INCOMPATIBILITY OF LOGNORMAL FORWARD-LIBOR AND SWAP MARKET MODELS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE,
FACULTY OF SCIENCE
AT THE UNIVERSITY OF CAPE TOWN
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR A DEGREE OF
MASTERS OF SCIENCE IN INFORMATION TECHNOLOGY

By
Wayne S. Goschen
September 2005

Supervised by
Prof. Ken MacGregor

Wayne S. Goschen

Abstract

The lognormal forward-Libor and Swap market models were formulated to price caps and swaptions. However, the prices computed by these two models, under equivalent measures, are reported to be unequal. This study investigates this incompatibility by computing the prices of caps and swaptions under both the forward Libor measure and the forward Swap measure, in both the Libor and Swap market models. This was done by building a computer program that implements the Monte Carlo versions of the models, using data from caps and swaptions traded in the South African market.

It was found that the actual price of caps, using the same implied volatility, were simulated accurately in both the Libor and Swap market models under both the forward Libor measure and the forward Swap measure. On the other hand, although the actual swaption prices were also simulated accurately in both the Libor and Swap market models under both the forward Libor measure and the forward Swap measure, a different implied volatility was used for each model. Therefore, the swaption price computed by the Libor market model was inconsistent with the price generated by the Swap market model; the two models are indeed incompatible.

In order to price interest rate derivatives consistently, either the Libor market model or the Swap market model must be chosen. Since the Libor market model priced consistently under the two forward measures, and the time taken to simulate a price in the Libor market model was much less than in the Swap market model, the practice in the market to use the Libor market model in favour of the Swap market model is justified.

Acknowledgements

I would like to thank the following:

- Dr. Michelle Kuttel, Department of Computer Science, U.C.T., for instigating this project and Dr. Peter Ouwehand, Department of Mathematics, U.C.T., for providing the financial mathematical knowledge to formulate the problem.
- Prof. Bradfield, of Cadiz, for allowing access to the resources at Cadiz and Brett Dugmore, of Cadiz, for giving me data on JIBAR rates, and cap and swaption trades. Brett Dugmore also volunteered, through discussions, valuable insight into the practice of trading interest rate derivatives.
- Rory Mackay from Investec, as the originator of the data.
- Richard Sang, of NedBank, and Crispin Swainston-Harrison, of SA Home Loans, for giving me several years of historical JIBAR data.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1 Introduction

The two so-called market models employed in the financial derivative industry are the Libor market model and the Swap market model. In developed markets, outside the borders of South Africa, the market models are one of the most popular and promising families of interest rate models (Brigo and Mercurio [BM01]). The reason for these models being so popular lies in the agreement between such models and well-established market formulas for two basic financial derivative products. The Libor market model prices caps with the Black cap formula (Black [Bla76]), which is the standard formula employed in the cap market. Similarly, the Swap market model prices swaptions with the Black swaption formula (also from Black [Bla76]), which is the standard formula in the swaption market. Libor is an acronym for London Interbank Offered Rate.

In South Africa, the cap and swaption markets are not well developed (Taylor and Brickhill [TB05]). Elsewhere in the world, the cap and swaption markets are the two main markets in interest-rate-options trading. In these markets, it is therefore important for the market models to be compatible with standard market formulas. However, even with full mathematical rigor given separately to the caps and swaptions formulas, there is a now classic problem: the Libor market model and Swap market models are not compatible (Brigo and Mercurio [BM01], Rebonato [Reb03]). If forward Libor rates are lognormal each under its own measure, as assumed by the Libor market model, forward swap rates cannot be lognormal at the same time under their measure, as assumed by the Swap market model. In other words, if a Libor market model is calibrated to market caplet prices, it allows swaptions to be priced, but those swaption prices do not correspond to the market swaption prices.

In this study, we investigate this incompatibility by simulating the prices of caps and swaptions under their own and under each other's forward measures, and then compare the computed prices. The options were taken from a set recently traded on one day in the South African interest rate derivative market. The method chosen for the computation of the prices was Monte Carlo simulation.

Chapter 2

Historical Perspective and Background

2 Historical Perspective and Background

This chapter discusses the background to interest rate model development and puts market models in a historical perspective. We begin with the famous Black and Scholes model, and discuss the role a close relation of it, the Black model, plays in the pricing of interest rate derivatives. Models evolved to modeling the term structure, and then to fitting the model to the yield curve, discussed in the following two sections. Deficiencies in these models to recover exactly the Black price, lead to the development of the market models, introduced in the final section.

2.1 Black and Scholes Model

Black and Scholes [BS73] made a major breakthrough in the arena of pricing of financial derivatives when they derived a differential equation that must be satisfied by the price of any derivative security dependent on a non-dividend-paying stock. They used the equation to derive values for European call and put options on the stock. A security is a financial asset, while stock is a fixed interest security issued by a government or an institution in fixed units. A non-dividend-paying stock may also mean a zero-coupon stock where payment is only made at the end of the life the security. An option gives its owner the right, not the obligation, to buy or sell the underlying (such as stock) at a particular date (maturity date) at a particular price (the exercise price). A European option is an option that can only be exercised at the maturity date of the option, in contrast to an American option, which can be exercised at any time during the life of an option.

Hull [Hul93] describe how Black and Scholes [BS73] obtained their equation by assuming that the stock price, S, follows a Wiener process, W, in the stochastic equation given in Eq. 2.1. A Weiner process is a particular type of Markov stochastic process that follows a geometric Brownian motion. Any variable whose value changes over time in an uncertain way is said to

follow a stochastic process. A Markov process is a particular type of stochastic process where only the present value of a variable is relevant to predicting the future.

A variable has a lognormal distribution if the natural logarithm of the variable is normally distributed. Black and Scholes [BS73] assumed that stock prices were log normally distributed, which means that if S is the stock price, $\ln S \sim \Phi [m, d]$ where Φ denotes a normal distribution with mean, m, and standard deviation, d. With stock prices, d (the uncertainty) is proportional to the square root of how far ahead a prediction is made.

Black and Scholes [BS73] also assumed that the short selling of securities with full use of proceeds is permitted, but security trading is continuous, and there are no transaction costs. Another assumption was that there are *no riskless arbitrage* opportunities (the arbitrage argument) and the *risk-free rate of interest* is constant and the same for all maturities.

The derivation of the Black and Scholes formula is discussed in any elementary textbook on financial derivatives, such as Hull [Hul93], and will not be rehashed in the project. For this discusion, it is sufficient to say that in their derivation they assumed that the stock price S is governed by the stochastic equation

$$dS = \mu S\, dt + \sigma S dW, \qquad \text{Eq. 2.1}$$

where μ is the expected rate of return on the stock and σ^2 is the variance of the proportional change in the stock price. By supposing that f is the price of a derivative security with the stock, S, as underlying, and using Ito's lemma, and with r as the risk-free interest rate, Black and Scholes derived a differential equation, equivalently given in Hull [Hul93] as

$$\frac{\partial f}{\partial t} + rS\frac{\partial f}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 f}{\partial S^2} = r f. \qquad \text{Eq. 2.2}$$

This equation is known as the Black-Sholes differential equation. Black and Scholes succeeded in solving the differential equation for the prices of European call and put options. The equation has many solutions corresponding to the different derivative securities (with price f) that can be defined with S as the underlying variable.

2.2 Black Model

Enhancements on the Black-Scholes formulae soon followed, all with the basic assumption that the underlying factor is log normally distributed. Variations of this formula were applied to financial derivatives such as options on futures, options on foreign exchange, and options on interest rates. Three years after the publication of the Black-Scholes formula, Black published a paper (Black [Bla76]) that obtained an almost similar pricing formula for a call option. Despite the superficial similarities between the two formulas, there was one fundamental difference in that the volatility of relevance was now the volatility of the forward price (as opposed to the spot price of Black and Scholes). This opened the way to further developments.

The Black formula could be applied to the pricing of a caplet and a swaption, discussed below. The formulas of Black are important to this study, because the Libor market model allows caps to be priced directly by the Black's cap formula and the Swap market model allows swaps to be priced directly by Black's swap formula. Our discussion follows that of Brigo and Mercurio [BM01].

This basic assumption, that the underlying rates were log normally distributed, lead to inconsistencies of these early models. For example, it could be shown that if a 1-year forward Libor rate is lognormal then a 2-year forward Libor rate cannot also be lognormal. A similar problem also arose with swap rates. However, by luck or insight, the trading practice continued to use the Black formula for swap and forward rates (Rebanato [Reb03]), despite having no theoretical grounding. This approach would later be theoretically justified (the use of Black formulas in these situations were later rigorously proved), but at that time they turned it into a market standard, which eventually lead to the modern Libor market models, as described below. Rebonato [Reb03]), even goes on to state that without these fortuitous choices made during the establishment of the standard, the "modern" pricing approach might have had a completely different structure. Hence, the history of the development of the models is important in the understanding of them, and so will be described in this introduction.

Another disadvantage of the models was that the volatility used must in general be different for different deals (Hull [Hul93]); it was not easy to find a way of relating the volatility used for one option to that used by another. Another serious drawback was that American bond options and other types of interest rate derivative securities could not be valued.

2.3 Term Structure models

The Black and Scholes and Black models were used to describe the movements over time of a single variable, such as a stock price or interest rate. A more sophisticated approach was to model the changes in the whole yield curve. A yield curve is a curve on a graph in which the yields of many fixed securities of the same type, but different maturities, are plotted against the length of time they have to run to maturity (used as a gauge to evaluate the future of interest rates). The yield is the income from an investment, which can be expressed as the difference between the nominal price of the security and its market price. These were known as yield curve models, or term structure models. These models generally start with a plausible stochastic process for the short-term rate, r, in a risk-neutral world and exploring what the process implies for bond prices and option prices.

A general form of the risk-free process for r is

$$dr = m(r)\,dt + s(r)\,dW ,$$

where the instantaneous drift, m, and the instantaneous standard deviation, s are assumed to be functions of r, but independent of time. The Wiener process is symbolized by dW. The process is therefore dependent on a single factor.

A risk-neutral valuation compatible with the Black-Scholes equation leads (Hull [Hul93]) to the value, f, of an interest rate security

$$f = \hat{E}[e^{-\check{r}(T-t)} f_T] ,$$

where $\hat{E}$ denotes the expected value in a risk-neutral world, $\check{r}$ is the average value of r in the time interval between t and T. The payoff at time T is given by f_T. By a series of algebraic steps, the term structure of interest rates can be obtained (not given here).

Historically, the first internally consistent term structure models were by Vasicek [Vas77] and Cox, Ingersoll and Ross [CIR85]. Rendleman and Bartter [RB80] also proposed a model where they assumed

$$dr = Mr\,dt + Sr\,dW .$$

This meant that r followed a geometric Brownian motion with a constant expected growth rate of M and a constant volatility of S in a risk-neutral world. However, their model did not capture

the pull-to-par, or mean reversion, phenomena of interest rate, where interest rates are pulled back to some long-term average.

A model that did incorporate mean reversion was that of Vasicek [Vas77], who proposed that

$$dr = a(b-r)\,dt + \sigma\,dW,$$

where a, b and σ were constants. The short rate, r, is pulled to a level b at a rate a, with a superimposed normally distributed stochastic term $\sigma\,dW$.

Jamshidian [Jam89] showed that the prices of options on discount bonds, and coupon-bearing bonds, could be obtained using the Vasicek [Vas77] model.

One theoretical problem of the Visicek model was that it allowed interest rates to go negative. Cox, Ingersoll and Ross [CIR85] solved this problem by using

$$dr = a(b-r)\,dt + \sigma\sqrt{r}\,dW,$$

where the stochastic term has a standard deviation proportional to $\sqrt{r}$.

2.4 No Arbitrage Models

A fundamental problem affected these models (Rebonato [Reb03]). Because of their stationary nature, (a, b and σ were all constants), they could not recover for an arbitrary observed yield curve. Moreover, when pricing a derivative security the models did not price the underlying correctly (Rebonato[Reb03]). Models that followed in order to fit the yield curve were known as no arbitrage models.

The first of these were by Ho and Lee [HL86]. They used

$$dr = \theta(t)\,dt + \sigma\,dW,$$

where σ is again the standard deviation of the short rate, and is constant, and θ(t) is chosen to fit the initial term structure. This term was concerned with the market price of risk.

This model, however, has no mean reversion (among other problems), so Hull and White [HW90] proposed that

$$dr = (\theta(t) - ar)\,dt + \sigma\,dW,$$

where the new constant, a, is include with an extra degree of freedom. This lead to the fitting of the yield curve, and interest rate options could be priced with confidence. In addition, the underlying bonds were also priced correctly.

Around the same time other models were produced Black, Dermon and Toy [BDT90] and Black and Karisinski [BK91]. Models such as these, and others, were essentially Markov models driven by the short rate.

Not long after, Heath, Jarrow and Morton [HJM92] published the first non-Markovian model. Their approach was to specify the volatilities of all instantaneous forward rates at all future times. That is, they provided a framework for modeling the term structure of interest rates; they modeled the entire yield curve. Their risk-neutral non-arbitrage process for the price of a discount at time t with a unit principal amount maturing at time T was

$$dP(t, T) = r(t)\, P(t, T)\, dt + \upsilon(t, T)\ P(t, T)\ dW(t),$$

where r(t) is the short term risk-free interest rate at time t, $\upsilon(t, T)$ is the volatility of P(t, T) and $dW(t)$ the Weiner process driving term structure movements.

The main results of this approach were (Rebonato [Reb03]):

- The instantaneous forward rates can be used as the fundamental building blocks of the yield curve. This ensures that the market yield curve can always be recovered.
- All no-arbitrage previous short-rate-based or bond-price-based models can be regarded as special cases of the HJM approach.
- The non-arbitrage drifts of the forward rates are uniquely specified once their volatilities and correlations are assigned. The apparent ability of the previous models to do the same was just a consequence of them using a special set of coordinates.
- The dynamics of the yield curve is non-Markovan, which means that an up-down shock does not give rise to the same yield curve as a down-up shock.

For the traders, one disadvantage of using the HJM model was that since instantaneous forward rates are not directly observable, nor linked to the price of any instrument, the calibration of the HJM model was difficult. Another difficulty was that since the pricing of caplets was still linked to the Black model with lognormal forward rates, it would be natural to impose lognormal behavior to the instantaneous forward rates. Unfortunately, in that model lognormal forward rates were guaranteed to reach infinity with the probability of one in a finite time. The HJM was the prototype for the market models, discussed in the following section.

2.5 Market Models

Using the above approach (Section 2.4), each caplet and swaption in a deal set was priced by the Black formula on its own, and therefore lognormal under its own measure. This meant that these options were independently and simultaneously assumed to be lognormally distributed. This is adequate to price the caplets and swaptions in isolation, but inadequate for more complex products where the payoff depends on the joint probability of the realization of several individual instruments. The market models address this problem.

The market models were designed to recover *exactly* the market-standard (Black) prices of all the compatible lognormal caplets and swaptions combined in a product, while at the same time specifying a coherent and desirable dynamics for all underlying forward rates. These are models of forward rate behavior (Libor, Swap), which are consistent with their corresponding Black formulae. They are called "market" because they take real market interest rates.

The first of two varieties of market models was the lognormal forward-Libor model (Libor market model, abbreviated in this document as LFM), originally proposed by Brace *et. al.* [BGM96], Miltersen *et. al.* [MSS97], Musiela and Rutkowski [MR97], and Jamshidian [Jam97]. The models were developed in the framework of Heath *et. al.* [HJM92]. A concise overview of the mathematical formulation of the early market models by the above authors are given in the paper by Rutkowski [Rut99]. In the same paper Rutkowski also mentions how Miltersen *et. al.* [MSS97] were the first to attempt a rigorous construction of a lognormal model of forward Libor rates. These early models are also discussed comprehensively in Rebonato [Reb98]. Within the Libor market model N forward Libor rates are modeled as lognormal processes under their respective forward (or spot) measure, with the forward Libor L_k satisfying

$$\frac{dL_k(t)}{L_k(t)} = \sigma_k(t)\, dW^{k+1}(t)\,, \qquad k = 1,\ldots N\,,$$

where $\sigma_k(t)$ denotes the instantaneous volatility function and W^{k+1} denotes the Brownian motion under the k^{th} forward measure. The forward measure is associated with a discount bond that matures at t_{k+1}, which is the payment time of the k^{th} Libor deposit. This choice of numeraire, which lead to the above stochastic equation of forward Libor rates having no drift term, means that this measure is a martingale under its forward probability measure. In other words, given the Q^N processes of these N forward Libor rates, all numeraire-dominated bond prices can be

determined. Under the probability measure Q^N associated with the numeraire $B_k(t)$, the Libor rate $dL_k(t)$ has zero drift.

The second variety was the lognormal forward-Swap model (or Swap market model, abbreviated in this document as LSM) by Jamshidian [Jam97]. Within the Swap market model, N forward swap rates are modeled as lognormal processes under their respective forward (or spot) measures, with the forward swap rate $S_{n,N}$ satisfying

$$\frac{dS_{n,N}(t)}{S_{n,N}(t)} = \sigma_{n,N}(t)\, dW^{n,N}(t)\ , \qquad n = 1,\ldots N\ ,$$

where, again, $\sigma_{n,N}(t)$ denotes the instantaneous volatility function and $W^{n,N}$ denotes the Brownian motion under the n^{th} forward swap measure. The forward measure is associated with a portfolio of discount bonds, weighted by the respective count fractions and with maturity times corresponding to the payment times of the swap. As with the Libor rates, this choice of numeraire leads to the above stochastic equation of forward swap rates having no drift term, which means that this measure is a martingale under its forward swap probability measure.

In a recent experiment on the performance of Bermudan swaptions, Pietersz and Pelsser [PP05] compared single factor Markov-functional and multi-factor market models. They found that the single factor model was sufficient. Hence, in this study, only single factor models will be considered. The formulations of these models are described in Chapter 5, after a discussion on spot and forward interest rates (Chapter 3), and caps and swaptions (Chapter 4).

Chapter 3

Interest Rates

3 Interest Rates

Forward interest rates are the underlying of caps and swaptions. These forward rates are based on interest rates that are simply compounding, such as the Libor and the JIBAR (Johannesburg Interbank Acceptance Rate) in South Africa (JIBAR rates may be substituted for Libor rates wherever found). This chapter introduces interest rates, beginning with continually and simply compounding interest rates, and then discusses their respective forward rates. The Swap market model is built on forward swap rates, so swap rates and forward swap rates are also briefly introduced.

3.1 Spot Interest Rates

Following Brigo and Mercurio [BM01], we begin by considering an initial investment of a unit amount of money in a money market account (bank account) at a risk-free rate where profit is accrued continuously. We define A(t) to be the value of the account at $t \geq 0$, with $A(0) = 1$. The account then evolves according to

$$dA(t) = r_t A(t)\, dt,$$

where r_t is a positive function of time, and is usually referred to as the instantaneous short rate, or just the short rate. This yields a value for the account at time t

$$B(t,T) = \exp\left(\int_t^T r_s\, ds\right). \qquad \text{Eq. 3.1}$$

If one assumes that the interest rate, r, and the account, A, are deterministic, this leads to the value of one unit of currency payable at time T, as seen from time t, to be

$$D(t, T) = \frac{A(t)}{A(T)} = \exp\left(-\int_t^T r_s \, ds \right). \qquad \text{Eq. 3.2}$$

This is defined as the discount factor.

A zero coupon (pure discount) bond is a contract that guarantees its holder the payment of one unit of currency at a maturity time T, with no intermediate payment. The contract value at time t < T is denoted by B(t, T). Since there is a fixed payment at time T, B(T, T) = 1. If the rates, r, are deterministic, then D(t, T) is also deterministic and D(t, T) = B(t, T) for each pair of (t, T).

In the modeling of interest rates, the interest rates themselves are variable. This leads to the requirement that r evolves through a stochastic process, so that the money market account and the discount factors (Eq. 3.1 and Eq. 3.2) must also be modeled as stochastic processes. In this scenario, under a particular probability measure, B(t, T) is the expectation of the random variable D(t, T).

3.2 Spot Libor Rates

Libor rates are simply compounding interest rates, as opposed to continually compounding interest rates. Simply compounding interest rates are interest rates that are compounded over a certain fixed period, such as 3 months. Since Libor rates are central to Market Models, let us define them (after Brigo and Mecurio [BM01]). A simply-compounded interest rate, L(t, T), is a constant rate at which an investment has to be made to produce one unit of currency at maturity, T, starting from P(t, T) units of currency at time t, accruing proportionally to the investment time, and is defined as

$$L(t, T) \equiv \frac{1 - B(t, T)}{\delta(t,T) * B(t, T)}.$$

From the above equation one may deduce that Libor rates are linked to zero coupon bond prices by the local day-count convention for computing, δ(t,T), the year fraction between t and T.

Reversing the above equation results in bond prices that may be express in terms of Libor rates

$$B(t, T) \equiv \frac{1}{1 + \delta(t,T)^* L(t, T)}.$$

3.3 Forward Interest Rates

We define forward rates in terms of a forward rate agreement (FRA). A FRA is a contract that gives its holder an interest rate payment at time t, for the period between T and M, where T_e is the expiry time $T_e > t$ and T is the maturity time $T > T_e$. At maturity, T, a fixed payment based on a fixed rate K is exchanged against a floating payment based on the spot rate $S(T_e,T)$, resetting in T_e and with maturity T. With a nominal value N, one receives $\delta(T_e,T)^*K^*N$ units of currency and pays $\delta(T_e,T)^* S(T_e,T)^*N$. This gives the value of the contract at maturity

$$FRA(T_e, T)= \delta(T_e,T)^*(K - S(T_e,T))^*N .$$

At time t it can be shown that the total value of the contract is

$$FRS(t,\ldots) = N\{B(t,T) * \delta(T_e,T)^*K - B(t, T_e) + B(t,T)) . \qquad \text{Eq. 3.3}$$

By equating this to zero gives a fair contract at time t, resulting in the equation for simply-compounded forward interest rates

$$F(t; T_e, T) \equiv \frac{1}{\delta(T_e,T)} \left(\frac{B(t, T_e)}{B(t, T)} - 1 \right) . \qquad \text{Eq. 3.4}$$

In the region where the forward rate $F(t; T_e, T)$ nears expiry, we get the instantaneous forward interest rate

$$f(t,T) = \lim_{T \to Te} F(t; T_e, T) = - \frac{\partial B(t,T)}{\partial T} ,$$

which gives

$$B(t,T) = \exp\left(\int_t^T f(t,u)\,du\right) .$$

3.4 Forward Libor Rates

We begin by defining a finite set of dates representing the dates on which the derivatives mature, called the tenor structure,

$$T_i < T_{i+1} < \ldots < T_N < T_{N+1}, \quad i = 1,\ldots,N .$$

We also define a tenor,

$$\delta_i = T_{i+1} - T_i, \quad i = 1,\ldots,N ,$$

as the period between instrument payoffs, usually equal to 3 or 6 months. We will assume that all δ_i are equal, $\delta_i = \delta_0$, (example, 0,25 for 3 months or 0.5 for 6 months) although in practice they will be different due to the influence of the day-count basis. We have it so that at each tenor date, T_i, a zero-coupon bond matures. The price of the bond at time t is denoted by $B_i(t)$, where $t \in [0, Ti]$ and $B_i(T_i) = 1$.

The forward Libor rate at time t for the accrual period $[T_i, T_{i+1}]$, $t \leq T_i$, is related to a series of bond prices $B_i(t)$ by the industry formula

$$L_i(t) = \frac{1}{\delta_i}\left(\frac{B_i(t)}{B_{i+1}(t)}\right) - 1, \qquad 0 \leq t \leq Ti,\ i = 1,\ldots,N . \qquad \text{Eq. 3.5}$$

By the notation $B_i(t)$ we imply a Bond priced at i for maturity at time t.

By setting $L_i(t) = L_i(T_i)$ for $t > T_i$ (extending the definition of L_i beyond the i-th tenor date), the above equation may be manipulated to give the price of any bond, B_n, for $n > i$, that has not yet matured. Thus at any tenor date, T_i, the inversion gives

$$B_n(T_i) = \prod_{j=i}^{n-1} \frac{1}{1 + \delta_j L_j(t)}, \quad n = i+1,\ldots,N+1 . \qquad \text{Eq. 3.6}$$

More generally, for an arbitrary date, t, $T_i < t < T_{i+1}$, the forward rates, $L_i(t)$, do not determine the bond prices because they do not determine the discount factors at times between the tenor dates. We must then discount the bond's payment at T_n back to T_{i+1} and then multiply it by a factor to adjust the time T_{i+1} to t. This factor turns out to be the current price of the shortest maturity bond.

Thus if we want to price a bond between the accrual periods, $T_i < t < T_{i+1}$, we must use

$$B_n(t) = B_{\eta(t)}(t) \prod_{j=\eta(t)}^{n-1} \frac{1}{1 + \delta_j L_j(t)}, \quad 0 \le t \le T_n , \qquad \text{Eq. 3.7}$$

where $B_{\eta(t)}(t)$ is the current price of the shortest maturity bond.

3.5 Swap rates and Forward Swap Rates

An Interest Rate Swap is a generalization of a FRA, discussed in Section 3.3. A swap allows a company to interchange fixed rate payments for the floating rate payments of another company. This works by allowing the first company to pay floating (for example) Libor rates to a bank and receive fixed rates from the same bank, while the second company receives floating Libor rates from the same bank and pays a fixed rate to it.

Consider a typical forward-start interest rate swap contract, with principal N, where two parties agree to exchange the floating Libor rate $\{ L_n(t),\ldots, L_{N-1}(t) \}$ for a fixed rate at dates $\{T_{n+1},\ldots, T_N \}$. In this case the floating-leg rate resets at dates $T_n, T_{n+1},\ldots, T_{N-1}$ and pays at dates $T_{n+1},\ldots T_N$. The party who pays the fixed leg and receives the floating leg is called the Payer (PFS) while the party that receives the fixed and pays the floating is called the Receiver (RFS). Assume also, that the fixed-rate payments and floating rate payments occur at the same dates, and with the same year fractions, δ_i, and that the fixed interest rate is K. Then at each payment date T_i, the fixed leg pays the amount

$$N\delta_i K$$

while the floating leg pays the amount

$$N\delta_i L(T_{i-1}, T_i) .$$

If we consider the RFS, the discounted payoff at time $t < T_n$ is

$$\sum_{i=n+1}^{N} D(t, Ti)\, N\, \delta_i\, (K - L(T_{i-1}, T_i)) .$$

From Section 3.3 this contract may be viewed as a portfolio of FRAs.

$$\begin{aligned} RFS(t, \ldots) &= \sum_{i=n+1}^{N} FRA(t, \ldots) . \\ &= N \sum_{i=n+1}^{N} \delta_i\, B(t, Ti)\, (K - L(T_{i-1}, T_i)) . \\ &= -N\, B(t,T_n) + NP(t, T_N) + N \sum_{i=n+1}^{N} \delta_i\, K\, B(t, T_i) . \end{aligned} \qquad \text{Eq. 3.8}$$

A similar formula may be derived for PFS.

We require that the interest rate swap be a fair contract at the present time, which implies a value for the fixed rate K for which RFS(t, …) = 0. Some manipulation yields

$$S_{n,N}(t) = \frac{B(t, T_n) - B(t, T_N)}{\sum_{i=n}^{N} \delta_i\, K\, B(t, T_i)} .$$

Recall that B_n (t) can be given as a function of L(t)

$$B_n(t) = B_{n+j}(t) \prod_{j=1}^{N} \frac{1}{1 + \delta_i L_i(t)}.$$

Substituting this into the $S_{n,N}(t)$ given above, it can be seen that the forward swap rate depends on several forward Libor rates, giving the forward swap rate formula

$$S_{n,N}(t) = \frac{1 - \prod_{j=n}^{N} \frac{1}{1 + \delta_j L_j(t)}}{\sum_{i=n}^{N} \prod_{j=i+1}^{N} \frac{1}{1 + \delta_j L_j(t)}}. \qquad \text{Eq. 3.9}$$

The term $\frac{1}{1 + \delta_j L_j(t)}$ is called the forward discount factor.

Eq. 3.9 can be expressed in terms of an exponential function of the underlying forward rates

$$S_{n,N}(t) = \exp(\psi(L_{n+1}(t), L_{n+2}(t), \ldots, L_N(t))) \qquad \text{Eq. 3.10}$$

This indicates that the forward swap rate is actually a nonlinear function of the underlying forward Libor rates so that the variance of a swap rate is a function of both the variance and covariances (or correlations) of forward Libor rates. Hence, it cannot be lognormal if the Libor rates are lognormal. In contrast, from the Bond price equation in Section 5 it can be seen that caplet prices only depend on the conditional variance of one forward Libor rate, which does follow a lognormal process. This theoretical result is at the core of this study, and shown to be true through the computation of the prices of caps and swaption prices.

Chapter 4

Caps and Swaptions

4 Caps and Swaptions

The market models were designed to price caps and swaptions. This chapter derives the mathematical formulation of caps and swaptions, and shows their relation to Black's formulas.

4.1 Caps

A cap is a collection of caplets. A caplet is a call option on an interest rate, which will provide a positive payoff if the prevailing Libor rates goes above a certain level, the strike rate. An option to buy is known as a call option and is usually purchased in the expectation of a rising price. With caps it is assumed that the future Libor rates are also log normally distributed. Again, following Brigo and Mercurio [BM01], it may be viewed as a pay interest rate swap that is executed only if it has positive value, and is given by

$$\sum_{i=n+1}^{N} D(t, T_i)\, N\delta_j\, [L(T_{i+1}, T_i) - K]^+ ,$$

where the price of one caplet is just

$$D(t, T_i)\, N\delta_j\, [L(T_{i+1}, T_i) - K]^+ .$$

The notation $[L(T_{i+1}, T_i) - K]^+$ is equivalent to

$$\max(L(T_{i+1}, T_i) - K, 0) ,$$

which means that its value is $L(T_{i+1}, T_i) - K$ if $L(T_{i+1}, T_i) > K$, or 0 otherwise.

Looking at a cap as forward rate agreements with P as a zero-coupon bond and L as the forward Libor rate, this gives

$$N \sum_{i=n+1}^{N} \delta_j B(t, T_i) [L(T_{i+1}, T_i) - K]^+ ,$$

which gives the Black formula for a cap, as

$$Cap^{Black}(0,..) = N \sum_{i=n+1}^{N} \delta_j B(0, T_i) [L(0, T_i) \Phi(d_1) - K \Phi(d_2)]^+ ,$$

with

$$d_1 = \frac{\ln (F_n(0)/K) + 0.5 \sigma^2_n}{\sigma_n} \text{ and}$$

$$d_2 = \frac{\ln (F_n(0)/K) + 0.5 \sigma^2_n}{\sigma_n} = d_1 - \sigma \sqrt{t T_{i-1}} .$$

where Φ is the standard Gaussian cumulative distribution function.

4.2 Swaptions

A swaption is an option to enter into a swap at some fixed swap rate at some future date. The basic assumption is that swap rates are also log normally distributed. There are two types of swaptions, the payer and receiver swaptions.

Let us consider the payer contract for a European option. A European payer swaption is an option giving the right (not obligation) to enter a payer interest rate swap (IRS), described in Section 3.5, at a given future date, the swaption maturity. We can write the discounted payoff of a payer swaption by considering the value of the underlying payer IRS at it first reset date, T_n, which is also taken to be the swaption maturity. Hence from Eq. 3.8, this is

$$\sum_{i=n+1}^{N} N\,\delta_j\,D(T_n, T_i)\,[L(T_{i-1}, T_i) - K]^+.$$

To compute the value of the PFS we take the expectation at time t

$$PFS(t, \ldots) = E\left(\sum_{i=n+1}^{N} N\,\delta_i\,D(T_n, T_i)\,[L(T_{i-1}, T_i) - K]^+\right)$$

$$= N\sum_{i=n+1}^{N} \delta_i\,B(t, T_i)\;E\left([L(T_{i-1}, T_i) - K]^+\right)$$

$$= N\sum_{i=n+1}^{N} \left(\delta_i\,B(t, T_i)\;[L(T_{i-1}, T_i) - K]^+\right) \qquad \text{Eq. 4.1}$$

$$= N\sum_{i=n+1}^{N} \left(B(t, T_{i-1}) \;-(1 + \delta_i K)\,B(t, T_i)\right).$$

In particular the payoff risk-neutral expectation at time T_n is

$$PFS = N\sum_{i=n+1}^{N} \delta_i\,B(t, T_i)\;[L(T_n, T_i) - K]^+. \qquad \text{Eq. 4.2}$$

It follows that the discounted payoff at time t is

$$PFS = N\;D(t, T_n)\sum_{i=n+1}^{N} \delta_i\,B(T_n, T_i)\;[L(T_n, T_i) - K]^+.$$

The option will be exercised only if this value is positive

$$PFS = N\ D(T_n, T_i) \left(\sum_{i=n+1}^{N} B(T_n, T_i)\ \delta_j\ (L(T_{i-1}, T_i) - K) \right)^{+} .$$

Using the equation for forward swap rates, this may be expressed as

$$D(0, T_n)\ [S_{n,N}(T_n) - K]^{+}\ N \sum_{i=n+1}^{N} \delta_i B(T_n, T_i)$$

or

$$D(0, T_n)\ \max(S_{n,N}(T_n) - K, 0)\ N \sum_{i=n+1}^{N} \delta_i\ B(T_n, T_i)\ . \qquad \text{Eq. 4.3}$$

This is the Black formula for a payer swaption, with S now the forward swap rate, given below (without displaying formal parameters)

$$Swptn^{Black}(..) = N\ [S(0, T_i)\ \Phi(d_1) - K\ \Phi(d_2)] \sum_{i=n+1}^{N} \tau_i\ P(0, T_i)\ ,$$

with

$$d_1 = \frac{\ln (S_n(0)/K) + 0.5\ \sigma^2_n}{\sigma_n} \quad \text{and}$$

$$d_2 = \frac{\ln (S_n(0)/K) + 0.5\ \sigma^2_n}{\sigma_n} = d_1 - \sigma \sqrt{t T_{i-1}}\ .$$

Similarly for the RFS (receive fixed leg, pay floating leg).

Chapter 5

Market Models

5 Market Models

This chapter discusses the mathematical formulation of lognormal forward-Libor and Swap market models. For each of these models, the formulas for pricing caps and swaptions are derived. The derivations are by no means mathematically rigorous; only major steps in the construction of the models are given.

5.1 The lognormal forward-Libor Market Model

In order to describe the main features of a Libor market model we follow Glasserman [Gla04], who based his description on Jamshidian [Jam97] and Musiela and Rutkowski [MR97], and that of Brigo and Mecurio [BM01]. Other similar descriptions of the Libor market models may be found in London [Lon04].

For the Libor market model it is assumed that the forward Libor rates follow an Ito process (Appendix C), and so the dynamics are given by

$$\frac{dL_k(t)}{L_k(t)} = \mu_k(t)\,dt + \sigma_k(t)\,dW_k(t), \quad 0 \le t \le T_k \quad k = 1,\ldots,N, \qquad \text{Eq. 5.1}$$

where $\mu_k(t)$ is the drift function of the forward Libor rate, $\sigma_k(t)$ is the volatility function and $W_k(t)$ is again a standard Brownian motion.

Consider a Libor market model based on N forward Libor rates

$$L_k(t), \quad k = 1,\ldots,N,$$

which is "alive" up to time T_{k-1}, where it coincides with the simply compounded spot rate prevailing at time S for maturity T.

In this notation $L_k(t) = L(t; T_{k-1}, T_k)$ denotes a simply compounded forward Libor rate at time t for expiry T_{k-1} and maturity T_k, where expiry and maturity are related pairs. Let these pairs be taken from a set of dates $T_0, T_1, \ldots, T_N$. Let the year fractions, or tenors, belong to a set $\delta_0, \delta_1, \ldots \delta_N$ and be denoted by $\delta_k = T_k - T_{k-1}$ for $k > 0$. The tenor from settlement to T_0 is denoted by δ_0. Time T_k are expressed in years from the current time, $t = 0$. We set $T_{-1} = 0$.

Consider now the T_k-forward measure, in other words the forward (adjusted) measure for the maturity T_k. This probability measure is associated with the $B(\cdot, T_k)$ numeraire, the price of a bond whose maturity coincides with the maturity of the forward rate. Under simple compounding it follows, by definition, that $L_k(t)B(t, T_k)$ is a tradable asset, given by

$$L_k(t)B(t, T_k) = (B(t, T_{k-1}) - B(t, T_k))/ \delta_k .$$

Thus, by definition of our measure associated with a numeraire, when its price is expressed with respect to the numeraire $B(\cdot, T_k)$, it has to be a martingale under the Q^k measure. But the price $L_k(t)B(t, T_k)$ of the tradable asset divided by this numeraire is simply $L_k(t)$ itself, it follows that $L_k(t)$ is a martingale under Q^k. Thus if $L_k(t)$ is modeled according to a diffusion process, its dynamics need to be driftless under the Q^k measure.

From the above argument, that since the price of the tradable asset $L_k(t)B(t, T_k)$ divided by the zero-coupon bond numeraire $B(t, T_k)$ is a martingale, and a diffusion process is being modeled, the dynamics are driftless and the *k*th forward rate evolves by

$$dL_k(t) = \sigma_k(t)\, L_k(t)\, dW_k(t) \quad , \quad \text{for } t \leq T_{k-1} . \qquad \text{Eq. 5.2}$$

In general, we need to know the dynamics of the forward rate $L_k(t)$ under a measure Q^i different from Q^k, for $t \leq \min(T_i, T_{k-1})$. Both the chosen numeraire and the forward rate being modeled must be "alive" in t. We also assume that each forward rate is driven by a standard Brownian motion W_k with time dependent lognormal volatility and a time-dependent correlations structure $E[dWi\, dWj] = \rho_{ij}\, dt$. Under this general framework the dynamics are given by Eq. 5.2.

Consider now the forward rate $L_k(t) = L(t; T_{k-1}, T_k)$ and suppose we wish to derive its dynamics first under the T_i-forward measure Q^i with $I < k$. We know from the above analysis that the dynamics under the T_k-forward measure Q^k has null drift, and so we can recover the dynamics under Q^i. This can be done by applying the change of numeraire formulas derived in Appendix A to the specific case of our forward rate dynamics.

Substituting the following into Eq. 10.7 gives

$$X = L_k(t) = L(t;T_{k-1}, T_k) ,$$
$$S = B(\cdot, T_k) ,$$
$$U = B(\cdot, T_i) ,$$

where the dynamics of X is driftless and is of the form given by Eq. 5.2, we obtain the percentage drift as

$$\mu^i(t) = - \frac{d \langle ln\, L_k(t), \ln(B(\cdot, T_k) / B(\cdot, T_i)) \rangle_t}{dt} . \qquad \text{Eq. 5.3}$$

Consider the case i < k, and recall that

$$ln \left(\frac{B(t, T_i)}{B(t, T_k)} \right) = ln \left(1/ \left(\prod_{j=k+1}^{i} (1 + \delta_j L_j(t)) \right) \right)$$
$$= - \sum_{j=i+1}^{k} ln\, (1 + \delta_j L_j(t)) .$$

which when substituted into Eq. 5.3 gives

$$\mu^i(t) = \sum_{j=i+1}^{k} \frac{d \langle ln\, L_k(t), \ln(1 + \delta_j L_j(t)) \rangle_t}{dt}$$

$$= \sum_{j=i+1}^{k} \frac{\delta_j \; d \langle ln\, L_k(t), \ln(L_j(t)) \rangle_t}{1 + \delta_j L_j(t) \; dt}$$

$$= \sum_{j=i+1}^{k} \frac{\rho_{j,k}\, \delta_j\, \sigma_k()\sigma_j()\, L_j(t)}{1 + \delta_j L_j(t)} .$$

Now consider the case i > k. The derivation is similar to i < k, but in this case the numeraire is a bond with maturity longer than the maturity of the forward rate being modeled. This yields

$$ln\left(\frac{B(t, T_i)}{B(t, T_k)}\right) = \sum_{j=k+1}^{i} ln\,(1 + \delta_j L_j(t)) \ ,$$

from which the drift emerges as

$$\mu^i(t) = \sum_{j=i+1}^{k} \frac{\rho_{j,k}\,\delta_j\,\sigma_k(t)\sigma_j(t)\,L_j(t)}{1 + \delta_j L_j(t)} \ . \qquad \text{Eq. 5.4}$$

Applying Ito's formula (Appendix C) gives us the dynamics of the forward Libor rates $L_k(t)$ under the forward adjusted measure Q^i, with the lognormal assumption, in the three cases i < k, i = k, and i > k

$$i < k,\ t \le T_i : dL_k(t) = \sigma_k(t)L_k(t) \sum_{j=i+1}^{k} \frac{\rho_{k,j}\,\delta_j\,\sigma_j(t)\,L_j(t)}{1 + \delta_j L_j(t)}\,dt + \sigma_k(t)L_k(t)\,dW_k(t) \ ,$$

$$i = k,\ t \le T_{k-1} : dL_k(t) = \sigma_k(t)L_k(t)\,dWk(t) \ ,$$

$$i > k,\ t \le T_{k-1} : dL_k(t) = -\sigma_k(t)L_k(t) \sum_{j=i+1}^{k} \frac{\rho_{k,j}\,\delta_j\,\sigma_j(t)\,L_j(t)}{1 + \delta_j L_j(t)}\,dt + \sigma_k(t)L_k(t)\,dW_k(t) \ ,$$

where $\sigma_k(t)$ and $W_k(t)$ are as before.

The above result implies that only the volatility functions, $\sigma_N(t)$, have to be determined in order to price an interest rate derivative. If it is assumed that $\sigma_N(t)$ is deterministic then $L_N(t)$ has lognormal distribution

$$L\,N(-\bar{\sigma}^2_N(t))/2,\ \bar{\sigma}^2_N(t)),$$

with

$$\bar{\sigma}_N(t) = \sqrt{\left(1/t \int_0^t \| \sigma_N(u) \|^2 \, du \right)} .$$

5.1.1 Caplet pricing in the LFM

Consider a caplet for the accrual period $[T_n, T_{n+1}]$ with a strike rate K. Recalling that a cap is a collection of caplets and that each caplet may be viewed as a call option on a simple forward rate, the underlying rate for this caplet is L_n where the value $L_n(T_n)$ is fixed at T_n. The caplet payoff at time t, $C_n(t)$, is

$$C_n(t) = \delta_n (L_n(T_n) - K)^+.$$

By the change of numeraire equation in Appendix A.1, this is equivalent to

$$C_n(t) = B_{n+1}(t)\, E^{n+1}[\delta_n (L_n(T_n) - K)^+],$$

where E^{n+1} is the expectation with respect to the forward measure Q^{n+1} associated with maturity T_{n+1}.

To find the initial value, $C_n(0)$, under the forward measure, Q^{n+1}, associated with maturity T_{n+1}, the martingale property applies to

$$\frac{C_n(t)}{B_{n+1}(t)} .$$

Using the numeraire equation again we must write, where E^{n+1} is the expectation under the Q^{n+1} measure,

$$C_n(0) = B_{n+1}(0)\, E^{n+1}\left[\frac{\delta_n\,(L_n(T_n) - K)^+}{B_{n+1}(T_{n+1})}\right]. \qquad \text{Eq. 5.5}$$

This formula lends itself to pricing by Monte Carlo simulation, discussed in Chapter 6 , where the expectation is taken as the average of the simulation.

Because $L_n(t)$ follows a driftless lognormal process under the Q^{n+1} measure, the above equation leads to the Black pricing formula for a caplet, and this caplet price is determined by the conditional variance (under the Q^{n+1} measure) of the forward Libor rate over the maturity of the caplet, which is equal to

$$\int_t^T \sigma_n(s)^2\, ds\,.$$

Thus

$$C(t, T_n, K) = \delta_n\, B_{n+1}(t)\,[\,(L_n(t)\,N(h) - K\,N(h-\xi_n))\,]\,,$$

where

$$h = \frac{\ln(L_n(t)/K) + 0.5\,\xi^2_n}{\xi_n} \qquad \text{and} \qquad \xi^2_n = \int_t^{T_n} \sigma_n(s)^2\, ds\,.$$

The price of a single cap can therefore be determined from the sum of caplets for different maturities.

5.1.2 Swaption pricing in the LFM

Recall that a payers swaption price may be considered as the expectation

$$PS = E^{n+1}\left[\, D(0,T_n)\max(S_{n,N} - K, 0)\sum_{i=n+1}^{N} \delta_i\, B(T_n, T_i)\right]\,,$$

where the expectation is taken with respect to the numeraire associated with the forward measure discussed above.

The above equation is equivalent to

$$PS = B(0,T_n) \quad E^{n+1}\left[\max(S_{n,N} - K, 0) \sum_{i=n+1}^{N} \delta_i B(T_n, T_i) \right].$$

From the Eq. 3.9,

$$S_{n,N}(t) = \frac{1 - \prod_{j=n}^{N} \frac{1}{1 + \delta_j L_j(t)}}{\sum_{i=n}^{N} \prod_{j=i+1}^{N} \frac{1}{1 + \delta_j L_j(t)}},$$

it is seen that the swap rate may be expressed in terms of spanning forward rates at time T_n,

$$L_{n+1}(T_n), L_{n+2}(T_n), \ldots, L_N(T_n).$$

From this it is obvious that the forward rate correlations impact the price, unlike in the case for the pricing of caplets.

An alternative approach is to take the payer swaption payoff at time T_n

$$PFS = N\left[\sum_{i=n+1}^{N} \delta_i B(t, T_i) \ (L(T_n, T_{i-1}) - K)^+ \right]$$

As with the caplets, to price this derivative at T_0 it is necessary to use the change of numeraire equation again. We can then write

$$C_n(0) = B_{n+1}(0)\, N\, E^{n+1} \frac{\left(\sum_{i=n+1}^{N} \delta_i B(t, T_i)\ (L(T_n, T_{i-1}) - K)^+ \right)}{B_{n+1}(T_{n+1})}, \qquad \text{Eq. 5.6}$$

where E^{n+1} is the expectation under the Q^{n+1} measure.

5.2 The lognormal forward-Swap Market Model

If it is assumed that the forward swap prices given in Eq. 3.9 follow a lognormal process (which follow Itô processes) then under the standard probability measure the dynamics of swap rates is given by

$$\frac{dS_{n,N}(t)}{S_{n,N}(t)} = \mu_{n,N}(t)\, dt + \sigma_{n,N}(t)\, dW_{n,N}(t), \quad 0 \le t \le T_n, \quad n = 1,\ldots,N. \qquad \text{Eq. 5.7}$$

Jamshidian [Jam97], also referred to in Glasserman and Zhao [GZ00], shows that the drift, $\mu_{n,N}(t)$, in Eq. 5.7 is given by

$$\mu_{n,N}(t) = -\frac{\sum_{i=n}^{N} \sum_{k=n+1}^{i} \delta\, S_k(t)\, \sigma_k(t) \prod_{j=n+1, j\neq k}^{i} (1 + \delta\, S_j(t))}{\sum_{i=n}^{N} \prod_{j=n+1}^{i} (1 + \delta\, S_j(t))}.$$

Given the N deterministic volatility functions $\sigma_{n,N}(t)$ of the forward swap rates, the N volatility functions of bond prices cannot be determined. As with the LFM, this indeterminacy is resolved by choosing a bond price as numeraire. Assuming lognormal dynamics, for the forward swap rate $dS_{n,N}(t)$ the most convenient choice of numeraire is the present value of a basis point

$$C_{n,N}(t) = \sum_{i=n+1}^{N} \delta_{n,N} B(t, T_i) \quad . \qquad \text{Eq. 5.8}$$

This numeraire is such that

$$C_{n,N}(t)\ S_{n,N}(t) = B(t, T_n) - B(t, T_N) ,$$

and this gives the price of a tradable asset, that, if expressed in $C_{n,N}(t)$ units, coincides with the forward swap rate. Thus, under the $Q^{n,N}$ forward swap measure the numeraire $C_{n,N}(t)$ leads to a driftless dynamics of $S_{n,N}(t)$.

If this $C_{n,N}(t)$ numeraire is introduced to above equation, and assuming no arbitrage, it can be shown that $S_{n,m(t)}$ follows a driftless lognormal process under the equivalent martingale,

$$\frac{dS_{n,N}(t)}{S_{n,N}(t)} = \sigma_{n,N}(t)\, dW_{n,N}(t) , \quad 0 \le t \le T_n , \quad n = 1,\ldots,N , \qquad \text{Eq. 5.9}$$

where $\sigma_{n,N}(t)$ is the deterministic instantaneous percentage forward swap rate volatility, and $W_{n,N}(t)$ is the standard Brownian motion under the $Q^{n,N}$ forward swap measure.

Eq. 5.9 gives the dynamics of the forward swap rate, and is known as the lognormal forward-Swap rate market model (LSM) since each swap rate $S_{n,N}(t)$ has a lognormal distribution under its forward swap measure $Q^{n,N}$ associated with numeraire $C_{n,N}(t)$.

5.2.1 Swaption pricing in the LSM

Consider the interest rate swap described in Section 3.5, in the discussion of swap rates and forward swap rates. It was shown that the swaption payoff for the payer leg was

$$D(0, T_n) \max(S_{n,N}(T_n) - K, 0)\ N \sum_{i=n+1}^{N} \delta_i B(T_n, T_i) .$$

Substituting $C_{n,N}(t)$ from Eq. 5.8 into this equation, and taking the risk free expectation of the discounted payoff under the forward swap measure, we get

$$E^Q [D(0, T_n) \max(S_{n,N}(T_n) - K, 0) C_{n,N}(T_n)] .$$

Realizing that $D(0,T_n) = 1/B(t,Tn)$ and $B(0) = 1$, $T=T_n$, and using the numeraire formula with $Z_T = \max(S_{n,N}(T_n) - K, 0)\ C_{n,N}(T_n)$, $U = B = Q$ and $N = C_{n,N}$ gives

$$E^Q \left(\frac{B(0) \max(S_{n,N}(T_n) - K, 0)\ C_{n,N}(T_n)}{C_{n,N}(T_n)\ B(T_n)} \right)$$

$$= E^{n,M} \left(\frac{\max(S_{n,N}(T_n) - K, 0)\ C_{n,N}(T_n)}{C_{n,N}(T_n)} \right)$$

and hence

$$PFS^{LSM} = E^Q [D(0, T_n) \max(S_{n,N}(T_n) - K, 0) C_{n,N}(T_n)]$$

$$= C_{n,N}(0)\ E^{n,M}\ [\max(S_{n,N}(T_n) - K, 0)]$$

$$= C_{n,N}(0)\ BL(\ldots) ,$$

where BL(…) is Black's formula defined in Section 4.2.

The LSM, discussed in the previous section, resulting in the Eq. 5.9 directly implies that the LSM price of a swaption with maturity T_N and swap maturity $T_N - T_n$ is given by the Black formula.

As with the LFM, this is a closed form solution, all that is needed to price the swaption is the determination of the volatilities of the LSM, $\sigma_{n,N}(t)$.

5.2.2 Cap pricing in the LSM

In the LSM, Jamshidian [Jam97] shows how the forward Libor rates may be recovered through the inversion of the forward swap rates (Eq. 3.9). Glasserman and Zhao [GZ00] give the formula as

$$1+\delta L_n(t) = \frac{1+\delta S_n \sum_{i=n}^{N} \prod_{j=n+1}^{i}(1+\delta S_j)}{1+\delta S_{n+1} \sum_{i=n+1}^{N} \prod_{j=n+2}^{i}(1+\delta S_j)}.$$

The swap rate derived from the swap dynamics under the forward swap rate may be substituted into the above equation. The resulting forward Libor rate $L_n(t)$ may then be substituted into the formula for pricing caps (given in Section 5.2.2), to give the price of caps in the LSM.

5.3 Theoretical Incompatibility of the Market Models

Let us look at the dynamics of the forward Libor and forward Swap rates under each others numeraire; that is the Libor rates under the forward Swap measure (FSM), and the Swap rates under the forward Libor measure (FLM).

5.3.1 Forward swap rates under the numeraire used for the forward Libor rates

Following London [Lon04] and Brigo and Mercurio [BM01], we look at the incompatibility by expressing the dynamics of the forward swap rates under the numeraire used for the forward Libor rates, namely the forward (terminal) measure Q^{n+1}. By applying the change-of-drift formula given in Appendix A.1 we obtain the percentage drift $m^{n,N}(t)$ for $S_{n,N}$ under Q^{n+1} as

$$m^n = 0dt - d\log(S_{n,m}(t))\, d\log\left(\frac{C_{n,N}(t)}{B(T_n(t))}\right)$$

The covariance term is

$$\log\left(\frac{C_{n,N}(t)}{B(T_n(t))}\right) = \log\left(\frac{\sum_{i=n+1}^{N} \delta_i B(t, T_i)}{B(t, T_i)}\right)$$

$$= \log\left(\sum_{i=n+1}^{N} \delta_i \prod_{j=n+1}^{i} \frac{1}{\delta_i B(t, T_i)}\right)$$

$$= \chi\,(L_{n+1}(t), L_{n+2}(t), \ldots, L_N(t))$$

Recalling from Eq. 3.10 that

$$S_{n,N}(t) = \exp(\psi(L_{n+1}(t), L_{n+2}(t), \ldots, L_N(t)))\,,$$

so that

$$\log S_{n,N}(t) = \psi(L_{n+1}(t), L_{n+2}(t), \ldots, L_N(t))$$

we find

$$m^n\, dt = \sum_{j=n+1}^{N} \sum_{i=n+1}^{N} \frac{\partial \psi}{\partial L_i} \frac{\partial \chi}{\partial L_i}\, dL_i(t)\, dL_j(t)\,.$$

After some straight forward but lengthy calculations, Brigo and Mercurio [BM01] have shown that the forward swap rate $S_{n,N}(t)$ under the Q^{n+1} forward Libor measure follow the dynamics

Some lengthy manipulation by them yield the dynamics of the forward Libor rates

$$dL_k(t) = \sigma_k(t)\, L_k(t)\, \mu_k^{n,N}(t)\, dt + \sigma_k(t)\, L_k(t)\, dW_k(t) \qquad \text{Eq. 5.12}$$

where W is the $Q^{n,N}$ standard Brownian motion, and the drift is given by

$$\mu_k^{n,N}(t) = \sum_{i=n+1}^{N} (2*1_{(j\le k)}) - 1)\, \delta_i \frac{B(t,Tj)}{C_{n,N}(t)} \sum_{i=\min(k+1,\, j+1)}^{\max(k,j)} \frac{\delta_i\, \rho_{k,i}\, \sigma_i(t)\, L_i(t)}{1 + \delta_i\, L_i(t)} \qquad \text{Eq. 5.13}$$

where $1_{(j\le k)}$ is the indicator function equal to 1 if $j \le k$ and 0 otherwise. This is a closed set of SDE when k ranges from n+1 to N, since the terms $B(t,Tj)/C_{n,N}(t)$ can easily be expressed as suitable functions of spanning forward rates $L_{n+1}(t),\ldots L_N(t)$. By substituting the forward Libor rates from derived from Eq. 5.12 into the equation for forward swap rates,

$$S_{n,N}(t) = \frac{1 - \prod_{j=n}^{N} \frac{1}{1 + \delta_j L_j(t)}}{\sum_{i=n}^{N} \prod_{j=i+1}^{N} \frac{1}{1 + \delta_j L_j(t)}}, \qquad \text{Eq. 5.14}$$

we get the LFM swap rate under the forward swap measure. This swap rate can then be compared to the swap rate derived from the LSM model.

5.3.3 Incompatibility

The swap rate computed from the LSM dynamics (Eq. 5.11) is lognormally distributed, while the swap rate computed from the LFM dynamics (Eq. 5.13 and Eq. 5.14) is not lognormal. It is also observed that the dynamics of the forward Libor and Swap rates under each others numeraire, shown in Eq. 5.10 and Eq. 5.12, are not driftless, and hence they are not lognormal under those measure. This is what we talk about when we say that the two models are theoretically incompatible (Brigo and Mercurio [BM01], Brigo and Liinev [BL01]).

$$dS_{n,N}(t) = m^n(t)\, S_{n,N}(t)\, dt + \sigma^s(t)\, S_{n,N}(t)\, dW(t)\ , \qquad \text{Eq. 5.10}$$

where W is the Q^{n+1} standard Brownian motion.

The drift $m^n(t)$ is given by

$$m^n\, dt = \frac{\sum_{h,k=n+1}^{N} \mu_{h,k}(t)\, \delta_h\, \delta_k\, FP_h(t)\, FP_k(t)\, \rho_{h,k}\, \sigma_h(t)\, \sigma_k(t)\, F_h(t)\, F_k(t)}{1 - FP_{n,N}(t)}\ , \qquad \text{Eq. 5.11}$$

where

$$\mu_{h,k}(t) = \frac{\sum_{i=k}^{N} \delta_i\, FP_{n,i}(t) \cdot \left(FP_{n,N}(t) \sum_{i=n+1}^{h-1} \delta_i\, FP_{n,i}(t) + \sum_{i=h}^{N} \delta_i\, FP_{n,i}(t) \right)}{\left(\sum_{i=n+1}^{N} \delta_i\, FP_{n,i}(t) \right)^2}$$

$$\text{and } FP_{n,i}(t) = \frac{B_i(t)}{B_n(t)} = \prod_{i=n+1}^{N} \frac{1}{1 + \delta_i L_i(t)}\ .$$

5.3.2 Forward Libor rates under the numeraire used for the forward swap rates

Again following Brigo and Mercurio [BM01], we can compute the forward rate Libor dynamics under the forward swap measure $Q^{n,N}$. In this case the drift is

$$m^{n,N} = 0dt - d\log(S_{n,m}(t))\, d\log\left(\frac{B(t,T_n)}{C_{n,N}(t)} \right).$$

With respect to the pricing of instruments, if swaptions are priced with the LSM based on the swap rate dynamics in Eq. 5.11 then the swap rate distribution is exactly lognormal and the LSM expectation reduces to the Black's formula. However, if this expectation is taken with the LFM then the computations of the forward Libor rates are not lognormal.

The remainder of this study is dedicated to implementing the models to determine if this incompatibility exists in practice.

Chapter 6

Implementation of the Models

6 Implementation of the Models

In general, the differential equations of the models for pricing financial derivatives cannot be solved explicitly. However, it is possible to simulate them through a discretization scheme, and then implement them through a Monte Carlo computer program simulation. Since Monte Carlo simulation is central to this study, we start the chapter with a brief description of Monte Carlo simulation. This is followed by a model of the implementation, illustrated early on in the chapter to give the reader an overview of the processes taken to implement the market models. We introduce discretization and martingale measures, to support the derivation of the discretized formulas for the forward Libor and Swap rates under the forward Libor and forward Swap measures. The resultant forward rates are then used in the simulation of caps and swaptions in the Libor and Swap market models. Pseudo code is given for each simulation, as a prelude to its implementation by a computer program, discussed in Chapter 7. This chapter concludes with a brief discussion of the calibration methods used in the implementation to calibrate the prices generated by the models to the actual market trades.

6.1 Monte Carlo Simulation

In Monte Carlo simulations, expected values from a discretized probability space are substituted by the average of finite samples. Following Glasserman [Gla04], consider the problem of estimating the integral of a function f over a unit interval (0 to 1). The integral may be given as

$$\alpha = \int_0^1 f(\mathrm{x})\, d\mathrm{x} .$$

This integral can be represented as an expectation $\hat{\mathrm{E}}[f(\mathrm{U})]$, with U uniformly distributed between 0 and 1. Now suppose there is a mechanism for drawing points $\mathrm{U}_1, \mathrm{U}_2, \ldots, \infty$ independently and

uniformly from 0 and 1. Evaluating the function f at n for these random points and averaging the results produces the Monte Carlo estimate

$$\acute{\alpha}_n \approx 1/n \sum_{i=1}^{n} f(U_i) .$$

By the strong law of numbers, $\acute{\alpha}_n \rightarrow \alpha$ with probability 1 as $n \rightarrow \infty$.

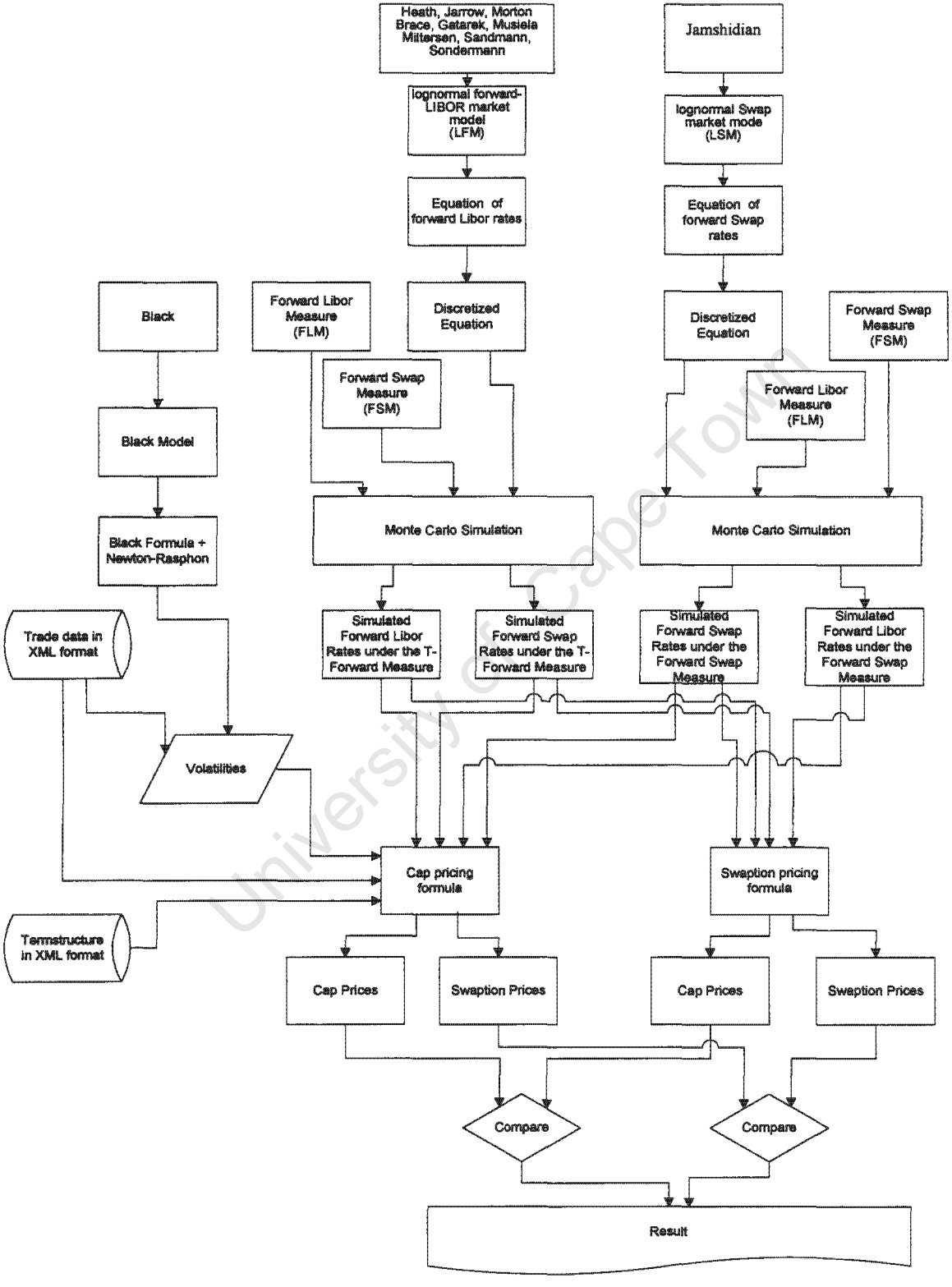
In order to implement the lognormal forward-rate market models, there are two stochastic differential equations that must be solved from input data. These equations are the dynamics of the forward Libor rates in the LFM (Eq. 5.1) and dynamics of the forward Swap rates in the LSM (Eq. 5.7). Since these equations do not have explicit solutions, we choice Monte Carlo simulation to solve these equations. For Monte Carlo simulation, the equations must be discretized. The discretization schemes are discussed in Section 6.3.

6.2 A Model of the Implementation

Figure 1 illustrates the implementation of the Libor and Swap market models used to determine the incompatibility of the two models. The details of the implementation are discussed in the remainder of the chapter. A broad description of Figure 1 is the following:

- Models: the formulation of the Libor and Swap market models by the different authors.
- Discretization: discretization of the equations of forward rate dynamics by Euler's scheme.
- Data: trade and term structure data is read from the XML files.
- Implied Volatility: the implied volatility is calculated from the trade data, using Black's formulas and Newton- Raphson techniques.
- Forward Rates: Monte Carlo simulation under the forward measures to give the simulated forward rates.
- Prices: computation through the cap and swaption formulas to give the simulated prices.
- Result: comparison of the prices generated by the two models under the different forward measures to give a result.

Figure 1 Implementation of the Models



6.3 Discretization

Pietersz *et. al.* [PMR04] discuss four discrete approximations as applied to the BGM (Brace, Gatarek, Musiela [BGM97]) model: Euler, predictor-corrector, Milstein second order scheme and the Brownian Bridge. Andersen and Andersen [AA00] use a Crank-Nicholson finite difference scheme with their Monte Carlo simulations, after pointing out the systematic biases and random sample errors introduced by the simpler Euler or log-Euler discretization. They also discuss the merits of other discretization schemes. Hull and White [HW99] discuss an approximation that simplifies the Monte Carlo implementation of the model, which is that the drift of $ln L_i$ is constant between times t_j and t_{j+1}. In their analysis of the LFM, Hunter *et. al.* [HJJ01] show that the predictor-corrector method of discretization is better than the Euler method. They suggest that their result could probably be applied to other models, such as the LSM.

The methods discussed above generally discretize the rates to be simulated. Glasserman and Zhao [GZ00] go further and use a technique that transforms the Libor or swap rates into positive martingales, discretizes the martingales and then recovers that Libor and swap rates from those discretized variables. In their analysis particular attention is paid to precluding arbitrage amongst bonds and keeping interest rates positive even after discretization. They show how this can be done under both the spot and terminal measures.

We follow the simplest discretization scheme, which in the limit case leads to the Itô integral, which is the Euler, or log-Euler, discretization scheme. In this scheme, the Monte Carlo method discretizes probability space and substitutes expected values with averages of finite samples.

6.4 Martingale measures

Another issue to consider in the implementation of the market models is under which martingale measure the prices must be evaluated. Andersen and Andreasen [AA00] used the spot martingale measure in their analysis. Glasserman and Zhao [GZ00] illustrate the use of both spot and forward (terminal) measures for both caplets and swaps. In a theoretical comparison between the LFM and LSM, Brigo and Mercurio [BM01] derive swaption prices in both models under the forward rate swap measure.

In this study we follow Jamshidian [Jam97], de Jong *et. al.* [DDP01] and Glasserman and Zhao [GZ00], who choose to simulate the forward rates under the forward (terminal) martingale measures (for forward Libor and Swap rates).

for i = 1,…,n

generate Wi

set $F^i(T) = F(0) \exp([r - \frac{1}{2}\sigma]T + \sigma(\sqrt{T})W_i$

set $C_i = e^{-rT}(F(T) - K)^+$

set $\hat{C} = (C_1 + \ldots + C_n)/n$

The above algorithm applies to the pricing of caps in the case of the LFM, but similarly for swaptions in the LSM.

6.6 The Libor Market Model

6.6.1 The discretized forward Libor rate equation

Following Glasserman [Gla04], since caps deal with a finite set of maturities, we only need to discretize the time argument. We fix a time grid $0 = t_0 < t_1 < \ldots < t_n < t_{n+1}$ over which to simulate, and include the tenor dates $T_1, T_2,\ldots,T_{N+1}$ among the simulation dates.

Recall that the dynamics of forward Libor rates is given by

$$\frac{dL_i(t)}{L_i(t)} = \mu_j(t)\, dt + \sigma_i(t)\, dW^{M+1}(t), \quad i = n+1,\ldots,N, \qquad \text{Eq. 6.1}$$

where $\mu_j(t)$ is the drift.

Often the lognormal is easier and faster to work with. By taking logs and applying Ito's formula to Eq. 6.1 gives

$$\log L_k^{\Delta t}(t+\Delta t) = \log L_i^{\Delta t}(t) + \mu_j(t)\ \Delta t - \frac{\sigma_k^2(t)}{2}\Delta t + \sigma_i(t)\,(W_i(t+\Delta t) - W_i(t)), \qquad \text{Eq. 6.2}$$

where i = n+1, …, N.

It is known (Hull [Hul93], Glasserman [Gla04]) that $W_t+\Delta t - W_t \sim \sqrt{(\Delta t)}\, N(0,\rho)$, and that Z(t3)-Z(t2) is independent of Z(t2)-Z(t1) for all t1<t2<t3 so that the joint shocks can be seen as independent draws from a multivariate normal distribution N(0,1). This gives the final formula to compute the simulation of the forward Libor rates

for i = 1,…,n

generate Wi

set $F^i(T) = F(0) \exp([r - \frac{1}{2}\sigma]T + \sigma(\sqrt{T})W_i$

set $C_i = e^{-rT}(F(T) - K)^+$

set $\hat{C} = (C_1 + \ldots + C_n)/n$

The above algorithm applies to the pricing of caps in the case of the LFM, but similarly for swaptions in the LSM.

6.6 The Libor Market Model

6.6.1 The discretized forward Libor rate equation

Following Glasserman [Gla04], since caps deal with a finite set of maturities, we only need to discretize the time argument. We fix a time grid $0 = t_0 < t_1 < \ldots < t_n < t_{n+1}$ over which to simulate, and include the tenor dates $T_1, T_2, \ldots, T_{N+1}$ among the simulation dates.

Recall that the dynamics of forward Libor rates is given by

$$\frac{dL_i(t)}{L_i(t)} = \mu_j(t)\, dt + \sigma_i(t)\, dW^{M+1}(t), \quad i = n+1,\ldots,N, \qquad \text{Eq. 6.1}$$

where $\mu_j(t)$ is the drift.

Often the lognormal is easier and faster to work with. By taking logs and applying Ito's formula to Eq. 6.1 gives

$$\log L_k^{\Delta t}(t+\Delta t) = \log L_i^{\Delta t}(t) + \mu_j(t)\, \Delta t - \frac{\sigma_k^2(t)}{2}\Delta t + \sigma_i(t)\,(W_i(t+\Delta t) - W_i(t)), \qquad \text{Eq. 6.2}$$

where i = n+1, …, N.

It is known (Hull [Hul93], Glasserman [Gla04]) that $W_t+\Delta t - W_t \sim \sqrt{(\Delta t)}\, N(0,\rho)$, and that Z(t3)-Z(t2) is independent of Z(t2)-Z(t1) for all t1<t2<t3 so that the joint shocks can be seen as independent draws from a multivariate normal distribution N(0,1). This gives the final formula to compute the simulation of the forward Libor rates

6.5 Implementation

Consider an interest rate derivative with a payoff at time T, specified through a function f of the prices of the underlying asset, F. To price the derivative, we model the dynamics of the underlying assets under the risk-neutral measure, ensuring that the discounted asset prices are martingales, typically through the choice of drift. The price of the derivative is then given as (Glasserman [Gla04])

$$\hat{E}[e^{-rT} f(F(T))] ,$$

where $\hat{E}$ is the expectation, as before.

To evaluate this expectation, we simulate according to the risk-neutral dynamics. On each path we calculate the discounted payments of $e^{-rT} f(F(T)$, then average across the paths to give our estimate of the derivative price. A simplified summary of this procedure is as follows:

- Replace the rates with risk-free interest rates.
- Calculate the drifts and simulate the forward rate paths.
- Calculate the payoff of the derivative security on each path.
- Discount the payoffs at the risk-free rate.
- Calculate the average over each path.

6.5.1 Simulation of Black formulas for caplets and swaptions

The caplet formula in the LFM and the swaption formula the LSM both reduce to their respective equivalent Black formula for caplets and swaptions, as discussed in Chapter 5. Hence, there is no real need to resort to Monte Carlo simulation for these formulas. However, (following Glasserman, [Gla04]), as an illustration we will show the simulation here. All that is needed is to draw samples of independent standard normal random variables from a standard normal distribution, using the following algorithm to estimate $\hat{E}[e^{-rT} f(F(T))]$:

$$\log L_k^{\Delta t}(t+\Delta t) = \log L_i^{\Delta t}(t) + \sigma_j \mu_j(t)\ \Delta t - \frac{\sigma_k^2(t)}{2}\Delta t + \sigma_i(t)\sqrt{(\Delta t)}\, N(0,\rho). \qquad \text{Eq. 6.3}$$

6.6.2 Simulation of forward Libor rates under the forward Libor measure

In the above equation (Eq. 6.3) the drift $\mu_j(t)$ under the forward Libor measure is given by

$$\mu_j(t) = -\sigma_j(t) \sum_{j=n+1}^{i} \frac{\rho_{i,j}\delta j\, \sigma_j(t)\, L_j^{\Delta t}(t)}{1 + \delta j\, L_j^{\Delta t}(t)}, \qquad \text{Eq. 6.4}$$

from Eq. 5.4.

To simulate N forward Libor rates under the forward Libor measure, with M time steps and starting at time n, we use the following algorithm:

for t = n,…,M-1
  for k = n+1,…,N
    generate W_j
    for i = k+1,…,N-1
      set $\mu_i = (\rho_{k,i}\, \delta i\, \sigma_j\, L_{t,i-1})/(1+ \delta_i\, L_{t,i-1})$
    set $\mu_k = (\mu_1 - \mu_2 - \ldots - \mu_{m-1})$ //drift
    set $\log L = \log(L_{t,k-1}) + \sigma_k\, \mu_k\, \Delta t - \frac{1}{2}\, \sigma_k \sigma_k\, \Delta t + \sigma_k W_j \sqrt{(\Delta t)}$
    set $L_{t+1,k} = \exp(\log L)$

The code is listed in Appendix E.4.

6.6.3 Simulation of forward Libor rates under the forward Swap measure

In Eq. 6.3 the drift $\mu_j(t)$ under the forward Swap measure is give by Eq. 5.13,

$$\mu_k^{n,N}(t) = \sum_{j=n+1}^{N} (2*1_{(j\le k)} - 1)\,\delta_i \frac{B(t,Tj)}{C_{n,N}(t)} \sum_{i=\min(k+1,\,j+1)}^{\max(k,j)} \frac{\delta_i\,\rho_{k,i}\,\sigma_i(t)\,L_i(t)}{1+\delta_i\,L_i(t)} . \qquad \text{Eq. 6.5}$$

Starting at time n, with M time steps, we simulate N forward Libor rate using the following algorithm:

for t = n,…,M-1
  for k = n,…,N
    for i = k,…,N
      set $B^n_i = 1/(1+ \delta_i L_{t,i})$
    set $B^n_k = (B^n_1 * B^n_2 *_{\ldots} * B^n_N)$ // B^{n+1} numeraire
    for i = k,…,N
      set $C^n_i = 1/(1+ \delta i\, L_{t,i})$
    set $C^n = (C^n_1 * C^n_2 *_{\ldots} * C^n_N)$ // C^{n+1} numeraire
    set $C^n_k = \delta_k * C^n$
  for k = n+1,…,N
    for j = n+1,…,N
      generate Wj
    set ind_j = (j <= k) ? 1 : 0
      set $LT_j = (2*ind-1) * \delta j * (Bn_j/Cn_j)$
    for i = min(k+1,j+1),…, max(k,j)
      set $SR_j = (\rho_{k,i}\,\delta i\,\sigma_j\,L_{t,i-1})/(1+\delta_i\,L_{t,i-1})$
    set $SR_j = (SR_1 + SR_2 +_{\ldots} + SR_{max(k,j)})$
  set $\mu_j = LT_j * SR_j$
set $\mu_k = (\mu_1 + \mu_2 +_{\ldots} + \mu_N)$ //drift
  set $logL = \log(L_{t,k-1}) + \sigma_k\,\mu_k\,\Delta t - \tfrac{1}{2}\,\sigma_k\sigma_k\,\Delta t + \sigma_k W_j \sqrt{(\Delta t)}$
  set $L_{t+1,k} = \exp(logL)$

The code is listed in Appendix E.5.

$$\mu_k^{n,N}(t) = \sum_{j=n+1}^{N} (2*1_{(j\le k)} - 1)\, \delta_i \frac{B(t,Tj)}{C_{n,N}(t)} \sum_{i=\min(k+1,\, j+1)}^{\max(k,j)} \frac{\delta_i\, \rho_{k,i}\, \sigma_i(t)\, L_i(t)}{1 + \delta_i\, L_i(t)} . \qquad \text{Eq. 6.5}$$

Starting at time n, with M time steps, we simulate N forward Libor rate using the following algorithm:

for t = n,…,M-1
  for k = n,…,N
    for i = k,…,N
      set $B^n_i = 1/(1+ \delta_i\, L_{t,i})$
    set $B^n_k = (B^n_1 * B^n_2 *_{\ldots} * B^n_N)$ // B^{n+1} numeraire
    for i = k,…,N
      set $C^n_i = 1/(1+ \delta i\, L_{t,i})$
    set $C^n = (C^n_1 * C^n_2 *_{\ldots} * C^n_N)$ // C^{n+1} numeraire
    set $C^n_k = \delta_k * C^n$
  for k = n+1,…,N
    for j = n+1,…,N
      generate Wj
    set ind_j = (j <= k) ? 1 : 0
      set $LT_j = (2*ind-1) * \delta j * (Bn_j/Cn_j)$
    for i = min(k+1,j+1),…, max(k,j)
      set $SR_i = (\rho_{k,i}\, \delta i\, \sigma_j\, L_{t,i-1})/(1+ \delta_i\, L_{t,i-1})$
    set $SR_j = (SR_1 + SR_2 +_{\ldots} + SR_{max(k,j)})$
  set $\mu_j = LT_j * SR_j$
set $\mu_k = (\mu_1 + \mu_2 +_{\ldots} + \mu_N)$ //drift
  set $logL = \log(L_{t,k-1}) + \sigma_k\, \mu_k\, \Delta t - \tfrac{1}{2}\, \sigma_k \sigma_k\, \Delta t + \sigma_k W_j \sqrt{(\Delta t)}$
  set $L_{t+1,k} = \exp(logL)$

The code is listed in Appendix E.5.

6.6.4 Simulation of caps in the LFM

In order to simulate cap prices in the LFM we need to simulate N discretized variables L_1, $L_2,\ldots,L_N$ of the forward Libor rates using the LFM forward rate dynamics under the forward measure. To do this, we compute the caplet payoff from

$$C_n(t) = \delta_n (L_n(T_n) - K)^+.$$

for each realization, and then average over all M = N-n payoffs.

The following algorithm shows the steps in simulating SIMS paths of N transitions each for M forward rates. The *j*th Libor rate along the $_i$th path is denoted by Zi,j

for t = n,…,N

$B^0_k = 1/(1+ \delta_k L_{0,k+1})$

set $B^0 = (B^0_1 * B^0_2 *_{\ldots} * B^0_N)$ //zero coupon bond

for s = 1,…,SIMS

generate $F_{t,k}$

for t = n+1,…,N

for k= t,…,N

$B^n_k = 1/(1+ \delta_k F_{t,k})$

set $B^n_t = (B^n_1 * B^n_2 *_{\ldots} * B^n_N)$ //B_{n+1} numeraire

set $C_t = (B^0 / B^n_t) * \delta_t * P * (Ft,t - K)^+$

set $\hat{C}_t = (C_{t1} + \ldots + C_{t,m})$

set $\hat{C}ap = (C_1 + \ldots + C_n)/SIMS$

The code is listed in Appendix E.8.

Another example of a caplet formula approximation may be found in Hull and White [HW99], which from the literature appears to be the simplest to implement with Monte Carlo. Glasserman and Zhao [GZ00] give a similar formula, but they simulate $1 + \delta F_i$ instead of F_i to ensure that the process remains positive and also discretize the martingales.

6.6.5 Simulation of swaptions in the LFM

Used in this analysis is an alternate discretization formula for Monte Carlo pricing of swaptions in the LFM, an example of which is given in London [Lon04]. If the reset dates for the swap

coincide with the reset dates for the caplets underlying the Libor market model then the forward swap rate may be written as (Eq. 5.14)

$$S_{n,N}(t) = \frac{1 - \prod_{j=n}^{N} \frac{1}{1 + \delta_j L_j(t)}}{\sum_{i=n}^{N} \prod_{j=i+1}^{N} \frac{1}{1 + \delta_j L_j(t)}} .$$

From this formula it is seen that the swap rate may be expressed in terms of spanning forward rates at time T_n,

$$L_{n+1}(T_n), L_{n+2}(T_n), \ldots, L_N(T_n) ,$$

and it is also obvious that the forward rate correlations impact the price, unlike in the case for the pricing of caplets.

To get a value for the forward swap rate we need simulate N discretized variable $L_1, L_2, \ldots, L_N$ of the forward Libor rates in above using the LFM forward rate dynamics under the forward Libir measure. For each realization we compute the swap rate from the above equation and then calculate the swaption payoff

$$PS = P(0,T_n) \max(S_{n,N} - K, 0) \sum_{i=n+1}^{N} \delta_i B(T_n, T_i),$$

and then average over all M = N-n payoffs.

The following algorithm shows the steps in simulating SIMS paths of N transitions each for M forward rates. The *j*th Libor rate from the along the $_i$th path is denoted by $Zi_{,j}$

for t = n,…,N

$B^0_k = 1/(1+ \delta_k L_{0,k+1})$

set $B^0 = (B^0_1 * B^0_2 * \ldots * B^0_N)$ //zero coupon bond

for s = 1,…,SIMS

generate $F_{j,k}$

for t = n+1,…,N

for k= t,…,N

$B^n_k = 1/(1+ \delta_k F_{t,k})$

set $B^n_t = (B^n_1 * B^n_2 *_{...} * B^n_{m-1})$ //B_{n+1} numeraire

for t = n+1,...,M

for k = t,...,N

$UP_k = 1/(1+ \delta_k F_{k,k})$

set $UP_t = (UP_1 * UP_2 *_{...} * UP_N)$ //upper product

for i = t,...,N

for k= i+1,...,N

$LP_k = 1/(1+ \delta_k F_{k,k})$

set $LP_i = (LP_1 * LP_2 *_{...} * LP_N)$

set $LP_t = (LP_1 + LP_2 + \ldots + LP_N)$ //lower product

set $FSR_t = (1- UP_t)/LP_t$

set $S_t = (B^0/ B^n_t) * \delta_t * P * (FSR_t - K)^+$

set $\hat{S}_t = (S_{t1} + S_{t2} + \ldots + S_{m-1})$

set $\hat{S} = (S_1 + \ldots + S_n)/SIMS$

The code to implement this algorithm is listed in Appendix E.11 .

Other approximations to pricing swaptions with the LFM have been suggested by Brace *et. al.* [BGM97] and De Jong *et. al.* [DDP01], who follow the analysis by Brace *et. al.* [BGM97]. Hull and White [HW99], who based their approximation on Andersen and Andreasen [AA00], also provide an approximation for pricing a swaption in the LFM. They suggest that their method is much easier to implement than other methods, but nonetheless very accurate. Their derivation is motivated by the fact that when forward Libor rates are lognormal, the swap rates are approximately lognormal and approximately linearly dependent of forward Libor rates. By exploiting the relationship between bond prices and forward rates, a formula for the spot option volatility can be derived. When this is substituted into the equation for the forward swap price, and using Ito's Lemma, the swaption price in the LFM can be derived.

6.7 The Swap market Model

6.7.1 The discretized forward Swap rate equation

Since swaptions deal with a finite set of maturities, as with caps, we also only need to discretize the time argument. We fix a time grid $0 = t_0 < t_1 < \ldots < t_n < t_{n+1}$ over which to simulate, and include the tenor dates $T_1, T_2, \ldots, T_{N+1}$ among the simulation dates.

Recall from Eq. 5.7 that the dynamics of forward swap rates are given by

$$\frac{dS_i(t)}{S_i(t)} = \mu_j(t)\ dt\ + \sigma_i(t)\, dW^{M+1}(t), \quad i = n+1,\ldots,N, \qquad \text{Eq. 6.6}$$

where $\mu_j(t)$ is the drift.

Often the lognormal is easier and faster to work with. By taking logs and applying Ito's formula to the Eq. 6.1 gives

$$\log S_k^{\Delta t}(t+\Delta t) = \log S_i^{\Delta t}(t) + \mu_j(t)\ \Delta t - \frac{\sigma_k^2(t)}{2}\Delta t + \sigma_i(t)\,(W_i(t+\Delta t) - W_i(t)), \qquad \text{Eq. 6.7}$$

where $i = n+1, \ldots, N$. Using the fact that $W_t+\Delta t - W_t \sim \sqrt{(\Delta t)}\, N(0,\rho)$, this gives

$$\log S_k^{\Delta t}(t+\Delta t) = \log S_i^{\Delta t}(t) + \mu_j(t)\ \Delta t - \frac{\sigma_k^2(t)}{2}\Delta t + \sigma_i(t)\sqrt{(\Delta t)}\, N(0,\rho). \qquad \text{Eq. 6.8}$$

6.7.2 Simulation of forward Swap rates under the forward Swap measure

It was shown in Section 5.2 that the drift $\mu_j(t)$ of the forward swap rate dynamics given in Eq. 6.6, under the forward Swap measure, is

$$\mu_{n,N}(t) = -\frac{\sum_{i=n}^{N}\sum_{k=n+1}^{i} \delta\ S_k(t)\ \sigma_k(t) \prod_{j=n+1, j\neq k}^{i}(1+\delta\ S_j)}{\sum_{i=n}^{N}\prod_{j=n+1}^{i}(1+\delta\ S_j)}. \qquad \text{Eq. 6.9}$$

To simulate the N forward swap rates under the forward swap measure, with M time steps and starting at time n, we use the following algorithm:

for t = n,…,M-1

for k = n+1,…,N

generate Wj

for i = n,…,N

for l = n+1,…,i

for j = n+1,j!=l,…,i

set $p_j = 1+ \delta i\ L_{t,k-1}$)

set $p_l = (p_1 * p_2 *_{\ldots} * p_i)$

set $s_l = \delta t * L_{t,k-1} * \sigma_t * p_l$

set $s_i = (s_1 + s_2 +_{\ldots} + s_i)$

$US = (s_1 + s_2 +_{\ldots} + s_N)$ // upper sum

for i = n,…,N

for j = n+1,…,i

set $pj = 1+\delta_t L_{t,k-1}$

set $p_i = (p_1 * p_2 *_{\ldots} * p_i)$

$LS = (p_1 + p_2 +_{\ldots} + p_N)$ // lower sum

set μ_k = -US/LS

set $\log L = \log(L_{t,k-1}) + \sigma_k\ \mu_k\ \Delta t - \frac{1}{2}\ \sigma_k \sigma_k\ \Delta t + \sigma_k W_j \sqrt{(\Delta t)}$

set $L_{t+1,k} = \exp(\log L)$

The code is listed in Appendix E.6.

6.7.3 Simulation of forward Swap rates under the forward Libor measure

From Section 5.3, it was shown that the drift $\mu_j(t)$ in Eq. 6.6, under the forward Libor measure is

$$\mu_n(t) = \frac{\sum_{h,k=n+1}^{N} m_{h,k}(t)\ \delta_h\ \delta_k\ FP_h(t)\ FP_k(t)\ \rho_{h,k}\ \sigma_h(t)\ \sigma_k(t)\ F_h(t)\ F_k(t)}{1 - FP_{n,N}(t)}, \qquad \text{Eq. 6.10}$$

where

$$m_{h,k}(t) = \frac{\sum_{i=k}^{N} \delta_i FP_{n,i}(t) \left(FP_{n,N}(t) \sum_{i=n+1}^{N} \delta_i FP_{n,i}(t) + \sum_{i=h}^{N} \delta_i FP_{n,i}(t) \right)}{\left(\sum_{i=n+1}^{N} \delta_i FP_{n,i}(t) \right)^2}$$

$$\text{and } FP_{n,i}(t) = \frac{B_i(t)}{B_n(t)} = \prod_{i=n+1}^{N} \frac{1}{1 + \delta_i L_i(t)}$$

We can simulate the N forward swap rate dynamics under the forward Libor measure, with M time steps starting at time n by the following algorithm:

```
for t = n,…,M-1
        for k = n+1,…,N
                generate Wj
                for h = n+1,…,M
                   for g = n+1,…,M
                              for u = g,…,M
                                  for i = h,…,M
                                     for j = n+1,…,M
                                        set FPj = 1/(1+ δj Lt,j-1)
                                              set FPi = (FP1* FP2 *…* FPm)
                           set USi = δi* FPi
                                        set USMU3u = (US1+ US2 +…+ USm)  // MU3rd term
                                  for i = n+1,…,h-1
                                     for j = n+1,…,M
                                        set FPj = 1/(1+ δj Lt,j-1)
                                              set FPi = (FP1* FP2 *…* FPh-1)

                                     set USi += δi* FPi
```

set $USMU2_u = (US_1 + US_2 + \ldots + US_m)$ // MU 2rd term

for $j = n+1,\ldots,M$

set $FP_n = 1/(1+\delta j\, L_{t,j-1})$

set $FP^N_u = (FP_1 * FP_2 * \ldots * FP_m)$

for $j = n+1,\ldots,i$

set $FP_i = 1/(1+\delta j\, L_{t,j-1})$

set $FP^i_u = (FP_1 * FP_2 * \ldots * FP_i)$

$UMS_u = \delta u * FP^i_u * (FP^N_u * USMU2_u + USMU3_u)$

set $USMU1_u = (UMS_1 + UMS_2 + \ldots + UMS_m)$ // MU 1st

for $j = n+1,\ldots,M$

for $j = n+1,\ldots,M$

set $FP_i = 1/(1+\delta j\, L_{t,j-1})$

set $FP^n j = (FP_1 * FP_2 * \ldots * FP_{m-1})$

set $LSMU = (\delta_1 * FP_1 + \delta_2 * FP_2 + \ldots + \delta_m * FP_m)$

set $denMU_u = LSMU_u * LSMU_u$

set $MU_{h,g} = USMU1_u / denMU_u$

for $i = h+1,\ldots,M$

set $FP_h = 1/(1+\delta i\, L_{t,i-1})$

set $FP_k{}^h = (FP_1 * FP_2 * \ldots * FP_m)$

for $i = g+1,\ldots,M$

set $FP_g = 1/(1+\delta i\, L_{t,i-1})$

set $FP_k{}^g = (FP_1 * FP_2 * \ldots * FP_m)$

set $num_i = MU_{h,g} * \delta_h * \delta_g * FP_h\, FP_g * \rho_{g,h} * \sigma_h * \sigma_k * L_{t,g-1} * L_{t,h-1}$

$num_{h,g} = (num_1 + num_2 + \ldots + num_m)$

set $\mu_k = num_k/(1 - FP_n)$

set $logL = \log(L_{t,k-1}) + \sigma_k\, \mu_k\, \Delta t - \tfrac{1}{2}\, \sigma_k \sigma_k\, \Delta t + \sigma_k W_j \sqrt{(\Delta t)}$

set $L_{t+1,k} = \exp(logL)$

The code for the implementation of this algrothim is listed in Appendix E.7.

6.7.4 Simulation of swaptions in the LSM

The Monte Carlo simulation of Black's swaption formula for the swaption price can be used to price swaptions in the LSM, similar to the LFM case in Section 6.6.4. It will not be described again here.

Another swaption formula is given by Jamshidian [Jam97], while Glasserman and Zhao [GZ00] derive swaption formula in the LSM by discretizing the terminal martingales. Gullicco *et. al.* [GHLS04] show that the price of a swaption can be obtained through their co-terminal Swap market model. However, these authors do not discuss discretized versions of their formulas.

The following algorithm shows the steps in simulating SIMS paths of N transitions each for M forward rates. The *j*th Libor rate along the ith path is denoted by Zi,j

for $t = n,\ldots,N$
    $B^0_k = 1/(1+ \delta_k L_{0,k+1})$
set $B^0 = (B^0_1 * B^0_2 * \ldots * B^0_N)$ //zero coupon bond
for $s = 1,\ldots,SIMS$
    generate $F_{t,k}$
    for $t = n+1,\ldots,N$
        for $k= t,\ldots,N$
        $C^n_k = 1/(1+ \delta_k F_{t,k})$
    set $C^n_t = (C^n_1 * C^n_2 * \ldots * C^n_N)$ //C_{n+1} swap numeraire
        set $S_t = (C^0 / C^n_t) * \delta_t * P * (Ft,t - K)^+$
    set $\hat{S}_t = (S_{t1} + \ldots + S_{t,m})$
set $\hat{S} = (S_1 + \ldots + S_n)/SIMS$

Appendix E.10 lists the code for this implementation.

6.7.5 Simulation of caps in the LSM

In the terminal measure, the forward Libor rates can be recovered from the forward swap rates through the relation (Jamshidian [Jam97], Glasserman and Zhao [GZ00])

$$1 + \delta L_n(t) = \frac{1 + \delta S_n(t) \sum^N_{t=n} \prod_{j=n+1}(1 + S_j)}{1 + \delta S_{n+1}(t) \sum^N_{t=n+1} \prod_{j=n+2}(1 + S_j)}.$$

In this implementation, the forward swap rates will be simulated by Monte Carlo simulation in the LSM, and then inserted into the above equation.

The following algorithm shows the steps in simulating SIMS paths of N transitions each for M forward rates. The *j*th Libor rate from the along the $_i$th path is denoted by Zi,j.

for t = n,…,N

$B^0_k = 1/(1+\delta_k L_{0,k+1})$

set $B^0 = (B^0_1 * B^0_2 *_{\ldots} * B^0_N)$ //zero coupon bond

for s = 1,…,SIMS

generate $F_{j,k}$

for t = n+1,…,M

for k = t,…,N

set $C^n_k = 1/(1+\delta_k F_{t,k})$

set $C^n_t = (C^n_1 * C^n_2 *_{\ldots} * C^n_N)$ //C_{n+1} swap numeraire

for k = n,…,M

for i = n+1,…,k

set $P_i = 1/(1+\delta_k F_{t,t})$

set $UP_i = (P_1 * P_2 *_{\ldots} * P_m)$ //upper product

set $US_t = (UP_1 + UP_2 +_{\ldots} + UP_m)$ //upper sum

for k= n+1,…,M

for i = n+2,…,k

set $P_i = 1/(1+\delta_k F_{t,t})$

set $LP_i = (P_1 * P_2 *_{\ldots} * P_m)$ //lower product

set $LS_t = (LP_1 + LP_2 + \ldots + LP_{m-1})$ //lower sum

set $U_t = 1 + \delta_k F_{t,t} * US_t$

set $D_t = 1 + \delta_k F_{t,t} * LS_t$

set $FLR_t = (U_t/D_t - 1)/\delta_k$ // forward Libor rate

set $C_t = (B^0/B^n_t) * \delta_t * P * (FLR_t - K)^+$

set $\hat{C}_t = (C_{t1} + C_{t2} + \ldots + C_m)$

set $\hat{Cap} = (C_1 + \ldots + C_n)/SIMS$

The code for this implementation is found in Appendix E.9. The caplet price in the LSM model can also be obtained through the co-terminal LSM of Gullicco *et. al.* [GHLS04], however their method is not discussed further in this study.

6.8 Calibration

Calibration is the computation of the volatility function of the market models so that the market observed prices of actively traded derivatives match as closely as possible the model derived prices. This can be done in two ways: either by using the volatilities usually given with the market prices or by calculating the implied volatilities and correlations from market or historical data. For example, Hull White [HW99], Longstaff *et. al.* [LSS01] and Rebinato and Joshi [RJ01] have used historical data, while de Jong *et. al.* [DDP01] have shown the benefits of using historical data (with error measurement corrections) with market data. Others, such as Schoenmakers and Coffey [SC00] and Brigo and Mercurio [BM01] have formulated semi-parametric or parametric forms for the forward rate correlation matrices.

If the volatilities are given by the market in the form of Black-like implied volatilities for cap prices then the calibration of caps for the LFM models is almost automatic, since one can simply input the volatilities in the model. Caplets, which are a series of caps, can then be priced in market by the Black formula. Similarly for the pricing of swaptions in the LSM.

If historical price data is used, without accompanying volatilities, the implied volatilities must be calculated. In this case, parameters of the models are then found in such a way that the market volatilities are reproduced. These newly found market volatilities are then fed back into the model to simulate the real prices. Hull and White [HW99] also point out that there may be problems with this because the data might be stale, or there may be a temporal misalignment. Alexander and Lvov [AL03] also mention that when historical data on forward rates are used, there are many source of model risk. The market convention is to use liquid instruments such as futures, FRA (forward rate agreements) and swap rates to obtain forward Libor rates by a bootstrap method. Moreover, significant noise is introduced by this method. Alexander and Lvov [AL03] suggest methods for getting historical forward rates from different yield curve fitting techniques.

Brigo *et. al.* []BCM01 point out the actual problems involved in calibration the LFM market model, and discuss some possible solutions. They show that for a successful calibration, there must be small calibration error, regular correlation structures and a relatively smooth evolution of the term structure of volatilities over time. Apparently, this is solved by Choy *et. al.* [CDS03] who use the Pederson calibration in the LFM for at-the-money caps and swaptions, which gives a good fit to market conditions.

6.8.1 Implied Volatility

Following London [Lon04], and remaining in the confines of interest rate derivates, consider a cap expiring at time T with strike price X. Denote C_{market} as the market quote of the cap. The Black model gives a fair value for this option, $C_B(\sigma)$ which depends on the historical volatility estimate. We can estimate the historical volatility as follows. Suppose we have a time series of daily prices data, for N days, S_i, i = 1, 2,...N. The continuously compounded, not annualized, return in the ;th interval is

$$u_i = ln\left(\frac{S_i}{S_{i-1}}\right), \quad i = 1,2,\ldots,N.$$

The usual unbiased estimate of volatility υ is given by

$$\upsilon^2 \Delta t = \frac{1}{N-1} \sum_{i=1}^{N} (u_i - \bar{u})^2,$$

where ū is the mean of the u_i's and Δt is the interval between observations. Implied volatility $\sigma^{implied}$ is such a value of the volatility parameter σ that the Black price patches the observed price, C_{market}, so that

$$C_B(\sigma^{implied}) = C_{market}.$$

Since the Black price is a known function of (constant) volatility, this equation can be inverted to find $\sigma^{implied}$. The closed-form solution to this problem does not exist, but it can be solved by using the Newton-Raphson numerical procedure.

6.8.2 Newton-Raphson

In the Newton-Raphson numerical procedure, we make an initial guess σ_0 of the volatility. Next we estimate σ_1 by using the Newton-Raphson equation, which is a first-order Taylor approximation of $C_B(\sigma)$

$$\sigma_1 = \sigma_0 - \frac{C_B(\sigma) - C_{market}}{\kappa_B(\sigma_1)},$$

where

$$\kappa_B(\sigma_1) = \frac{\partial C_B(\sigma_1)}{\partial \sigma}$$

is the Greek hedge statistic vega. We then use σ_1 in the next iteration to find σ_2. We continue the following iteration for each step i

$$\sigma_{i+1} = \sigma_i - \frac{C_B(\sigma) - C_{market}}{\kappa_B(\sigma_i)}$$

iteratively until the desired accuracy is reach:

$$| C_{market} - C_B(\sigma_{i+1})| \leq \xi,$$

where ξ is a very small value, such as 0.0001.

6.8.3 Calibration of caps and swaptions to market prices

One way to determine the volatility via calibration is to minimize the following function with respect to σ, where sqrt denotes the square root

$$\min_\sigma = \text{sqrt}\left(\sum_{i=1}^{N}\left(\frac{\text{model}_i(\sigma) - \text{market}_i}{\text{market}_i}\right)^2\right).$$

Caps and swaption prices in the South African financial market are quoted in percentage of the nominal. In order to compare LFM and LSM model prices with market and historical prices, we therefore must convert these quoted prices to implied volatilities and then use these volatilities for input into the models.

Chapter 7

The Computer Program

7 The Computer Program

This chapter describes the computer system needed to implement the Monte Carlo versions of the market models. We begin by describing the higher-order components that make up the computer system. Thereafter, we describe the software design, which encompasses the classes that represents the objects of the market model pricing application. The class inheritances and relationships are also discussed. Some of these classes act as a container for the implementation of the discretized versions of the market models.

7.1 System overview

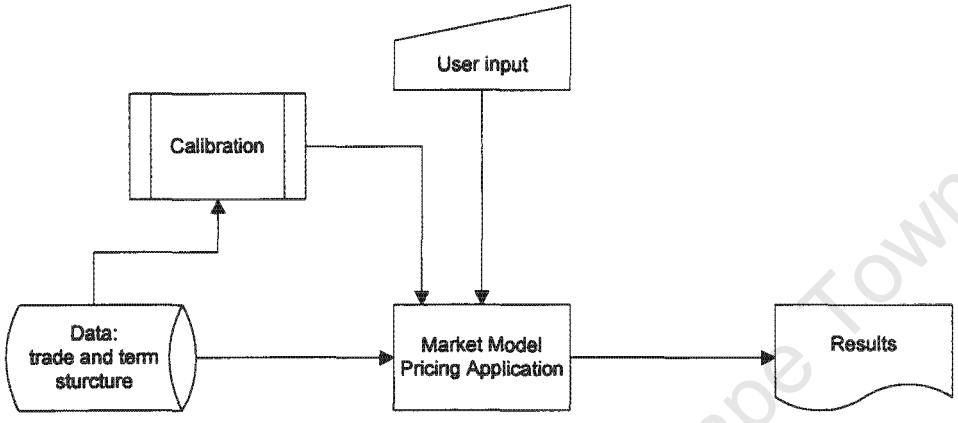
The computer system designed for pricing cap and swaption interest rate derivatives consists of the following modules: data, calibration, the market model pricing application and the results output. The user's interaction with the application is limited mainly to selecting which trade to simulate and to enter the implied volatility. A high-level view of the system is illustrated in Figure 2.

The Data file

Market data consisting of trade details and traded prices of caps and swaps are housed in a plain text file in XML format. The cap trades XML file is listed in Appendix E.14 and the swaption trades XML file is listed in Appendix E.15. The term structure was also stored in a XML file, listed in Appendix E.16. This format was chosen because it allowed data to be entered, updated and viewed easily. XML is a relatively recent development in the transfer of information, making it an interesting format to choose. It also allows the program to read the data by unmarshalling the XML structure. A novel way to parse XML data from an multi layer XML file using C/C++ and the Xerces C++ XML Library is introduced (Xerces [Xer05]). The code is shown in Appendix E.13. In the unmarshalling, each child node is read downwards until the final entity

name and value pair are read. These values are then stored in their relevant data class (FileData), from which they can be transferred to the Instrument object, which is the traded instrument that is selected for simulation. The C++ classes that represent the objects in the market model application are discussed in Section 7.2.

Figure 2 A high-level view of the computer system



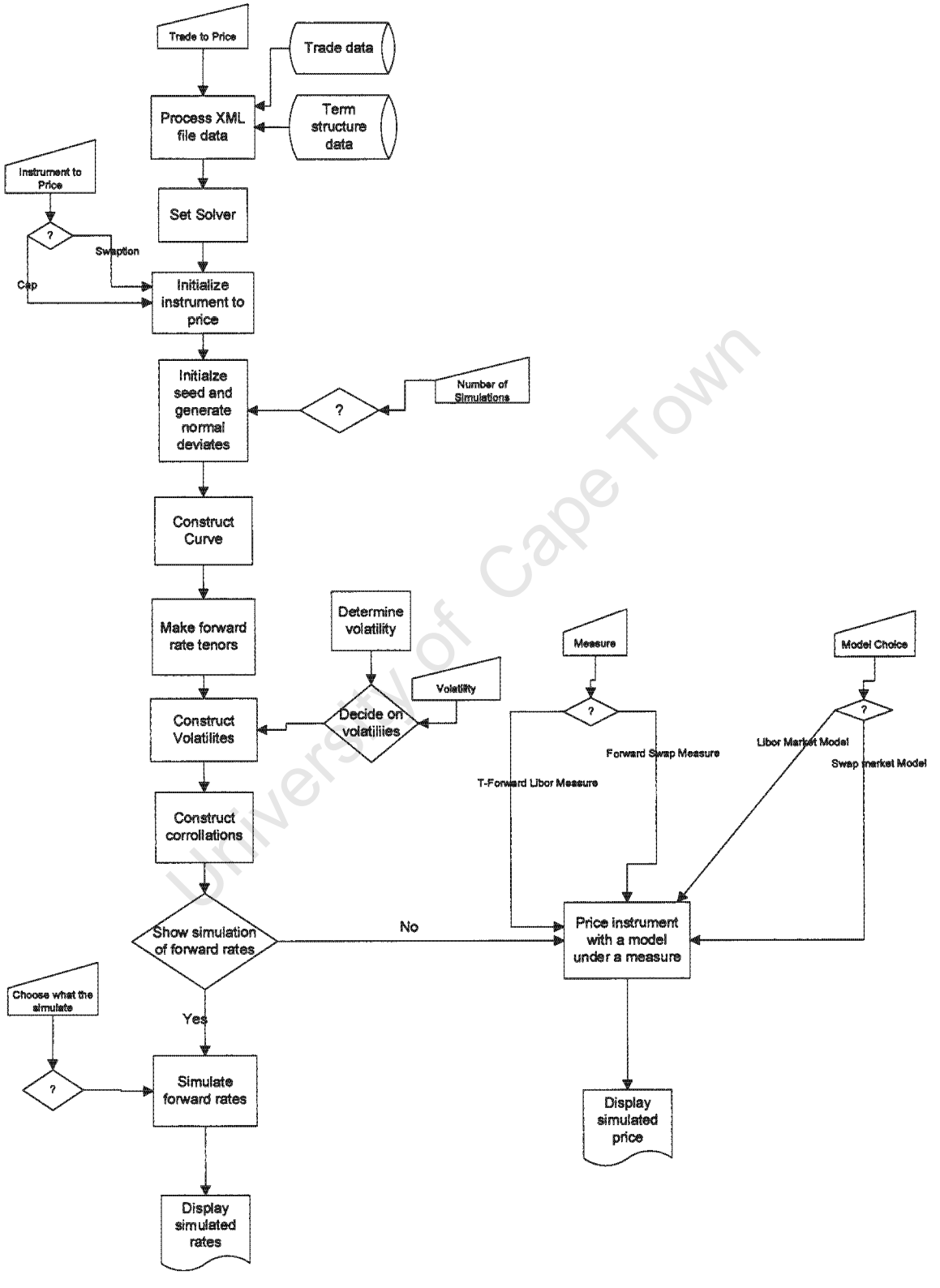
Calibration

The models were calibrated to the actual trade data by using Black's model for pricing caps and swaptions and then using a variant of the Netwon-Rasphon technique to determine the implied volatility. The implied volatility was then adjusted slightly to reflect the actual traded prices.

Market Model Pricing Application

The majority of code, comprising the application, was written to implement the Monte Carlo simulation of the forward rates under the different measures. Since the Monte Carlo simulation requires many reiterations, the speed at which the code runs plays a critical role in the viability of the implementation. As such, the code was written in C++ using object-oriented programming techniques. However, for the benefit of the reader, the code has not been optimized for speed, but has been written as an illustration of the processes involved. The code was written in the Microsoft Visual C++ development environment, on a Windows 2000 operating system. Components of the C\C++ Standard Template Library were used, in particular strings, lists and vectors.

Figure 3 Processes and information flow in the Market Model Application



The Results Output

The results of the Monte Carlo simulation were printed to a DOS (Disk Operating System) screen. When a cap was simulated, the caplet payoffs were printed together with the final cap price. Similarly with the pricing of swaptions. From there the prices were recorded on paper and stored for future reference and analysis. Three simulation runs were done for each instrument in the two models under the two measures, to give an estimate of the average of the price. The application also allows the simulated forward rates to be printed to screen, so that one can see the results of any single simulation.

The data flow and main processes in the system are shown in Figure 3.

7.2 Software Design

In this section, a description is given of the high-level overview of the classes and their components that were developed for the computer program.

7.2.1 Class inheritances and relationships

The major classes (shown in Figure 4) are the following:

- Simulate
    - SimulateCap
    - SimulateSwaption
- Model
    - InterestRateModel
        - Black
        - MarketModel
            - LiborMarketModel
            - SwapMarketModel
- Instrument
    - Bond
    - InterestRateOption
        - Cap
        - Swaption
- PayoffLeg

    - Caplet
    - SwaptionLeg
- TermStructure
- TermStructureRow
    - MonteCarlo
- Data
    - FileData
        - TradeData
        - TermStructureData
- FileTrade
- FileTermStructure
- Solver
    - MonteCarlo
- MathUtil
    - RandonNumber
    - RandomNormalVariable
    - CumulativeNormal

The utility classes from Xerces are used.

A diagramatical representation of the classes may be found in Appendix D.1.

7.2.2 Class descriptions

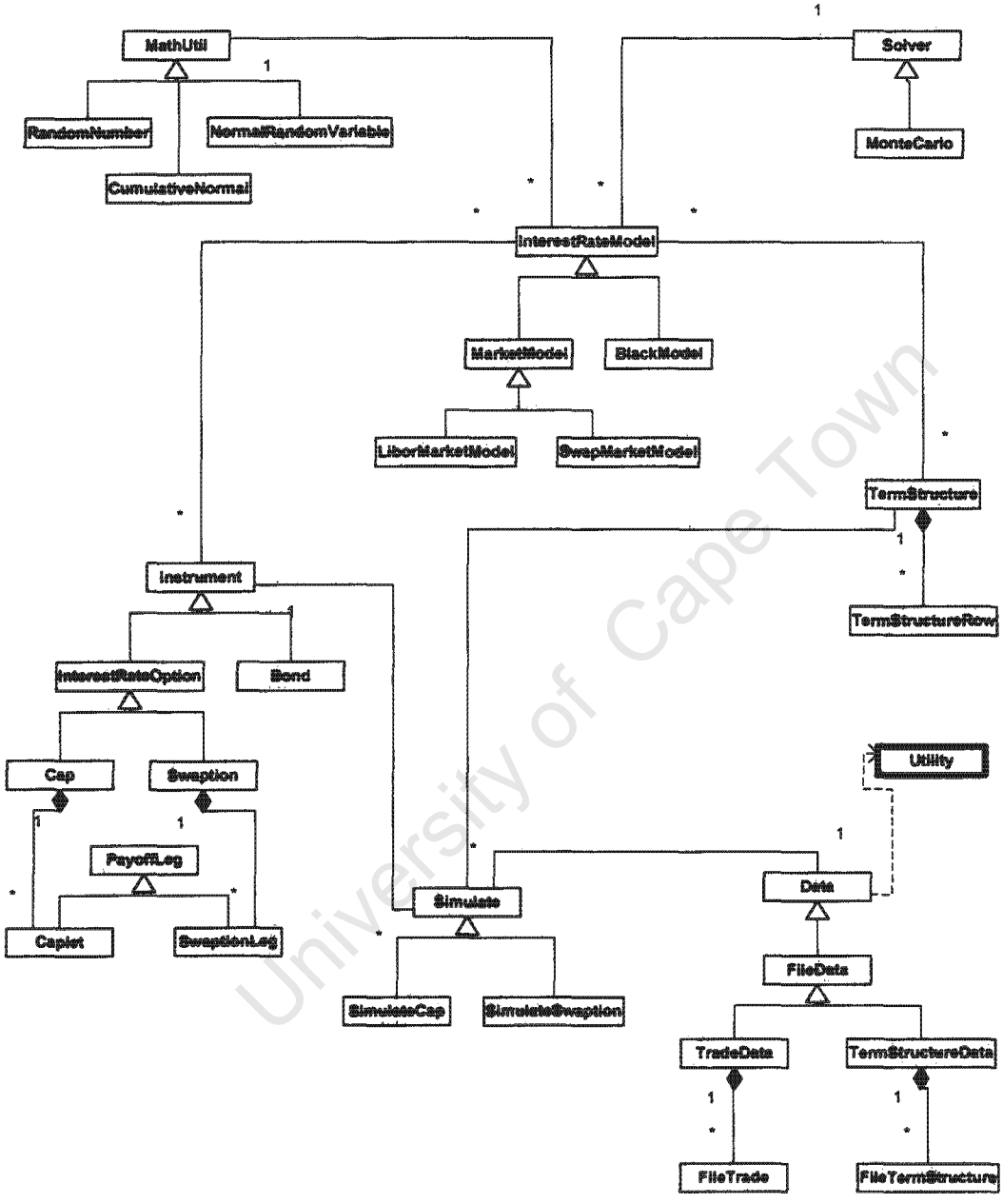
Simulate

The main purpose of the computer program is to calculate the price of a certain set of financial derivatives. This class represents the simulation of that data set. Its function is to interact with the data classes to get a specific set of trade and term structures from the XML files and make that data available to the other classes, described below, that are utilized to calculate the derivative price.

Model

This class forms the base class from which other model classes are derived. Apart from the market models, other classes of models such as Black-Scholes, Black, Vasicek or Hull-White (which are not part of this project) could be derived from this base model class. MarketModel, a child class of Model, forms the parent class of InterestRateModel, since interest rate (or term

Figure 4 Class inheritances and associations



structure) models form part of the group called market models. From the later class the LiborMarketModel and the SwapMarket Model are derived. The Market Model class has the data members and methods that are common to all of its derived classes, and in that class are placed the methods to simulate the forward rates. The caplet pricing of the Libor market model uses an appropriate method from the BlackCapletPrice of the class Black. Likewise, the swaption pricing of the Swap market model will use an appropriate method from the BlackSwaptionPrice class (also derived from Black), both of which use the Monte Carlo simulation. The class Black will be derived from Models and will represent Black's model for caplets and swaptions.

Instrument

The Instrument class represents a traded instrument. In other words, it is defined as an interest rate security that was traded in the financial markets. The price of this traded instrument is simulated by the model. In this study, we are only interested in interest rate options and their underlying securities, as well that any other tradable that are relevant to the equations. Thus, we develop classes, derived from InterestRateOption, with Instrument as a base class, such Cap and Swaption. These classes contain the relevant data members, such as strike price, nominal and maturity date (among others) and methods that are relevant to this project. Other tradable securities can easily be derived from this class.

PayoffLeg

Each Cap traded in the market has one or more payoff dates (Caplet), and each Swaption traded in the market had one or more payoff legs. This class represents those caplets and payoff legs. It is linked to the Cap and Swaption instruments by a one-to-many relationship. These classes may be used as a printout class to the payoff that comprises the final option price.

Solver

There are several numerical methods to give solutions to unsolvable stochastic differential equation, such as Monte Carlo simulation and finite differences. The models in this study use Monte Carlo simulations. In this project, the Solver class acts as a base class for Monte Carlo simulations.

TermStructure

Since this project is concerned with the modeling the term structures, it seems logical to define a class that holds a term structure. The term structure consists of a list of year fractions, for example 0.25, 0.5,...,8.0. Constructors and public functions of this class will allow the term structure to be set and retrieved.

MathUtil

At the core of Monte Carlo simulation is a random number taken from the normal distribution. RandomNumberGenerator is a class that represents the generation of this number. The random number generator is used to generate the normal deviate, which is contained in the NomalDeviate class. The CumulativeNormal represents a class that calculates the cumulative normal used in Black's formula. Several other methods are included in the model; these were not used in the simulations but were used in the preliminary results as a comparison to the method chosen for the eventual simulation.

Chapter 8

Results

8 Results

In this chapter, we present the results of the simulations. Since at the core of Monte Carlo simulation is the generation of normal deviates, we give the results of a number of normal deviate test runs. Then we give examples of the generation of forward Libor and swap rates under the forward Libor and forward swap measures. Thereafter, we present the results from the Monte Carlo simulations of prices of caps and swaption by the two models under the two forward measures. Finally, we use the models to give prices of similarly traded caps and swaption.

8.1 Market Data

Market data used in the simulations were real trades made in the South Africa interest rate derivatives market, given to us by Cadiz. The trades were made on 7 July 2005, and consisted of three cap trades and four swaption trades.

The cap trades are shown in Table 1. The caps were struck at the money (ATM) at the equivalent forward rates for the resets. ATM trades are equivalent to stating that the forward rate was equal to the current strike rate at the time of the trade. The underlying rate for the caps was the 3-month JIBAR rate. The traded prices are shown in the last column of Table 1. These prices were calculated by excluding the first caplet payment, since that payment was known (discussions with Brett Dugmore). The calculation of the implied volatility and the Monte Carlo simulation of the prices were based on this knowledge.

The swaptions, given in Table 2, were likewise real swaptions traded in the South African market. The swaptions were struck at the money (ATM) at the equivalent forward rates for the reset. The underlying rates were the 3-month and 6-month JIBAR rate. As with the caps, the first payment was known, and was therefore excluded from the quoted price. The Monte Carlo simulation of the actual price was constructed so as to also exclude the first payment leg.

The swap curve used by the market at the time these cap and swaption trades were made is given in Table 3.

Table 1 Cap ATM trades, compliments of Cadiz

Trade	Term of Option (months)	Maturity (months)	Notional (R)	Strike Rate (%)	Cap Price (R)
9m fwd 6.833 zar 1226 ATM	3	9	1,000,000	6.833	1,226
12m fwd 6.86 zar 2116 ATM	3	12	1,000,000	6.860	2,116
15m fwd 6.935 zar 3460 ATM	3	15	1,000,000	6.935	3,460

Table 2 Swaption ATM trades, compliments of Cadiz

Trade	Term of Option (months)	Maturity (years)	Notional (R)	Strike Rate (%)	Swaption Prices (R)
3m 2y fwd 7.308 ZAR 4526 ATM	3	2	1,000,000	7.308	4,526
6m 2y fwd 7.469 ZAR 6784 ATM	6	2	1,000,000	7.469	6,784
3m 5y fwd 7.868 ZAR 11504 ATM	3	5	1,000,000	7.868	11,504
6m 5y fwd 7.956 ZAR 16209 ATM	6	5	1,000,000	7.956	16,209

Table 3 Zero coupon yield curve

Date	JIBAR Rate (%)	Date	JIBAR Rate (%)	Date	JIBAR Rate (%)	Date	JIBAR Rate (%)
2005/07/08	6.7000	2005/12/07	6.8634	2006/10/09	6.9392	2012/07/09	7.9594
2005/08/08	6.7828	2006/01/09	6.9006	2007/01/08	7.0073	2013/07/08	8.0194
2005/09/07	6.8684	2006/02/07	6.8665	2007/04/10	7.0900	2015/07/07	8.0795
2005/10/07	6.9540	2006/03/07	6.8439	2007/07/09	7.1798	2017/07/07	8.0696
2005/10/13	6.9487	2006/04/07	6.8745	2008/07/07	7.4790	2020/07/07	7.9796
2005/10/20	6.9349	2006/05/08	6.8586	2009/07/07	7.6593	2025/07/07	7.8296
2005/10/27	6.9250	2006/06/07	6.8482	2010/07/07	7.7894	2030/07/08	7.7046
2005/11/07	6.9078	2006/07/07	6.8904	2011/07/07	7.8795	2035/07/09	7.5996

8.2 Normal Deviates

In the computer system used in this investigation, the random numbers and normal deviates were generated independently of the forward rate simulations. That is, for each simulation run a unique set of normal deviates were generated, stored, and then used by that forward rate simulation. This was done so that the samples of the normal deviates could be viewed to see if there were any anomalies in the series, and to print out the mean and variance of the series to check that the series conformed to that of normal deviates (with a mean of zero and a variance of 1). However, the normal deviates could have been generated precisely when needed for each simulation.

Table 4 shows an example of the mean and variance of five series of normal deviates, each series consisting of 1 million generated normal deviates. This shows that an independent and legitimate series of normal deviates were in fact generated for each simulation. The code that generated these series of normal deviates is shown in Appendix E.2.

Table 4 Simulation runs of normal deviates

Series	Number of Simulations	Mean	Variance
0	1,000,000	-0.00017	0.99985
1	1,000,000	-0.00013	0.99992
2	1,000,000	-0.00860	1.01313
3	1,000,000	0.00006	0.99962
4	1,000,000	-0.00016	1.00081
5	1,000,000	0.00029	1.00024

8.3 Forward Rate Simulations

8.3.1 Forward JIBAR rates

Table 5 and Table 6 show examples of the Monte Carlo simulation of the forward Libor (JIBAR) rates under the forward Libor measure and the forward swap measure, respectively. The

simulation of the forward rates give a sample path of Brownian motion. The data used in the forward rate simulations were taken from the first cap trade listed in Table 1. In the simulation, the number of time steps equaled the number of forward rates, M=N= 3. The term of the trade was 0.75 years with a tenor of 0.25 years. In the South African financial markets, a day-count basis of Actual/365 is used in the calculation of financial derivatives (West [Wes04]), whereas in our calculations we chose to used a simpler year fraction, such as 0.25 and 0.5. The initial spot rate was set at 0.06833 and it was assumed to have an initial flat structure. The volatility was kept constant at 12.04%. The code to simulate the forward Libor rates under the two measures is shown in Appendixes E.4 and E.5.

Table 5 Simulation of forward JIBAR rates under the forward Libor measure

	T_0	T_1	T_2	T_3
$F_0(T_t)$	0.06833			
$F_1(T_t)$	0.06833	0.07035		
$F_2(T_t)$	0.06833	0.06795	0.07221	
$F_3(T_t)$	0.06833	0.06526	0.06596	0.07409

Table 6 Simulation of forward JIBAR rates under the forward swap measure

	T_0	T_1	T_2	T_3
$F_0(T_t)$	0.06833			
$F_1(T_t)$	0.06833	0.07291		
$F_2(T_t)$	0.06833	0.06470	0.07870	
$F_3(T_t)$	0.06833	0.07096	0.05079	0.07222

8.3.2 Forward Swap rates

An example of the simulation of forward swap rates under the forward swap measure is illustrated in Table 7, and an example of the simulation of forward swap rates under the forward Libor measure is given in Table 8. In these examples, the data from the first swap trade listed in Table 2 were used . The number of time steps, M, and the number of forward rates, N, was taken as M=N=8. The time to maturity was 2 years, with a tenor of 0.25 years. The initial spot forward

rate was 7.208% and was assumed initially to be flat. The code to simulate the forward swap rates under the two measures is shown in Appendixes E.6 and E.7.

Table 7 Simulation of forward swap rates under the forward swap measure

	T_0	T_1	T_2	T_3	T_4	T_5	T_{36}	T_7	T_8
$F_0(T_t)$	0.07308								
$F_1(T_t)$	0.07308	0.07234							
$F_2(T_t)$	0.07308	0.07235	0.07242						
$F_3(T_t)$	0.07308	0.07269	0.07261	0.07280					
$F_{34}(T_t)$	0.07308	0.07292	0.07231	0.07270	0.07318				
$F_5(T_t)$	0.07308	0.07283	0.07338	0.07263	0.07261	0.07341			
$F_6(T_t)$	0.07308	0.07304	0.07317	0.07312	0.07213	0.07275	0.07366		
$F_7(T_t)$	0.07308	0.07268	0.07304	0.07342	0.0735	0.07171	0.07253	0.07390	
$F_8(T_t)$	0.07308	0.07375	0.07301	0.07325	0.0738	0.07359	0.07123	0.07254	0.07330

Table 8 Simulation of forward swap rates under the forward Libor measure

	T_0	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8
$F_0(T_t)$	0.07308								
$F_1(T_t)$	0.07308	0.07330							
$F_2(T_t)$	0.07308	0.07293	0.07359						
$F_3(T_t)$	0.07308	0.07371	0.07351	0.07417					
$F_{34}(T_t)$	0.07308	0.07284	0.07322	0.07342	0.07402				
$F_5(T_t)$	0.07308	0.07324	0.07279	0.07311	0.07358	0.07370			
$F_6(T_t)$	0.07308	0.07274	0.07266	0.07312	0.07343	0.07335	0.07439		
$F_7(T_t)$	0.07308	0.07236	0.07271	0.07342	0.07316	0.07319	0.07314	0.07449	
$F_8(T_t)$	0.07308	0.07280	0.07257	0.07325	0.07334	0.07294	0.07361	0.07283	0.07397

8.4 Results of Monte Carlo Simulation of Cap Prices

The variables used in the simulations of the three cap trades are shown in Table 9. In the headings of the tables, fwd stands for forward, Vol stand for volatility, Mat stands for time to maturity in years, Tenor is the length of time between payments, Sims is the number of

simulations. Time Steps is the number of time steps and Forward rates are the number of forward rates that are simulated, and are set equal to each other in this study.

To get a statistical average, three simulations runs of one million simulations were execute for each of the cap trades. The results are shown in Table 10. The actual traded prices are shown in the second column of the tables, while the prices recorded for the three simulation runs in the two models under the two forward measures are shown in the other columns. In the table headings, LFM stands for Libor Market Model and LSM stands for the Swap Market Model.

The results show that the actual cap prices were simulated accurately by both the LFM and the LSM under both the forward Libor measure and the forward swap measure. In general the simulated prices (in particular the shorter trades) had an error of approximately R10, which worked out to be 0.001% of the nominal. In terms of the nominal, this is considered a small error.

The average time, in minutes, taken by one simulation run is shown in Table 11, and illustrates that the time taken to complete one simulations run was faster in the LFM than the LSM. The LSM under the Libor forward measure recorded the longest time of 146 minutes.

Table 9 Variables used in the Monte Carlo simulation of Cap trades

Trade	Notional (R)	Risk free rate (%)	Spot fwd rate (%)	Vol (%)	Mat (yrs)	Tenor (yrs)	Sims	Time steps	Forward rates
9m fwd 6.833 zar 1226 ATM	1,000,000	6.945	6.833	12.04	0.75	0.25	1,000,000	3	3
12m fwd 6.86 zar 2116 ATM	1,000,000	6.945	6.860	12.83	1.0	0.25	1,000,000	4	4
15m fwd 6.935 zar 3460 ATM	1,000,000	6.945	6.935	14.80	1.25	0.25	1,000,000	5	5

Table 10 Results for Cap trades of three Monte Carlo simulation runs for each trade

		Simulated Prices			
Trade	Traded Price (R)	LFM Forward Libor Measure (R)	LFM Forward Swap Measure (R)	LSM Forward Swap Measure (R)	LSM Forward Libor Measure (R)
9m fwd 6.833 zar 1226 ATM	1226	1227.05	1224.45	1223.30	1229.00
		1225.88	1223.81	1223.21	1228.00
		1230.02	1231.89	1224.26	1226.77
12m fwd 6.86 zar 2116 ATM	2116	2117.46	2121.00	2109.32	2125.68
		2113.51	2115.95	2105.83	2115.80
		2122.10	2131.14	2107.91	2119.45
15m fwd 6.935 zar 3460 ATM	3460	3466.01	3459.39	3448.67	3468.24
		3460.44	3472.75	3435.79	3461.35
		3460.43	3484.01	3463.96	3463.09

Table 11 Average time, in minutes, taken for each Cap simulation run of 1million simulations

	Number of simulations	LFM Forward Libor Measure (minutes)	LFM Forward Swap Measure (minutes)	LSM Forward Swap Measure (minutes)	LSM Forward Libor Measure (minutes)
9m fwd 6.833 zar 1226 ATM	1,000,000	1.05	1.83	2.61	11.72
12m fwd 6.86 zar 2116 ATM	1,000,000	1.82	3.90	6.25	45.54
15m fwd 6.935 zar 3460 ATM	1,000,000	2.65	7.01	14.07	146.12

8.5 Results of Simulation of Swaption Prices

Four at–the-money (ATM) swaption trades were chosen for simulation. They are given in Table 12, with the variables used in each simulation. The heading descriptions are as for the caps, described in Section 8.4, except in the swaption case there was an underlying JIBAR rate of 3 months and 6 months with maturities of 2 and 5 years. Since the time taken for one simulation run of 1 million simulations in the LSM under the forward Libor measure would have taken an unrealistic amount of time (Table 15), it was decided to only do 100,000 simulations in those cases. This would give a less accurate simulated price, but to execute 1 million simulations for each run was considered impractical.

Table 12 Variables used in the Monte Carlo simulation of Swaption trades

Trade	Notional (R)	Risk free rate (%)	Spot fwd rate (%)	Vol (%)	Mat (yrs)	Option Term (mnths)	Tenor (mnths)	Sims	Time steps	Fwd rates
3m 2y fwd 7.308 ZAR 4526 ATM	1,000,000	6.945	7.308	9.93	2	3	0.25	1,000,000	8	8
6m 2y fwd 7.469 ZAR 6784 ATM	1,000,000	6.945	7.469	17.10	2	6	0.5	1,000,000	4	4
3m 5y fwd 7.868 ZAR 11504 ATM	1,000,000	6.945	7.868	5.50	5	3	0.25	1,000,000*	20	20
6m 5y fwd 7.956 ZAR 16209 ATM	1,000,000	6.945	7.956	8.77	5	6	0.5	1,000,000*	10	10

*Number of simulations was 10,000 for the LSM under the Forward Libor Measure

The results of the four swaption trades simulated in the LFM and LSM under the forward Libor measure and the forward swap measure are shown in Table 13. The actual live traded price is shown in the second column, while the other columns show the simulated price. Three simulation runs were executed for each trade in the two models under the different forward measures, to give a more accurate assessment of the simulated price.

From the results (Table 13) it may be seen that the LFM simulated the actual swaption prices very well under both the forward Libor measure and the forward swap measure. The simulated prices in the LSM were higher than the actual traded price, and higher than those simulated by the LFM. Nevertheless, the swaption prices simulated in the LSM under the two different

measures were similar. Since we have chosen the LFM as our primary model, the implied volatility used was that calculated for the LFM.

Table 13 Results for Swaption trades of three Monte Carlo simulation runs for each trade

		Simulated Prices			
Trade	Traded Price (R)	LFM Forward Libor Measure (R)	LFM Forward Swap Measure (R)	LSM Forward Swap Measure (R)	LSM Forward Libor Measure (R)
3m 2y fwd 7.308 ZAR 4526 ATM	4526	4519.12	4533.90	5094.62	5151.01
		4526.01	4518.18	5126.73	5198.45
		4527.83	4575.43	5110.54	5175.34
6m 2y fwd 7.469 ZAR 6784 ATM	6784	6732.01	6849.33	8530.79	8680.37
		6790.80	6810.15	8507.09	8537.86
		6799.34	6859.00	8535.92	8771.25
3m 5y fwd 7.868 ZAR 11504 ATM	11504	11538.37	11658.42	13004.24	12773.34
		11499.64	11416.24	12905.01	12394.77
		11525.09	11509.34	12803.33	12988.12
6m 5y fwd 7.956 ZAR 16209 ATM	16209	16102.06	16238.12	20146.97	21878.55
		16209.33	12617.32	20393.01	21555.35
		16270.07	16592.56	20072.42	20225.74

Table 14 Results for Swaption trades of three Monte Carlo simulation runs for each trade, using the LSM as the principle model

			Simulated Prices			
Trade	Traded Price (R)	Volatility (%)	LSM Forward Swap Measure (R)	LSM Forward Libor Measure (R)	LFM Forward Swap Measure (R)	LFM Forward Libor Measure (R)
3m 2y fwd 7.308 ZAR 4526 ATM	4526	8.83	4525.32	4529.02	3946.34	3890.12
			4573.14	4511.76	3941.24	3884.76
			4529.36	4524.98	3942.08	3872.59
6m 2y fwd 7.469 ZAR 6784 ATM	6784	13.54	6778.07	6720.19	5056.54	5064.67
			6794.32	6800.01	5052.71	5042.73
			6833.00	6774.73	5055.49	5050.43
3m 5y fwd 7.868	11504	4.60	11751.32	11488.25	9152.76	9141.91

ZAR 11504 ATM			11459.57	11511.92	9132.81	9161.37
			11559.92	11494.48	9148.22	9125.63
6m 5y fwd 7.956	16209	6.75	15985.65	15942.66	11527.99	11645.80
ZAR 16209 ATM			16219.31	16109.71	11501.81	11573.65
			16176.06	16156.02	11588.75	11655.61

We could just as easily as calibrated the LSM to an implied volatility and simulated accurate swaption prices with that implied volatility. However, with that implied volatility the LFM would have under-predicted the price. Results for the Swaption trades of three Monte Carlo simulation runs for each trade, using the LSM as the principle model confirm this prediction (Table 14).

Table 15 Average time, in minutes, taken for each Swaption simulation run of 1 million simulations

	Number of simulations	LFM Forward Libor Measure (minutes)	LFM Forward Swap Measure (minutes)	LSM Forward Swap Measure (minutes)	LSM Forward Libor Measure (minutes)
3m 2y fwd 7.308 ZAR 4526 ATM	1,000,000	0.78	3.13	9.38	194.57
6m 2y fwd 7.469 ZAR 6784 ATM	1,000,000	0.25	0.52	0.52	4.68
3m 5y fwd 7.868 ZAR 11504 ATM	1,000,000	0.78	88.50	659.91	3,800.00
6m 5y fwd 7.956 ZAR 16209 ATM	1,000,000	1.31	7.03	22.53	688.90

8.6 Using the Models to Predict Prices

The purpose of option pricing models is to calculate the, as yet undetermined, price of interest rate derivative under slightly different market conditions. If we take the results of the previous sections as an exercise in the calibration of the LFM and LSM, we can then simulate the price of similarly structured caps and swaptions under different market conditions. Here we take the first of the traded caps and swaptions from Table 9 and Table 12, respectively, and simulate new trades that are struck in-the-money (ITM) and out-the-money (OTH). ITM options means that

the strike price is below the current spot forward price, and OTM options are trades that are stuck at a price above the current spot forward rate.

The cap results of three simulation runs of 1 million simulations, using the same volatilities as in the previous results but with different strike rates, are shown Table 16. Again, it is seen that the Libor and Swap market models under the forward Libor measure and forward swap measure give prices that are very close to each other. One could then use either model under whichever measure to give an accurate cap price.

The ITM and OTM swaption simulations are shown in Table 17. Since the LFM and LSM give different prices for the swaption with the same volatility, one must choose which model to calibrate and use to predict swaption prices.

Table 16 ITM and OTM caps priced by the market models under the respective forward measures

			Simulated Prices			
Cap Trade	Strike Price (%)	Volatility (%)	LFM Forward Libor Measure (R)	LFM Forward Swap Measure (R)	LSM Forward Swap Measure (R)	LSM Forward Libor Measure (R)
9m fwd 6.833 ITM	6.50	12.04	2169.48 2175.12 2166.35	2169.52 2167.98 2176.50	2166.50 2169.94 2167.50	2175.54 2173.14 2170.01
9m fwd 6.833 OTM	7.10	12.04	725.56 721.54 725.20	720.20 724.22 722.41	722.51 722.75 721.53	726.41 726.63 722.16

Table 17 ITM and OTM swaption priced by the market models under the different forward measures

			Simulated Prices			
Swaption Trade	Strike Price (%)	Volatility (%)	LFM Forward Libor Measure (R)	LFM Forward Swap Measure (R)	LSM Forward Swap Measure (R)	LSM Forward Libor Measure (R)
3m 2y fwd 7.308 ITM	7.10	9.93	6094.27 6095.09 9094.56	6078.84 6085.98 6098.42	6906.67 6900.99 6924.12	6876.56 6835.64 6901.76

3m 2y fwd 7.308	7.50	9.93	3365.99	3344.83	3806.57	3775.49
OTM			3319.78	3363.96	3745.56	3883.46
			3337.64	3345.22	3751.62	3698.72

Chapter 9

Discussion

9 Discussion

When computing cap and swaption prices we have two possibilities for each interest rate derivative, corresponding to the two forward measures. Once we have selected which forward measure as our basic measure for pricing, we can compare the situations induced by the two different models. For a distributional comparison to make sense, we need to compare the models under the same measure. The theory behind the computation of the price under their different measures was illustrated in Section 5.3.

For caps, we can so this in the lognormal forward-Libor market model (LFM) where the forward Libor rates are expressed in terms of the LFM dynamics (under the forward Libor measure), or with the lognormal forward-Swap market model (LSM) where the forward Libor rates are expressed in terms of the LSM dynamics (under the forward swap measure). If we chose the fist method to price caps, it means we are basing our computations directly on Libor-rate dynamics. Then the Libor-rate distribution is exactly lognormal and the expectation reduces to the well know Black formula or caps. However, if we chose the second method, we are basing our computation on the dynamics of the forward Swap rates. Since forward Swap rates define a forward Libor rate only through a algebraic relationship, there is no reason for the distribution of the computer Libor rate to be lognormal.

For the swaption we can do likewise; we can either do this with the LSM where the forward swap rates are express in terms of the LSM dynamics (under the forward swap measure), or with the LFM where the forward swap rate is expressed in terms of the LFM dynamics (under the forward Libor measure). As with caps, if we chose the first method to price a swaption, it means we are basing our computations directly on swap-rate dynamics. In this case, the swap-rate distribution is exactly lognormal and the expectation reduces to the well-known Black formula. However, if we chose the second method, we are basing our computations on the dynamics of the forward Libor rates. These forward Libor rates define a forward swap rates through an algebraic relationship, and thus there is no reason for the distribution of the calculated swap rate to be lognormal.

From the above it may be deduced that while the Libor rate coming from the LFM dynamics is lognormally distributed, the Libor rate coming from the LSM dynamics is not lognormal. Like wise, while the swap rate coming from the LSM dynamics is lognormally distributed, the swap rate coming from the LFM dynamics is not lognormal. This explains the incompatibility issue from a theoretical point of view. However, is this incompatibility of the lognormal distributions also evident from a practical point of view? The results of Brigo and Mecurio (2001), and Brigo and Liinev (2003), suggest that this incompatibility is mostly theoretical, since in practice the LFM distribution of the forward swap rate is almost lognormal.

In this study, we investigated the incompatibility of the two models, not by looking at the lognormal distributions, but by computing the Monte Carlo prices of actual traded caps and swaptions under the different measures, and observing the results. The results show that, for the pricing of caps, the LFM and the LSM give prices that were close to the traded price under both the forward Libor measure and the forward swap measure. For example, the first trade from Table 10 gave average prices of 1227, 1226 in the LFM and 1223 and 1227 in the LSM. The traded price was 1226. However, in the pricing of swaptions, it is evident from Table 13 and Table 14 that the LFM and LSM give different results. The same result is observed when the LSM was taken as the primary model to calibrate against the traded data.

These results are support by those of Driessen and Pelsser [DP01], who empirically compared the LFM and LSM for the pricing of interest rate derivates using US data on caplets and swaptions. As in this study, they found that the LFM was better at predicting derivative prices than the LSM. Indeed, they found that the LFM could be directly calibrated to observed prices of caplets (caps), whereas the LSM was only calibrated to a certain set of swaption prices. In this study it was found that caps were priced consistently by the LFM and LSM under both forward measures, while the pricing of swaptions, although consistent within each model under the different forward measures, were inconsistent between the LFM and LSM. This result suggests that the LSM is less tractable the LFM, and is supported by the fact that the time taken for each simulation run was much less in the LFM than the LSM (compare Table 11 and Table 15). It is not surprising, therefore, that the dealers in the interest rate derivative market favour the LFM over the LSM (Rebonato [Reb03]).

From a theoretical point of view, the approaches used by the Libor and Swap market models are very close in their mathematical construction. Nevertheless, and partly for the reasons discussed above, the Swap market models is considered less tractable than the Libor market model (Driessen and Pelsser [DP01], Gulliccio *et. al.* [GHLS04]).This view may be justified by noticing that forward rates may be thought of as deterministic linear combinations of forward Libor rates. In addition, the cap/floor markets are more liquid that the swaption markets, which would justify a Libor-based approach in pricing. This bias is reflected in the literature, where

there are extensive publications on the Libor market model, but very little on the Swap market model. The results of this study justify this view, nevertheless Gulliccio *et. al.* [GHLS04]) argue in favour of the Swap market model. They show that the Libor market model may be thought of as a particular specification of a general swap-based approach, which they call the Market Model Approach. They analyze the properties of a so-called co-terminal (same ending) Swap market model and demonstrate that this model enjoys the same degree of mathematical tractability as the Libor market model. In particular, their drift approximations work equally well and are no less user-friendly in comparison to the Libor market model. They also show that the joint calibration of caplets and swaptions can be achieved in a faster, more robust and efficient way that the Libor market model.

Following the lead of Gulliccio *et. al.* [GHLS04]) , and others, it appears that the direction of research into the market models is towards a construction of a generic market models which encompass the Libor and swap market models. Pieterz and van Regenmortal [PR05], at present, are working on a generic market model framework, wherein the Libor and Swap market models are special cases. They formulate the model by analyzing forward rates that span periods other than the classical Libor and Swap periods. They developed generic expressions for the drift terms occurring in the dynamic equations of forward rates. However, they only show that their generic model is considered particularly apt to pricing a certain class of swaptions, such as Bermudan swaptions, fixed-maturity Bermudan swaptions and callable hybrid coupon swaps.

Chapter 10

Conclusion

10 Conclusion

In this study, we investigated the incompatibility of the Libor and Swap Market Models. It was found that the two models were indeed incompatible, both from a theoretical and practical point of view. This result was expected from the few studies in the comparison between the two models. This study went further than previous known studies by implementing the models under the different forward measures, and then comparing the results with interest rate derivatives recently traded in the South African market. The use by the market practitioners of the Libor market model for pricing, as opposed to the Swap market model, is justified by the results of the study.

The study began with a brief investigation into the background behind the market models, and the practicalities needed to implement the models were discussed. The original theories of the models are based on rigorous mathematics, and these are needed to describe them fully. Nevertheless, in this investigation the theories and derivation of the pricing formulas were only loosely described. This was considered acceptable in the context of the study, which was not intended to be mathematical in nature.

Since the investigation was computer oriented, a software system was designed and a computer program developed to implement the pricing models. The program used object-oriented techniques that allowed the addition of new classes, and hence could be expanded into a complete system for derivative pricing. The pricing data produced from the computer program illustrated the inconsistency between the two models. This incompatibility was discussed, as well as the research into the theory of market models to eradicate this inconsistency.

It was originally proposed that a graphical user interface (GUI) be attached to the system, but once the Monte Carlo simulation had been implemented, it was considered impractical due to the length of time taken for each simulation run; Monte Carlo simulation is very computer intensive. The simulations were best run as background processes administered from a consol, as opposed to having a GUI that was in a suspended mode while the simulation took place.

From a coding point of view, the program could be improved by optimizing it for speed, possibly using multi-threading. Another direction would be to expand the design into a full

interest rate option pricing system. As a side to coding, a novel way to read a multi-leveled XML by C++ code was introduced. Although only two layers were present in the structure of the XML files, this process allows the XML to be scaled to many more levels. An interesting project would be to generalize this into an algorithm, and back it with an investigation into other algorithms that are used to read XML files.

Little study has been done on the new generic market models. A exciting project would be to investigate and implement these generic models, and compare the results from the classical Libor and swap market models. There is also very little literature on the application of the market models in the South African interest rate derivative markets, so this would be another area for empirical and theoretical research.

A Appendix

A.1 Change of Numeraire

This discussion on the change of numeraire follows that of Brigo and Mercurio [BM01] and London [Lon04]. We begin by defining the numeraire and then by showing how the change of numeraire formula can be applied to a change of drift.

A.2 Numeraire Definition

A numeraire is defined as any positive non-dividend-paying asset, and is identified with a self-financing strategy φ, defined as

$$\varphi \equiv \{ (\Delta_T, N_t), 0 \leq t \leq T \}$$

in that $N_t = V_t(\varphi)$, where V is the market value of the portfolio φ for each t. From this definition, it is seen that a numeraire is a reference asset chosen so as to normalize all other asset prices, S_i, $i = 0,\ldots,n$, with respect to it so that the relative prices S_i/N are considered rather than the asset prices themselves.

Consider a trading strategy, φ, and let N be a numeraire. Then φ is self-financing if and only if

$$V_t(\varphi)/N = V_0(\varphi) + \sum_{i=1}^{n} \varphi_i \, \Delta S_i/N \, . \qquad \text{Eq. 10.1}$$

This can be extended to any numeraire, so that any self-financing strategy remains self-financing after a change of numeraire. in addition, an attainable claim is also attainable under any numeraire. This is seen from the self-financing condition

$$dV_t(\varphi) = \sum_{k=0}^{n} \varphi_t^k \, dS_t^k$$

implying that

$$d\left(\frac{V_t(\varphi)}{N_t}\right) = \sum_{k=0}^{n} \varphi_t^k \; d\left(\frac{S_t^k}{N_t}\right).$$

Now, assume that there exists a numeraire N and a probability measure Q^N, equivalent to the initial measure Q_0, such that the price of any traded asset X relative to N is a martingale under Q^N, assuming there are no intermediate payments. This gives the expectation, E^N, in filtration space, such that

$$\frac{X_t}{N_t} = E^N\left(\frac{X_t}{N_t} \mid F_t\right).$$

Generalizing, it may be shown that if U is an arbitrary numeraire, then there exits a probability measure Q^u, equivalent to the initial Q_0, so that the price of an attainable claim X, normalized by U, is a martingale under Q^u, giving the equation

$$\frac{X_t}{U_t} = E^U\left(\frac{X_t}{U_t} \mid F_t\right). \qquad \text{Eq. 10.2}$$

A.3 Radon-Nikodym derivatives

By the definition of Q^N, we know that for any tradable asset price Z,

$$E^N\left(\frac{Z_t}{N_t}\right) = E^U\left(\frac{U_0}{N_t}\,\frac{Z_t}{U_t}\right), \qquad \text{Eq. 10.3}$$

since both are equal to Z_0/N_0.

Also by definition of the Radon-Nikodym derivatives, we know that for all Z

$$E^N\left[\frac{Z_t}{N_t}\right] = E^U\left[\frac{Z_t}{N_t}\frac{dQ^N}{dQ^U}\right]. \qquad \text{Eq. 10.4}$$

By comparing the right hand side of Eq. 10.3 and Eq. 10.4, from the arbitrariness of Z, and noticing that

$$E^U\left[\frac{Z_t}{N_t}\frac{dQ^N}{dQ^U}\right] = \frac{Z_0}{N_0} \qquad \text{Eq. 10.5}$$

we obtain the Radon-Nikodym derivative defining the measure Q^U

$$\frac{dQ^U}{dQ^N} = \frac{U_t\ N_0}{U_0\ N_t}. \qquad \text{Eq. 10.6}$$

When computing the expected value of an integrable random variable X, it is often convenient to switch from one measure Q^a to another equivalent measure Q. With E^a denoting the expectation under the Q^a measure, this gives

$$E^a[X] = \int X\, dQ^a = \int X \frac{dQ^a}{dQ} dQ = E\left[X\frac{dQ^a}{dQ}\right].$$

Thus the expectation of the random variable X under the new measure is the expectation of that variable multiplied by the Radon-Nikodym derivative.

A.4 Change of Drift

When changing for a numeraire N, associated with the measure Q^N to a different numeraire U, associated with the measure Q^U the drift in the dynamics of the asset also change.

Consider the dynamics off two numeraires, S and U, which evolve under the Q^U measure according to

$$dS(t) = \mu^U(X_t, t)dt + \sigma^S(t)\, C\, dz^U(t)\,,$$
$$dU(t) = \mu^U(X_t, t)dt + \sigma^U(t)\, C\, dz^U(t)\,,$$

where the drift is denoted by $\mu^U(X_t, t)$, $\sigma^S(t)$ and $\sigma^U(t)$ are 1 x vectors of volatility, $dz^U(t)$ is an n-dimensional Brownian motion, and C is a variance-covariance matrix of Brownian motions such that CC' = ρ. Then it can be shown that drift of the process X under the numeraire Q^U is

$$\mu^U(X_t, t) = \mu^N(X_t, t) - \sigma(X_t, t)\,\rho \left(\frac{\sigma_N}{S(t)} - \frac{\sigma_U}{U(t)} \right) .$$

Since we are interested in log normality, let us assume that the volatilities take a "level-proportional" functional form

$$\sigma^S(t) = \upsilon^S(t)\, S(t)$$
$$\sigma^U(t) = \upsilon^U(t)\, U(t)$$
$$\sigma(X_t, t) = \mathrm{diag}(X_t)\mathrm{diag}(\upsilon^X(t))$$

where the υ's are deterministic 1 x n-vector functions of time and $\mathrm{diag}(X_t)$ denotes the diagonal matrix whose diagonal elements are entries of vectors X. Then according the B&M we get

$$\mu^U(X_t, t) = \mu^S(X_t, t) \; - \mathrm{diag}(X_t) \frac{d\,\langle \ln X,\, \ln(S/U) \rangle_t}{dt},$$

where the quadratic covariation and the logarithms, when applied to vectors, are meant to act on each component. When the drift of X under Q^S is deterministically level proportional, as it is in the fully lognormal case, we get

$$\mu^S(X_t, t) = \text{diag}(X_t)\, m^S(t)\,,$$

where $m^S(t)$ is a deterministic n x 1 vector. It turns out that the drift under the new measure, Q^U, is of the same type,

$$\mu^U(X_t, t) = \text{diag}(X_t)\, m^U(t)\,,$$

but with

$$m^U(t) = m^S(t) - \text{diag}(\upsilon^S(t) - \upsilon^U(t))$$

$$= m^S(t) - \frac{d\,\langle lnX,\, ln(S/U)\rangle_t}{dt}$$

$$= m^S(t) - (d\, lnX_t)(\, d\, ln(S_t/U_t)\,) \quad . \qquad \text{Eq. 10.7}$$

B Appendix

B.1 The Forward Measure

In many situations, an ideal numeraire to use is a zero-coupon bond, B(t) whose maturity T coincides with that of the derivative to price S(t), since at maturity

$$S_T = B(T,T) = 1 .$$

This simplifies calculations, since the pricing of the derivative can be achieved by computing the expectation of its payoff, divided by 1.

The measure associated with the bond maturing at time T is referred to in the literature as the T-forward risk-adjusted measure, or more simply the T-forward measure, shortened here to the "forward measure". We denote this measure by Q^T and the expectation related to it by E^T.

By applying Eq. 10.2 to the forward measure numeraire, B(T,T), it can be seen that the price of a derivative at time t, P_t, is given by

$$P_t = B(t,T)\, E^T\{H_T \mid F_t\}$$

for $0 \leq t \leq T$, and where H_T is the option payoff at time T.

Consider the definition of a simply compounded forward rate from Eq. 3.5

$$F(t; S, T) \equiv \frac{1}{\delta(S,T)} \left(\frac{B(t, S)}{B(t, T)} - 1 \right)$$

where $\delta(S,T)$ is the year fraction from S to T. Then

$$F(t; S, T)\, B(t, T) = \frac{1}{\delta(S,T)} \left(B(t, S) - B(t, T) \right)$$

is the price of a traded asset, since it is the multiple of the difference of two bonds. By definition, of the forward measure

$$\frac{F(t; S, T)\, B(t, T)}{B(t, T)} = F(t; S, T)$$

is a martingale under such a measure.

Since the expected value of any future instantaneous spot interest rate, R_T, under the corresponding forward measure, is equal to the related instantaneous forward rate, F_T, in other words

$$F(t; S, T) = R(S, T)$$

it follows that

$$E^T\{R(S,T) \mid \mathcal{F}_t\} = F(t;S,T)$$

for each $0 \leq t \leq S \leq T$.

In general, Brigo and Mercurio [BM01] state that in any simply compounded forward rate spanning a time interval ending in T is a martingale under the forward measure, i.e.

$$E^T\{F(t;S,T) \mid \mathcal{F}_t\} = F(u;S,T)$$

for each $0 \leq u \leq t \leq S \leq T$.

C Appendix

C.1 Ito's Formula

A concise derivation of Ito's formula is given below. Let S(t) be a process such that

$$dS = \mu S dt + \sigma S dW \ .$$

Consider the process $f(S(t), t)$ where f is a smooth function. Using a Taylor series expansion, we obtain

$$df = \frac{\partial f}{\partial t} dt + \frac{\partial f}{\partial S} dS + \frac{1}{2} \frac{\partial^2 f}{\partial S^2} (dS)^2$$

$$= \frac{\partial f}{\partial t} dt + \frac{\partial f}{\partial S} (\mu dS + \sigma S dW) + \frac{1}{2} \frac{\partial^2 f}{\partial S^2} \sigma S^2 dt$$

$$= \left(\frac{\partial f}{\partial t} + \frac{\partial f}{\partial S} \mu dS + \frac{1}{2} \frac{\partial^2 f}{\partial S^2} \sigma S^2 \right) dt + \sigma S \frac{\partial f}{\partial S} dW$$

where $(dW)^2 = dt$ and ignored higher order terms. If we let $X = f(S) = lnS$ then we obtain the following Wiener process

$$dX = (\mu - \tfrac{1}{2} \sigma^2) dt + \sigma dW.$$

This process is the continuous lognormal model for stock behavior.
In a risk-neutral world the drift term must earn the risk-free rate, so that

$$\frac{\partial f}{\partial t} + \frac{\partial f}{\partial S}\mu dS + \frac{1}{2}\frac{\partial^2 f}{\partial S^2}\sigma S^2 dt = rf,$$

which is equivalent to the Black-Scholes partial differential diffusion equation.

D Appendix

D.1 Market Model Application Classes

Figure 5 The MathUtil class

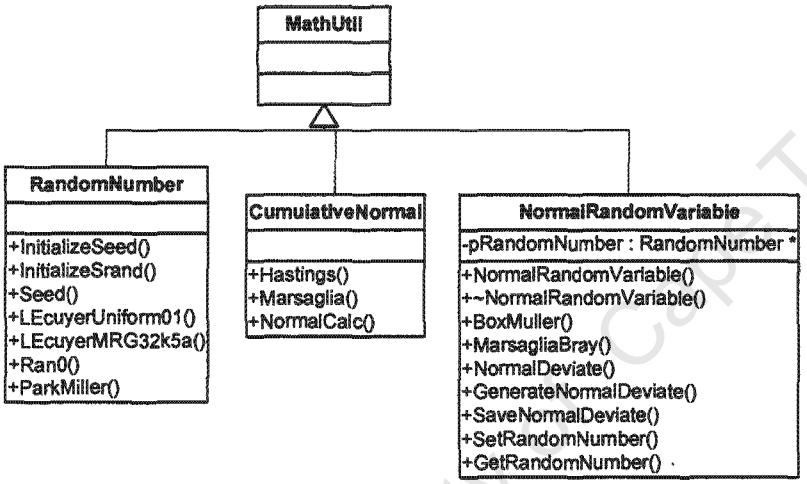


Figure 6 The Solver class, which forms the base class for MonteCarlo

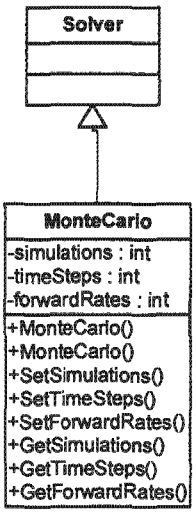


Figure 7 The Instrument class, representing the traded instruments that are simulated

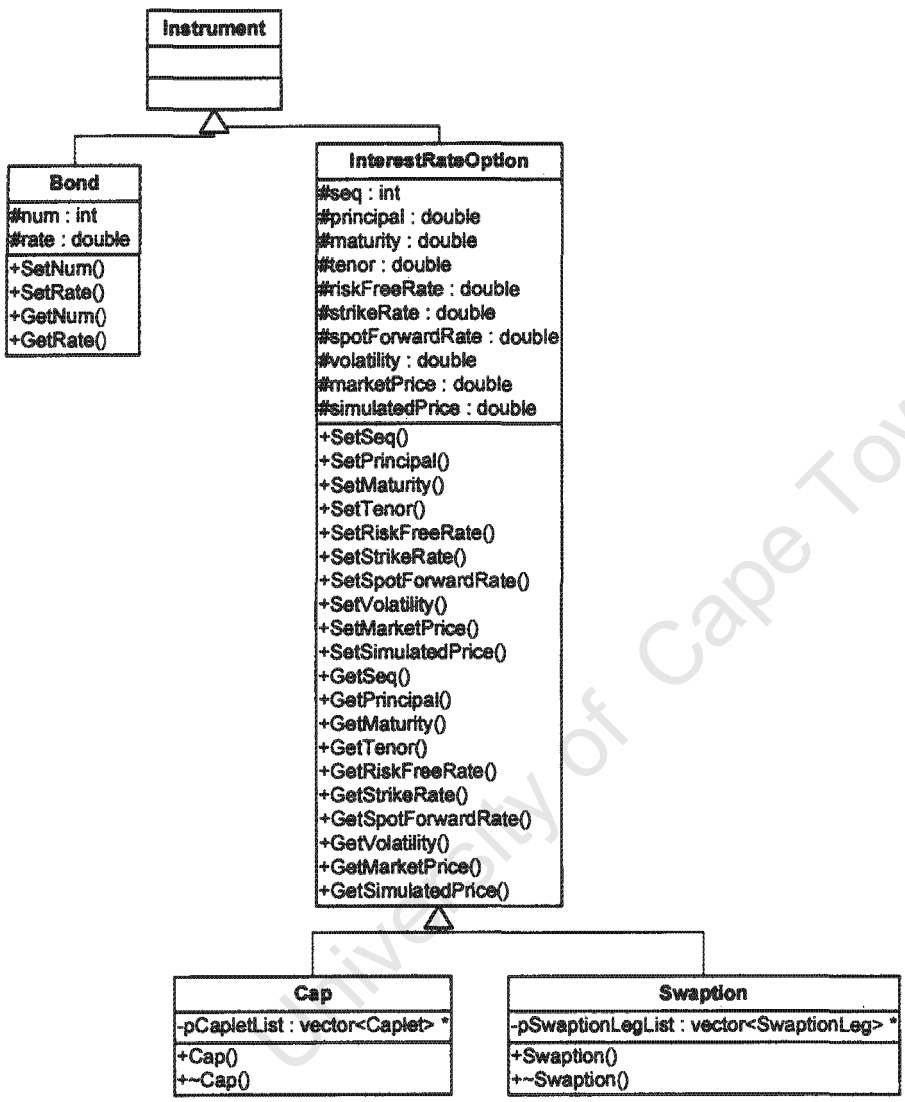


Figure 8 The InterestRateModel class. The Libor Market Model and the Swap Market Model are derived from this class

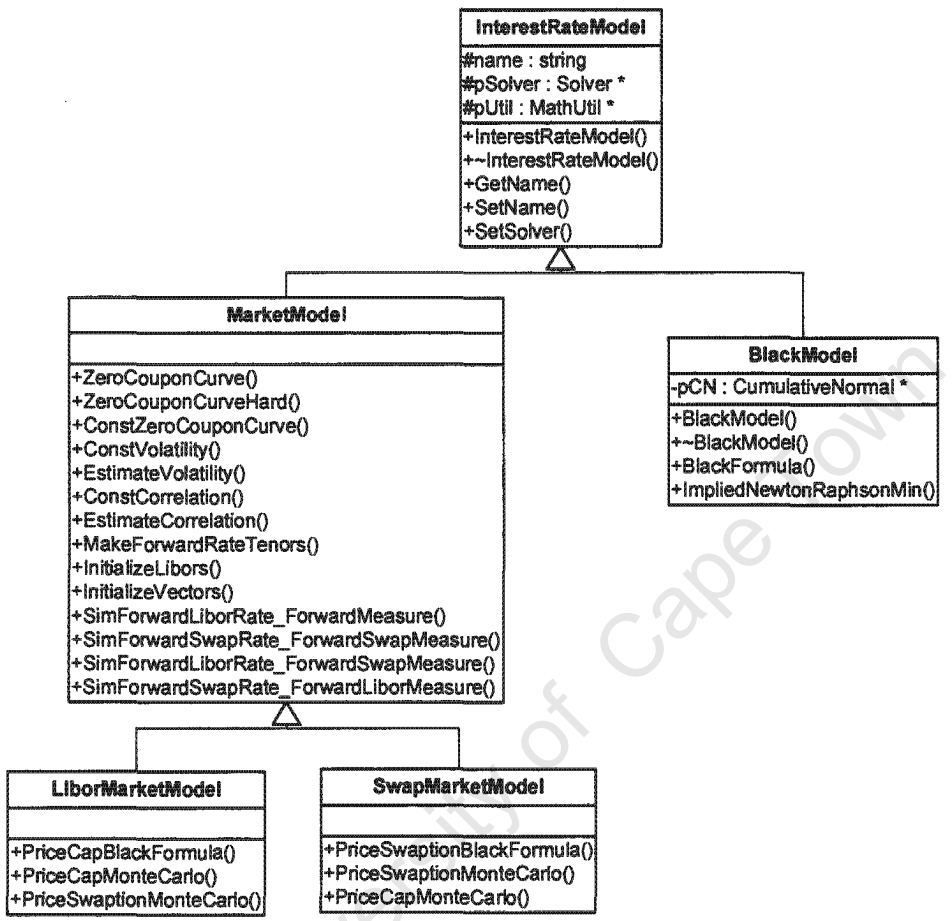


Figure 9 The Simulated class, used as a transfer link between the data in the XML files and the instruments to be modeled

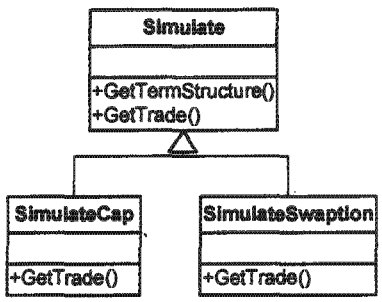


Figure 10 The base Data class representing the Trade and TermStructure data that are stored in XML files

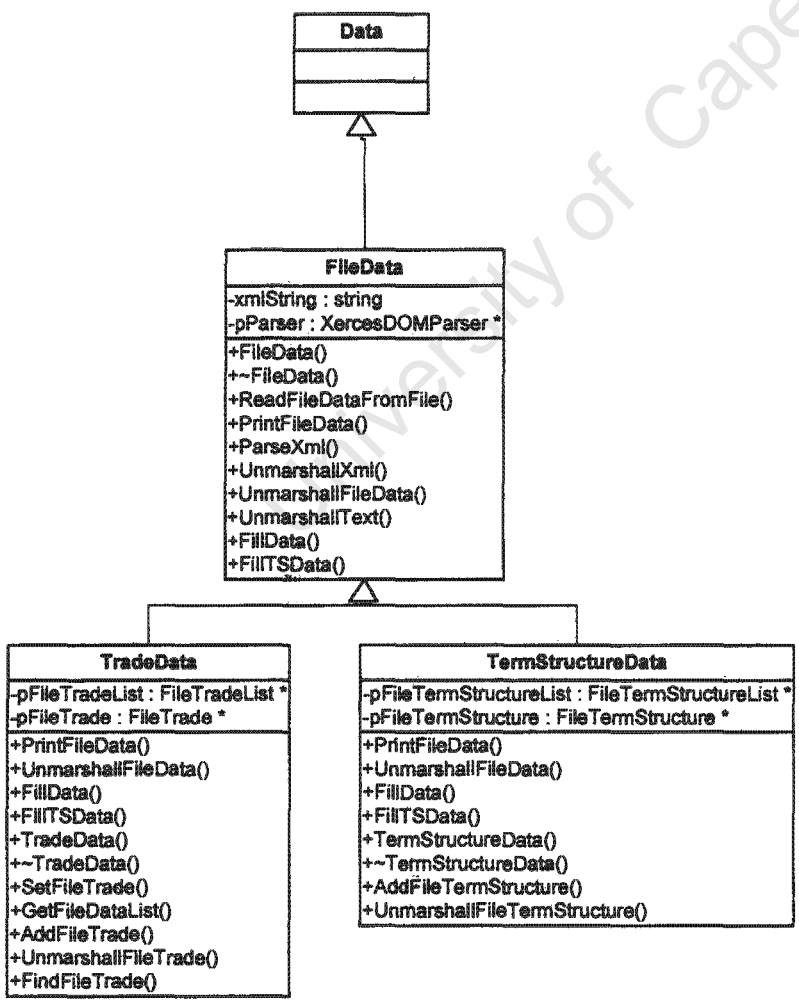


Figure 11 Classes representing each Trade and TermStructure record that are stored in XML files

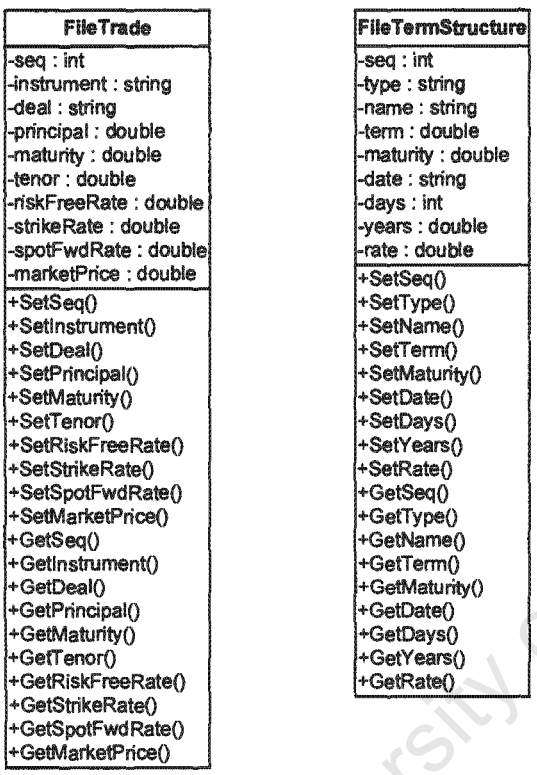
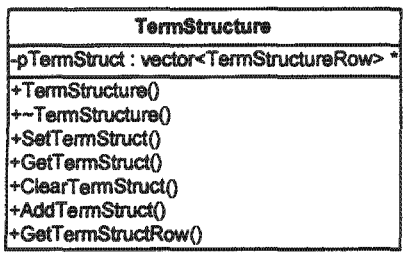
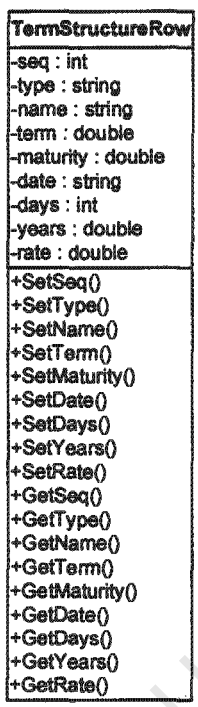


Figure 12 The term structure used by the models, associated by one-to-many to the rows in the term structure





E Appendix

E.1 Code

Appendix E lists the major components of the code that comprise the market model application to price cap and swaption interest rate derivatives in the lognormal forward-Libor and Swap market models using Monte Carlo simulation.

E.2 Generation of Normal Deviates

```
void NormalRandomVariable::GenerateNormalDeviate(long N,
                                                 long seed,
                                                 vector<double>& Z,
                                                 double* mean,
                                                 double* variance)

{
    int i;
    double sum;

    sum = 0.0;
    *mean = 0.0;
    Z.resize(N);
    for (i = 0; i < N; i++)
    {
        Z[i] = NormalDeviate(&seed);
        sum += Z[i];
    }
    *mean = sum/N;

    sum = 0.0;
    *variance = 0.0;
    for (i=0; i<N; i++)
    {
        sum += pow(Z[i] - *mean,2);
    }
    *variance = sum/(N-1);
}

double NormalRandomVariable::NormalDeviate(long* idum)
{

    // Adapted from Press et al. (1992) p289 (gasdev)
    static int iset = 0;
```

```
    static double gset;
    double fac, rsq, v1, v2;

    long *p = idum;

    if (*p < 0)          //reinitialize
        iset = 0;

    if (iset == 0)
    {
        do
        {
            v1 = 2.0*pRandomNumber->ParkMiller(p) - 1.0;
            v2 = 2.0*pRandomNumber->ParkMiller(p) - 1.0;

            rsq = v1*v1 + v2*v2;

        } while (rsq >= 1.0 || rsq == 0.0);

        fac = sqrt (-2.0*log(rsq)/rsq);

        // Box-Muller transformation to get two normal deviates, return one and save other
        gset = v1*fac;

        iset = 1;

        return v2*fac;
    }
    else
    {
        iset = 0;

        return gset;
    }
}

double RandomNumber::ParkMiller(long* idum)
{
    // Adapted from Press et. al. (1992) page 280 (ran1) after Park and Miller

    const long ia = 16807.0;
    const long im = 2147483647.0;
    const double am = 1.0/im;
    const long iq = 127773.0;
    const long ir = 2836.0;
    const long ntab = 32;
    const double ndiv = 1.0 + ((im - 1.0)/ntab);
    const double eps = 1.2e-7;
    const double rnmx = 1.0 - eps;

    int j;
    long k;
    static long iy = 0;
    static long iv[ntab];
```

```
    double temp;

    if (*idum <= 0 || !iy) // initialize
    {
        if (-(*idum) < 1)  // prevent idum = 0
            *idum = 1;
        else
            *idum = -(*idum);

        for (j = ntab+7; j >= 0; j--)
        {
            k = (*idum)/iq;

            *idum = ia*(*idum - k*iq) - ir*k;

            if (*idum < 0)
                *idum += im;

            if (j < ntab)
                iv[j] = *idum;
        }

        iy = iv[0];

    }

    k = (*idum)/iq;                 // start here when not initializing

          //compute idum=(ia*idum)%im without overflows from Schrage's method
    *idum = ia*(*idum - k*iq) - ir*k;
    if (*idum < 0)
        *idum += im;

    j = iy/ndiv;                    // will be in range 0..ntab-1

    iy = iv[j];                     // output previously stored value  and refill the shuffle table

    iv[j] = *idum;

    temp = am*iy;

    if ( temp > rnmx ) //users don't expect endpoint values
        return rnmx;
    else
        return temp;
}

void RandomNumber::InitializeSeed(long* seed)
{
    NormalRandomVariable nrv;
    nrv.GetRandomNumber()->InitializeSrand();

    *seed = (long)rand();
}
```

```
void RandomNumber::InitializeSrand()
{
    struct _timeb timebuffer;
    struct tm *lt;
    char localTime[256];

    // Get current date/time
    _ftime( &timebuffer );
    lt = localtime(&timebuffer.time);

    sprintf(localTime, "%02d%02d%03hu", lt->tm_min, lt->tm_sec, timebuffer.millitm);

    long t = atol(localTime);

    srand(t); // initialize the seeding
}
```

E.3 Black Model

```
double LiborMarketModel::PriceCapBlackFormula(  Cap& cap,                       // cap
                                                vector<vector<double> > F,      // simulated forward Libor rate
                                                vector<vector<double> > B,      // zero coupon bonds
                                                vector<double>& C,              // vector of caplets
                                                const vector<double> V,         // volatility
                                                const vector<double> T,         // tenor
                                                const vector<double> M,         // maturity
                                                const long P,                   // principal amount
                                                const long N,                   // number of forward rates
                                                const double K,                 // strike rate
                                                const double spot,              // spot (current )rate
                                                double& price)                  // cap price
{
    BlackModel blackModel;

    int i,k;

    // caplet prices
    C.resize(N+1);
    for (i = 0; i <= N; i++)
    {
        C[i] = 0.0;
    }

    // calculate caplets through Blacks Formula
    price = 0;
    for (i = 1; i <= N; i++)
    {
        C[i] = blackModel.BlackFormula(  F[i][i],       // forward rate
                                         B[0][i],       // price of pure discount bond
                                         P,             // principal amount of bond
                                         K,             // cap rate (strike
                                         V[i],          // cap volatility
                                         M[i],          // payment dates
                                         T[i]);         // tenor

        // cap price
        price += C[i];
    }

    return price;
}

double SwapMarketModel::PriceSwaptionBlackFormula(Swaption& swaption            // swaption
                                                  vector<vector<double> > F,    // simulated forward Libor rate
                                                  vector<vector<double> > B,    // zero coupon bonds
                                                  vector<double>& S,            // vector of swaptions
                                                  const vector<double> V,       // volatility
                                                  const vector<double> T,       // tenor
```

```
                                        const vector<double> M,     // maturity
                                        const long P,               // principal amount
                                        const long N,               // number of forward rates
                                        const double K,             // strike rate
                                        const double spot,          // spot (current )rate
                                        double& price)              // cap price
{
    BlackModel blackModel;

    int i;

    // swaption prices
    S.resize(N+1);
    for (i = 0; i <= N; i++)
    {
        S[i] = 0.0;
    }

    // calculate swaption through Blacks Formula
    price = 0;
    for (i = 1; i <= N; i++)
    {
        S[i] = blackModel.BlackFormula(F[i][i],    // forward rate
                                       B[0][i],    // price of pure discount bond
                                       P,          // principal amount of bond
                                       K,          // cap rate (strike)
                                       V[i],       // cap volatility
                                       M[i],       // payment dates
                                       T[i]);      // tenor

        // swaption price
        price += S[i];
    }

    return price;

}

// Computes the price of a interest rate derivative using Black's 1976 model
double BlackModel::BlackFormula(const double F,  // forward rate
                                const double B,  // price of pure discount bond
                                const double P,  // principal amount of bond
                                const double K,  // strike rate
                                const double V,  // market volatility
                                const double M,  //  time to maturity
                                const double T)  // fixed tenor between reset times
{
    double d1 = (log(F/K) + (V*V/2)*M)/(V*sqrt(M));

    double d2 = d1 - V*sqrt(M);

    return B*P*(T/(1+F*T))*(F*pCN->normalCalc(d1) - K*pCN->normalCalc(d2));
}

// Calculates the implied volatility for price input by Newton-Raphson
```

```
double BlackModel::ImpliedNewtonRaphsonMin(    const double F,   // forward rate
                                               const double B,   // price of pure discount bond
                                               const double P,   // principal amount of bond
                                               const double K,   // strike
                                               const double Vl,  // initial low guess of implied volatility
                                               const double Vh,  // initial high guess of implied volatility
                                               const double M,   // time to maturity
                                               const double T,   // fixed tenor between reset times
                                               const double ep,  // epsilon
                                               const double MP) // market price
{

    double BPl = 0.0; // Black price low
    double BPh = 0.0; // Black price high
    double BPi = 0.0; // Black price i+1
    double voll = Vl; // vol low
    double volh = Vh; // vol low
    double voli = 0.0; // vol i+1
    double error = 0.0;

    do
    {
        BPl = BlackFormula(F, B, P, K, voll, M, T);
        BPh = BlackFormula(F, B, P, K, volh, M, T);

        voli = voll + (MP - BPl)*((volh-voll)/(BPh-BPl));

        BPi = BlackFormula(F, B, P, K, voli, M, T);

        if (BPi < MP)
            voll = voli;
        else if (BPi > MP)
            volh = voli;
        else
            return voli;

    }
    while (fabs(MP - BPi) >= ep);

    return voli;

}
```

E.4 Simulation of Libor rates under the Forward Libor Measure

```
// Simulate forward Libor rates under the forward (terminal) measure for Libor rates
void MarketModel::SimForwardLiborRate_ForwardMeasure(
                                    vector<vector<double> >& F,  // simulated forward Libor rate
                                    vector<double> L,            // Libor rate
                                    vector<vector<double> > R,   // corralation matrix of forward rates
                                    vector<double> V,            // volatility
                                    vector<double> T,            // tenor
                                    const long seed,             // seed for random number generator
                                    const long M,                // number of time steps
                                    const long N,                // number of forward rates
                                    const double term,           // time to maturity
                                    const double spot,           // initial libor spot rate
                                    const vector<double>& Z,     // normal deviates
                                    const long s)                // simulation run
{
    int i,k,t;
    double dt = (double)term/M;
    int z = 0;
    int n = 0; // start

    // initialize forward rates
    F.resize(M+2);
    for (t = 0; t <= M; t++)
    {
        F[t].resize(N+2);
        for (k = 0; k <= N; k++)
        {
            F[t][k] = 0.0;

            // set to initial spot
            F[0][k] = spot;
        }
    }


    // loop through M number of timesteps
    for (t = n; t < M; t++)
    {
        // for each timestep generate N forward rates
        for (k = n+1; k <= N; k++)
        {
            double drift = 0.0;

            for (i = k+1; i < N; i++)
                drift -= ((R[k][i]*T[i]*V[k]*F[t][i-1])/(1.0 + T[i]*F[t][i-1]));
```

```
                    double logF = log(F[t][k-1]) + V[k]*drift*dt - 0.5*(V[k]*V[k])*dt  V[k]*Z[s+(z++)]*sqrt(dt);

            F[t+1][k] = exp(logF);

        } //forward rates k

    } //timesteps t
}
```

E.5 Simulation of Libor rates under the Forward Swap Measure

```
void MarketModel::SimForwardSwapRate_ForwardSwapMeasure(
                                        vector<vector<double> >& F,  // simulated forward Libor rate
                                        vector<double> L,            // Libor rate
                                        vector<vector<double> > R,   // corralation matrix of forward rates
                                        vector<double> V,            // volatility
                                        vector<double> T,            // tenor
                                        const long seed,             // seed for random number generator
                                        const long M,                // number of time steps
                                        const long N,                // number of forward rates
                                        const double term,           // time to maturity
                                        const double spot,           // initial libor spot rate
                                        const vector<double>& Z,     // normal deviates
                                        const long s)                // simulation run
{
    int i,k,t,l,j;
    double dt = (double)term/M;
    int z = 0;
    int n = 0; // start

    // initialize forward rates
    F.resize(M+2);
    for (t = 0; t <= M; t++)
    {
        F[t].resize(N+2);
        for (k = 0; k <= N; k++)
        {
            F[t][k] = 0.0;

            // set to initial spot
            F[0][k] = spot;
        }
    }


    // loop through M number of timesteps
    for (t = n; t < M; t++)
    {
        // for each time step generate N forward rates
        for (k = n+1; k <= N; k++)
        {
            double y = F[t][k-1];
            double drift = 0.0;

            // calculate the drift
            //upper sum
```

```
            double upperSum = 0.0;
            for (i = n; i <= N; i++)
            {
                double sum = 0.0;
                //lower sum over product
                for (l = n+1; l <= i; l++)
                {
                    double prod = 1.0;
                    for (j = n+1,j != l; j <= i; j++)
                    {
                        prod *= (1 + T[t]*F[t][k-1]);
                    }
                    sum += T[t]*F[t][k-1]*V[t]*prod;
                }
                upperSum += sum;
            }

            //lower sum
            double lowerSum = 0.0;
            for (i = n; i <= N; i++)
            {
                double prod = 1.0;
                for (j = n+1; j <= i; j++)
                {
                    prod *= (1 + T[t]*F[t][k-1]);
                }

                lowerSum += prod;
            }

            drift = -upperSum/lowerSum;

            double logF = log(F[t][k-1]) + V[k]*drift*dt - 0.5*(V[k]*V[k])*dt + V[k]*Z[s+(z++)]*sqrt(dt);

            F[t+1][k] = exp(logF);

        } //forward rates k

    } // timesteps t

}
```

E.6 Simulation of Swap rates under the Forward Swap Measure

```
void MarketModel::SimForwardLiborRate_ForwardSwapMeasure(
                                        vector<vector<double> >& F,  // simulated forward Libor rate
                                        vector<double> L,            // Libor rate
                                        vector<vector<double> > R,   // corralation matrix of forward rates
                                        vector<double> V,            // volatility
                                        vector<double> T,            // tenor
                                        const long seed,             // seed for random number generator
                                        const long M,                // number of time steps
                                        const long N,                // number of forward rates
                                        const double term,           // time to maturity
                                        const double spot,           // initial libor spot rate
                                        const vector<double>& Z,     // normal deviates
                                        const long s)                // simulation run

{
    int i,k,t,j;
    double dt = (double)term/M;
    int z = 0;
    vector<double> Cn;
    vector<double> Bn;
    int n = 0; // start

    Cn.resize(M+2);
    Bn.resize(M+2);
    for (t = 0; t <= M+1; t++)
    {
        Cn[t] = 1.0;
        Bn[t] = 1.0;
    }

    // initialize forward rates
    F.resize(M+1);
    for (t = 0; t <= M; t++)
    {
        F[t].resize(N+1);
        for (k = 0; k <= N; k++)
        {
            F[t][k] = 0.0;

            // set to initial spot
            F[0][k] = spot;
        }
    }
```

```
// loop through M number of timesteps
for (t = n; t < M; t++)
{
    // Pn(t) P(t, Tn) numaire
    for (k = 0; k <= M; k++)
    {
        double bn = 1.0;
        for (i = k; i <= M; i++)
        {
            bn *= 1.0/(1.0 + T[i]*F[t][i]);
        }
        Bn[k] = bn;
    }

    // forward swap numeraire Cn,N(t)
    double cn = 0.0;
    for (k = 0; k <= M; k++)
    {
        // forward swap numeraire Cn,N(t)
        double bn = 1.0;
        for (i = k; i <= M; i++)
        {
            bn *= 1.0/(1.0 + T[i]*F[t][i]);
        }
        Cn[k] += T[k]*bn;
    }

    // for each timestep generate M forward rates
    for (k = n+1; k <= N; k++)
    {
        // calculate the drift
        double drift = 0.0;
        for (j = n+1; j <= N; j++)
        {
            int ind = (j <= k) ? 1 : 0;

            // left sum
            double leftTerm = ((double)(2*ind - 1))*T[j]*(Bn[j]/Cn[j]);

            // right sum term
            double sumRight = 0.0;
            for (i = __min(k+1,j+1); i <= __max(k,j); i++)
            {
                sumRight += (T[i]*R[k][i]*V[i]*F[t][k-1])/(1.0 + T[i]*F[t][k-1]);
            }

            // drift
            drift += leftTerm*sumRight;
        }

        double logF = log(F[t][k-1]) + V[k]*drift*dt - 0.5*(V[k]*V[k])*dt + V[k]*Z[s+(z++)]*sqrt(dt);

        F[t+1][k] = exp(logF);
```

```
        } //forward rates k

    } //timesteps t

}
```

E.7 Simulation of Swap rates under the Forward Libor Measure

```
void MarketModel::SimForwardSwapRate_ForwardLiborMeasure(
                                    vector<vector<double> >& F,  // simulated forward Libor rate
                                    vector<double> L,            // Libor rate
                                    vector<vector<double> > R,   // corralation matrix of forward rates
                                    vector<double> V,            // volatility
                                    vector<double> T,            // tenor
                                    const long seed,             // seed for random number generator
                                    const long M,                // number of time steps
                                    const long N,                // number of forward rates
                                    const double term,           // time to maturity
                                    const double spot,           // initial libor spot rate
                                    const vector<double>& Z,     // normal deviates
                                    const long s)                // simulation run
{
    int i,j,k,t,g,h, u;
    long seedIt = seed;
    double dt = (double)term/M;
    vector<vector<double> > MU; //MUh,g(t)
    vector<vector<double> > D; // drift
    double z = 0;
    int l = 0;
    int n = 0; // start

    // initialize vectors
    F.resize(M+2);
    MU.resize(M+2);
    D.resize(M+2);
    for (t = 0; t <= M; t++)
    {
        F[t].resize(N+2);
        MU[t].resize(N+2);
        D[t].resize(N+2);
        for (k = 0; k <= N; k++)
        {
            F[t][k] = 0.0;
            MU[t][k] = 0.0;
            D[t][k] = 0.0;

            // set to initial spot
            F[0][k] = spot;
        }
    }


    // loop through M number of timesteps
```

```
for (t = n; t < M; t++)
{
    // for each timestep generate N forward rates
    for (k = n+1; k <= N; k++)
    {
        // calculate the drift
        double drift = 0.0;

        ////////////// Mn(t) ////////////////////////
        double num = 0.0;
        double FPn = 1.0;
        for (h = n+1; h <= M; h++)
        {
            for (g = n+1; h <= M; h++)
            {
                // MU(t)
                ////////////////////////////////////////////////////
                //upperSum1 for MU
                double upperSumMU1 = 0.0;
                for (u = g; u <= M; u++)
                {
                    //upperSum3 for MU
                    double upperSumMU3 = 0.0;
                    for (i = h; i <= M; i++)
                    {
                        //FP(t;Tn;Ti)
                        double FPi = 1.0;
                        for (j = n+1; j <= M; j++)
                        {
                            FPi *= 1.0/(1.0 + T[j]*F[t][j-1]);
                        }

                        upperSumMU3 += T[i]*FPi;
                    }

                    //upperSum2 for MU
                    double upperSumMU2 = 0.0;
                    for (i = n+1; i <= h-1; i++)
                    {
                        //FP(t;Tn;Ti)
                        double FPi = 1.0;
                        for (j = n+1; j <= M; j++)
                        {
                            FPi *= 1.0/(1.0 + T[j]*F[t][j-1]);
                        }

                        upperSumMU2 += T[i]*FPi;
                    }

                    //FP(t;Tn;TN)
                    FPn = 1.0;
                    for (j = n+1; j <= M; j++)
                    {
                        FPn *= 1.0/(1.0 + T[j]*F[t][j-1]);
```

```
        }

        //FP(t;Tn;Ti)
        double FPi = 1.0;
        for (j = n+1; j <= M; j++)
        {
            FPi *= 1.0/(1.0 + T[j]*F[t][j-1]);
        }

        upperSumMU1 += T[u]*FPi*(FPn*upperSumMU2 + upperSumMU3);
    }


    //lower sum for MU
    double denMU = 0.0;
    double lowerSumMU = 0.0;
    for (i = n+1; i <= M; i++)
    {
        //FP(t;Tn;Ti)
        double FPi = 1.0;
        for (j = n+1; j <= M; j++)
        {
            FPi *= 1.0/(1.0 + T[j]*F[t][j-1]);
        }

        lowerSumMU += T[i]*FPi;
    }
    // denominator for MU
    denMU = pow(lowerSumMU,2);

    MU[h][g] = upperSumMU1/denMU;


    ////////////////////////////////////////////////////////

    double FPh = 1.0;
    for (i = h+1; i <= M; i++)
    {
        FPh *= 1.0/(1.0 + T[i]*F[t][i-1]);
    }

    double FPg = 1.0;
    for (i = g+1; i <= M; i++)
    {
        FPg *= 1.0/(1.0 + T[i]*F[t][i-1]);
    }

    num += MU[h][g]*T[h]*T[g]*FPh*FPg*R[g][h]*V[h]*V[g]*F[t][g-1]*F[t][h-1];

  } //g
}// h


double den = 1.0 - FPn;
```

```
            drift = num/den;

            D[t][k] = drift;

            ////////////////////////////////////////////////////

            double logF = log(F[t][k-1]) + V[k]*drift*dt - 0.5*(V[k]*V[k])*dt + V[k]*Z[s+(z++)]*sqrt(dt);

            F[t+1][k] = exp(logF);

        } //forward rates k

    } // time steps t

}
```

E.8 Pricing of Caps in the Libor Market Model

```
double LiborMarketModel::PriceCapMonteCarlo(
                                Cap& cap,                          // cap
                                const vector<double> D,            // Libor rates
                                const vector<double> L,            // Libor rates
                                vector<vector<double> >& F,        // simulated forward Libor rate
                                vector<double>& C,                 // vector of caplets
                                const vector<vector<double> > R,   // corralation matrix of forward rates
                                const vector<double> V,            // volatility
                                const vector<double> T,            // tenor
                                const long seed,                   // seed for random number generator
                                const long sim,                    // number of simulations
                                const long M,                      // number of time steps
                                const long N,                      // number of forward rates
                                const double term,                 // time to maturity
                                const long P,                      // principal amount
                                const double K,                    // strike rate
                                const double spot,                 // spot forward rate
                                const char* measure,               // measure
                                vector<double>& Z,                 // deviates
                                double &price,                     // calculated price
                                double &std,                       // standard deviation
                                double &stdErr)                    // standard error
{
    // See Glasserman [Gla04], p179

    int k, t, s;          // looping

    int n = 0; // start

    double sum1 = 0.0;
    double sum2 = 0.0;

    vector<double> B0;
    vector<double> Bn;

    C.resize(M+2);
    B0.resize(M+2);
    Bn.resize(M+2);
    for (t = 0; t <= M+1; t++)
    {
        C[t] = 0.0;
        B0[t] = 0.0;
        Bn[t] = 0.0;
    }

    // zero coupon bond
    double bo = 1.0;
    for (k = n; k <= N; k++)
```

```
{
    bo *= 1.0/(1.0 + T[k]*L[k+1]);

    B0[k] = bo;
}

for (s = 1; s <= sim; s++)
{
    // simulate under the T-forward libor measure
    if (strcmp(measure, "FLM" ) == 0)
    {
        SimForwardLiborRate_ForwardMeasure(  F,     // simulated forward Libor rate
                                             L,     // initial libor rates
                                             R,     // corralation matrix of forward rates
                                             V,     // volatility
                                             T,     // tenor
                                             seed,  // seed for random number generator
                                             M,     // number of time steps
                                             N,     // number of forward rates
                                             term,  // time to maturity
                                             spot,  // initial libor spot rate
                                             Z,     // deviates
                                             s);    // simulation
    }
    // simulate under the forward swap measure
    else
    {
        SimForwardLiborRate_ForwardSwapMeasure(  F,     // simulated forward Libor rate
                                                 L,     // initial libor rates
                                                 R,     // corralation matrix of forward rates
                                                 V,     // volatility
                                                 T,     // tenor
                                                 seed,  // seed for random number generator
                                                 M,     // number of time steps
                                                 N,     // number of forward rates
                                                 term,  // time to maturity
                                                 spot,  // initial libor spot rate
                                                 Z,     // deviates
                                                 s);    // simulation
    }

    // Bn+1 numeraire
    for (t = n+1; t <= N; t++)
    {
        double bn = 1.0;
        for (k = t; k <= N; k++)
        {
            bn *= 1.0/(1.0 + T[k]*F[t][k]);
        }

        Bn[t] = bn;
    }

    // for each time step
```

```
        double po = 0;
        for (t = n+1; t <= M; t++)
        {
            po = B0[M]/Bn[t]*T[t]*P*(__max(F[t][t] - K, 0)); // payoff is the caplet price

            C[t] += po;

            sum1 += po;
            sum2 += po*po;

        } //timeSteps t

    } // simulations s

    price = 0.0;
    for (int i = 0; i <= M; i++)
    {
        // cap
        C[i] = C[i]/(double)sim;

        // first caplet is ignored
        if (i > 1)
            price += C[i];
    }

    cap.SetSimulatedPrice(price);

    std = sqrt( (sum2 - sum1*sum1/sim)/(sim-1) );

    stdErr = std/sqrt(sim);

    return price;
}
```

E.9 Pricing of Caps in the Swap Market Model

```
double SwapMarketModel::PriceCapMonteCarlo(   Cap& cap,                          // cap
                                              const vector<double> D,            // libor rates
                                              const vector<double> L,            // libor rates
                                              vector<vector<double> >& F,        // simulated forward Libor rate
                                              vector<double>& C,                 // vector of caplets
                                              const vector<vector<double> > R,   // corralation matrix of forward rates
                                              const vector<double> V,            // volatility
                                              const vector<double> T,            // tenor
                                              const long seed,                   // seed for random number generator
                                              const long sim,                    // number of simulations
                                              const long M,                      // number of time steps
                                              const long N,                      // number of forward rates
                                              const double term,                 // time to maturity
                                              const long P,                      // principal amount
                                              const double K,                    // strike rate
                                              const double spot,                 // spot (current )rate
                                              const char* measure,               // measure
                                              vector<double>& Z,                 // deviates
                                              double &price,                     // calculated price
                                              double &std,                       // standard deviation
                                              double &stdErr)                    // standard error
{
    // See Glasserman & Zhoa [GZ01], p52

    int i,k,t,s;          // looping

    int n = 0; // start

    double sum1 = 0.0;
    double sum2 = 0.0;

    vector<double> FLR; // forward libor rate
    vector<double> B0;   // B0 zero coupon bond
    vector<double> Bn;   // Bn+1 numeraire

    // Initialize
    C.resize(M+2);
    B0.resize(M+2);
    Bn.resize(M+2);
    FLR.resize(M+2);
    for (t = 0; t <= M+1; t++)
    {
        C[t] = 0.0;
        B0[t] = 0.0;
        Bn[t] = 0.0;
        FLR[t] = 0.0;
    }

    // zero coupon bond
```

```
double bo = 1.0;
for (k = 0; k <= N; k++)
{
    bo *= 1.0/(1.0 + T[k]*L[k+1]);

    B0[k] = bo;
}

for (s = 1; s <= sim; s++)
{
    // simulate under the forward swap measure
    if (strcmp(measure, "FSM" ) == 0)
    {
        SimForwardSwapRate_ForwardSwapMeasure( F,     // simulated forward swap rate
                                               L,     // libor rates
                                               R,     // corralation matrix of forward rates
                                               V,     // volatility
                                               T,     // tenor
                                               seed,  // seed for random number generator
                                               M,     // number of time steps
                                               N,     // number of forward rates
                                               term,  // term
                                               spot,  // spot
                                               Z,     // normal deviates
                                               s);    // number of simulations
    }
    // simulate under the forward Libor measure
    else
    {
        SimForwardSwapRate_ForwardLiborMeasure( F,     // simulated forward swap rate
                                                L,     // libor rates
                                                R,     // corralation matrix of forward rates
                                                V,     // volatility
                                                T,     // tenor
                                                seed,  // seed for random number generator
                                                M,     // number of time steps
                                                N,     // number of forward rates
                                                term,  // term
                                                spot,  // spot
                                                Z,     // normal deviates
                                                s);    // number of simulations
    }

    // Bn+1 numeraire
    for (t = n+1; t <= N; t++)
    {
        double bn = 1.0;
        for (k = t; k <= N; k++)
        {
            bn *= 1.0/(1.0 + T[k]*F[t][k]);
        }

        Bn[t] = bn;
    }
```

```
    // recover forward libor rates from the forward swap rates
    double n = 0;
    for (t = n+1; t <= M; t++)
    {
        //upper sum
        double upperSum = 0.0;
        for (k = n; k <= M; k++)
        {
            double prod = 1.0;
            for (i = n+1; i <= k; i++)
            {
                prod *= (1.0 + T[t]*F[t][t]);
            }
            upperSum += prod;
        }

        //lower sum
        double lowerSum = 0;
        for (k = n+1; k <= M; k++)
        {
            double prod = 1.0;
            for (i = n+2; i <= k; i++)
            {
                prod *= (1.0 + T[t]*F[t][t]);
            }
            lowerSum += prod;
        }


        double upper = 1.0 + T[t]*F[t][t]*upperSum;

        double lower = 1.0 + T[t]*F[t][t]*lowerSum;

        FLR[t] = ((upper/lower) - 1.0)/T[t];

    }


    // for each time step
    double po = 0.0; //payoff
    for (t = n+1; t <= N; t++)
    {
        double po = B0[N]/Bn[t]*T[t]*P*(__max(FLR[t] - K, 0)); //payoff

        C[t] += po;

        sum1 += po;
        sum2 += po*po;

    } //timesteps t

} // simulations s
```

```
    price = 0.0;
    for (i = 0; i <= M; i++)
    {
        // caplets
        C[i] = C[i]/(double)sim;

        // first caplet is ignored
        if (i > 1)
            price += C[i];

        FLR[t] = FLR[t]/(double)sim;
    }

    cap.SetSimulatedPrice(price);

    std = sqrt( (sum2 - sum1*sum1/sim)/(sim-1) );

    stdErr = std/sqrt(sim);

    return price;
}
```

E.10 Pricing of Swaptions in the Swap Market Model

```
double SwapMarketModel::PriceSwaptionMonteCarlo(  Swaption& swaption,                  // swaption
                                                  vector<vector<double> >& F,          // simulated forward Libor rate
                                                  vector<double>& S,                   // swappayoffs
                                                  const vector<vector<double> > R,     // corralation matrix of forward rates
                                                  const vector<double> V,              // volatility
                                                  const vector<double> T,              // tenor
                                                  const vector<double> L,              // Libor rate
                                                  const long seed,                     // seed for random number generator
                                                  const long sim,                      // number of simulations
                                                  const long M,                        // number of time steps
                                                  const long N,                        // number of forward rates
                                                  const long term,                     // time to maturity
                                                  const long P,                        // principal amount
                                                  const double K,                      // strike rate
                                                  const double riskFree,               // risk free rate
                                                  const double spot,                   // spot (current )rate
                                                  const char* measure,                 // measure
                                                  vector<double>& Z,                   // deviates
                                                  double &price,                       // simulated price
                                                  double &std,                         // standard deviation
                                                  double &stdErr)                      // standard error
{
    // See Glasserman & Zhoa [GZ01], p54

    int i,k, t, s;          // looping

    int n = 0; // start

    double sum1 = 0.0;
    double sum2 = 0.0;

    vector<double> B0;
    vector<double> Bn;

    B0.resize(M+2);
    Bn.resize(M+2);
    for (t = 0; t <= M+1; t++)
    {
        B0[t] = 1.0;
        Bn[t] = 1.0;
    }

    // zero coupon bond
    double bo = 1.0;
    for (k = 0; k <= N; k++)
    {
        bo *= 1.0/(1.0 + T[k]*L[k+1]);
```

```
    B0[k] = bo;
}

for (s = 1; s <= sim; s++)
{
    // simulate under the forward swap measure
    if (strcmp(measure, "FSM" ) == 0)
    {
        SimForwardSwapRate_ForwardSwapMeasure( F,     // simulated forward swap rate
                                               L,     // libor rates
                                               R,     // corralation matrix of forward rates
                                               V,     // volatility
                                               T,     // tenor
                                               seed,  // seed for random number generator
                                               M,     // number of time steps
                                               N,     // number of forward rates
                                               term,  // term
                                               spot,  // spot
                                               Z,     // normal deviates
                                               s);    // number of simulations
    }
    // simulate under the forward libor measure
    else
    {
        SimForwardSwapRate_ForwardLiborMeasure( F,     // simulated forward swap rate
                                                L,     // libor rates
                                                R,     // corralation matrix of forward rates
                                                V,     // volatility
                                                T,     // tenor
                                                seed,  // seed for random number generator
                                                M,     // number of time steps
                                                N,     // number of forward rates
                                                term,  // term
                                                spot,  // spot
                                                Z,     // normal deviates
                                                s);    // number of simulations
    }

    // Bn+1 numeraire
    for (t = n+1; t <= N; t++)
    {
        double bn = 1.0;
        for (k = t; k <= N; k++)
        {
            bn *= 1.0/(1.0 + T[k]*F[t][k]);
        }
        Bn[t]  = bn;
    }

    // for each time step
    double po = 0; //payoff
    for (t = n+1; t <= M; t++)
    {
```

```
            double po = B0[M]/Bn[t]*P*T[t]*(__max(F[t][t] - K, 0)); //payoff

            S[t] += po;

            sum1 += po;
            sum2 += po*po;

        } //timeSteps t

    } // simulations s


    price = 0.0;
    for (i=0; i<=M; i++)
    {
        // swaption legs
        S[i] = S[i]/(double)sim;

        // first swaption leg is ignored
        if (i > 1)
            price += S[i];
    }

    swaption.SetSimulatedPrice(price);

    std = sqrt( (sum2 - sum1*sum1/sim)/(sim-1) );

    stdErr = std/sqrt(sim);

    return price;
}
```

E.11 Pricing of Swaptions in the Libor Market Model

```
double LiborMarketModel::PriceSwaptionMonteCarlo(  Swaption& swaption,                  // swaption
                                                   vector<vector<double> >& F,          // simulated forward Libor rate
                                                   vector<double>& S,                   // swappayoffs
                                                   const vector<vector<double> > R,     // corralation matrix of forward rates
                                                   const vector<double> V,              // volatility
                                                   const vector<double> T,              // tenor
                                                   const vector<double> L,              // Libor rate
                                                   const long seed,                     // seed for random number generator
                                                   const long sim,                      // number of simulations
                                                   const long M,                        // number of time steps
                                                   const long N,                        // number of forward rates
                                                   const long term,                     // time to maturity
                                                   const long P,                        // principal amount
                                                   const double K,                      // strike rate
                                                   const double riskFree,               // risk free rate
                                                   const double spot,                   // spot (current )rate
                                                   const char* measure,                 // measure
                                                   vector<double>& Z,                   // deviate
                                                   double &price,                       // calculated price
                                                   double &std,                         // standard deviation
                                                   double &stdErr)                      // standard error
{
    // See London  [Lon04],  p 652

    int s,i,k,t;          // looping

    int n = 0; // start

    double sum1 = 0.0;
    double sum2 = 0.0;

    vector<double> FSR;  // forward swap rate

    vector<double> B0;
    vector<double> Bn;

    FSR.resize(M+2);
    B0.resize(M+2);
    Bn.resize(M+2);
    for (t = 0; t <= M+1; t++)
    {
        FSR[t] = 0.0;
        B0[t] = 1.0;
        Bn[t] = 1.0;
    }

    // zero coupon bond
```

```
double bo = 1.0;
for (k = 0; k <= N; k++)
{
    bo *= 1.0/(1.0 + T[k]*L[k+1]);

    B0[k] = bo;
}

for (s = 1; s <= sim; s++)
{
    // simulate under the forward libor measure
    if (strcmp(measure, "FLM" ) == 0)
    {

        SimForwardLiborRate_ForwardMeasure( F,      // simulated forward Libor rate
                                            L,      // libor rates
                                            R,      // corralation matrix of forward rates
                                            V,      // volatility
                                            T,      //tenor
                                            seed,   // seed for random number generator
                                            M,      // number of time steps
                                            N,      // number of forward rates
                                            term,   // time to maturity
                                            spot,   // initial libor spot rate
                                            Z,      // deviates
                                            s);     // simulations
    }
    // simulate under the forward swap measure
    else
    {
        SimForwardLiborRate_ForwardSwapMeasure( F,      // simulated forward Libor rate
                                                L,      // libor rates
                                                R,      // corralation matrix of forward rates
                                                V,      // volatility
                                                T,      //tenor
                                                seed,   // seed for random number generator
                                                M,      // number of time steps
                                                N,      // number of forward rates
                                                term,   // time to maturity
                                                spot,   // initial libor spot rate
                                                Z,      // deviates
                                                s);     // simulations

    }

    // Bn+1 numeraire
    for (t = n+1; t <= N; t++)
    {
        double bn = 1.0;
        for (k = t; k <= N; k++)
        {
            bn *= 1.0/(1.0 + T[k]*F[t][k]);
        }
```

```
            Bn[t] = bn;
        }

        // calc forward swap rate
        for (t = n+1; t <= M; t++)
        {
            // upper product
            double UP = 1.0;
            for (k = t; k <= N; k++)
            {
                UP *= (1.0/(1.0 + T[k]*F[k][k]));
            }

            // lower product
            double sumLP = 0.0;
            for (i = t; i <= N; i++)
            {
                // lowerProduct
                double LP = 1.0;
                for (k = i+1; k <= N; k++)
                {
                    LP *= (1.0/(1.0 +  T[k]*F[k][k]));
                }
                sumLP += T[i]*LP;
            }

            // forward swap rate
            FSR[t] = (1.0 - UP)/sumLP;

        } //timeSteps k

        // calc swaption payoff
        for (t = n+1; t <= M; t++)
        {
            //payoff
            double po = B0[M]/Bn[t]*P*T[t]*(__max(FSR[t] - K, 0));

            S[t] += po;

            sum1 += po;
            sum2 += po*po;

        } //timeSteps k

    } // simulations s

    // calc swaption price
    price = 0.0;
    for (i=0; i<=M; i++)
    {
        // swaption legs
        S[i] = S[i]/(double)sim;
```

```
            // first swaption leg is ignored
            if (i > 1)
                price += S[i];
        }

        swaption.SetSimulatedPrice(price);

        std = sqrt( (sum2 - sum1*sum1/sim)/(sim-1) );

        stdErr = std/sqrt(sim);

        return price;
}
```

E.12 The Main function of MarketModelApp

```
using namespace std;

#include "Instrument.h"
#include "Model.h"
#include "Data.h"
#include "Simulate.h"
#include "TermStructure.h"

int main(int argc, char* argv[])
{
    int i,j;

    // models
    MarketModel mm;
    LiborMarketModel LFM;
    SwapMarketModel LSM;
    RandomNumber rn;

    // maths
    NormalRandomVariable nrv;

    // simulate
    Simulate* pSimulate = NULL;

    // term structure
    TermStructure aTermStructure;
    TermStructureData termStructureData;

    // instruments
    Cap aCap;
    Swaption aSwaption;

    // initial pricing choices
    int instrument;      // traded instrument to simulated price
    int seq;             // sequence of instrument in traded data
    int model;           // selected model to use
    int measure;         // measure to price with
    int numsims;         // number of simulations
    char yn[2];          // yes no to show simulation of rates

    // pricing variables
    int sim;             // number of simulations
    int M;               // number of time steps
    int N;               // number of forward rates
    double term;         // time to maturity years
    double riskFree;     // initial libor spot rate
    double spot;         // initial forward spot rate
    double K;            // strike rate
```

```
double P;              // principal amount
double tenor;          // tenor
double vol;            // volatility
float volatility;
double price;
long seed;             // seed for random number generator
double std = 0;        // standard deviation
double stdErr = 0;     // standard error
double mean = 0.0;     // mean
double var = 0.0;      // variance

vector<double> L;              // libor rates
vector<double> D;              // days in libor rates
vector<vector<double> > F;     // forward libor rates
vector<vector<double> > S;     // forward swap rates
vector<vector<double> > R;     // corralation matrix of forward rates
vector<double> V;              // volatility
vector<double> T;              // vector of tenors
vector<double> A;              // swaplets
vector<double> C;              // caplets
vector<double> Z;

try
{
    printf("\n");
    printf("\nMonte Carlo Simulation of Caps and Swaptions");
    printf("\n");

    printf("\n~~~~~~~~~~~~~~~~~~~~");
    printf("\nWhich instrument to price:");
    printf("\n1. Cap");
    printf("\n2. Swaption");
    printf("\nEnter: ");
    scanf("%d",&instrument);

    printf("\n~~~~~~~~~~~~~~~~~~~~");
    printf("\nWhich model to use:");
    printf("\n1. Libor Market Model");
    printf("\n2. Swap Market Model");
    printf("\nEnter: ");
    scanf("%d",&model);

    printf("\n~~~~~~~~~~~~~~~~~~~~");
    printf("\nUnder which measure:");
    printf("\n1. Forward Libor measure ");
    printf("\n2. Forward Swap measure");
    printf("\nEnter: ");
    scanf("%d",&measure);

    printf("\n~~~~~~~~~~~~~~~~~~~~");
    printf("\nHow many simulations:");
    printf("\nEnter: ");
    scanf("%d",&numsims);
```

```
printf("\n~~~~~~~~~~~~~~~~~~~");
printf("\nVolatility (%%):");
printf("\nEnter: ");
scanf("%f",&volatility);
volatility = volatility/100;

printf("\nInstrument: %d\n", instrument);

// get pricing variables
/*************************************************************************/

switch (instrument)
{
    case 1:
        {
            printf("\nWhich trade to price:");
            printf("\n 0 - 2 for a cap");
            printf("\nEnter: ");
            scanf("%d",&seq);

            SimulateCap simCap;
            TradeData tradeData;

            pSimulate = &simCap;
            pSimulate->GetTrade(seq, &aCap, &tradeData);

            aCap.SetVolatility(volatility);

            pSimulate->GetTermStructure(aCap.GetTenor(), &aTermStructure, &termStructureData);

            // set the solver, in this case Monte Carlo
            MonteCarlo mc;
            mc.SetSimulations(numsims);
            mc.SetTimeSteps((int)(aCap.GetMaturity()/aCap.GetTenor()));
            mc.SetForwardRates((int)(aCap.GetMaturity()/aCap.GetTenor()));

            sim = mc.GetSimulations();
            M = mc.GetTimeSteps();
            N = mc.GetForwardRates();
            term = aCap.GetMaturity();
            riskFree = aCap.GetRiskFreeRate();
            spot = aCap.GetSpotForwardRate();
            K = aCap.GetStrikeRate();
            P = aCap.GetPrincipal();
            tenor = aCap.GetTenor();
            vol = aCap.GetVolatility();

            break;
        }
    case 2:
        {
            printf("\nWhich trade to price:");
            printf("\n 0 - 3 for a swaption");
```

```
            printf("\nEnter: ");
            scanf("%d",&seq);

            SimulateSwaption simSwaption;
            TradeData tradeData;

            pSimulate = &simSwaption;
            pSimulate->GetTrade(seq, &aSwaption, &tradeData);

            aSwaption.SetVolatility(volatility);

            pSimulate->GetTermStructure(aSwaption.GetTenor(), &aTermStructure, &termStructureData);

            // set the solver, in this case Monte Carlo
            MonteCarlo mc;
            mc.SetSimulations(numsims);
            mc.SetTimeSteps((int)(aSwaption.GetMaturity() /aSwaption.GetTenor()));
            mc.SetForwardRates((int)(aSwaption.GetMaturity() /aSwaption.GetTenor()));

            sim = mc.GetSimulations();
            M = mc.GetTimeSteps();
            N = mc.GetForwardRates();
            term = aSwaption.GetMaturity();
            riskFree = aSwaption.GetRiskFreeRate();
            spot = aSwaption.GetSpotForwardRate();
            K = aSwaption.GetStrikeRate();
            P = aSwaption.GetPrincipal();
            tenor = aSwaption.GetTenor();
            vol = aSwaption.GetVolatility();

            break;
        }
    default:
        printf("\nInstrument 1 or 2 is required");

};

// pricing variables
/*************************************************************************/
printf("\nPricing :");
printf("\nSimulations  = %d",sim);
printf("\nTimeSteps    = %d",M);
printf("\nForwardRates = %d",N);
printf("\nPrincipal    = %.5f",P);
printf("\nMaturity     = %.5f",term);
printf("\nTenor        = %.5f",tenor);
printf("\nRiskFreeRate = %.5f",riskFree);
printf("\nStrikeRate   = %.5f",K);
printf("\nSpotFwdRate = %.5f",spot);
printf("\nVolatility = %.5f",vol);

// convert data to vectors
/*************************************************************************/
```

```
// initialize vectors
mm.InitializeVectors(N, M, F, S, R, C, A);

// initialize seed
rn.InitializeSeed(&seed);
seed = -1*seed;

// generate normal deviates
long num = sim*M*N;
nrv.GenerateNormalDeviate(num, seed, Z, &mean, &var);
printf("\n~~~~~~~~~~~~~~~~~~~");
printf("\nNormal Deviate");
printf("\nmean= %.5f",mean);
printf("\nvar= %.5f",var);

// set zero coupon curve
mm.ZeroCouponCurve(aTermStructure, L);

// make forward rate tenors
mm.MakeForwardRateTenors(tenor, M, T);

// constant volatility
mm.ConstVolatility(V, vol, M );

// correlation;
LFM.ConstCorrelation(R,M,N);

// Show simulation of forward rates
/*************************************************************************/

printf("\n~~~~~~~~~~~~~~~~~~~");
printf("\nOnly show simulation of forward rates? (Y/N):");
scanf("%s",&yn);

if (strcmp(yn,"Y") == 0 || strcmp(yn,"y") == 0 )
{
    int fm;
    printf("\n~~~~~~~~~~~~~~~~~~~");
    printf("\nWhich forward rate simulation:");
    printf("\n1. Simulate forward Libor rate under the forward measure");
    printf("\n2. Simulate forward Swap rate under the forward Swap measure");
    printf("\n3. Simulate forward Libor rate under the forward Swap measure");
    printf("\n4. Simulate forward Swap rate under the forward Libor measure");
    printf("\nEnter: ");
    scanf("%d",&fm);

    switch (fm)
    {
        case 1 :
            cout << endl << "SimForwardLiborRate_ForwardMeasure"<< endl;
            for (i=0; i< sim; i++)
                mm.SimForwardLiborRate_ForwardMeasure(F,L,R,V,T,seed,N,M,term, spot,  Z, i);
            break;
```

```
        case 2:
            cout << endl << "SimForwardSwapRate_ForwardSwapMeasure"<< endl;
            for (i=0; i< sim; i++)
                mm.SimForwardSwapRate_ForwardSwapMeasure (F,L,R,V,T,seed,N,M,term, spot,  Z, i);
            break;

        case 3:
            cout << endl << "SimForwardLiborRate_ForwardSwapMeasure"<< endl;
            for (i=0; i< sim; i++)
                mm.SimForwardLiborRate_ForwardSwapMeasure (F,L,R,V,T,seed,N,M,term, spot,  Z, i);
            break;

        case 4:

            cout << endl << "SimForwardSwapRate_ForwardLiborMeasure"<< endl;
            for (i=0; i< sim; i++)
                mm.SimForwardSwapRate_ForwardLiborMeasure (F,L,R,V,T,seed,N,M,term, spot,  Z, i);
            break;

        default:
            cout << endl << "Wrong choice"<< endl;

    }

    for (i=0; i<=M; i++)
    {
        for (int j=0; j<=N; j++)
            printf("\n[%d][%d]Libor = %.5f",i,j,F[i][j]);
        printf("\n");
    }

}

else
{
    // Pricing of instruments
    /*************************************************************************/
    switch (instrument)
    {
        // cap
        case 1:
            switch(model)
            {
                // LFM
                case 1 :
                    switch (measure)
                    {
                        // forward Libor measure
                        case 1 :
                            {
                                cout << endl << "Libor Market Model simulation of a Cap under the forward Libor measure"<<
                                endl;
                                price = LFM.PriceCapMonteCarlo(aCap,D, L, F, C, R, V, T ,seed, sim, M,N,term, P, K,
                                spot,"FLM", Z, price,std, stdErr);
```

```
                }
                break;
            // forward swap measure
            case 2 :
                {
                    cout << endl << "Libor Market Model simulation of a Cap under the forward Swap measure"<<
                    endl;
                    price = LFM.PriceCapMonteCarlo(aCap,D, L, F, C, R, V, T ,seed, sim, M,N,term, P, K, spot,"FSM",
                    Z, price,std, stdErr);
                }
                break;

            default:
            cout << endl << "Wrong choice"<< endl;

        }
        break;
    // LSM
    case 2 :
        switch (measure)
        {
            // forward Libor measure
            case 1 :
                {
                    cout << endl << "Swap Market Model simulation of a Cap under the forward Libor measure"<<
                    endl;
                    price = LSM.PriceCapMonteCarlo(aCap,D, L, F, C, R, V, T ,seed, sim, M,N,term, P, K,
                    spot,"FLM", Z, price, std, stdErr);
                }
                break;
            // forward swap measure
            case 2 :
                {
                    cout << endl << "Swap Market Model simulation of a Cap under the forward Swap measure"<<
                    endl;
                    price = LSM.PriceCapMonteCarlo(aCap,D, L, F, C, R, V, T ,seed, sim, M,N,term, P, K, spot,"FSM",
                    Z, price, std, stdErr);
                }
                break;

            default:
            cout << endl << "Wrong choice"<< endl;

        }
        break;

    default:
    cout << endl << "Wrong choice"<< endl;

}
break;

// swaption
case 2:
```

```
switch(model)
{
    // LFM
    case 1 :
        switch (measure)
        {
            // forward Libor measre
            case 1 :
                {
                    cout << endl << "Libor Market Model simulation of a Swaption under the forward Libor
                    measure"<< endl;
                    price = LFM.PriceSwaptionMonteCarlo(aSwaption, F, A, R, V, T ,L, seed, sim, M,N,term, P, K,
                    riskFree, spot,"FLM",Z, price,std,stdErr);
                }
                break;
            // forward swap measure
            case 2 :
                {
                    cout << endl << "Libor Market Model simulation of a Swaption under the forward Swap
                    measure"<< endl;
                    price = LFM.PriceSwaptionMonteCarlo(aSwaption, F, A, R, V, T ,L, seed, sim, M,N,term, P, K,
                    riskFree, spot,"FSM",Z, price,std,stdErr);
                }
                break;

            default:
            cout << endl << "Wrong choice"<< endl;
        }
        break;
    // LSM
    case 2 :
        switch (measure)
        {
            // forward Libor measre
            case 1 :
                {
                    cout << endl << "Swap Market Model simulation of a Swaption under the forward Libor
                    measure"<< endl;
                    price = LSM.PriceSwaptionMonteCarlo(aSwaption, F, A, R, V, T ,L, seed, sim, M,N,term, P, K,
                    riskFree, spot,"FLM",Z, price,std,stdErr);
                }
                break;
            // forward swap measure
            case 2 :
                {
                    cout << endl << "Swap Market Model simulation of a Swaption under the forward Swap
                    measure"<< endl;
                    price = LSM.PriceSwaptionMonteCarlo(aSwaption, F, A, R, V, T ,L, seed, sim, M,N,term, P, K,
                    riskFree, spot,"FSM",Z, price,std,stdErr);
                }
                break;

            default:
            cout << endl << "Wrong choice"<< endl;
```

```
                    }
                    break;

                default:
                cout << endl << "Wrong choice"<< endl;
            }

            break;
        default:
            cout << endl << "Wrong choice"<< endl;

        break;
    }

    // Show the price
    /*************************************************************************/
    // cap
    if (instrument == 1)
    {
        printf("\n");
        for (j=0; j<= M; j++)
            printf("\n[%d] Caplet Price = %.5f",j,C[j]);
        printf("\n");
        printf("\nCap Price = %.5f", price);
        printf("\nstd = %.5f", std);
        printf("\nstdErr = %.5f", stdErr);
        printf("\n");
    }
    // swaption
    else if (instrument == 2)
    {
        for (i=0; i<= M; i++)
            printf("\n[%d] Swap Payoff = %.5f",i,A[i]);
        printf("\n");
        printf("\nprice = %.5f", price);
        printf("\nstd = %.5f", std);
        printf("\nstdErr = %.5f", stdErr);
        printf("\n");

    }
  }
}
catch (...)
{
    printf("\nError in MarketModelApp");
}

printf("\n\n");

return 0;
}
```

E.13 C++ data extraction from the XML files

```
FileData::FileData()
{

    // Initialize XML Parser environment
    try
      {
        XMLPlatformUtils::Initialize();

        pParser = new XercesDOMParser();

        pParser->setValidationScheme(XercesDOMParser::Val_Auto);
        pParser->setDoSchema(false);
        pParser->setDoNamespaces(false);
        pParser->setValidationSchemaFullChecking(false);
        pParser->setCreateEntityReferenceNodes(false);

      }
      catch (const XMLException& e)
      {
            char tempString[255];
            sprintf(tempString, "\nError while initializing XML (DOMParser) system!: %s", (char*)e.getMessage());
            printf(tempString);
      }
    catch(...)
    {
        printf("\nError while constructing FileData");
    }

}

FileData::~FileData()
{
    try
    {
        // Delete the parser
        delete pParser;

        // Call the termination method
        XMLPlatformUtils::Terminate();
    }
    catch (const XMLException& e)
    {
            char tempString[255];
                    sprintf(tempString, "\nError while destroying XML (DOMParser) system: %s", (char*)e.getMessage());
            printf(tempString);
    }
    catch(...)
    {
```

```
        printf("\nError while destroying FileData");
    }

}


void FileData::ReadFileDataFromFile(string xmlFilePathName)
{
    FILE* xmlFile = NULL;

    char buf[4096];

    xmlString = "";

    // Open XML file
    xmlFile = fopen((const char*)xmlFilePathName.c_str(), "r");

    if (xmlFile == NULL)
    {
        printf("File %s not found.\n", (const char*)xmlFilePathName.c_str());
        exit(-1);
    }

    // Read from xmlFile
    while ( !feof(xmlFile) )
    {
        fgets(buf, 4096, xmlFile);
        xmlString.append(buf);
    }

    fclose(xmlFile);

}

void FileData::PrintFileData() const
{

}

void FileData::ParseXml(string xmlFilePathName)
{

    DOMDocument* pDoc = NULL;
    DOMNode* pRootNode = NULL;

    try
    {
        pParser->parse((char*)xmlFilePathName.c_str());
    }
    catch (const XMLException& e)
    {
        printf("\n");
        printf((char*)e.getMessage());
    }
```

```
    catch (const DOMException& e)
    {
        printf("\n");
        printf((char*)XMLString::transcode(e.msg));
    }
    catch (...)
    {
        printf("\n");
        printf("Unknown XML Parsing error!");
    }

    try
    {
        pDoc = pParser->getDocument();

        if (!pDoc)
        {
            printf("\n");
            printf("Unable to get DOMDocument");
        }

        pRootNode = pDoc->getDocumentElement();

        UnmarshallFileData(pRootNode);

    }
    catch(...)
    {
        printf("\n");
        printf("Error in DOMDocument");
    }
}

// Unmarshall the TEXT NODE
string FileData::UnmarshallText( DOMNode* pRootNode)
{
    char tempString[255];
    string value;
    char* rawValue;

    DOMNode* pNode;
    DOMNodeList* pNodeList = pRootNode->getChildNodes();

    for( int i=0; i<pNodeList->getLength(); i++ )
    {
        pNode = pNodeList->item( i );

        if( pNode->getNodeType() == DOMNode::TEXT_NODE )
        {
            rawValue = XMLString::transcode( pNode->getNodeValue());

            value.assign(rawValue);

            XMLString::release(&rawValue);
```

```
        }

        //printf("\n");
        //sprintf(tempString, "value = : %s", (char*)value.c_str());
        //printf(tempString);
    }

    return value;
}

bool FileData::FillData(const int seq, InterestRateOption* pIRO)
{
        return true;
}

////////////////////////////TradeData//////////////////////////////////////////////////////
////////////////////////////TradeData//////////////////////////////////////////////////////
////////////////////////////TradeData//////////////////////////////////////////////////////

TradeData::TradeData()
{
    pFileTradeList = new FileTradeList();
    pFileTrade = new FileTrade();
}

TradeData::~TradeData()
{
    try
    {
        //Delete the data set list
        pFileTradeList->clear();

        delete pFileTradeList;

        //Delete the data set
        delete pFileTrade;
    }
    catch (const XMLException& e)
    {
        char tempString[255];
        sprintf(tempString, "\nError while destroying XML (DOMParser) system: %s", (char*)e.getMessage());
        printf(tempString);
    }
    catch(...)
    {
        printf("\nError while destroying FileData");
    }

}
void TradeData::SetFileTrade(FileTrade* pFileTrade)
{
    TradeData::pFileTrade = pFileTrade;
}
```

```
FileTrade* TradeData::GetFileDataList() const
{
    return pFileTrade;
}
void TradeData::AddFileTrade(FileTrade* tradeSet)
{
    pFileTradeList->push_back(tradeSet);
}

void TradeData::PrintFileData() const
{
    if (pFileTradeList == NULL)
    {
        printf("\nFileData SetList = NULL");
        return;
    }

    printf("\nPrintFileData::pFileTradeList size %d", pFileTradeList->size());

    printf("\n");
    FileTradeList::iterator dataIterator;
    dataIterator = pFileTradeList->begin();
    while (dataIterator != pFileTradeList->end())
    {
        FileTrade* pFileTrade = (FileTrade*)*dataIterator;

        int tradeSetSeq = pFileTrade->GetSeq();
                printf("[%d] GetInstrument : %s \n", tradeSetSeq, (const char*)pFileTrade->GetInstrument().c_str());
        printf("[%d] GetDeal : %s \n", tradeSetSeq, (const char*)pFileTrade->GetDeal().c_str());
        printf("[%d] GetPrincipal : %.4f \n", tradeSetSeq, pFileTrade->GetPrincipal());
        printf("[%d] GetMaturity : %.4f \n", tradeSetSeq, pFileTrade->GetMaturity());
        printf("[%d] GetTenor : %.4f \n", tradeSetSeq, pFileTrade->GetTenor());
        printf("[%d] GetRiskFreeRate : %.4f \n", tradeSetSeq, pFileTrade->GetRiskFreeRate());
        printf("[%d] GetStrikeRate : %.4f \n", tradeSetSeq, pFileTrade->GetStrikeRate());
        printf("[%d] GetSpotFwdRate : %.4f \n", tradeSetSeq, pFileTrade->GetSpotFwdRate());
        printf("[%d] GetMarketPrice : %.4f \n", tradeSetSeq, pFileTrade->GetMarketPrice());

        dataIterator++;
    }
}

FileTrade* TradeData::FindFileTrade(int dateSetSeq)
{
    if (pFileTradeList == NULL)
    {
        printf("\nFileData SetList = NULL");
        return NULL;
    }

    FileTradeList::iterator dataIterator;
    dataIterator = pFileTradeList->begin();
    while (dataIterator != pFileTradeList->end())
    {
```

```
        FileTrade* pFileTrade = (FileTrade*)*dataIterator;

        int seq = pFileTrade->GetSeq();

        if (seq == dateSetSeq )
        {
            return  pFileTrade;
        }

        dataIterator++;
    }

    return NULL;
}

bool TradeData::FillData(const int seq, InterestRateOption* pIRO)
{
        if (pFileTradeList == NULL)
        {
            printf("\nFileData SetList = NULL");
            return false;
        }

        FileTrade* pFileTrade = NULL;
        FileTradeList::iterator dataIterator;
        dataIterator = pFileTradeList->begin();
        while (dataIterator != pFileTradeList->end())
        {
            pFileTrade = (FileTrade*)*dataIterator;

            int dateSetSeq = pFileTrade->GetSeq();

            if (seq == dateSetSeq )
            {
                break;
            }

            dataIterator++;
        }

        if (pFileTrade != NULL)
        {
            pIRO->SetSeq(pFileTrade->GetSeq());
            pIRO->SetPrincipal(pFileTrade->GetPrincipal());
            pIRO->SetMaturity(pFileTrade->GetMaturity());
            pIRO->SetTenor(pFileTrade->GetTenor());
            pIRO->SetRiskFreeRate(pFileTrade->GetRiskFreeRate()/100);
            pIRO->SetStrikeRate(pFileTrade->GetStrikeRate()/100);
            pIRO->SetSpotForwardRate(pFileTrade->GetSpotFwdRate()/100);
            pIRO->SetMarketPrice(pFileTrade->GetMarketPrice()/100);

            return true;
        }
        else
```

```
        {
            return false;
        }
}

void TradeData::UnmarshallFileData( DOMNode* pRootNode )
{
    char tempString[1026];
    char* rawName;

    DOMNode* pNode;
    DOMNodeList* pNodeList = pRootNode->getChildNodes();

    try
    {
        for( int i=0 ; i< pNodeList->getLength(); i++ )
        {
            pNode = pNodeList->item(i);

            short rawType= pNode->getNodeType();

            if( rawType == DOMNode::ELEMENT_NODE )
            {
                rawName = XMLString::transcode( pNode->getNodeName());
                _strupr(rawName);

                if ( strcmp(rawName, "TRADE") == 0 )
                {
                    AddFileTrade(UnmarshallFileTrade(pNode));
                }
                else
                {
                    sprintf(tempString, "Unexpected element in FileData : %s", rawName);
                    printf("\n");
                    printf(tempString);
                }

                XMLString::release(&rawName);
            }
        }
    }
    catch(...)
    {
        sprintf(tempString, "Error in UnmarshallFileData");
        printf("\n");
        printf(tempString);
    }
}

// Unmarshall FileTrade
FileTrade* TradeData::UnmarshallFileTrade( DOMNode* pRootNode)
{
    char tempString[1026];
    char* rawName;
```

```
try
{
    pFileTrade = new FileTrade();
}
catch(...)
{
    printf("\n");
    sprintf(tempString, "Memory allocation error (FileTrade)");
    printf(tempString);
}

if (pFileTrade == NULL)
{
    printf("\n");
    sprintf(tempString, "Memory allocation error (FileTrade): NULL pointers");
    printf(tempString);
}

DOMNode* pNode;
DOMNodeList* pNodeList = pRootNode->getChildNodes();

try
{
    for( int i=0 ; i< pNodeList->getLength(); i++ )
    {
        pNode = pNodeList->item(i);

        if( pNode->getNodeType() == DOMNode::ELEMENT_NODE )
        {
            rawName = XMLString::transcode( pNode->getNodeName());
            _strupr(rawName);

            if ( strcmp(rawName, "SEQ") == 0 )
            {
                string str = UnmarshallText( pNode );
                pFileTrade->SetSeq( atoi((char*)str.c_str()) );
            }
            else if ( strcmp(rawName, "INSTRUMENT") == 0 )
            {
                pFileTrade->SetInstrument( UnmarshallText( pNode ) );
            }
            else if ( strcmp(rawName, "DEAL") == 0 )
            {
                pFileTrade->SetDeal( UnmarshallText( pNode ) );
            }
            else if ( strcmp(rawName, "PRINCIPAL") == 0 )
            {
                string str = UnmarshallText( pNode );
                pFileTrade->SetPrincipal( atof((char*)str.c_str()) );
            }
            else if ( strcmp(rawName, "MATURITY") == 0 )
            {
                string str = UnmarshallText( pNode );
```

```
                pFileTrade->SetMaturity( atof((char*)str.c_str()) );
            }
            else if ( strcmp(rawName, "TENOR") == 0 )
            {
                string str = UnmarshallText( pNode );
                pFileTrade->SetTenor( atof((char*)str.c_str()) );
            }
            else if ( strcmp(rawName, "RISK_FREE_RATE") == 0 )
            {
                string str = UnmarshallText( pNode );
                pFileTrade->SetRiskFreeRate( atof((char*)str.c_str()) );
            }
            else if ( strcmp(rawName, "STRIKE_RATE") == 0 )
            {
                string str = UnmarshallText( pNode );
                pFileTrade->SetStrikeRate( atof((char*)str.c_str()) );
            }
            else if ( strcmp(rawName, "SPOT_FWD_RATE") == 0 )
            {
                string str = UnmarshallText( pNode );
                pFileTrade->SetSpotFwdRate( atof((char*)str.c_str()) );
            }
            else if ( strcmp(rawName, "MARKET_PRICE") == 0 )
            {
                string str = UnmarshallText( pNode );
                pFileTrade->SetMarketPrice( atof((char*)str.c_str()) );
            }
            else
            {
                printf("\n");
                sprintf(tempString, "Unexpected element in FileTrade : %s", rawName);
                printf(tempString);
            }

            XMLString::release(&rawName);
        }
    }
}
catch(...)
{
        printf("\n");
        sprintf(tempString, "Error in UnmarshallFileTrade");
        printf(tempString);
}

return pFileTrade;
}

////////////////////////////TermStructureData//////////////////////////////////////////////////
////////////////////////////TermStructureData//////////////////////////////////////////////////
////////////////////////////TermStructureData//////////////////////////////////////////////////


TermStructureData::TermStructureData()
```

```
{
    pFileTermStructureList = new FileTermStructureList();
    pFileTermStructure = new FileTermStructure();

}
TermStructureData::~TermStructureData()
{
    try
  {
        //Delete the data set list
        pFileTermStructureList->clear();

        delete pFileTermStructureList;

        //Delete the data set
        delete pFileTermStructure;
    }
  catch (const XMLException& e)
  {
        char tempString[255];
        sprintf(tempString, "\nError while destroying XML (DOMParser) system: %s", (char*)e.getMessage());
        printf(tempString);
  }
    catch(...)
    {
        printf("\nError while destroying FileData");
    }
}


void TermStructureData::AddFileTermStructure(FileTermStructure* pPoint)
{
    pFileTermStructureList->push_back(pPoint);
}

void TermStructureData::UnmarshallFileData( DOMNode* pRootNode )
{
    char tempString[1026];
    char* rawName;

    string type;
    string name;
    double term;
    double maturity;

    DOMNode* pNode;
    DOMNodeList* pNodeList = pRootNode->getChildNodes();

    try
    {
        for( int i=0 ; i< pNodeList->getLength(); i++ )
        {
            pNode = pNodeList->item(i);
```

```
            short rawType= pNode->getNodeType();

            if( rawType == DOMNode::ELEMENT_NODE )
            {
                rawName = XMLString::transcode( pNode->getNodeName());
                _strupr(rawName);

                if ( strcmp(rawName, "TYPE") == 0 )
                {
                    type = UnmarshallText( pNode );
                }
                else if ( strcmp(rawName, "NAME") == 0 )
                {
                    name = UnmarshallText( pNode );
                }
                else if ( strcmp(rawName, "TERM") == 0 )
                {
                    string strTerm = UnmarshallText( pNode );
                    term = atof((char*)strTerm.c_str());
                }
                else if ( strcmp(rawName, "MATURITY") == 0 )
                {
                    string strMat = UnmarshallText( pNode );
                    maturity = atof((char*)strMat.c_str());
                }
                else if ( strcmp(rawName, "POINT") == 0 )
                {
                    AddFileTermStructure(UnmarshallFileTermStructure(type, name, term, maturity, pNode));
                }
                else
                {
                    sprintf(tempString, "Unexpected element in FileData : %s", rawName);
                    printf("\n");
                    printf(tempString);
                }

                XMLString::release(&rawName);
            }
        }
    }
    catch(...)
    {
        sprintf(tempString, "Error in UnmarshallFileData");
        printf("\n");
        printf(tempString);
    }
}

// Unmarshall File Term Structure
FileTermStructure* TermStructureData::UnmarshallFileTermStructure(const string type, const string name,const double term, const double
maturity, DOMNode* pRootNode)
{
    char tempString[1026];
    char* rawName;
```

```
try
{
    pFileTermStructure = new FileTermStructure();
}
catch(...)
{
    printf("\n");
    sprintf(tempString, "Memory allocation error (FileTermStructure)");
    printf(tempString);
}

if (pFileTermStructure == NULL)
{
    printf("\n");
    sprintf(tempString, "Memory allocation error (FileTermStructure): NULL pointers");
    printf(tempString);
}

pFileTermStructure->SetType(type);
pFileTermStructure->SetName(name);
pFileTermStructure->SetTerm(term);
pFileTermStructure->SetMaturity(maturity);

DOMNode* pNode;
DOMNodeList* pNodeList = pRootNode->getChildNodes();

try
{
    for( int i=0 ; i< pNodeList->getLength(); i++ )
    {
        pNode = pNodeList->item(i);

        if( pNode->getNodeType() == DOMNode::ELEMENT_NODE )
        {

            rawName = XMLString::transcode( pNode->getNodeName());
            _strupr(rawName);

            if ( strcmp(rawName, "SEQ")  == 0 )
            {
                string str = UnmarshallText( pNode );
                pFileTermStructure->SetSeq( atoi((char*)str.c_str()) );
            }
            else if ( strcmp(rawName, "DATE")  == 0 )
            {
                pFileTermStructure->SetDate( UnmarshallText( pNode ) );
            }
            else if ( strcmp(rawName, "DAYS")  == 0 )
            {
                string str = UnmarshallText( pNode );
                pFileTermStructure->SetDays( atoi((char*)str.c_str()) );
            }
            else if ( strcmp(rawName, "YEARS")  == 0 )
```

```
                {
                    string str = UnmarshallText( pNode );
                    pFileTermStructure->SetYears( atof((char*)str.c_str()) );
                }
                else if ( strcmp(rawName, "RATE")  == 0 )
                {
                    string str = UnmarshallText( pNode );
                    pFileTermStructure->SetRate( atof((char*)str.c_str()) );
                }
                else
                {
                    printf("\n");
                    sprintf(tempString, "Unexpected element in FileTrade : %s", rawName);
                    printf(tempString);
                }

                XMLString::release(&rawName);
            }
        }
    }
    catch(...)
    {
            printf("\n");
            sprintf(tempString, "Error in UnmarshallFileTrade");
            printf(tempString);
    }

    return pFileTermStructure;
}


void TermStructureData::PrintFileData() const
{

    if (pFileTermStructureList == NULL)
    {
        printf("\nFileTermStructure SetList = NULL");
        return;
    }

    printf("\nPrintFileDTermStructure::pFileTermStructureList size %d", pFileTermStructureList->size());

    printf("\n");
    FileTermStructureList::iterator dataIterator;
    dataIterator = pFileTermStructureList->begin();
    while (dataIterator != pFileTermStructureList->end())
    {
        FileTermStructure* pFileTermStructure = (FileTermStructure*)*dataIterator;

        int seq = pFileTermStructure->GetSeq();
        printf("[%d] GetType: %s \n", seq, (const char*)pFileTermStructure->GetType().c_str());
        printf("[%d] GetName: %s \n", seq, (const char*)pFileTermStructure->GetName().c_str());
        printf("[%d] GetTerm : %.4f \n", seq, pFileTermStructure->GetTerm());
        printf("[%d] GetMaturity : %.4f \n", seq, pFileTermStructure->GetMaturity());
```

```
        printf("[%d] GetDate : %s \n", seq, (const char*)pFileTermStructure->GetDate().c_str());
        printf("[%d] GetDays : %d \n", seq, pFileTermStructure->GetDays());
        printf("[%d] GetYears : %.4f \n", seq, pFileTermStructure->GetYears());
        printf("[%d] GetRate : %.4f \n", seq, pFileTermStructure->GetRate());

        dataIterator++;
    }
}

bool TermStructureData::FillTSData(TermStructure* pTS)
{
        if (pFileTermStructureList == NULL)
        {
            printf("\nFileDTermStructure SetList = NULL");
            return false;
        }

        TermStructureRow tsr;
        pTS->ClearTermStruct();

        FileTermStructure* pFileTermStructure = NULL;
        FileTermStructureList::iterator dataIterator;
        dataIterator = pFileTermStructureList->begin();
        while (dataIterator != pFileTermStructureList->end())
        {
            pFileTermStructure = (FileTermStructure*)*dataIterator;

            tsr.SetSeq(pFileTermStructure->GetSeq());
            tsr.SetType(pFileTermStructure->GetType());
            tsr.SetName(pFileTermStructure->GetName());
            tsr.SetTerm(pFileTermStructure->GetTerm());
            tsr.SetMaturity(pFileTermStructure->GetMaturity());
            tsr.SetDate(pFileTermStructure->GetDate());
            tsr.SetDays(pFileTermStructure->GetDays());
            tsr.SetYears(pFileTermStructure->GetYears());
            tsr.SetRate(pFileTermStructure->GetRate()/100);

            pTS->AddTermStruct(tsr);

            dataIterator++;
        }

        return true;
}
```

E.14 The cap trades XML data file

```
<?xml version="1.0" ?>
<DATA>
    <TRADE>
        <SEQ>0</SEQ>
        <INSTRUMENT>Cap</INSTRUMENT>
        <DEAL>9m fwd 6.833 zar 1226</DEAL>
        <PRINCIPAL>1000000</PRINCIPAL>
        <MATURITY>0.75</MATURITY>
        <TENOR>0.25</TENOR>
        <RISK_FREE_RATE>6.954</RISK_FREE_RATE>
        <STRIKE_RATE>6.833</STRIKE_RATE>
        <SPOT_FWD_RATE>6.833</SPOT_FWD_RATE>
        <MARKET_PRICE>1226</MARKET_PRICE>
    </TRADE>
    <TRADE>
        <SEQ>1</SEQ>
        <INSTRUMENT>Cap</INSTRUMENT>
        <DEAL>12m fwd 6.86 zar 2116</DEAL>
        <PRINCIPAL>1000000</PRINCIPAL>
        <MATURITY>1</MATURITY>
        <TENOR>0.25</TENOR>
        <RISK_FREE_RATE>6.954</RISK_FREE_RATE>
        <STRIKE_RATE>6.890</STRIKE_RATE>
        <SPOT_FWD_RATE>6.890</SPOT_FWD_RATE>
        <MARKET_PRICE>2116</MARKET_PRICE>
    </TRADE>
    <TRADE>
        <SEQ>2</SEQ>
        <INSTRUMENT>Cap</INSTRUMENT>
        <DEAL>15m fwd 6.935 zar 3460</DEAL>
        <PRINCIPAL>1000000</PRINCIPAL>
        <MATURITY>1.25</MATURITY>
        <TENOR>0.25</TENOR>
        <RISK_FREE_RATE>6.954</RISK_FREE_RATE>
        <STRIKE_RATE>6.935</STRIKE_RATE>
        <SPOT_FWD_RATE>6.935</SPOT_FWD_RATE>
        <MARKET_PRICE>2116</MARKET_PRICE>
    </TRADE>
</DATA>
```

E.15 The swaption trades XML data file

```
<?xml version="1.0" ?>
<DATA>
    <TRADE>
        <SEQ>0</SEQ>
        <INSTRUMENT>Swaption</INSTRUMENT>
        <DEAL>3m 2y fwd 7.308  ZAR 4526 ATM</DEAL>
        <PRINCIPAL>1000000</PRINCIPAL>
        <MATURITY>2</MATURITY>
        <TENOR>0.25</TENOR>
        <RISK_FREE_RATE>6.954</RISK_FREE_RATE>
        <STRIKE_RATE>7.308</STRIKE_RATE>
        <SPOT_FWD_RATE>7.308</SPOT_FWD_RATE>
        <MARKET_PRICE>4526</MARKET_PRICE>
    </TRADE>
    <TRADE>
        <SEQ>1</SEQ>
        <INSTRUMENT>Swaption</INSTRUMENT>
        <DEAL>6m 2y fwd 7.469  ZAR 6784 ATM</DEAL>
        <PRINCIPAL>1000000</PRINCIPAL>
        <MATURITY>2</MATURITY>
        <TENOR>0.5</TENOR>
        <RISK_FREE_RATE>6.954</RISK_FREE_RATE>
        <STRIKE_RATE>7.469</STRIKE_RATE>
        <SPOT_FWD_RATE>7.469</SPOT_FWD_RATE>
        <MARKET_PRICE>6784</MARKET_PRICE>
    </TRADE>
    <TRADE>
        <SEQ>2</SEQ>
        <INSTRUMENT>Swaption</INSTRUMENT>
        <DEAL>3m 5y fwd 7.868  ZAR 11504 ATM</DEAL>
        <PRINCIPAL>1000000</PRINCIPAL>
        <MATURITY>5</MATURITY>
        <TENOR>0.25</TENOR>
        <RISK_FREE_RATE>6.954</RISK_FREE_RATE>
        <STRIKE_RATE>7.868</STRIKE_RATE>
        <SPOT_FWD_RATE>7.868</SPOT_FWD_RATE>
        <MARKET_PRICE>11504</MARKET_PRICE>
    </TRADE>
    <TRADE>
        <SEQ>3</SEQ>
        <INSTRUMENT>Swaption</INSTRUMENT>
        <DEAL>6m 5y fwd 7.956  ZAR 16209 ATM</DEAL>
        <PRINCIPAL>1000000</PRINCIPAL>
        <MATURITY>5</MATURITY>
        <TENOR>0.5</TENOR>
        <RISK_FREE_RATE>6.954</RISK_FREE_RATE>
        <STRIKE_RATE>7.956</STRIKE_RATE>
        <SPOT_FWD_RATE>7.956</SPOT_FWD_RATE>
        <MARKET_PRICE>16209</MARKET_PRICE>
```

```
    </TRADE>
</DATA>
```

E.16 The term structure XML data file

```
<?xml version="1.0" ?>
<DATA>
    <TYPE>TERM STRUCTURE</TYPE>
    <NAME>ZERO COUPON</NAME>
    <TERM>0.25</TERM>
    <MATURITY>5</MATURITY>
    <POINT> <SEQ>0</SEQ> <DATE>2005/07/08</DATE> <DAYS>0</DAYS> <YEARS>0.00</YEARS> <RATE>6.7000</RATE>
</POINT>
    <POINT> <SEQ>1</SEQ> <DATE>2005/10/07</DATE> <DAYS>91</DAYS> <YEARS>0.25</YEARS> <RATE>6.9540</RATE>
</POINT>
    <POINT> <SEQ>2</SEQ> <DATE>2006/01/09</DATE> <DAYS>185</DAYS> <YEARS>0.50</YEARS> <RATE>6.9006</RATE>
</POINT>
    <POINT> <SEQ>3</SEQ> <DATE>2006/04/07</DATE> <DAYS>273</DAYS> <YEARS>0.75</YEARS> <RATE>6.8745</RATE>
</POINT>
    <POINT> <SEQ>4</SEQ> <DATE>2006/07/07</DATE> <DAYS>364</DAYS> <YEARS>1.00</YEARS> <RATE>6.8904</RATE>
</POINT>
    <POINT> <SEQ>5</SEQ> <DATE>2006/10/09</DATE> <DAYS>458</DAYS> <YEARS>1.25</YEARS> <RATE>6.9392</RATE>
</POINT>
    <POINT> <SEQ>6</SEQ> <DATE>2007/01/08</DATE> <DAYS>549</DAYS> <YEARS>1.50</YEARS> <RATE>7.0073</RATE>
</POINT>
    <POINT> <SEQ>7</SEQ> <DATE>2007/04/10</DATE> <DAYS>641</DAYS> <YEARS>1.75</YEARS> <RATE>7.0900</RATE>
</POINT>
    <POINT> <SEQ>8</SEQ> <DATE>2007/07/09</DATE> <DAYS>731</DAYS> <YEARS>2.00</YEARS> <RATE>7.1798</RATE>
</POINT>
    <POINT> <SEQ>9</SEQ> <DATE>2007/10/07</DATE> <DAYS>821</DAYS> <YEARS>2.25</YEARS> <RATE>7.2520</RATE>
</POINT>
    <POINT> <SEQ>10</SEQ> <DATE>2008/01/06</DATE> <DAYS>913</DAYS> <YEARS>2.50</YEARS> <RATE>7.3310</RATE>
</POINT>
    <POINT> <SEQ>11</SEQ> <DATE>2008/04/06</DATE> <DAYS>1004</DAYS> <YEARS>2.75</YEARS> <RATE>7.4100</RATE>
</POINT>
    <POINT> <SEQ>12</SEQ> <DATE>2008/07/07</DATE> <DAYS>1095</DAYS> <YEARS>3.00</YEARS> <RATE>7.4790</RATE>
</POINT>
    <POINT> <SEQ>13</SEQ> <DATE>2008/10/06</DATE> <DAYS>1186</DAYS> <YEARS>3.25</YEARS> <RATE>7.5370</RATE>
</POINT>
    <POINT> <SEQ>14</SEQ> <DATE>2009/01/05</DATE> <DAYS>1278</DAYS> <YEARS>3.50</YEARS> <RATE>7.6000</RATE>
</POINT>
    <POINT> <SEQ>15</SEQ> <DATE>2009/04/06</DATE> <DAYS>1369</DAYS> <YEARS>3.75</YEARS> <RATE>7.6360</RATE>
</POINT>
    <POINT> <SEQ>16</SEQ> <DATE>2009/07/07</DATE> <DAYS>1460</DAYS> <YEARS>4.00</YEARS> <RATE>7.6593</RATE>
</POINT>
    <POINT> <SEQ>17</SEQ> <DATE>2009/10/06</DATE> <DAYS>1551</DAYS> <YEARS>4.25</YEARS> <RATE>7.6840</RATE>
</POINT>
    <POINT> <SEQ>18</SEQ> <DATE>2010/01/05</DATE> <DAYS>1643</DAYS> <YEARS>4.50</YEARS> <RATE>7.7250</RATE>
</POINT>
    <POINT> <SEQ>19</SEQ> <DATE>2010/04/06</DATE> <DAYS>1734</DAYS> <YEARS>4.75</YEARS> <RATE>7.7670</RATE>
</POINT>
    <POINT> <SEQ>20</SEQ> <DATE>2010/07/07</DATE> <DAYS>1825</DAYS> <YEARS>5.00</YEARS> <RATE>7.7894</RATE>
</POINT>
</DATA>
```

Bibliography

[AA00] L. Andersen and J. Andreasen. Volatility Skews and Extensions of the Libor Market Model. Applied Mathematical Finance, 7: 1-32, 2000.

[AL03] C. Alexander and D. Lvov. Statistical Properties of Forward Libor Rates. ISMA Discussion Papers in Finance 2003-03, January 2003. The Business School for Financial Markets, University of Reading, March 2003. Working paper, available at http://www.ismacentre.rdg.ac.uk/pdf/discussion/DP2003-03.pdf.

[Bla76] F. Black. The Pricing of Commodity Contracts. Journal of Financial Economics, 3: 167-179, 1976.

[BK91] F. Black and P. Karasinski, P. Bond and Option Pricing when Short Rates are Lognormal. Financial Analyst Journal, 47(4): 52-59, July-August 1991.

[BDT90] F. Black, E. Derman and W. Toy. A One-Factor Model of Interest Rates and Its Application to Treasury Bond Options. Financial Analyst Journal, 46(1): 33-39, January-February 1990.

[BS73] F. Black and M. Scholes. The Pricing of Options and Corporate Liabilities. Journal of Political Economy, 81: 637-659, May-June1973.

[BGM97] A. Brace, D. Gatarek and M. Musiela. The Market Model of Interest Rate Dynamics. Mathematical Finance, 7(2): 127-147, April 1997.

[BL03] D. Brigo and J. Liinev. On the distributional distance between the Libor and Swap market models. Working Paper, available at http://ww.damianobrigo.it, 5 June 2003.

[BM01] D. Brigo and F. Mercurio. Interest Rate Models: Theory and Practice. Springer Verlag Berlin Heidelberg, 2001.

[BCM01] D. Brigo, C. Capitani and F. Mercurio. On the joint calibration of the Libor market models to caps and swaptions market volatilities. Working Paper, available at http://ww.damianobrigo.it, July 17 2001.

[BMM02] D. Brigo, F. Mercurio and M. Morini. Different Covariance Parameterizations of the Libor Market Model and Joint Caps/Swaptions Calibrations. Working Paper available at http://ww.damianobrigo.it, June 10 2002.

[CIR85] J. C. Cox, J. E. Ingersoll Jr. and S. A. Ross. A Theory of the Term Structure of Interest Rates. Econometrica, 53(2): 385-408, March 1985.

[CDS03] B. Choy, T. Dun and E. Schlogl. Correlating Market Models. Working paper, available at http://ideas.repec.org/p/uts/rpaper/105.html, 2003.

[DDP01] F. De Jong, J. Driessen and A. Pelsser. Libor Market Models versus Swap Market Models for Pricing Interest Rate Derivatives: An Empirical Analysis. European Finance Review, 5: 201-237, 2001.

[GHLS04] S. Galluccio, Z. Huang, J.-M. Ly and O. Scaillet. Theory and Calibration of Swap Market Models. Research Paper No. 107, Match 2004, FAME – International Center for Financial Asset Management and Engineering, March 2004. Available at http://ideas.repec.org/p/fam/rpseri/rp107.html.

[Gla04] P. Glasserman. Monte Carlo Methods in Financial Engineering. Springer-Verlag, New York Inc., 2004.

[GZ00] P. Glasserman and X. Zhao. Arbitage-free discretization of lognormal forward Libor and swap rate models. Finance Stochastics, 4: 35-68, 2000.

[HJM92] D. Heath, R. Jarrow and A. Morton. Bond Pricing and the Term Structure of Interest Rates: A New Methodology for Contingent Claims Valuations. Econometrica, 60(1): 77-105, January 1992.

[HL86] T. S. Y. Ho and S.-B. Lee. Term structure Movements and Pricing Interest Rate Contingent Claims. Journal of Finance, 41(5): 1011-1029, December 1986.

[Hul93] J. C. Hull. Options, Futures and Other Derivative Securities. Prentice Hall International Editions, Second Edition, 1993.

[HW90] J. Hull and A. White. Pricing Interest Rate Derivative Securities. Review of Financial Studies, 3(4): 573-592, 1990.

[HW99] J. Hull and A. White. Forward rate volatilities, Swap rate volatilities, and the implementation of the Libor market model. Working paper found at www.rotman.utoronto.ca/~hull, August 1999.

[HJJ01] C. J. Hunter, P. Jäckel and M. S. Joshi. Drift Approximations in a Forward-Rate-Based LIBOR Market Model. Working paper March 2001. Published as Getting the Drift. Risk, July 2001. Available at http://www.quarchome.org/MarketModelPredictorCorrector.pdf

[Jam89] F. Jamshidian. An Exact Bond Option Formula. Journal of Finance, 44(1): 205-209, March 1989.

[Jam97] F. Jamshidian. LIBOR and Swap market models and measures. Finance and Stochastics, 1(4): 293-330, 1997.

[Lon04] J. London. Modeling Derivatives in C++. Wiley Finance, 2004.

[LSS01] F. A. Longstaff, P. Santa-Clara and E. S. Schwartz. The Relative Valuation of Caps and Swaptions: Theory and Empirical Evidence. The Journal of Finance, LVI(6):2067-2109, December 2001.

[MSS97] K. R. Miltersen, K. Sandmann and D. Sondermann. Closed form solutions for term structure derivatives with log-normal interest rates. Journal of Finance, 52(1): 409-430, March 1977.

[MR97] M. Musiela and M. Rutkowski. Continuous-time term structure models: Forward measure approach. Finance and Stochastics, 1: 261-291, 1977.

[PP05] R. Pietersz and A. A. J. Pelsser. A Comparison of Single Factor Markov-functional and Multi Factor Market Models. Journal of Economic Literature. Working paper, available at http://ideas.repec.org/p/wpa/wuwpfi/0502008.html, 6 January 2005.

[PR05] R. Pietersz and M. van Regenmortel. Generic Market Models. Working paper found at http://econwpa.wustl.edu:8089/eps/fin/papers/0502/0502009.pdf, 13 January 2005.

[PPR04] R. Pietersz, A. Pelsser and M. van Regenmortel. Fast drift approximated pricing in the BGM model. Journal of Computational Finance 8(1):93-124, 2004.

[PTVF92] W. H. Press, S. A. Teukolsky, W. T. Vetterling and B. P. Flannery. Numerical Recipes in C: The Art of Scientific Computing. Cambridge University Press, 1992.

[Reb98] R. Rebonato. Interest Rate Models, Theory and Practice. Springer Finance, 1998.

[Reb03] R. Rebonato. Term-Structure Models: a Review. QUARC – Royal Bank of Scotland. Working paper, available at http://www.quarchome.org/TSMRS.pdf, 25 February 2003.

[RJ01] R. Rebonato and M. Joshi. A Joint Empirical and Theoretical Investigation of the Modes of Deformation of Swaption Matrices: Implications for Model Choice. Working paper, available at http://www.ismacentre.rdg.ac.uk/pdf/discussion/DP2003-03.pdf, 24 July 2001.

[RB80] R. Rendleman Jr. and B. J. Bartter. The Pricing of Options on Debt Securities. Journal of Financial and Quantitative Analysis, 15(1): 11-24, March 1980.

[Rut97] M. Rutkowski. Modeling of Forward Libor and Swap Rates. Faculty of Mathematics and Information Science. Working Paper, University of New South Wales, 1977.

[Rut99] M. Rutkowski. Models of forward Libor and Swap rates. Applied Mathematical Finance, 6: 29-60, 1999.

[SC99] J. Schoenmakers and B. Coffey. LIBOR rate models, related derivatives and model calibration. Working paper available at http://www.wias-berlin.de/publications/preprints/480/wias_preprints_480.pdf, 6 April 1999.

[SC00] J. Schoenmakers and B. Coffey. Stable implied calibration of a multi-factor LIBOR model via a semi-parametric correlation structure. Presented at the Math Week 2200 Risk-conference, New York, Nov 13-17, 2000. Available at http://www.wias-berlin.de/publications/preprints/611, 2000.

[TB05] D. Taylor and G Brickhill. Mathematical Finance in South Africa. Found at http://www.cam.wits.ac.za/mfinance/papers/MathematicalFinanceSA.doc. July 2005.

[Vas77] O. Vasicek. An Equilibrium Characterization of the Term Structure. Journal of Financial Economics, 5: 177-188, 1977.

[Wes04] G. West. Interest Rate Derivatives. Published by the Financial Modelling Agency, http://www.finmod.co.za, 28 November 2004.

[Xer05] Xerces C++ Library. Found at http://xml.apache.org/xerces-c/, 1 July 2005.