UNIVERSITA' DEGLI STUDI DI PADOVA

DIPARTIMENTO DI SCIENZE ECONOMICHE ED AZIENDALI "M.FANNO"

DIPARTIMENTO DI MATEMATICA

CORSO DI LAUREA MAGISTRALE / SPECIALISTICA IN ECONOMICS AND FINANCE

TESI DI LAUREA

"CALIBRATION AND SIMULATION OF THE GAUSSIAN TWO-ADDITIVE-FACTOR INTEREST RATE MODEL"

RELATORE:

CH.MO PROF. MARTINO GRASSELLI

LAUREANDO/A: MATTEO FERRANTI

MATRICOLA N. 1061357

ANNO ACCADEMICO 2014 – 2015

Il candidato dichiara che il presente lavoro è originale e non è già stato sottoposto, in tutto o in parte, per il conseguimento di un titolo accademico in altre Università italiane o straniere.

Il candidato dichiara altresì che tutti i materiali utilizzati durante la preparazione dell'elaborato sono stati indicati nel testo e nella sezione "Riferimenti bibliografici" e che le eventuali citazioni testuali sono individuabili attraverso l'esplicito richiamo alla pubblicazione originale.

Firma dello studente

_____

Acknowledgments

Un ringraziamento alla famiglia, agli amici ed ai parenti per il supporto ricevuto. Un sincero ringraziamento a Silvia Negrato per l’accesso alla banca dati Bloomberg e al Prof. Martino Grasselli per il costante sostegno.

Abstract

Lo scopo di questa ricerca è di fornire una breve guida su come calibrare ed implementare un modello additivo a due fattori Gaussiani per simulare la dinamica del tasso d'interesse a breve. Per poter fare ciò è necessario fornire una breve spiegazione di alcune convenzioni usate dal mercato per prezzare i più importanti e liquidi strumenti vanilla come: cap, floor e swaption di tipo Europeo.

Nel secondo capitolo viene data una breve descrizione di come reperire le informazioni da un'importante piattaforma dati come Bloomberg Professional Terminal. Di seguito, si illustrano alcune procedure per costruire la curva dei tassi di sconto, meglio nota come zero-coupon curve, focalizzando l'attenzione specialmente sul classico metodo di bootstrapping in un contesto mono-curva e sulla calibrazione del modello parametrico di Nelson-Siegel-Svensson.

Dopo una accorta descrizione del modello G2++, il principale argomento del presente lavoro, si procede con la calibrazione, spiegando in dettaglio il funzionamento degli algoritmi di minimizzazione utilizzati. Una volta calibrati i parametri del modello, viene implementata una simulazione esatta della dinamica del tasso, ciò è possibile sfruttando la soluzione dell'equazione differenziale stocastica sotto la misura di probabilità forward. Attraverso questa simulazione è inoltre possibile prezzare contratti aventi caratteristiche più complesse rispetto ad una semplice opzione vanilla e, proprio per questo motivo, verrà fatto un esempio pratico di pricing di un contratto non vanilla fra due controparti. Il contratto, infatti, possiede alcune peculiarità che richiedono una simulazione affinché possa essere correttamente valutato.

Table of Contents

Introduction

The pricing of interest rate derivatives requires the development of specific models that must generate a fair price for the contract under analysis. However, before being able to use any of these powerful tools, every model has to be calibrated to a specific data set of vanilla instruments volatilities that must be available in the market. For what concerns the present work, also a specific category of vanilla instruments has to be selected, given the presence of numerous vanilla instruments based on interest rates, among which: caplets, floorlets, caps, floors and swaptions.

The calibration of such models is a complicated step, but highly important; however, what does it mean to calibrate an interest rate model to the market observed quotations? Since a specific stochastic process characterizes each model, by minimizing the squared percentage difference between the market and model implied volatilities or prices, the chosen model is able to capture the volatility surface observed in the market. However, in order to minimize a specific error function, it is necessary to choose a specific minimization algorithm.

As a matter of facts, when dealing with real world problems, the main issue is given by programming an efficient and effective algorithm; thus, a trade-off between computational time required (i.e. efficiency) and the likelihood of finding the true global minimum of our loss function (i.e. efficacy) is necessary. To solve this trade-off, a choice between deterministic or stochastic minimization algorithms has to be made. Indeed, if a deterministic minimization algorithm is selected, the calibration will be extremely fast, but the likelihood of achieving a global minimum will be rather low due to the dependency of the whole minimization from the initial starting point, while the opposite is true when a stochastic algorithm is implemented.

Once the initial calibration of the model to a set of vanilla instruments has been completed, it is possible to implement threes or Monte Carlo simulations for pricing more complex contracts, which can also involve some form of path dependency or the faculty of exercising the option before the final maturity of the agreement (i.e. the so-called American options).

The objective of this thesis is to give some hints on the calibration procedure for the two-addictive-factor Gaussian interest rate model (abbreviated as G2++), initialy developed by

Brigo and Mercurio (2006 pag. 137-175). The model will be implemented mainly by using analytical formulas and Monte Carlo simulations.

The first chapter gives a brief overview of some basic definitions and conventions commonly used by the market; indeed, starting from the basic mathematical definition of the bank account, we will go on explaining other recurring market terminologies (including the Libor, spot and forward rates). The final paragraph will introduce the famous Black analytical formula used by traders to price the tree most liquid vanilla contracts available for calibrating the G2++ model: caps, floors and European swaptions.

The second chapter gives some indications on where to find the data needed for the calibration while using a professional market data provider (i.e. Bloomberg). Moreover, two different methodologies for retrieving the discount factor curve in a mono-curve framework will be presented. Both these steps are crucial for the present work, since having wrong entries in the market volatility surface or in the discount factor curve may undermine the whole calibration procedure and the resulting model parameters. Thus, particular attention will be given to the Bloomberg terminal functions needed for downloading the right information and on Matlab implementation of the chosen methodology for calculating the discount factor curve. Indeed, when building the zero-coupon curve, another choice must be made between bootstrapping or calibrating a parametric form of the curve. Each procedure has its pros and cons and, always in the second chapter, an explanation of the two methodologies will be presented.

The third chapter develops the G2++ model as originally presented by Brigo and Mercurio (2006 pages 137-175); however, after that the procedure for finding the value of a zero-coupon bond will be enlighten, we will mainly focus on closed form equations for caps, floors and European swaptions prices and on their implementation in Matlab. In particular, the European swaptions formula poses some issues in practice; however, as we will see, different options arise for solving the improper integral present in the G2++ pricing formula for swaptions. Last, but not least, the third chapter concludes with some remarks on the model presented and on the mono-curve framework in general, taking into account the most recent developments of the multi-curve framework and its implementation by the market.

The calibration procedure is greatly discussed in the fourth chapter. After a review of the main loss functions used for calibrating models to market data, an in-depth analysis of the main algorithms implemented in Matlab for minimizing the sum of the percentage difference between model and market volatilities is presented. Despite the main objective of the present work is to simulate the interest rate dynamic of the G2++ model, the calibration using analytical formulas is faster than directly calibrating the Monte Carlo simulation; hence, when calculating the model prices, the analytical formulas presented in the third chapter will be used. Once the model has

been calibrated, it is advisable to try pricing out of the sample vanilla instruments and, given that the model will be calibrated to the swaptions prices, we will use caps prices as out of the sample instruments to better see if our model parameters are meaningful. Thus, this chapter is crucial for better understanding the nuances and limits of the calibration procedure presented in this thesis.

The following chapter focuses on building a Monte Carlo simulation for the short-term interest rate process under the T-forward measure. Despite more classical simulation techniques, the G2++ model has the advantage of having an explicit solution for the zero-coupon bond and, more importantly, the interest rate process transition density is completely known. Therefore, by using this analytical solution, it is possible to implement a faster simulation scheme that will not require the computation of the short rate at each simulation step. After the simulation scheme has been built, a specific contract will be priced with the Monte Carlo simulation scheme presented in this project and the resulting price will be compared with the one obtained by using swap manager (i.e. SWPM) in Bloomberg.

1st Chapter

Basic Market Concepts and Definitions

1.1 Basic Concepts

Before explaining in details the two-factor additive Gaussian interest rate model, it is necessary to present some general definitions for numerous basic concepts that will be frequently used in the following chapters, all the definitions in this first chapter have been adapted from Brigo and Mercurio (2006), if not stated otherwise.

The first definition concerns the money-market account, which implies indirectly also the definition of the discount factor and of the zero-coupon bond. Indeed, it is possible to give the following definitions:

Definition 1.1 (Money-Market account). *By defining $B(t)$ as the value of the bank account at time $t \geq 0$ and assuming that $B(t)$ follows the dynamics:*

$$dB(t) = r_t B(t)dt, \qquad B(0) = 1,$$

where r_t is a positive function of time, then:

$$B(t) = e^{\int_0^t r_s ds}.$$

From the above money-market account characterisation, the value of the discount factor as a ratio between different maturity bank accounts is retrieved.

Definition 1.2 (Discount factor). *In the interval delimited by t and T, the discount factor $D(t,T)$ is the amount of currency at time t that is equivalent to one unit of currency at time T, and is defined as:*

$$D(t,T) \equiv \frac{B(t)}{B(T)} = e^{-\int_t^T r_s ds}.$$

Definition 1.3 (Zero-coupon bond). *The zero-coupon bond with maturity T is a contract that ensure to its holder one unit of currency at time T; the value of the contract at time $t < T$ is indicated with $P(t,T)$.*

It is important to notice that: if the interest rate r is deterministic, also the discount factor will be deterministic (indeed the following equality holds $P(t,T) = D(t,T)$). However, since the present work is trying to price interest rate derivatives, it is necessary to let r have its own variance, implying that the interest rate r should be modelled by a specific stochastic dynamic. In this more general stochastic case, also $D(t,T)$ will be stochastic; however, its price at time t, which is equal to the zero-coupon bond by definition, must have a known value since it is an agreement between two counterparties. This known value is given by the following formula:

$$P(t,T) \equiv E_t[D(t,T)] = E_t\left[e^{-\int_t^T r_s ds}\right].$$

As one can see, the zero-coupon bond is given by an expectation on the value of the discount factor. This expected value is conditioned on the filtration at time t, where the filtration $\mathcal{F}_t$ identifies all the information available at time t.
Moreover, it is possible to establish that if the contract is defined between the interval $[T_\alpha, T_\beta]$ with its value set at time t, with $t \leq T_\alpha \leq T_\beta$, we call this type of contract forward bond.

Definition 1.4 (Forward Bond). *The value at time T_α of a zero-coupon bond with maturity T_β priced at time t, with $t \leq T_\alpha \leq T_\beta$, is given by:*

$$P_{\alpha\beta}(t) = P(t; T_\alpha, T_\beta) \equiv \frac{P(t,T_\beta)}{P(t,T_\alpha)}.$$

in case of a stochastic interest rate:

$$P_{\alpha\beta}(t) \equiv E_t[D(T_\alpha, T_\beta)] = E_t\left[e^{-\int_{T_\alpha}^{T_\beta} r_s ds}\right].$$

From the definition (1.3) and (1.4), we can find also the value of a coupon-bearing bond.

Definition 1.5 (Coupon bond). *Given a set of dates $T_i,\ i = 1, \dots, M$, such that $T_1 = T_{\alpha+1}$ and $T_M = T_\beta$. A coupon bond is a contract that, at each time $T_i,\ i = 1, \dots, M$, ensures to its holder a deterministic amount c_i and one unit of currency at time T_β. The value of a generic coupon bond at time $t \leq T_\alpha$ is equal to:*

$$CB_{\alpha\beta}(c;t) = \sum_{i=\alpha+1}^{\beta} c_i \tau_i P_{\alpha i}(t) + P_{\alpha\beta}(t).$$

Where τ_i stands for the interval $[T_{i-1}, T_i]$.

The notation usually adopted in practice for a zero-coupon bond valued in the interval $[T_{i-1}, T_i]$, defined by two consecutive maturities, is $P_i(t) = P(t; T_{i-1}, T_i)$. By meanings of the definition (1.4) and of the above equation, the following formula for a generic zero-coupon forward bond can be inferred.

$$P_{\alpha\beta}(t) = \prod_{i=\alpha+1}^{\beta} P_i(t).$$

The above formulas and definitions will be used frequently in future chapters with the same notation.

1.2 Time Conventions

Given two time instants T_1 and T_2, and the length of the interval $[T_1, T_2]$ measured as a yearly fraction, and specified as $\tau(T_1, T_2)$, if the two quantities T_1 and T_2 are expressed in terms of day-month-year ($T = (d, m, y)$), the quantity $\tau(T_1, T_2)$ depends from the day count convention chosen to calculate the number of days inside the interval $[T_1, T_2]$.
In money markets, each instrument agrees to a specific rule depending on the specific underling IBOR (IntraBank Offered Rate); however, for what concerns the Euro area and the Euribor, the most frequently used day count conventions in the market are:

- **Actual/360.** In this convention the year is composed by 360 days and the year fraction between two dates $\tau(T_1, T_2)$ is simply equal to:
$$\frac{d_2 - d_1}{360}.$$
- **Actual/365.** In this case the year is composed by 365 days and the year faction between two dates $\tau(T_1, T_2)$ is given by the following ratio:
$$\frac{d_2 - d_1}{365}.$$
- **30/360.** According to this convention, each month is composed by 30 days summing up to 360 days per year; to calculate the year fraction between two dates, $\tau(T_1, T_2)$, the following formula has to be used:
$$\frac{max(30 - d_1, 0) + min(d_2, 30) + 360(y_2 - y_1) + 30(m_2 - m_1 - 1)}{360}.$$

After choosing a specific day count convention, an adjustment for business holidays is also necessary to match the specific cash flow dates used by Bloomberg to calculate options' strikes and volatilities. Unfortunately Matlab does not adjust automatically for holidays when

calculating year fraction between dates; nevertheless, by simply adopting an iterative command, such as a `while` statement, this correction may become straightforward (for more precise information see Appendix I).

1.3 Interest rates

Each interest rate is modelled on different zero-coupon bonds; indeed, the starting building block of the whole interest rates market in a mono-curve framework is the zero-coupon bond. Nevertheless, before going on with the discussion, it is advisable to make a clear distinction between interbank interest rates and government interest rates. Indeed, while the latter category comprises interest rates that are based on government bonds, the formers rates are associated with deposits and other instruments traded among banks.

From the recent situation in financial markets, one can clearly imagine that government bonds include some type of country specific risks and will trade with a premium on the IBOR rate; thus, since we are trying to model expectations on quantity that is theoretically "risk-free", we should focus our attention only to interbank interest rates. Moreover, interbank rates are the underlings of many liquid contracts traded in money markets, such as deposits, forward rate agreements and interest rate swaps; thus, by choosing interbank interest rates, we are able to retrieve data on sufficient maturities, leading to a more accurate bootstrapped curve.

Therefore, in the upcoming chapters and paragraphs, the word interest rate refers to interbank interest rates; however, to be more precise, it will mainly refers to the EURIBOR. Since the EURIBOR will be the interest rate under analysis in this thesis, it is advisable to give a brief definition of the EURIBOR and to describe how it is calculated.

The EURIBOR, which stands for EURo InterBank Offered Rate, is the interbank interest rate relative to a panel of European banks and it is linked to the rate at which these banks borrow funds from each other. The panel of banks may vary over time, since the bank sample should be a diversified representation of the European money market. These banks are required to exchange information with the TARGET (Trans-European Automated Real-Time Gross-Settlement Express Transfer system) on their mutual borrowing costs for every maturity starting from seven days up to one year. Starting from these inputs, the interbank interest rate index is calculated by the GRSS (Global Rate Set System) and published at 11:02 C.E.T. (Central European Time) each business day by Bloomberg, Thomson Reuters and many other professional financial platforms.

Despite the fact that it is commonly referred to as the Euribor, suggesting the existence of only one Euribor rate, in reality there are eight different rates accordingly to the specific maturity associated with the interbank deposit.

Another important interbank interest rate is the LIBOR (London InterBank Offered Rate) which is an index that measure the cost of funds for different large banks, which operate in London financial markets. The Libor is published at 11:30 L.T. (London Time) by Thomson Reuters every business day.

These two rates are among the most important interbank interest rates and, in what follows, the word Libor is used interchangeably as Euribor or Libor as it may be any type of interbank interest rate under analysis.

After this brief review of the two key interbank interest rates, let us introduce a more precise mathematical definition for the spot Libor interest rate and the forward Libor interest rate and see how, from these definitions, we can retrieve the value of a zero coupon bond.

Definition 1.6 (Spot Libor). *The Libor rate $L(t,T)$ at time t with maturity T is the interest rate that has to be applied to an investment of $P(t,T)$ units of currency at time t to ensure one unit of currency at maturity T. For the time period $\tau(t,T)$, it is calculated by solving the following equations*

$$P(t,T)[1+\tau(t,T)L(t,T)] \coloneqq 1,$$

by isolating to the left the term of interest, $L(t,T)$, the following mathematical definition of the spot Libor is obtained

$$L(t,T) \coloneqq \frac{1-P(t,T)}{\tau(t,T)P(t,T)}.$$

By inverting the above formula, the zero coupon bond may be defined in term of the spot Libor rate, which is known and quoted in the market

$$P(t,T) = \frac{1}{1+\tau(t,T)L(t,T)}.$$

This formula is of particular interest since it will be used in order to obtain the discount factors from market quoted short-term deposits.

Definition 1.7 (Forward Libor). *The forward Libor $L(t;T_\alpha,T_\beta)$, set at time t, relatives to the time interval $[T_\alpha,T_\beta]$, is the interest rate that has to be applied to an investment of $P(t,T)$ units of currency at time T_α to ensure one unit of currency at maturity T_β. When calculated for the year fraction $\tau(T_\alpha,T_\beta)$, the forward Libor is defined as:*

$$L_{\alpha\beta}(t) = L\big(t; T_\alpha, T_\beta\big) \coloneqq \frac{P(t,T_\alpha) - P\big(t,T_\beta\big)}{\tau\big(T_\alpha,T_\beta\big)P\big(t,T_\beta\big)}.$$

Analogously to the spot Libor, from here we are able to find the value of the forward zero-coupon bond

$$P_{\alpha\beta}(t) = \frac{1}{1+\tau\big(T_\alpha,T_\beta\big)L_{\alpha\beta}(t)}.$$

Hence, we can extrapolate the value of a zero-coupon bond at time $t = \beta$ by using the above relation, as we will see in the following chapter.

The next definition may also be useful in characterizing the continuous spot zero rate, even if we can limit our attention only to zero-coupon bonds, as they are the starting point for calibrating the model under analysis.

Definition 1.8 (Zero rate). *The spot zero rate $R(t,T)$, set at time t with maturity T, is the interest rate that has to be applied to an investment of $P(t,T)$ units of currency at time t to ensure one unit of currency at maturity T if continuous compounding has been adopted by the investment. Following this definition, the zero rate is simply defined as:*

$$R(t,T) \coloneqq -\frac{\ln P(t,T)}{\tau(t,T)}.$$

Even in this case, we can identify the price of the related zero-coupon bond as a function of the zero rate by inverting the above equation

$$P(t,T) = e^{-R(t,T)\tau(t,T)}.$$

1.4 From Interest Rate Swaps to European Swaptions

Some general definitions about interest rate instruments will be given in this paragraph, these definitions will be frequently used in future chapters; indeed, it is advisable that the reader tries to understand the notation of each instrument, since this symbolisation is going to be used also in the following chapters.

Let us start with two crucial definition. The first definition is the interest rate swap, or more briefly swap. This particular contract is the underling of numerous interest rate derivatives and, among these, of special interest are caps, floors, and European swaptions. Indeed, the value of these instruments is used to calibrate the two-factor additive Gaussian interest rate model.

Definition 1.9 (Swap). *Given a set of dates $T_\alpha, \dots, T_\beta$, a swap is a contract stipulated at time $t \leq T_\alpha$ that involves an exchange of cash flows at each date T_i, $i = \alpha + 1, \dots, \beta$, and is characterized by:*

- *a* Fixed Leg *that at each reset time T_i, $i = \alpha + 1, \dots, \beta$, pays the following quantity:*

$$N\tau_i K$$

  where K is a fixed number;

- *a* Floating Leg *that at each time interval $[T_{i-1}, T_i]$, $i = \alpha + 1, \dots, \beta$, fixes the below quantity at time T_{i-1}, this quantity will be paid at the next reset time T_i by the holder of the interest rate swap*

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

The contract is called "Payer Swap" *if the counterparty "A" pays to "B" the fixed leg and receives in exchange the floating leg; otherwise, the contract is named* "Receiver Swap".

The actual value at time t of a payer swap is given by the sum of the actualized cash flows times the notional value of the contract

$$\sum_{i=\alpha+1}^{\beta} P(t, T_i) N\tau_i (L_i(t) - K),$$

where $L_i(t)$ stands for the forward Libor rate, which is given by the below relation accordingly to a no-arbitrage argument

$$L_i(t) \equiv L(t; T_{i-1}, T_i) = \frac{1}{\tau_i}\left(\frac{1}{P_i(t)} - 1\right).$$

A quantity of particular interest is the forward swap rate, since it is also the At-The-Money strike rate for caps, floors and European swaptions. These derivatives are among the most liquid instruments traded in the market; indeed, given the existence of analytical pricing equations, they are frequently used by practitioners to calibrate different interest rate models, including the two-factor additive Gaussian interest rate model, which is the model that will be analysed in the present work.

Definition 1.10 (Forward swap rate). *The forward swap rate (also known as par swap rate) is the value of the fixed rate K that makes the NPV (Net Present Value) of the swap equals to zero at the writing date t.*

$$S_{\alpha\beta}(t) = \frac{P(t, T_\alpha) - P\left(t, T_\beta\right)}{\sum_{i=\alpha+1}^{\beta} \tau_i P(t, T_i)},$$

By substituting $t = 0$, we obtain the most interesting case, since all the data available in the money market refers to quantities that are calculated at present time

$$S_{\alpha\beta}(0) = \frac{P(0,T_\alpha) - P(0,T_\beta)}{\sum_{i=\alpha+1}^{\beta} \tau_i P(0,T_i)}.$$

After the introduction of the forward swap rate, it is time to define two of the main interest rate derivatives traded by the market: caps and floors.

A Cap contract is an agreement that gives to its holder the possibility, but not the obligation, to pay a floating leg and receive a fixed leg at specific times. This type of contract can be decomposed as a sum of elementary contracts named *Caplets*.

Definition 1.11 (Caplet). *A caplet ,with a fixed rate K (known as the cap rate), defined over an interval* $[T_{i-1}, T_i]$*, is an option with the payoff at time* T_i *given by*

$$h_{caplet_i} = \tau_i [L(T_{i-1}, T_i) - K]^+.$$

A caplet is essentially a call option that has as underling the spot Libor rate calculated for the interval $[T_{i-1}, T_i]$.

A company may be interested in entering into a cap (or caplet) contract for hedging against interest rate fluctuation if its debt is linked to a variable Libor rate; indeed, the firm will be able to give a maximum value to the variable Libor rate by acquiring a cap contract. This maximum amount of the Libor rate will be equal to the cap strike rate *K*

$$L - (L - K)^+ = min(L, K).$$

Using the above reasoning, we can express a cap contract as a summation of caplets, hence the following definition.

Definition 1.12 (Cap). *A cap with cap rate K, defined over a set of maturities* $[T_{\alpha+1}, T_i]$*, is a contract that gives to its holder the related caplet payoff for each maturity; the total actualized value at time* t *of the contract is equal to the sum of all the caplets' payoffs taking into account the notional value of the contract N*

$$\sum_{i=\alpha+1}^{\beta} P(t,T_i) N \tau_i [L_i(t) - K]^+.$$

By analogy, we can also define a floor contract as an agreement that gives to its holder the right, but not the obligation to pay a fixed leg and receive a floating leg at given times. Moreover, this

type of agreement may also be broken into a summation of elementary contracts called *Floorlets*.

Definition 1.13 (Floorlet). *A floorlet with a fixed rate K (also known as the floor rate), defined over an interval $[T_{i-1}, T_i]$, is an option with the payoff at time T_i given by*

$$h_{floorlet_i} = \tau_i [K - L(T_{i-1}, T_i)]^+.$$

A generic floorlet may resemble a put option that has as underling the spot Libor rate defined in the interval $[T_{i-1}, T_i]$, analogously to the caplet case.
As opposed to a cap, a creditor may enter into a floor agreement in order to hedge a loan indexed to a variable Libor rate; indeed, the creditor is hedged against fluctuation of the interest rate to which the debt is linked, since the minimum amount that the creditor would obtain by hedging using a floor contract is given by:

$$L + (K - L)^+ = max(K, L).$$

As for a cap, also a floor payoff is commonly described as a summation of payoffs generated from the underling floorlets contracts.

Definition 1.14 (Floor). *A floor with a floor rate K, defined by a set of maturities $[T_{\alpha+1}, T_\beta]$, is a contract that at each maturity gives to its holder the related floorlet payoff; the total actualized value at time t of the agreement is equal to the sum of all the floorlets payoffs taking into consideration the notional value N of the contract*

$$\sum_{i=\alpha+1}^{\beta} P(t, T_i) N \tau_i [K - L_i(t)]^+.$$

By looking at these two contracts payoffs, one can notice how a cap (floor) definition is similar to a swap agreement where each payment becomes due only if its value is positive: more precisely a cap (floor) may be seen as a payer (receiver) swap that pays at each maturity only if the relative payoff is positive.
Caps (or floors) are typically quoted by the market in the form of implied volatilities from a Black-like formula. Indeed, given different maturities and notional values, the market has adopted the convention of representing prices as volatilities since volatilities do not depend on the notional value of the specific contract. However, before we can give a definition of the Black cap (floor) formula, it is necessary to briefly highlight the core Black formula, which is at the basis of many equations used conventionally by the market to price, not only interest rate

derivatives, but also contracts that have different underlines, such as: commodities, equities or exchange rates.

Proposition 1.1 (Black core formula). *The price at time t of a call (put) option on an underling future with value F and strike price K, is the result of*

$$Bl(K, F, \upsilon, \omega) = F\omega\Phi(\omega d_1) - K\omega\Phi(\omega d_2),$$

where $w = 1$ ($w = -1$) for call (put) options, and:

$$d_1(K, F, \upsilon) = \frac{\ln(F/K) + \upsilon^2/2}{\upsilon},$$

$$d_2(K, F, \upsilon) = \frac{\ln(F/K) - \upsilon^2/2}{\upsilon}.$$

From the Black core formula, it is possible to infer the value of a single caplet (or floorlet) by substituting the future Libor rate as the underling of the contract, and by discounting the expected value using the value of the zero-coupon bond having the same maturity of the option.

Proposition 1.2 (Black formula for caplet and floorlet). *The value of a caplet (floorlet) at time t, defined over the interval $[T_{i-1}, T_i]$, with a caplet (floorlet) rate K and notional N, is given by the following Black like formula*

$$caplet_i(t) = P(t, T_i)N\tau_i[L_i(t)\Phi(d_1) - K\Phi(d_2)]$$

with

$$d_1 = \frac{\ln(L_i(t)/K) + {\sigma_i^{mkt}}^2(T_i - t)/2}{\sigma_i^{mkt}\sqrt{T_i - t}},$$

$$d_2 = d_1 - \sigma_i^{mkt}\sqrt{T_i - t}.$$

While for floorlet contracts

$$floorlet_i(t) = P(t, T_i)N\tau_i[K\Phi(-d_2) - L_i(t)\Phi(-d_1)].$$

One can write both formulas in the following more compact way

$$CF_i(t) = P(t, T_i)N\tau_i Bl(K, L_i(t), \upsilon, \omega),$$

with $\omega = 1$ ($\omega = -1$) for caplets (floorlets), and $BL(\cdots)$ is the Black core formula defined in proposition (1.1).

It is easy to infer that the value of a cap (or floor) will be given by the summation of each caplet (floorlet) Black formula with maturities ranging from the first reset payment to the maturity of the contract $[T_{\alpha+1}, T_\beta]$, leading to

$$cap_{\alpha\beta}(t) = \sum_{i=\alpha+1}^{\beta} caplet_i(t),$$

$$floor_{\alpha\beta}(t) = \sum_{i=\alpha+1}^{\beta} floorlet_i(t).$$

When traded in the market, both caps and floors, may have two different lengths of the underlying cap: 3 or 6 months. However, in a two years cap written today on the 6 months Libor rate, there are only three caplets, since the first six months are not considered and the first caplet will start at 6 months and mature after one year from today.

Another derivative contract frequently used by practitioners when calibrating interest rate models is the European swaption, given that this instrument contains information on the correlation between different points of the zero-coupon curve and is able to capture the negative correlation between stochastic factors in multi factors models, including the G2++.

Definition 1.15 (Swaption). *A European swaption, with rate K and maturity T, is an option that gives to its holder the right, but not the obligation, to enter in an underling interest rate swap, with fixed rate K, at time T. The swaption can be a* "Payer Swaption" *or* "Receiver Swaption" *depending on the underling interest rate swap type.*

Usually the first cash flow is not equal to the maturity of the European swaption, $T = T_\alpha$, but start from the next period. The length of the interval $[T_\alpha, T_\beta]$ is generally referred to as *tenor*. The option will be exercised at maturity T_α, only if the value of the underling interest rate swap is positive.

Hence, the value of a payer European swaption is given by the positive part of the payoff of the underling interest rate swap

$$h_{PS}(T_\alpha) = \left[\sum_{i=\alpha+1}^{\beta} P(t,T_i)\tau_i(L_i(t) - K)\right]^+.$$

we can rewrite the above equation as a function of the forward swap rate

$$h_{PS}(T_\alpha) = BPV_{\alpha\beta}(T_\alpha)\left[S_{\alpha\beta}(T_\alpha) - K\right]^+.$$

In the market, swaptions are commonly quoted as volatilities implied by an equation that originates directly from the Black '76 formula already explained briefly in Proposition 1.1. From definition (1.15), we can infer that a European swaption is a contract acquired at time t

that gives to its holder the right, but not the obligation, to enter into a series of call (put) options with strikes K on the underling forward starting swap rate. The call or (put) options have each a different reset time T_i beginning from time T_α.We can now give the following formula for calculating the price of a European swaption.

Proposition 1.3 (Black formula for swaption). *The price at time t of a payer swaption with strike price K and defined in the interval $[T_\alpha, T_\beta]$, is given by the Black formula*

$$PS_{\alpha\beta}(t) = P(t,T_\alpha)BPV_{\alpha\beta}(t)\left[S_{\alpha\beta}(t)\Phi(d_1) - K\Phi(d_2)\right]$$

where $\Phi(\cdot)$ is the standard normal cumulated distribution and

$$d_1 = \frac{\ln\left({S_{\alpha\beta}(t)}/{K}\right)}{\sigma_{\alpha\beta}^{mkt}\sqrt{T_\alpha - t}} + \frac{1}{2}\sigma_{\alpha\beta}^{mkt}\sqrt{T_\alpha - t}, \qquad d_2 = d_1 - \sigma_{\alpha\beta}^{mkt}\sqrt{T_\alpha - t},$$

for the receiver swaption

$$RS_{\alpha\beta}(t) = P(t,T_\alpha)BPV_{\alpha\beta}(t)\left[K\Phi(-d_2) - S_{\alpha\beta}(t)\Phi(-d_1)\right].$$

It is possible to rewrite the formula in a more compact format as:

$$ES_{\alpha\beta}(t) = \omega P(t,T_\alpha)BPV_{\alpha\beta}(t)\left[S_{\alpha\beta}(t)\Phi(\omega d_1) - K\Phi(\omega d_2)\right]$$

$$ES_{\alpha\beta}(t) = P(t,T_\alpha)BPV_{\alpha\beta}(t)Bl\left(K, S_{\alpha\beta}(t), v, \omega\right)$$

with $\omega = 1$ ($\omega = -1$) if the swaption is a payer (receiver) contract.

The time intervals between cash flows are calculated using the 30/360 day count convention, and the term $\sigma_{\alpha\beta}^{mkt}$ is the implied volatility quoted by the market (which is a function of the maturity and of the tenor of the swaption, but this point will not be discussed in this thesis).
In this first chapter, we focused our attention on basic definitions and instruments that are often used by traders and practitioners in order to price the most traded and liquid vanilla options. Indeed, starting from these tools and contracts we are able to calibrate the G2++ model to the volatility surface of caps or swaptions, accordingly to our needs; however, it is time to see how data on these instruments may be obtained from a professional financial information provider such as Bloomberg.

2nd Chapter

The Market Data

2.1 Introduction

In this second chapter, a brief introduction to the data used for calibrating the model will be presented. As everyone may imagine, it is crucial to use the right data not only for building the discount factors curve, and the related spot curve, but also for the market implied volatilities surface. More importantly, the G2++ model presented in the next chapter has a deterministic shift (the function $\varphi(t)$) that is added to the two stochastic processes in order to match exactly the term structure of interest rate at time 0, making the whole model dependent on the data used for building the zero-coupon curve.

Moreover, data on the zero-coupon curve are frequently used also in other functions; indeed, as shown in the first chapter, the At-The-Money strike of a European swaption is given by the forward swap rate, which is calculated using the zero-coupon curve built using the Swap curve data as an input of a specific function in Matlab.

2.2 The Data on the Interest Rate Curve

Many approaches are possible when building the zero-coupon curve from market data; however, for the sake of brevity, only two different methods will be analysed in the present work:

I. mono-curve framework bootstrapping;
II. fitting a term-structure model, the Svensson model in the present chapter, which is an extended version of the Nelson-Siegel model.

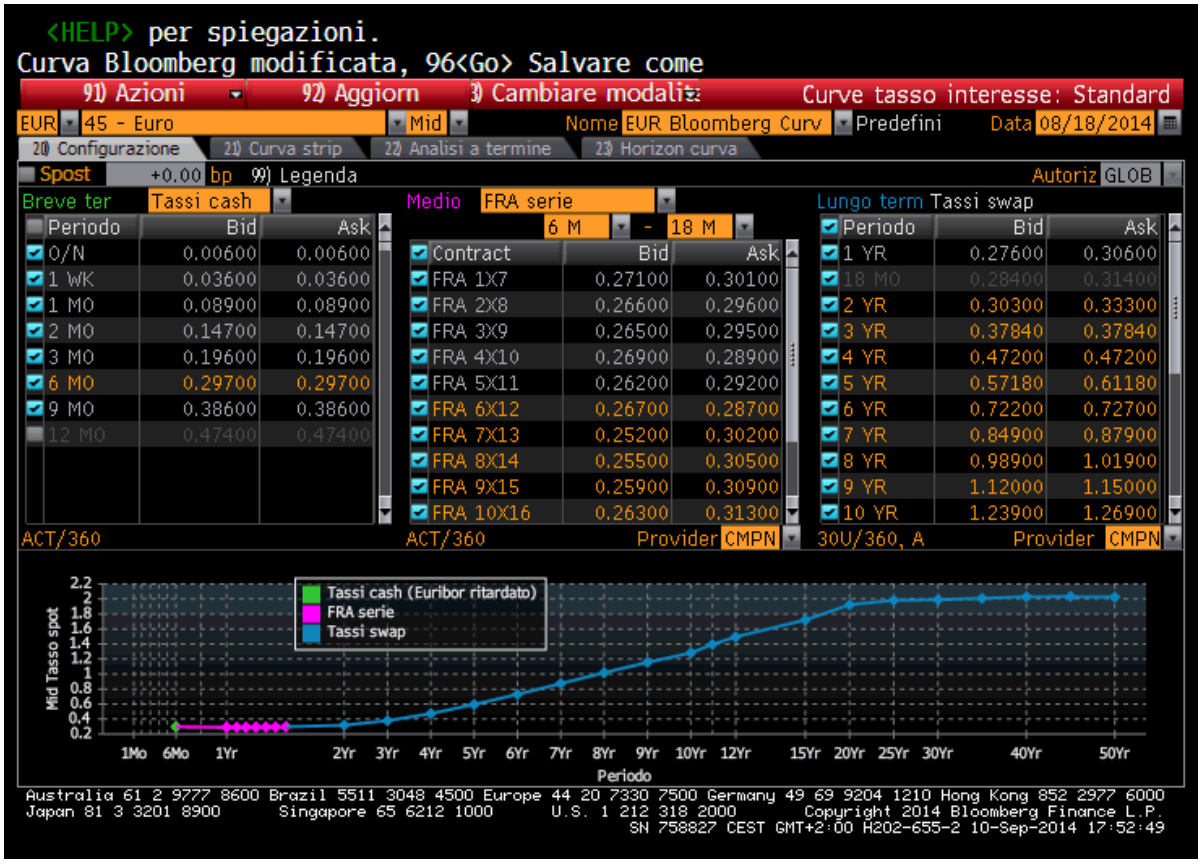
However, before describing these two procedures, we will briefly explain where to find the related data on a platform commonly used by traders to retrieve information on money market rates and interest rate derivatives contracts: Bloomberg. Bloomberg is one of the main service data provider for the financial industry (alongside with Thomson Reuters, Murex and other platforms), it offers some specific functions that summarise data on the zero-coupon curve observed in the swap market and used for discounting cash flows: SWDF.

From the SWDF function (Swap Discount Factors), one can easily obtain data on the Euro-area zero-coupon bond curve. By simply typing SWDF and GO in the command window, the trader is able to analyse the fundamental instruments used by Bloomberg to build the discount factor curve that is used in many other important money market and interest rate derivatives functions, among which:

- Swap Manager (SWPM), the built in Bloomberg pricer for interest rate derivatives;
- Interest Swap Curve Builder (ICSV), another important tool for bootstrapping curves based on money market instruments;
- Volatility Cube (VCUB), the function containing prices for interest rate derivatives, which are expressed as implied volatilities.

On Fig. 1, it is possible to see how data are organized by the SWDF function in Bloomberg.

Fig. 1: SWDF Function (source: Bloomberg)



The top section of the screen includes a series of options highlighted in orange regarding:

a) the currency of the curve, EUR standing for the euro-area curve;
b) the specific type of curve, in this case the option is set to 45 – Euro and we will see shortly after why this particular curve has been selected;
c) the market side of the data, MID stands for mid quotes;
d) the name of the curve, EUR Bloomberg Curve for the screen in hand;
e) the date of the data in mm/dd/yyyy format, 08/18/2014 in this case.

Above these options, it is possible to see a red bar that contains more options and some actions that the user can do with the data in hand. The most useful option for the present work is the one that let the trader download the data on the instruments used by Bloomberg to build the zero-curve into an Excel file, and this is done by opening the drop-down menu under Actions and clicking the Import to Excel command.

Moving on to the central section of the SWDF screen, one can notice a collection of market data divided into tree tabs, one for each part of the zero-curve. The left side tab contains data on short-term deposits or cash rates, as in Fig. 1, starting from S/N (spot next deposit or cash rate) up until one-year maturity. The tab in the middle contains data on Forward Rate Agreements (FRAs), or forward rates depending on the instrument selected by the user, up until two years and six months maturity. The tab on the right of the screen encompasses Interest Rate Swaps (IRS) rates up until fifty years maturity, starting from one year maturity IRS. As one can clearly notice, some instruments maturities overlap, but Bloomberg selects automatically which instrument to use accordingly to the liquidity of the instrument, giving precedence to the most frequently traded instruments, and the fixing of the instrument (i.e. to avoid overlapping due to different settlement day conventions).

At the bottom of the screen, the spot curve obtained from market instruments is represented in different colours accordingly to the type of instrument used to bootstrap the curve: the green part of the curve uses deposits or cash rates, the pink section is bootstrapped from FRAs or forwards contracts and the blue segment is obtained from IRS. Each dot in the curve is tied to a specific contract in one of the tree tab already described; therefore, Bloomberg highlights in orange the instruments used for building the curve, while the others are shown in light grey.

Before moving on with the discussion, it is important to analyse the SWDF options adopted for downloading the data for the present work.

Firstly, it is crucial to underline that we will rely on the default curve settings from Bloomberg, because changing any of these options may have repercussions also in other Bloomberg functions, something that we want to avoid. Hence, as far as the present work is concerned, we will rely on the same data points used by Bloomberg to build the spot rate curve and the discount factor curve, since our bootstrapped curve should match as much as possible the curve built by the Bloomberg proprietary algorithm.

Secondly, the only change that was made from the standard options, shown in Fig. 1, was to switch the shorter-term instruments from Cash Rates to Deposits. The main reason behind this small deviation from Bloomberg default curve is that Bloomberg calculates cash rates using some proprietary algorithms; thus, since Bloomberg does not explain the passages of this procedure, we are not able to calculate the zero-coupon value from directly the cash rates when

bootstrapping the curve. Moreover, since the majority of the literature refers to the short-end of the curve by using the word "deposits" as bootstrapping instruments, we have decided to modify this section of the SWDF function in order to consider deposits and not cash rates.
Finally yet importantly, since the curve used by Bloomberg in SWPM and VCUB is the so-called Euro 45 Bloomberg Curve, we will rely on the same curve for the sake of consistency with the volatility data presented by VCUB, making only the before mentioned adjustment.
The Bloomberg Euro 45 curve uses deposits, forward rate agreements and interest rate swaps contracts in order to bootstrap discount factors from the most liquid instruments quoted in the market. Thereof, after this brief discussion on the source of the data, it is time to analyse how the zero-coupon bond value is deducted from each type of instrument, starting from the short-end of the curve.
After this brief explanation on the SWDF function, we are able to describe how to build the discount factor curve starting from the usual bootstrapping procedure.

2.3 Bootstrapping the Discount Factors from SWDF Data

The function `DF_Bootstrapping` (see Appendix I) is implemented for bootstrapping the discount factors starting from market data on Deposits, Forward Rate Agreements (FRAs) and Interest Rate Swaps (IRSs), which are used respectively as short, medium and long terms instruments for building our own curve. In what follows, a brief description of the methodology used for each type of instrument is reported, starting from the short-end of the curve.

2.2.1 Deposits

Deposits with maturity starting from s/n (spot-next) until 6 months are used as building blocks for bootstrapping the discount factor curve for the interval $[T_0, T_{6months}]$ in the majority of the literature. Indeed, this type of instrument is constructed with the purpose of ensuring a quantity equals to $[1 + (T_i - T_0)L_i(T_0)]$ at time T_i with a unitary deposit at time T_0.
Nevertheless, despite the presence of shorter maturity, the first instrument will coincide with the six months deposits mainly because:

- according to Bloomberg, deposits with shorter maturities are not enough liquid;
- since we will use derivatives based on the six months tenor Euribor rate, we are trying to be as much consistent as possible with the underlying tenor of our discount factor curve and the underlying tenor of European swaptions and caps.

The zero-coupon bond $P(T_0, T_i)$ represents the value at time T_0 of one unit of currency at time T_i, so it is possible to define the zero-coupon bond as a function of the quoted deposit rate

$$P(T_0, T_i) = \frac{1}{1 + (T_i - T_0)L(T_0, T_i)},$$

where the interval $[T_0, T_i]$ is calculated using the Act/360 day count convention.
After the short end of the curve has been bootstrapped, it is necessary to move on to the bootstrapping of the middle section of our discount factor curve by using FRAs.

2.2.2 Forward Rate Agreements

A generic forward agreement is defined by using three time instants: the time of evaluation t, the expiry time of the forward S and the maturity of the underling deposit T, with $t \leq S \leq T$. Each forward rate is related to a future time investment and this suggests that each forward rate should be defined coherently with the related discount factor. This implies that, if we analyse different FRAs, we can find the related values for the zero-coupon bond curve.
A forward rate agreement is a contract in which two counterparties exchange future cash flows based on an interest rate defined in the interval between S and T. At time T the holder of the FRA receives a payment with a fixed rate K and pays a variable interest rate $L(S, T)$ (which in this particular case is the six months Euribor between S and T).
Forward rate agreements quoted by the market are defined such that the NPV of the contract is equal to zero, and are useful to bootstrap the discount factor curve for medium maturities, the interval between $[T_{6Months}, T_{2Years}]$. Quoted rates in the market actually stand for forward rates $L_i(T_0) = L(T_0; T_{i-1}, T_i)$ with, following the previous notation, $S = T_{i-1}$ and $T = T_i$.
Thus, for calculating the value of a forward bond, it is possible to invert the FRA formula

$$P_i(T_0) \equiv P(T_0; T_{i-1}, T_i) \equiv \frac{1}{1 + (T_i - T_{i-1})L_i(T_0)},$$

where the time interval is calculated using the Act/360 day count convention.
Using the curve calculated until this point, we can obtain the zero-coupon bond for the next maturity T_i by the following relation

$$P(T_0, T_i) = P(T_0, T_{i-1})P_i(T_0).$$

Before moving on to the last part of our zero-coupon bond curve, it is advisable to notice that, also for this section of the curve, we have adopted the same instruments used by the Bloomberg Euro 45 curve. Additionally, we have selected FRAs with 6 months tenors, given that we must

be consistent with the underlying Euribor tenor of European swaptions and caps used to calibrate the G2++ model.

2.2.3 Interest Rate Swaps

As already stated before, a forward swap may resemble a generalization of a FRA. At time T_0, the two counterparties establish a future exchange of cash flows, starting from T_α until T_β, at each intermediate date T_i (for a better explanation look at definition 1.10).
The market quotes only spot swap contracts (with $T_0 = T_\alpha$) for a given set of different maturities (including $T_\beta = 1, 2, \ldots, 10, 12, 20$ years) and from these data, we can obtain the par swap rate

$$S(T_0, T_\beta) = \frac{1 - P(T_0, T_\beta)}{\sum_{i=1}^{\beta} \tau_i P(T_0, T_i)}.$$

By inverting the above equation, it is also possible to calculate the zero-coupon bond for each IRS maturity quoted by the market

$$P(T_0, T_\beta) = \frac{1 - S(T_0, T_\beta) \sum_{i=1}^{\beta} \tau_i P(T_0, T_i)}{1 + \tau_\beta S(T_0, T_\beta)}.$$

In this case, the intervals τ_i are annual intervals calculated using the 30/360 day count convention, which is the convention used more frequently in the swap market. Beware that, in order to compute the above formula, the value of the zero-coupon bond at each reset time has to be known, so an intermediate zero curve has to be interpolated in order to extrapolate the value of each $P(T_0, T_i)$, given the absence of data for IRSs on a series of intermediate maturities.
From the framework depicted above and using the definitions already given, we can calculate the discount factors starting from the rates quoted by the market, through a procedure known as bootstrapping.
If we take into consideration the following transformation

$$R(T_0, T) = -\frac{\ln P(T_0, T)}{T_0 - T}$$

the zero rate curve can also be built through the same procedure, once we know the discount factors.
The above technique is implemented in Matlab by the `DC_Bootstrapping` function. In this function, raw data are divided into three different instrument categories (`'Deposits'`, `'FRA'` and `'IRS'`) and, once the data have been divided, the instrument specific formula is applied for each rate and maturity in order to calculate the value of the zero-coupon bond.

Following the line of reasoning depicted above, the algorithm moves from short-term deposits to long-term IRSs. However, for this last category of instrument a remark has to be made, as for each intermediate reset date it is necessary to:

1. adjust intermediate cash flows for holidays and non-trading days, unfortunately Matlab allows only for adjustments that follows the NYSE (New York Stock Exchange) holidays calendar, but this is sufficient to avoid weekends;
2. interpolate a temporary discount factor curve, as already stated above.

After we have interpolated all the zero-coupon bond values, using the Matlab built in function `interp1`, it is possible to build the whole zero-coupon curve through shape preserving piecewise cubic interpolation (`'pchip'`[1] in Matlab). By using this particular technique, the discount factor curve is able to maintain certain desirable characteristics, such as convexity and continuity.

Unfortunately, the above interpolation algorithm is not the "right" one; indeed, there are plenty of different methodologies available for interpolating different points and practitioners have not yet decided which is the most "correct" one for bootstrapping the zero-coupon curve. Nevertheless, we believe that having a convex and continuous discount factor curve is essential for pricing interest rate derivatives. Thus, without further discussions on this topic, that will require too much space and time to be discussed deeply, the shape-preserving piecewise cubic interpolation is considered to be the right trade-off between the computational time required and the characteristics of the interpolated function.

Once the zero-coupon curve has been bootstrapped from market's data, it is possible to retrieve the discount factor value for any maturity by building an anonymous function for the zero-curve value and by adding the `'extrap'` option into the `interp1` function

```
PM = @(t) interp1(CT,DF,t,'pchip','extrap');
```

where `CT` is the year fraction between T_0 and the maturity of the instrument used for bootstrapping the related value of `DF`, representing the bootstrapped zero-coupon value for the specific maturity; `@(t)` is the Matlab syntax for anonymous functions where `PM` depends from a specific value of `t` that stand for a generic time t; `'pchip'` and `'extrap'` are the two above discussed options for this particular Matlab function (for more information go to Appendix I). The semicolon at the end of the `PM` function is used in Matlab programming to suppress the output of a particular code line, so that if someone runs the code through a script,

[1] See Fritsch and Carlson (1980) for more information on monotonic piecewise cubic interpolation.

the results of the specific code line will not be reported in the command window, but are saved as matrices or objects (as for anonymous functions).

Our focus on this particular expression within the code is mainly due to the recurring presence of the above statement in many other functions, as someone can see in Appendix I; indeed, the above anonymous function has to be frequently rebuild in any function involving discount factors.

The bootstrapping procedure highlighted in the present paragraph is fast and precise; however, it has one main drawback: the shape of the instantaneous forward. As we will see in Chapter 4, Paragraph 2, the "market observed" instantaneous forwards are required to calculate the deterministic shift, $\varphi(t)$, for adjusting the G2++ model to the currently market observed zero-coupon curve. The shift is defined as:

$$\varphi(t) = f^M(0,t) + \frac{\sigma^2}{2a^2}(1-e^{-at})^2 + \frac{\eta^2}{2b^2}(1-e^{-bt})^2 + \rho\frac{\sigma\eta}{ab}(1-e^{-at})(1-e^{-bt}).$$

where the instantaneous forward is given by

$$f^M(0,t) = -\frac{\partial ln P^M(0,t)}{\partial t},$$

where $P^M(0,t)$ is the zero-coupon value at time t currently observed from the market. Practically speaking, the instantaneous forward rate depends on the value resulting from `PM(t)` in Matlab, which is calculated from the interpolation of the discount factor curve bootstrapped from the market data. The real problem is how to calculate the derivative of an interpolated function. One approach is to calculate the derivative numerically, but the resulting instantaneous forward curve rate will be non-continuous; another solution is to use Matlab for computing the derivative using the parameters implied by the interpolated `PM(t)` function, reaching a more continuous and pleasantly shaped curve for the instantaneous forward.

A further approach avoids the calculation of the deterministic shift and simulate the discount factors directly, since the value of the integral of the deterministic function required for calculating the zero-coupon bond in the G2++ model is known and it depends only on the value of `PM(t)`. Nevertheless, this solution is not practicable for the Monte Carlo simulation if the short interest rate has to be plotted. Thus, we propose an alternative methodology for building the zero-coupon bonds curve from market data that will automatically generate also the instantaneous forward function.

In Fig. 2, both the interest rate curve and the discount factor curve resulting from Matlab are reported and, as one can clearly see, both these curves are continuous and include all the market data points. Moreover, the error in absolute number between the discount factors from Matlab

and the ones given by the Bloomberg proprietary algorithm are represented in Fig. 3. We can notice by the errors in absolute number, that the `DF_Bootstrapping` function is able to reproduce the discount factor curve with small errors, despite some outliers observations especially for 30 years or above maturities. However, this does not pose any issues for our calibration process, given that we are interested in points below 30 years, since the European swaption with the longest combination of tenor and maturity pays its last cash flow at the end of the 30th year.

On the other hand, in Tab. 1 the results from `DF_Bootstrapping` function are presented both in term of the discount factors and the of zero-coupon bonds yields. The market is in a peculiar situation characterized by extremely low interest rates, thus implying low values for the strike prices of at-the-money European swaptions and at-the-money caps (or floors) currently quoted by the market.

Fig. 2: Interest Rate Curve and Discounting Curve on the 18th August 2014

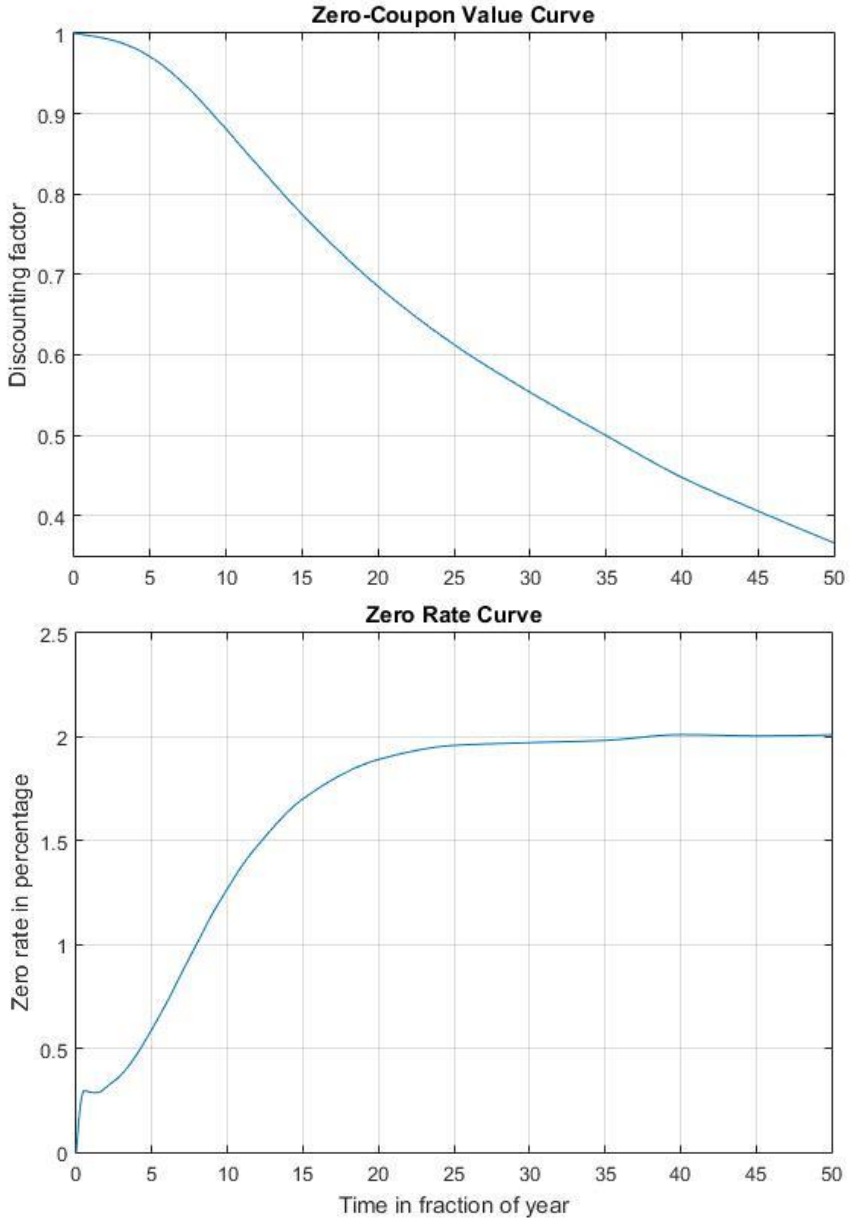
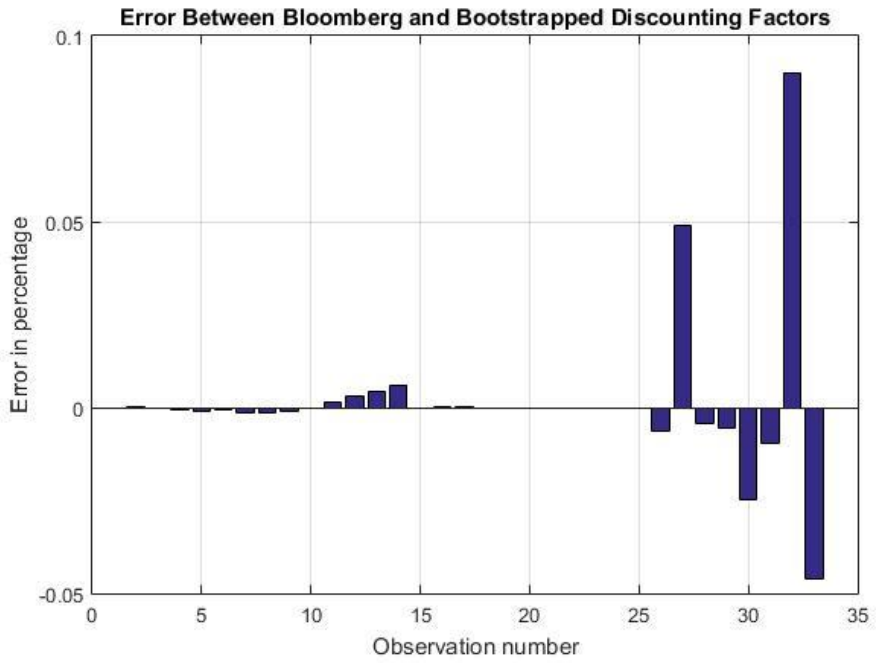


Fig. 3: Error Between Bloomberg Discount Factors and `DF_Bootstrapping` Ones



2.4 Calibrating the Svensson Model

Another commonly used approach for building the term-structure of interest rates is to fit some parametric form for the interest rate. The parametric form guarantees the continuity of the zero-curve and the analytical tractability of the instantaneous forward rate. One of the most important model has been proposed by Nielson and Siegel (1987). In this model the spot interest rate at time t takes the following form:

$$r(t) = \alpha_1 + \beta(\alpha_2 + \alpha_3)\frac{1 - e^{-\frac{t}{\beta}}}{t} - \alpha_3 e^{-\frac{t}{\beta}},$$

with:

$$\alpha_1 > 0,$$

$$\alpha_1 + \alpha_2 > 0,$$

$$\beta > 0.$$

The first parameter, α_1, represents the long-run yield level, as it does not depend on the time, thus it is assumed higher than 0, since a negative long run yield does not have any sense, logically speaking. The other parameters in the function generate a hump in the spot rate curve, in particular β weights the impact of α_2 and α_3, thus effecting the position of the hump.

Tab. 1: `DF_Bootstrapping` **Main Results**

Maturity Date	Instrument Type	Zero Rate	Discount Factor
20 February 2015	Deposit	0.2968%	0.998468
20 March 2015	Forward Rate Agreement	0.2954%	0.998246
20 April 2015	Forward Rate Agreement	0.2935%	0.998004
20 May 2015	Forward Rate Agreement	0.2921%	0.997771
22 June 2015	Forward Rate Agreement	0.2907%	0.997516
20 July 2015	Forward Rate Agreement	0.2895%	0.997302
20 August 2015	Forward Rate Agreement	0.2885%	0.997064
21 September 2015	Forward Rate Agreement	0.2875%	0.996818
20 October 2015	Forward Rate Agreement	0.2870%	0.996593
20 November 2015	Forward Rate Agreement	0.2868%	0.996350
21 December 2015	Forward Rate Agreement	0.2869%	0.996103
20 January 2016	Forward Rate Agreement	0.2874%	0.995858
22 February 2016	Forward Rate Agreement	0.2883%	0.995581
22 August 2016	Forward Rate Agreement	0.3155%	0.993674
21 August 2017	Interest Rate Swap	0.3753%	0.998775
20 August 2018	Interest Rate Swap	0.4718%	0.981279
20 August 2019	Interest Rate Swap	0.5924%	0.970782
20 August 2020	Interest Rate Swap	0.7236%	0.957476
20 August 2021	Interest Rate Swap	0.8676%	0.941029
22 August 2022	Interest Rate Swap	1.0087%	0.922368
21 August 2023	Interest Rate Swap	1.1481%	0.901743
20 August 2024	Interest Rate Swap	1.2692%	0.880742
20 August 2025	Interest Rate Swap	1.3834%	0.858776
20 August 2026	Interest Rate Swap	1.4770%	0.837513
20 August 2029	Interest Rate Swap	1.7015%	0.774671
21 August 2034	Interest Rate Swap	1.8911%	0.684967
22 August 2039	Interest Rate Swap	1.9591%	0.612631
22 August 2044	Interest Rate Swap	1.9721%	0.553305
20 August 2049	Interest Rate Swap	1.9830%	0.499500
20 August 2054	Interest Rate Swap	2.0108%	0.447349
20 August 2059	Interest Rate Swap	2.0051%	0.405598

However, the curve usually observed in the market is often characterized by two humps; thus, to better fit the interest rates observed in the market, Svensson (1994) expanded the Nelson-Siegel model by adding an additional term to the above equation

$$r(t) = \alpha_1 + \beta_1(\alpha_2 + \alpha_3)\frac{1 - e^{-\frac{t}{\beta_1}}}{t} - \alpha_3 e^{-\frac{t}{\beta_1}} + \alpha_4\beta_2\frac{1 - e^{-\frac{t}{\beta_2}}}{t} - \alpha_4 e^{-\frac{t}{\beta_2}},$$

with the additional bound $\beta_2 > 0$. The additional terms in the equation model a second hump in the spot rate curve.

The model is fitted by minimizing the sum of the squared error between model and market discount factors, which are calculated accordingly to the equations explained in the previous paragraph. Indeed, from the spot equation we retrieve the discount factors equation as

$$d(t) = exp\left[-\alpha_1 t - \beta_1(\alpha_2 + \alpha_3)\left(1 - e^{-\frac{t}{\beta_1}}\right) + \alpha_3 t e^{-\frac{t}{\beta_1}} - \alpha_4\beta_2\left(1 - e^{-\frac{t}{\beta_2}}\right) + \alpha_4 t e^{-\frac{t}{\beta_2}}\right].$$

The above expression is derived from the definition of the zero-rate in chapter 1, paragraph 3,

$$P(t,T) = e^{-R(t,T)\tau(t,T)} \xrightarrow{yields} d(t) = e^{-r(t)t}.$$

In Tab. 2, we report the model parameters resulting from a genetic[2] minimization, followed by a local minimization to refine the resulting parameters (for more information on `ga` and `fminsearch`, the non-constrained version of `fmincon` and `lsqnonlin`, see Chapter 4, Paragraph 3).

Tab. 2: Svensson Parameters for the 18th August 2014

α_1	α_2	α_3	α_4	β_1	β_2
0.020328	-0.014832	-2.876787	2.862163	4.005463	4.042529

Once the model is calibrated, we have a parametric form for the discounting curve, the spot curve and, more importantly, the instantaneous forward rate curve. By multiplying for t and taking the derivative of $r(t)$, we are able to find the instantaneous forward rate expression

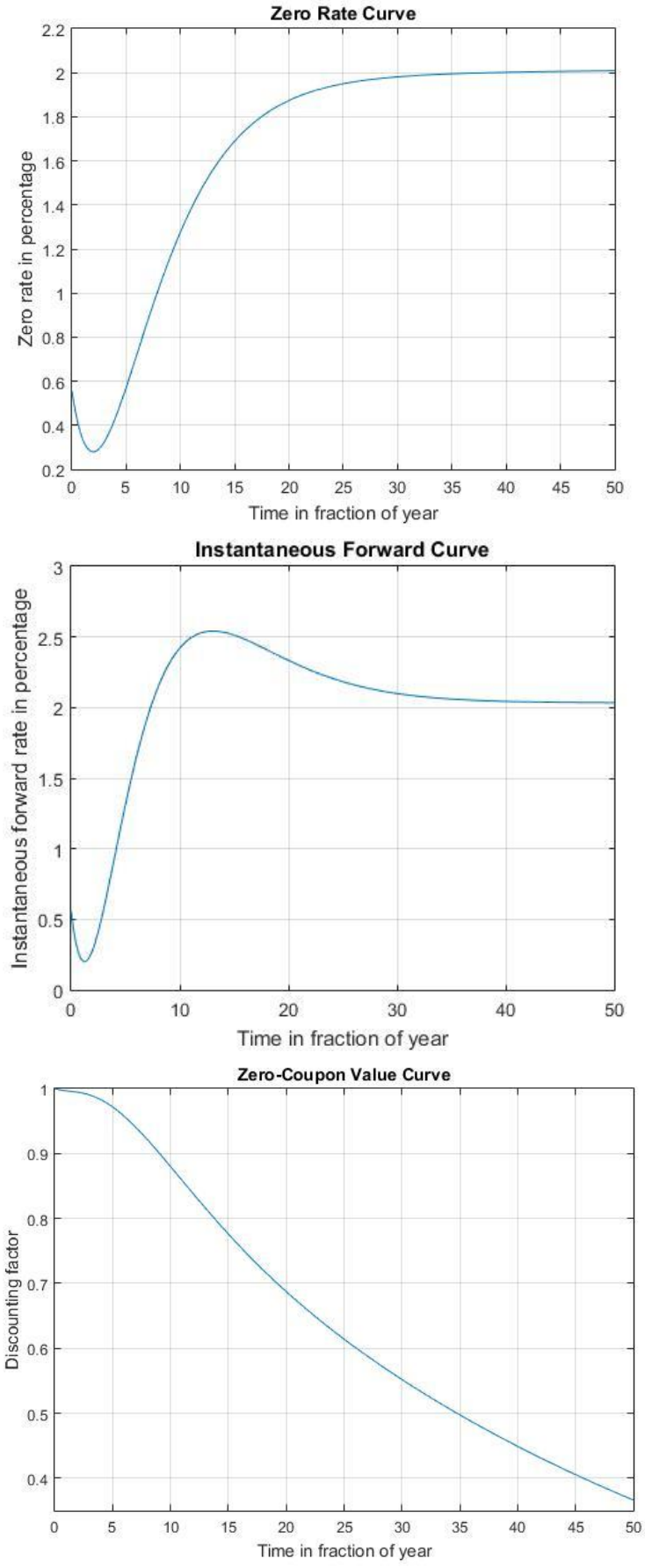
$$f(t) = \alpha_1 + \alpha_2 e^{-\frac{t}{\beta_1}} + \alpha_3\frac{t}{\beta_1}e^{-\frac{t}{\beta_1}} + \alpha_4\frac{t}{\beta_2}e^{-\frac{t}{\beta_2}},$$

thus, we can also implement a continuous $\varphi(t)$ function for the deterministic shift in the G2++ model. The above procedure is implemented in Matlab by the `DF_Fitting` function (see Appendix I). In Fig.

[2] See Sivanandam and Deepa (2008) for a complete introduction to genetic algorithm minimization.

4, the spot rate, the instantaneous forward rate and the discount factors resulting from the calibrated Svensson model are represented.

Fig. 4: Interest Rate, Instantaneous Forward and Zero-Coupon Bond by Svensson Model

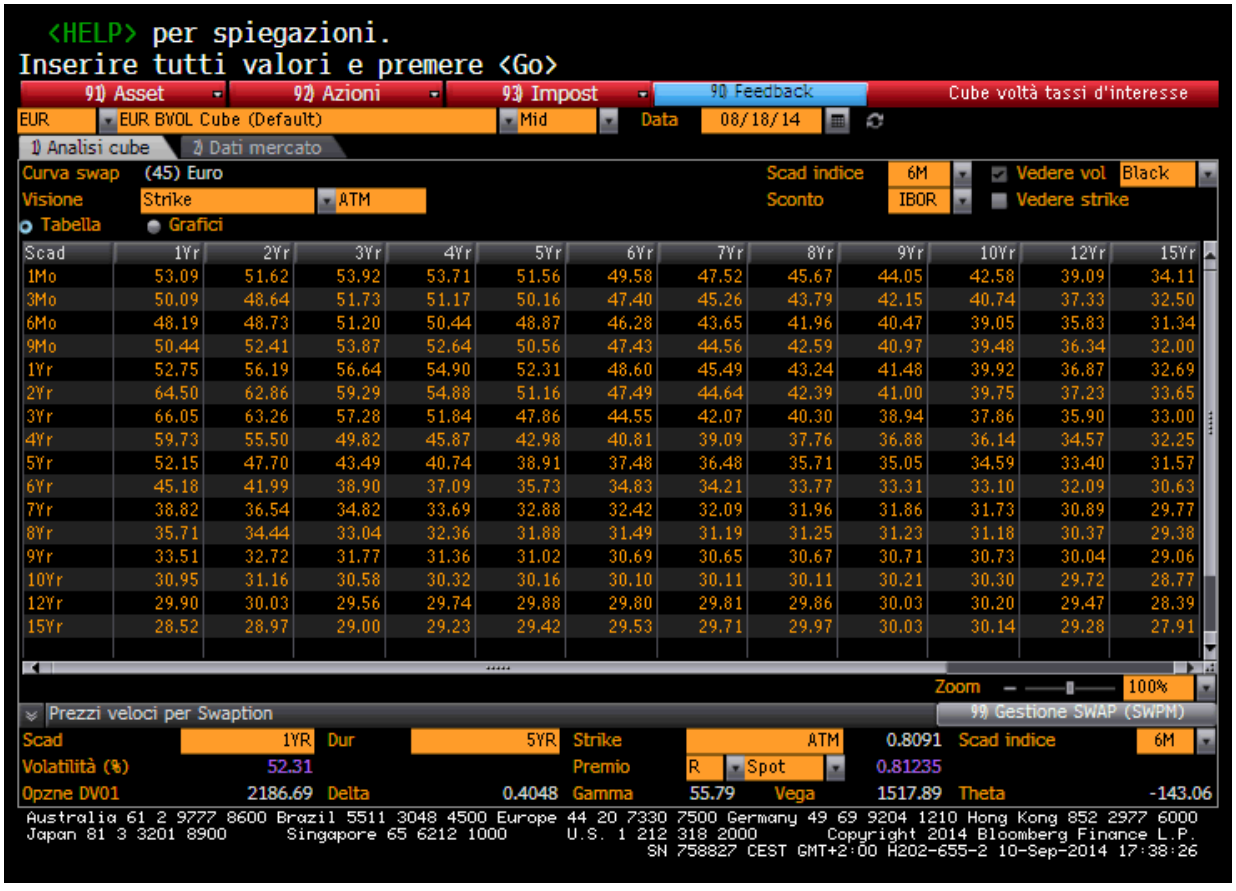


2.5 The Volatility Surface

The data that will be used to calibrate the model are also taken from Bloomberg VCUB function. VCUB, in the screen reported in Fig. 5, contains the implied volatilities for At-The-Money European swaptions, which are represented as Black implied volatilities in a matrix. The matrix is organized accordingly to the expiry (which is equivalent to the maturity of the underling interest rate swap, sometimes called also tenor) and the maturity of the swaption. Thus, each data point in the swaptions volatility matrix represents a different combination of tenor and maturity.

Inside the VCUB function, Bloomberg bootstraps the volatility for each instrument starting from different contributors and using different methodologies in order to take into consideration both the skew in caplet volatilities and the user pre-specified interest rate curve. The default settings of VCUB takes as input the Euro 45 discounting curve generated by SWDF in order to calculate options strikes and implied volatilities. Thus, if we set the discounting to IBOR (i.e. discount future cash flows using the InterBank Offered Rate, which for the euro area is the Euribor), we are able to use the data generated from our bootstrapped curve and from Bloomberg VCUB for calibrating the G2++ model.

Fig. 5: ATM European Swaptions Volatility Matrix from VCUB (source: Bloomberg)



The figure in the previous page contains the default Bloomberg volatility cube for the Euro area, using BVOL as the main contributors for the data, which are composed of mid quotations on the 18th August 2014. These options are set in the drop down menu in the top part of the screen; however, there are also other important fields that the user has to fill before refreshing the Bloomberg function (i.e. entering the GO command). Below the swap curve information on the upper section of the screen, two orange cells are used to select the type of surface, in this case Strike stands for European swaptions and ATM signals that we are seeing the At-The-Money surface, since it is possible to have data also for Out-of-The-Money (OTM) swaptions. Nevertheless, given the inability of the two-additive-factor Gaussian model to capture the skew implied in OTM options, adding OTM contracts to the calibration algorithm would not improve the efficiency of the G2++ model.

It is also crucial to set the volatility to Black in the cell in the northeast side of the first Bloomberg tab (i.e. data analysis), otherwise the swaptions data will be presented as normal volatilities implied by the Bachelier (1900 and 1912) model. Near the volatility option, it is possible to change the swaptions underling IRS tenor between 3 or 6 months. Excluding swaptions with maturity less than one year, that have 3 months tenor IRSs as underling, all the other swaptions a have 6 months tenor IRSs as underling contracts; hence, we have selected 6M as the underling IRS tenor. Once the tenor has been chosen, Bloomberg compute the equivalent volatility also for the contracts with different underlying Libor tenors, giving as a result a homogeneous volatility surface.

By modifying the input swap curve options in order to reflect the same data points used to build our discount factor curve and focusing only on combinations of tenor and maturity between 1 year and 10 years, we obtain the ATM European swaptions volatility matrix shown in Tab.3.

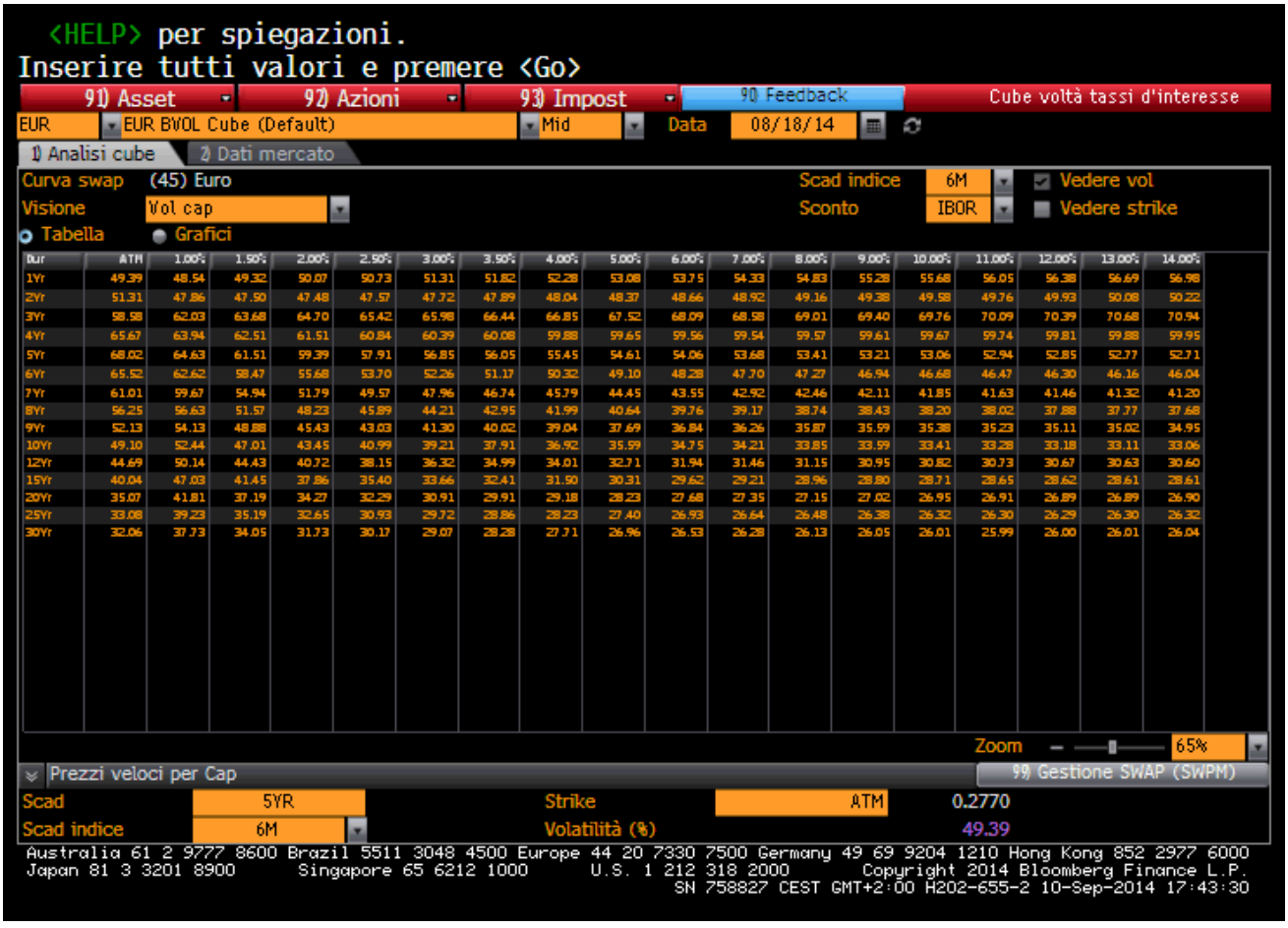
Tab. 3: ATM European Swaptions Volatilities from VCUB on the 18th August

	Tenor										
Maturity	**Year**	**1**	**2**	**2**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
	1	54.65	56.19	56.64	54.90	52.31	48.60	45.49	43.24	41.48	39.92
	2	62.82	62.86	59.29	54.88	51.16	47.49	44.64	42.39	41.00	39.75
	3	66.05	63.26	57.28	51.84	47.86	44.55	42.07	40.31	38.94	37.86
	4	59.73	55.50	49.82	45.88	42.98	40.81	39.09	37.76	36.88	36.14
	5	52.15	47.70	43.49	40.74	38.91	37.48	36.48	35.71	35.05	34.59
	6	45.18	41.99	38.90	37.09	35.74	34.86	34.18	33.74	33.35	33.09
	7	38.82	36.54	34.83	33.69	32.88	32.43	32.09	31.96	31.86	31.73
	8	35.71	34.44	33.09	32.40	31.81	31.49	31.25	31.23	31.21	31.16
	9	33.51	32.72	31.86	31.33	30.96	30.74	30.67	30.65	30.69	30.72
	10	30.95	31.16	30.58	30.32	30.16	30.10	30.11	30.11	30.21	30.30

Indeed, for calibrating and testing our model it is sufficient to retrieve maturities and tenors from 1 year until 10 years in both dimensions. As a matter of facts, not all the volatilities are evaluated at the same instant, given that some instruments are more frequently traded than other. This leads to unavoidable mistakes during the calibration of the model, which assumes that all the market data are updated at the same instant; thus, we will focus mainly on these contracts since they are the most liquid in the VCUB matrix.

Implied volatilities for caps are also available through VCUB if we switch the surface settings to "Vol cap", as depicted in Fig. 6. The page now shows the caps as a combination between a set of maturities, in rows, and different strikes, one for each column, starting from ATM caps on the left. The user selects the strikes by changing the surface settings under the Bloomberg menu "actions". Once we have selected the right surface options, it is possible to retrieve data for both ATM and OTM caps by using the discount factor curve that we have seen in the previous paragraph. Beware that, to maintain consistency with the swaptions data, the discount factor curve has been set to IBOR rate and the underling caplet tenor has been chosen equal to 6 months. As for European swaptions, caps contracts with maturity less or equal to 2 years are based on 3 caplets; thus Bloomberg adapt the entries for these two maturity in order to reflect the change from 3 to 6 months underling caplets and Euribor rate.

Fig. 6: Cap Volatility Matrix from VCUB (source: Bloomberg)



It is important to notice that caps volatilities are reported as flat volatilities, meanings that the volatility reported in the screen refers to a unique value that inserted in each underlying caplet will return the market price. This may lead to some issues while trying to calibrate the model to both surfaces; nonetheless, this particular procedure will not be adopted in this thesis given its poor calibration results, as stated also by Brigo and Mercurio (2006 page. 169).

As a recap to the present paragraph, the below figures represent various volatilities surfaces taken from Bloomberg on the 18th August 2014. In Fig. 7, the ATM European swaptions volatility surface is shown, it is possible to notice the differences between the volatility surface observed on the 13th February 2001 used by Brigo and Mercurio (2007 page. 168) to calibrate the G2++ model and the after crisis level of the swaptions volatilities. On one hand, Fig. 8 reports the ATM caps curve given by Bloomberg; on the other hand, Fig. 9 shows the OTM caps surface, starting from 0.20% strikes until 3.00% with 0.20% intervals. While Tab. 4, report the ATM caps volatility data always taken from VCUB.

Tab. 4: ATM Caps Volatilities from VCUB on the 18th August 2014

Maturity	1	2	3	4	5	6	7	8	9	10	12
Volatility	50.97	55.38	58.73	65.60	67.81	65.32	60.85	56.14	52.04	49.04	44.65

Maturity	15	20	25	30
Volatility	40.02	35.06	33.07	32.05

Fig. 7: ATM European Swaption Volatility Surface from VCUB on the 18th August 2014

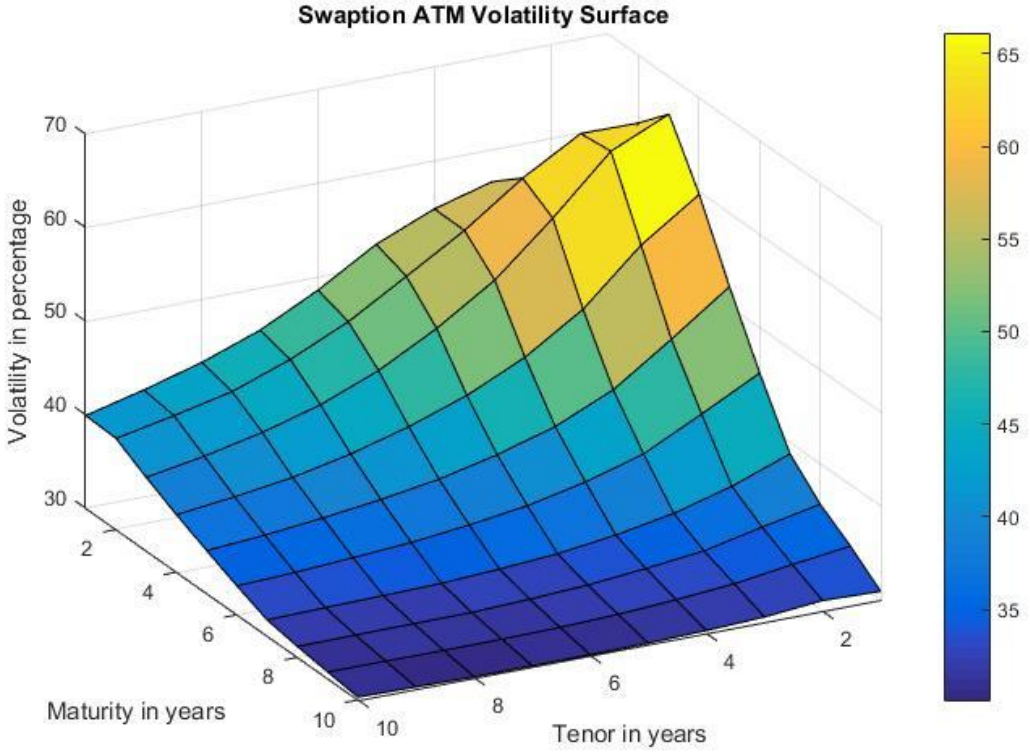


Fig. 8: ATM Cap Volatility Curve from VCUB on the 18th August 2014

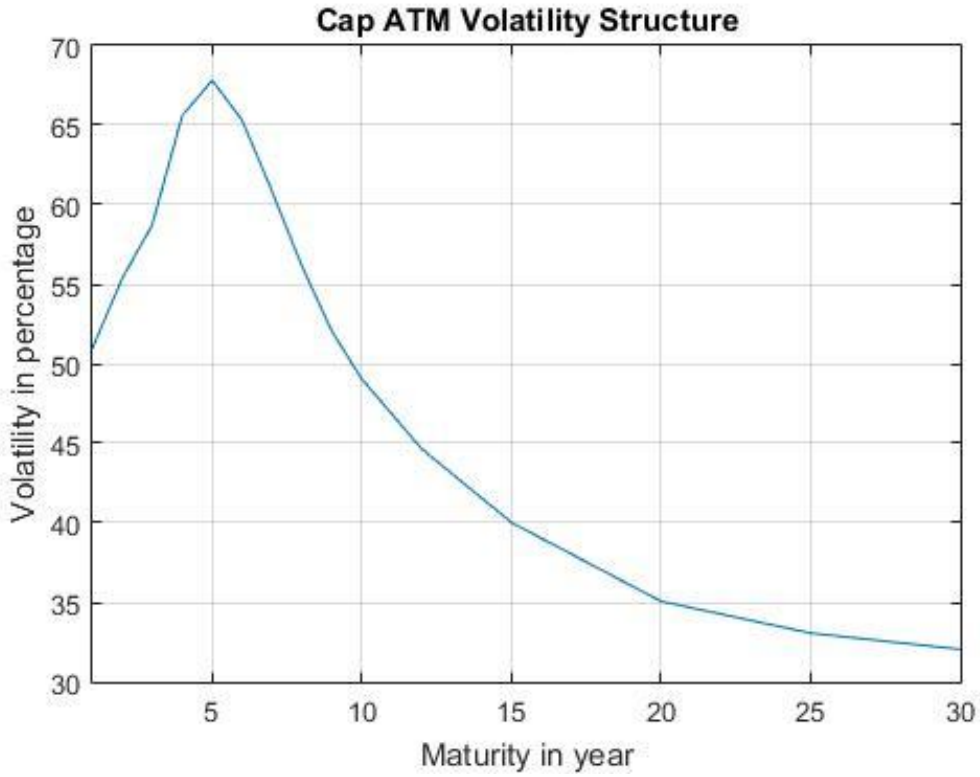
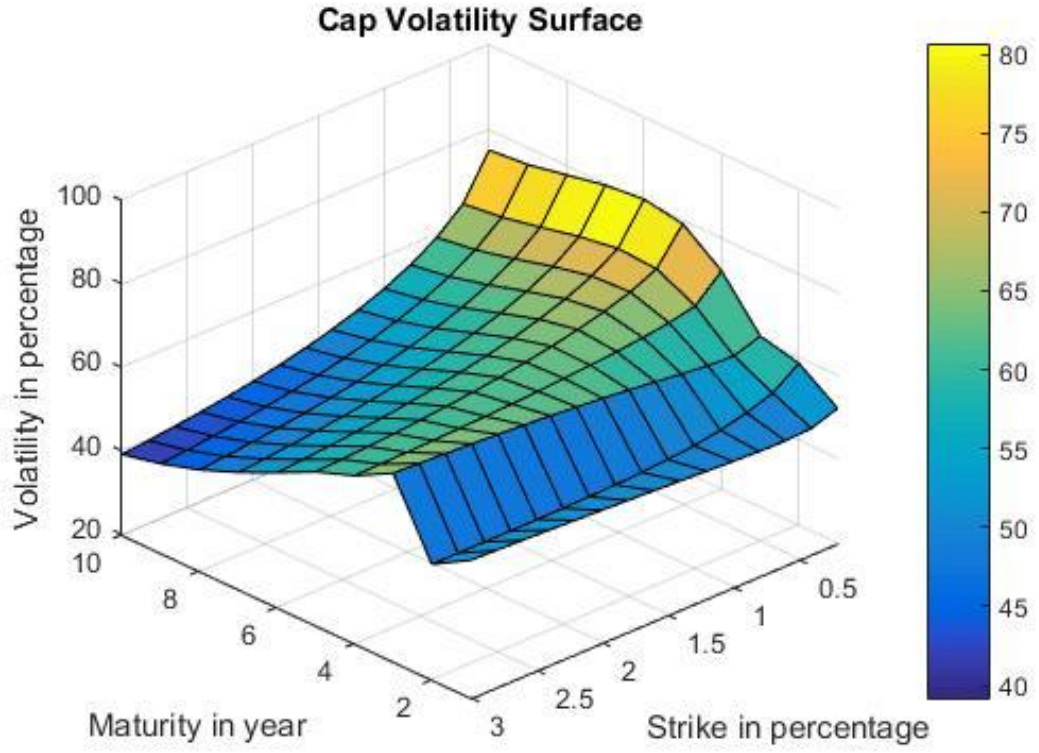


Fig. 9: OTM Cap Volatility Surface from VCUB on the 18th August 2014



3rd Chapter

The Two Factor Additive Gaussian Interest Rate Model

3.1 An Introduction to the G2++ Model

The Black model analytical prices for interest rate options, presented in the second chapter, assume that the interest rate probability distribution is lognormal. This hypothesis is useful when dealing with vanilla contracts, but if other more complex contracts, such as American-style options, or other path dependent derivatives have to be priced, it is necessary to use models that are able to describe the evolution of interest rate through times.

To solve this issue and to overcome the limitations of the Black framework, term structure models were used to describe the evolution through times of the entire zero-coupon curve.

Each term structure model depends on the assumptions made on the short rate r and on the existence of a risk-neutral probability measure. In this hypothetic world an individual earns $r(t)\Delta t$ over a short time period from t to $t + \Delta t$, with $\Delta t \to 0$.

Rendleman and Bartter (1980) proposed an initial risk-neutral process for the short rate r based on the classical geometric Brownian motion used also for describing stock prices

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

where μ and σ are two constants, and $W(t)$ is a Weiner process. However, it was soon noticed that, unlike stock prices, interest rates are characterized by a phenomenon known as mean reversion, since interest rates upward (or downward) movements tend to be followed by opposite sign changes that push the interest rate back around a set of specific values (i.e. the reversion level). Therefore, a well-designed short rate model has to take into account the mean reversion frequently observed in money market rates.

Vasicek (1977) modelled the dynamic of the short rate r under the risk-neutral measure as

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

where a, b and σ are constants, and $W(t)$ is a Weiner process. In this case the short rate is mean reverting; indeed, a represents the strength of the mean reversion, while b is the long term mean

of the process (a.k.a. the reversion level). The second term in the right hand of the dynamic equation represents a normally distributed random shock. Another desirable feature of the model is its ability to generate different shapes for the zero-coupon yield curve.

Another important equilibrium model is the Cox, Ingersoll and Ross (CIRR) model, developed in 1985. Despite all the other previous term structure models, in which the interest rate could assume negative values, the CIRR model generates interest rates that are always non-negative.

All the models presented until now are also called equilibrium models and, despite their theoretical soundness, they are not able to fit today observed term structures. Thereof, when adapted to real market data, most derivatives traders will find these models useless since, as stated by Hull and White (2009 page. 678), "a 1% error in the zero-coupon bond price may lead to a 25% error in an option price".

To ensure a perfect fit to the present term structure, no-arbitrage models were designed to take the current term structure not as an output, like equilibrium models, but as an input of the model. Without focusing on the mathematical proofs, it is possible to transform an equilibrium model into a no-arbitrage one by adding a deterministic function of time in the drift of the short rate, since this one time dependent parameter is able to capture information on the market observed zero-coupon curve.

Starting from the Vasicek model, Hull and White (1990) proposed the following specification for the short rate dynamics:

$$dr(t) = [\theta(t) - ar(t)]dt + \sigma dW(t).$$

If we rewrite the equation, we arrive at

$$dr(t) = a\left[\frac{\theta(t)}{a} - r(t)\right]dt + \sigma dW(t),$$

now we can clearly see how the b term in the Vasicek model has been replaced by the ratio between $\theta(t)$ and a. The $\theta(t)$ function is calculated from the present term structure to ensure a perfect match between model and market zero-coupon bond prices, and this is possible if and only if:

$$\theta(t) = \frac{\partial f(0,t)}{\partial t} + af(0,t) + \frac{\sigma^2}{a}(1 - e^{-2at}),$$

where $f(0,t)$ is the instantaneous forward rate obtained from the initial term structure observed in the market.

However, despite numerous possible patterns for the interest rate that a one-factor no-arbitrage model can generate, in order to provide even more alternatives, a multi-factor model has to be

considered. From the model proposed initially by Brennan and Schwartz (1979 and 1982) and their previous one-factor model, Hull and White (2011) were able to implement a two-factor interest rate model by using trees. Indeed, by adding a second stochastic process in the drift term, they created a model that was able to generate a richer set of interest rate term structures and volatilities structures. The short rate dynamic is specified by the following equations:

$$dr(t) = a[\theta(t) + y(t) - r(t)]dt + \sigma dW_1(t),$$

with

$$dy(t) = -by(t)dt + \eta dW_2(t).$$

The advantages of this two-factor model are numerous, but, unfortunately, the model is not analytically tractable and has to be implemented by using trees; thus, the calibration procedure to the market data will results computationally intensive and will require a sufficiently high amount of time. Nevertheless, Brigo and Mercurio (2006 page. 137-175) proposed a more tractable version of the Hull-White two-factor model in their book "Interest Rate Models – Theory and Practice", namely the G2++ model.

The present thesis focus its attention on the calibration of this model through the analytical formulas for caps (floors) and European swaptions and the subsequent Monte Carlo simulation of the term structure; hence, in what follows, we will discuss extensively the G2++ model.

The G2++ is a model initially proposed by Brigo and Mercurio as a more easily implementable Hull-White two-factor model. In this particular model the short-rate process, $r(t)$, is composed by the sum of two correlated normally distributed factors, $x(t)$ and $y(t)$, and of a deterministic function chosen to exactly fit the initial term structure of discount factors, $\varphi(t)$, leading to

$$r(t) = x(t) + y(t) + \varphi(t).$$

The main advantages of the G2++ model over others two-factor models (such as the CIR++) are:

- the analytical tractability of the model, since it offers analytical formulas for zero-coupon bonds and for other basic European vanilla instruments (among which caps, floors and European swaptions);
- the possibility of allowing some decorrelation between rates with different maturities; such opportunity is not feasible for the corresponding One-Factor Gaussian Interest Rate model, also known as the Hull-White One-Factor model.

These two points are the main reason why, even if the G2++ model allows the existence of negative rates, it is preferred to other short-rate models given its unique ability to capture shocks

at time t in the interest rate curve that are not equally transmitted throughout all maturities. Indeed, in various empirical researches, this is a commonly observed feature of interest rate curves and this particular characteristic of the model is critical in generating scenarios through Monte Carlo simulation for pricing some exotic or complex interest rate derivatives.
Always through this peculiar feature, the model is also able to generate instantaneous forward rates volatility curves with humped shapes; this behaviour, since it is also commonly observed in the market, is necessary in order to have a reasonable fit to market data on both the discount factor curve and the volatility of instruments used to calibrate the model.

3.2 The Short-Rate Dynamic and the Zero-Coupon Bond

As already noted before, the instantaneous-short-rate process under the risk-adjusted measure Q is equal to

$$r(t) = x(t) + y(t) + \varphi(t), \quad r(0) = r_0,$$

where the two processes $x(t)$ and $y(t)$ follow the two stochastic differential equations

$$dx(t) = -ax(t)dt + \sigma dW_1(t), \quad x(0) = 0,$$

$$dy(t) = -by(t)dt + \eta dW_2(t), \quad y(0) = 0,$$

the two terms W_1 and W_2 refer to bi-dimensional Brownian motions defined by an instantaneous correlation equals to

$$dW_1(t)\, dW_2(t) = \rho dt,$$

with r_0, a, b, σ, η positive constants, the instantaneous correlation is contained in the following interval for definition $-1 \leq \rho \leq 1$ and the starting value of the deterministic function phi is equal to the currently observed short interest rate $\varphi(0) = r_0$.
By simple integration of the interest rate dynamics, we can retrieve the following relation

$$r(t) = x(s)e^{-a(t-s)} + y(s)e^{-b(t-s)} + \sigma \int_s^t e^{-a(t-u)}\, dW_1(u) + \eta \int_s^t e^{-b(t-u)} dW_2(u) + \varphi(t)$$

This expression leads to the normal distribution of the process. Moreover, under the probability measure Q and conditional on the filtration $\mathcal{F}_s$ (where for filtration we mean all the available information at time s), we can also find the value for both the expected value and the variance of the interest rate

$$E^Q\{r(t)|\mathcal{F}_s\} = x(s)e^{-a(t-s)} + y(s)e^{-b(t-s)} + \varphi(t)$$

$$Var^Q\{r(t)|\mathcal{F}_s\} = \frac{\sigma^2}{2a}\left[1 - e^{-2a(t-s)}\right] + \frac{\eta^2}{2b}\left[1 - e^{-2b(t-s)}\right] + 2\rho\frac{\sigma\eta}{a+b}\left[1 - e^{-(a+b)(t-s)}\right].$$

These relations are feasible due to the martingality property of the above integrals and to the basic property of integrals calculated at their starting points, which are always equal to zero.
It is important to notice that the dynamics of the two process can be also rewritten in terms of two independent Brownian motions thanks to the Cholesky decomposition, leading to the following two processes

$$dx(t) = -ax(t)dt + \sigma d\widetilde{W}_1(t),$$

$$dy(t) = -by(t)dt + \eta\rho d\widetilde{W}_1(t) + \eta\sqrt{1-\rho^2}d\widetilde{W}_2(t),$$

by integrating the above expression, as done in the previous case, the following definition of the interest rate at a generic time t is obtained, with $t > s$

$$r(t) = x(s)e^{-a(t-s)} + y(s)e^{-b(t-s)} + \sigma\int_s^t e^{-a(t-u)}\,d\widetilde{W}_1(u) + \eta\rho\int_s^t e^{-b(t-u)}d\widetilde{W}_1(t)$$
$$+ \eta\sqrt{1-\rho^2}\int_s^t e^{-b(t-u)}d\widetilde{W}_2(u) + \varphi(t).$$

The deterministic function $\varphi(t)$ is always built in order to exactly fit the observed zero-coupon rates in the market, as already stated before.
To obtain the price of a zero-coupon bond with maturity equal to T, which is the starting point for many analytical pricing formulas in this model, we need to find the value of this expected value:

$$P(t,T) = E^Q\left\{e^{-\int_t^T r_s ds}|F_t\right\}.$$

From the above expression it appears necessary to identify the first two moments of the below quantity, as it represent the stochastic part of the zero-coupon bond

$$I(t,T) = \int_t^T [x(u) + y(u)]du.$$

This particular random variable is normally distributed and, leaving the proofs to Brigo and Mercurio (2006 page. 169-171), it is possible to identify its mean $M(t,T)$ and its variance $V(t,T)$ conditional on the filtration $\mathcal{F}_t$

$$M(t,T) = \frac{1-e^{-a(T-t)}}{a}x(t) + \frac{1-e^{-b(T-t)}}{b}y(t),$$

$$\begin{aligned} V(t,T) = \frac{\sigma^2}{a^2}\left[T-t+\frac{2}{a}e^{-a(T-t)} - \frac{1}{2a}e^{-2a(T-t)} - \frac{3}{2a}\right] \\ + \frac{\eta^2}{b^2}\left[T-t+\frac{2}{b}e^{-b(T-t)} - \frac{1}{2b}e^{-2b(T-t)} - \frac{3}{2b}\right] \\ + 2\rho\frac{\sigma\eta}{ab}\left[T-t+\frac{e^{-a(T-t)}-1}{a} + \frac{e^{-b(T-t)}-1}{b} - \frac{e^{-(a+b)(T-t)}-1}{a+b}\right]. \end{aligned}$$

We can now give a first incomplete definition of the price at time t of a zero-coupon bond maturing at time T:

$$P(t,T) = exp\left\{-\int_t^T \varphi(u)du - M(t,T) + \frac{1}{2}V(t,T)\right\}.$$

As a matter of facts, once the first two moments of the random variable $I(t,T)$ have been identified, it is easy to show that the zero-coupon bond is nonetheless the expected value of lognormal random variable with a deterministic shift that can be calculated as a simple deterministic integral.

Here it comes the crucial point: since the deterministic function $\varphi(t)$ is chosen in order to perfectly fit the current term structure of discount factors observed in the market, if we identify the current instantaneous forward rate at time 0 with a maturity equal to t as

$$f^M(0,t) = -\frac{\partial lnP^M(0,t)}{\partial t},$$

we can also specify $\varphi(t)$.Indeed, the model is able to fit perfectly the current term structure if and only if, for each maturity t, $\varphi(t)$ satisfies the following equation

$$\varphi(t) = f^M(0,t) + \frac{\sigma^2}{2a^2}(1-e^{-at})^2 + \frac{\eta^2}{2b^2}(1-e^{-bt})^2 + \rho\frac{\sigma\eta}{ab}(1-e^{-at})(1-e^{-bt}).$$

Moreover, we can also avoid the calculation of the instantaneous forward rate given that we are interested in the integral of the above equation

$$exp\left\{-\int_t^T \varphi(u)du\right\} = \frac{P^M(0,T)}{P^M(0,t)}exp\left\{-\frac{1}{2}[V(0,T)-V(0,t)]\right\}.$$

Finally, we can now give a full specification for the bond price in the G2++ model by using the value of the above integral and the first two moments of $I(t,T)$.

Definition 3.1 (Zero-Coupon Bond price in the G2++ model). *The price at time t of a zero-coupon bond with unitary face value and maturing at time T is equal to*

$$P(t,T) = \frac{P^M(0,T)}{P^M(0,t)} exp\left\{-M(t,T) + \frac{1}{2}[V(t,T) - V(0,T) + V(0,t)]\right\}.$$

Now that the main passages for the derivation of the zero-coupon bond price, which is chosen in order to fit exactly the term structure of interest rate observed in the market, have been described, we can move on to the value of a generic European call option on this particular zero-coupon bond. Indeed, after the price of the zero-coupon bond is known, it is possible to find also a series of analytical formulas for pricing other vanilla contracts under the G2++ model.

The price of a zero-coupon bond call may be stated as the expected value of a discounted payoff under the probability measure Q, given the information available at time $t < T$ (i.e. the filtration $\mathcal{F}_t$):

$$ZBC(t,T,S,K) = \mathbb{E}^Q\left\{e^{-\int_t^T r(s)ds}(P(T,S) - K)^+ | \mathcal{F}_t\right\}$$

with T, S and K representing respectively the maturity of the option, the maturity of the underlying unitary bond and the strike of the option.

To price the above claim, a change in the probability measure has to be adopted, switching from the bank accounting measure Q to the Q^T T-forward measure. While under the former probability measure the discounting rate used was set equal to the bank account, $B(t,T)$, under the latter probability space the discounting adopted has to be equal to the value of the zero-coupon bond $P(t,T)$.

By using the Radon-Nikodym derivative, it is possible to redefine the dynamics of the two processes as

$$dx(t) = -\left\{ax(t) + \frac{\sigma^2}{a}\left[1 - e^{-a(T-t)}\right] + \rho\frac{\sigma\eta}{b}\left[1 - e^{-b(T-t)}\right]\right\}dt + \sigma d{W_1}^{Q^T}(t),$$

$$dy(t) = -\left\{by(t) + \frac{\eta^2}{b}\left[1 - e^{-b(T-t)}\right] + \rho\frac{\sigma\eta}{a}\left[1 - e^{-a(T-t)}\right]\right\}dt + \eta d{W_2}^{Q^T}(t).$$

Finally, we can integrate both x and y under the forward measure Q^T leading to

$$x(t) = x(s)e^{-a(t-s)} - M_x^T(s,t) + \sigma\int_s^t e^{-a(t-u)}dW_1^T(u)$$

$$y(t) = y(s)e^{-b(t-s)} - M_y^T(s,t) + \eta \int_s^t e^{-b(t-u)} dW_2^T(u),$$

with

$$M_x^T(s,t) = \left(\frac{\sigma^2}{a^2} + \rho\frac{\sigma\eta}{ab}\right)\left[1 - e^{-a(t-s)}\right] - \frac{\sigma^2}{2a^2}\left[e^{-a(T-t)} - e^{-a(T+t-2s)}\right] - \frac{\rho\sigma\eta}{b(a+b)}\left[e^{-b(T-t)} - e^{-bT-at+(a+b)s}\right],$$

$$M_y^T(s,t) = \left(\frac{\eta^2}{b^2} + \rho\frac{\sigma\eta}{ab}\right)\left[1 - e^{-b(t-s)}\right] - \frac{\eta^2}{2b^2}\left[e^{-b(T-t)} - e^{-b(T+t-2s)}\right] - \frac{\rho\sigma\eta}{a(a+b)}\left[e^{-a(T-t)} - e^{-aT-bt+(a+b)s}\right].$$

As a matter of facts, under the new probability measure Q^T the distribution of $r(t)$ conditional on the filtration $\mathcal{F}_s$, with $s < t$, is normal and is characterized by the following mean and variance

$$E^{Q^T}\{r(t)|\mathcal{F}_s\} = x(s)e^{-a(t-s)} - M_x^T(s,t) + y(s)e^{-b(t-s)} - M_y^T(s,t) + \varphi(t),$$

$$Var^{Q^T}\{r(t)|\mathcal{F}_s\} = \frac{\sigma^2}{2a^2}\left[1 - e^{-2a(t-s)}\right] + \frac{\eta^2}{2b^2}\left[1 - e^{-2b(t-s)}\right] + 2\rho\frac{\sigma\eta}{a+b}\left[1 - e^{-(a+b)(t-s)}\right].$$

We have now covered all the basic tools needed evaluate an European call option on the zero-coupon bond, so we can now give a complete definition of an European call (or put) option on the zero-coupon bond.

Definition 3.2 (Value of a Zero-Coupon Bond Call and Put). *The unitary value at time t of a call option with maturity T and strike K, having as underlying a generic zero-coupon bond with maturity equal to S, is given by*

$$ZBC(t,T,S,K) = P(t,S)\Phi\left(\frac{ln\frac{P(t,S)}{KP(t,T)}}{\Sigma(t,T,S)} + \frac{1}{2}\Sigma(t,T,S)\right) - P(t,T)K\Phi\left(\frac{ln\frac{P(t,S)}{KP(t,T)}}{\Sigma(t,T,S)} - \frac{1}{2}\Sigma(t,T,S)\right),$$

where the square of the standard deviation, $\Sigma(t,T,S)$, can be written as

$$\Sigma(t,T,S)^2 = \frac{\sigma^2}{2a^3}\left[1-e^{-a(S-T)}\right]^2\left[1-e^{-2a(T-t)}\right] + \frac{\eta^2}{2b^3}\left[1-e^{-b(S-T)}\right]^2\left[1-e^{-2b(T-t)}\right] + 2\rho\frac{\sigma\eta}{ab(a+b)}\left[1-e^{-a(S-T)}\right]\left[1-e^{-b(S-T)}\right]\left[1-e^{-(a+b)(T-t)}\right].$$

In the same fashion, we can also define a put option on the zero-coupon bond as:

$$ZBP(t,T,S,K) = -P(t,S)\Phi\left(\frac{ln\frac{P(t,T)}{KP(t,S)}}{\Sigma(t,T,S)} - \frac{1}{2}\Sigma(t,T,S)\right) + P(t,T)K\Phi\left(\frac{ln\frac{P(t,T)}{KP(t,S)}}{\Sigma(t,T,S)} + \frac{1}{2}\Sigma(t,T,S)\right).$$

One of the main reasons why the G2++ model is preferred to other short-term interest rate models is given by its highly analytical tractability; the above formula of a European call on the zero bond is at the basis of many analytical price equations that will be soon presented in this chapter. Nevertheless, as we will soon discover, the market still heavily relies on numerical implementation through Monte Carlo simulations or bi-dimensional trinomial trees, given their power and flexibility for pricing exotic derivatives such as American options and other path dependent option contracts.

3.3 The Pricing of Caplets, Floorlets, Caps and Floors

Once the value of a zero-coupon bond vanilla option has been defined in the G2++ model, it is possible to derive an analytical formula to price other instruments frequently traded in the market for hedging against interest rate fluctuations. As already stated in the first chapter, two of the main contracts used by banks and companies are caps and floors. Nevertheless, before we define the prices for these two contracts, we should see the price of the two fundamentals instruments that compose these two options, respectively, caplets and floorlets.

The payoff at maturity of a general caplet with strike K and notional N is equal to

$$[L(T_1,T_2) - K]^+\tau(T_1,T_2)N$$

where $\tau(T_1, T_2)$ is the year fraction between T_1 and T_2, with $T_2 > T_1$; and $L(T_1, T_2)$ representing the Libor rate always between T_1 and T_2.

Leaving the derivation to Brigo and Mercurio's book (2006 pag. 156-157), we can compare interest rate caplets to put options on a zero-coupon bond: the put option has expiry set at time T_1, while the underling zero-coupon bond matures at time T_2. Given the above statement, it is feasible to arrive at the following formula for pricing caplets at time t under the G2++ model

$$Cpl(t, T_1, T_2, N, X)$$
$$= -N'P(t, T_2)\Phi\left(\frac{\ln\frac{NP(t,T_1)}{N'P(t,T_2)}}{\Sigma(t,T_1,T_2)} - \frac{1}{2}\Sigma(t, T_1, T_2)\right)$$
$$+ P(t, T_1)N\Phi\left(\frac{\ln\frac{NP(t,T_1)}{N'P(t,T_2)}}{\Sigma(t,T_1,T_2)} + \frac{1}{2}\Sigma(t, T_1, T_2)\right),$$

where X is the strike of the caplet, N is the notional value of the contract, $P(t, T)$ is the price of a zero-coupon bond in the G2++ model, as defined in the previous section, and X' and N' are two quantities that depend on the strike, the notional value of the contract and the year fraction τ between T_1 and T_2

$$X' = \frac{1}{1 + X\tau(T_1, T_2)},$$

$$N' = N\big(1 + X\tau(T_1, T_2)\big).$$

Conversely, a floorlet contract is characterized by a payoff at maturity equal to

$$[K - L(T_1, T_2)]^+\tau(T_1, T_2)N.$$

As for caplets, even a floorlet resembles a call option on a zero-coupon bond if we persist with the usual no-arbitrage argument. Indeed, we can arrive to the following formula for calculating the value at time t of a floorlet under the G2++ model:

$$Fll(t,T_1,T_2,N,X)$$

$$= N'P(t,T_2)\Phi\left(\frac{\ln\frac{N'P(t,T_2)}{NP(t,T_1)}}{\Sigma(t,T_1,T_2)}+\frac{1}{2}\Sigma(t,T_1,T_2)\right)$$

$$- P(t,T_1)N\Phi\left(\frac{\ln\frac{N'P(t,T_2)}{NP(t,T_1)}}{\Sigma(t,T_1,T_2)}-\frac{1}{2}\Sigma(t,T_1,T_2)\right).$$

By combining the two above formulations, we can represent the two equations in a more compact way, which looks a lot like the Black formula presented in the first chapter for caplets and floorlets. However, it is important to notice the presence of $N'$and the fact that when referring to the quantity $P(t,T)$, we are now using the zero-coupon bond as defined under the G2++ model.

$$CapletFloorlet(t,T_1,T_2,N,X,\omega) = -\omega N'P(t,T_2)\Phi(\omega d_1) + \omega P(t,T_1)N\Phi(\omega d_2)$$

with $\omega = 1$ ($\omega = -1$) for caplets (floorlets) and

$$d_1 = \frac{\ln\frac{NP(t,T_1)}{N'P(t,T_2)}}{\Sigma(t,T_1,T_2)} - \frac{1}{2}\Sigma(t,T_1,T_2),$$

$$d_2 = d_1 + \Sigma(t,T_1,T_2).$$

After we have obtained the specification of caplets and floorlets options in the model, we can find the price formula also for caps (or floors). Indeed, as briefly explained in the first chapter, the value of a cap (or floor) contract is given by the sum of the prices of all the underlying caplets (or floorlets). By using this definition, the following definition for caps and floors under the G2++ appears intuitive.

Definition 3.3 (Cap-Floor price in the G2++ model). *The arbitrage-free price of a cap (floor) contract with the cap (floor) payment dates and first reset date, T_0, denoted by $\mathcal{T} = \{T_0, T_1, T_2, \dots, T_n\}$ and the corresponding year fractions $\tau = \{\tau_0, \dots, \tau_n\}$, is given by the following formula*

$$Cap(t,\mathcal{T},\tau,N,X)$$

$$= \sum_{i=1}^{n}\left[-N(1+X\tau_i)P(t,T_i)\Phi\left(\frac{\ln\frac{P(t,T_{i-1})}{(1+X\tau_i)P(t,T_i)}}{\Sigma(t,T_{i-1},T_i)} - \frac{1}{2}\Sigma(t,T_{i-1},T_i)\right)\right.$$

$$\left. + P(t,T_{i-1})N\Phi\left(\frac{\ln\frac{P(t,T_{i-1})}{(1+X\tau_i)P(t,T_i)}}{\Sigma(t,T_{i-1},T_i)} + \frac{1}{2}\Sigma(t,T_{i-1},T_i)\right)\right],$$

the price of the corresponding floor is

$$Floor(t,\mathcal{T},\tau,N,X)$$

$$= \sum_{i=1}^{n}\left[N(1+X\tau_i)P(t,T_i)\Phi\left(\frac{\ln\frac{(1+X\tau_i)P(t,T_i)}{P(t,T_{i-1})}}{\Sigma(t,T_{i-1},T_i)} + \frac{1}{2}\Sigma(t,T_{i-1},T_i)\right)\right.$$

$$\left. - P(t,T_{i-1})N\Phi\left(\frac{\ln\frac{(1+X\tau_i)P(t,T_i)}{P(t,T_{i-1})}}{\Sigma(t,T_{i-1},T_i)} - \frac{1}{2}\Sigma(t,T_{i-1},T_i)\right)\right],$$

with X, N, $P(t,T)$ and $\Sigma(t,T,S)$ defined previously for caplet and floorlet.

As for caplet and floorlet, it is advisable to rewrite the previous equations in a more compact way, indeed

$$CapFloor(t,\mathcal{T},\tau,N,X,\omega) = \sum_{i=1}^{n}[-\omega N(1+X\tau_i)P(t,T_i)\Phi(\omega d_1) + \omega P(t,T_{i-1})N\Phi(\omega d_2)]$$

where $\omega = 1$ ($\omega = -1$) for cap (floor) and

$$d_1 = \frac{\ln\frac{P(t,T_{i-1})}{(1+X\tau_i)P(t,T_i)}}{\Sigma(t,T_{i-1},T_i)} - \frac{1}{2}\Sigma(t,T_{i-1},T_i),$$

$$d_2 = d_1 + \Sigma(t,T_{i-1},T_i).$$

In Matlab, the price of a European cap (floor) is calculated using the `CF` function, which simply implement the G2++ closed formula for the price for these instruments. Indeed, the price of this type of contracts is simply the results of an expected value of a normal random variable under a suitable probability measure, in this case the T-forward measure.

Even if the formula for calculating the price of caps and floors is relatively fast and easy to code in Matlab, the calibration of the model to such instruments will lead to a correlation parameter, ρ, equal or close to minus one. This issue, given by the absence of information on the correlation between interest rates with different maturities, will result in a degeneration of the G2++ model to a one-factor model; however, as underlined by Brigo and Mercurio (2006), the resulting process will still be a non-trivial and non-Markovian one.

Thus, in order to obtain a true two-factor model, practitioners prefer to calibrate the model to the European swaptions at-the-money volatility surface, as this procedure will lead to a correlation between the two processes ρ different from minus one (at least before the 2007 crisis).

This is the main reason why it is necessary to give also a brief review of the European swaption price formula for the G2++ model and its implementation in Matlab.

3.4 The Pricing of European Swaptions

Brigo and Mercurio (2006 pag. 158) proposed the following "analytical" formula for a European swaption value at time zero ($t = 0$), referencing to their book for the proofs (pag. 173-175).

Definition 3.4 (European Swaption price in the G2++). *The arbitrage-free price at time* $t = 0$ *of a European Swaption with strike* X *and notional* N *is given by numerically computing the following one-dimensional integral:*

$$ES(0,T,\mathcal{T},N,X,\omega) = N\omega P(0,T)\int_{-\infty}^{+\infty}\frac{e^{-\frac{1}{2}\left(\frac{x-\mu_x}{\sigma_x}\right)}}{\sigma_x\sqrt{2\pi}}\left[\Phi\big(-\omega h_1(x)\big) - \sum_{i=1}^{n}\lambda_i(x)e^{\kappa_i(x)}\Phi\big(-\omega h_2(x)\big)\right]dx,$$

where $\omega = 1$ *(*$\omega = -1$*) for a payer (receiver) swaption,*

$$h_1(x) \coloneqq \frac{\bar{y}-\mu_y}{\sigma_y\sqrt{1-{\rho_{xy}}^2}} - \frac{\rho_{xy}(x-\mu_x)}{\sigma_x\sqrt{1-{\rho_{xy}}^2}}$$

$$h_2(x) \coloneqq h_1(x) + B(b,T,t_i)\sigma_y\sqrt{1-{\rho_{xy}}^2}$$

$$\lambda_i(x) \coloneqq c_i A(T,t_i) e^{-B(a,T.t_i)x}$$

$$\kappa_i(x) \coloneqq -B(b,T,t_i)\left[\mu_y - \frac{1}{2}\left(1-{\rho_{xy}}^2\right){\sigma_y}^2 B(b,T,t_i) + \rho_{xy}\sigma_y \frac{x-\mu_x}{\sigma_x}\right],$$

$\bar{y} = \bar{y}(x)$ is the unique solution of the following equation

$$\sum_{i=1}^{n} c_i A(T,t_i) e^{-B(a,T,t_i)x - B(b,T,t_i)\bar{y}} = 1,$$

and

$$\mu_x \coloneqq -M_x^T(0,T),$$

$$\mu_y \coloneqq -M_y^T(0,T),$$

$$\sigma_x \coloneqq \sigma\sqrt{\frac{1-e^{-2aT}}{2a}},$$

$$\sigma_y \coloneqq \eta\sqrt{\frac{1-e^{-2bT}}{2b}},$$

$$\rho_{xy} \coloneqq \frac{\rho\sigma\eta}{(a+b)\sigma_x\sigma_y}\left[1-e^{-(a+b)T}\right].$$

However, the above equation is not truly an "analytical" formula; as a matter of facts, the above integral does not have clear boundaries and it is necessary to truncate the integration region to calculate the price of the swaption.

For what concerns the Matlab implementation, the function `ES_G2` (see Appendix I) allows the user to switch among tree different methodologies in order to calculate the value of a generic European swaption. The first two methods try to find a solution to the above formula; while the third function adopts an approximation made by Schrager and Pelsser (2006).

In the first two functions, it is necessary to set suitable boundaries to the above integral. However, as one may have already notice, it is important to remark that the integral above is calculated on a random variable that follows a normal distribution (i.e. $X \sim N(\mu_x, {\sigma_x}^2)$); therefore, it is possible to place some boundaries to the integral as a function of the mean and standard deviation of the distribution.

The integration region has been limited to the linear space between

$$[\mu_x - 10\sigma_x, \mu_x + 10\sigma_x],$$

given that for values of the function above and below these boundaries, the changes in swaption prices are negligible. Nevertheless, a better approach may be to let the boundaries change accordingly to each swaption price by using a `while` loop. Unfortunately, if we let the boundaries change using a `while` statement, we will slower the calibration procedure with only small changes in the final value of the swaption model price and also implied volatility, hence we will still rely on a pre-specified fixed interval.

The first two functions differ in the methodology used to approximate the value of the integral in the above swaption formula; indeed, the need to compute the value for $\bar{y}$ limits drastically the possible methodologies that one can implement in Matlab. Our two final choices rely on integration by Gauss-Legendre or by using the trapezoidal rule.

The trapezoidal rule has been used to evaluate the integral in the formula for the European swaption price `ES_G2byTR`. This simple approach consists in dividing the area below the function into different trapezoids and evaluating each trapezoid area, the final value of the integral is given by the sum of each trapezoid area. Thanks to the Matlab built in `trap` function, this type of integration is not difficult to implement; however, it requires the division of the integration region in a reasonable amount of small intervals (i.e. 1000) and, after calculating the value of $\bar{y}$ for each interval point, the results is given as a summation of small trapezoids areas.

Nevertheless, this division of the integration region into different small intervals and the subsequent need to find a solution to the following equation for each point

$$\sum_{i=1}^{n} c_i A(T,t_i) e^{-B(a,T,t_i)x - B(b,T,t_i)\bar{y}} = 1,$$

slower the whole code and may result in errors in the root-finding algorithm adopted. Indeed, the above system does not always have a solution and, as one can clearly imagine, the more equations the function needs to evaluate, the higher the probability that, for some points of integration, the above equation does not have a solution. Indeed, if `fsolve`[3] (the built in Matlab function for solving nonlinear systems of equations) will fail to find a solution for the above equation, the algorithm will stop the calculation of the remaining swaption prices.

With the aim of avoiding this problematic situation that could lead to a general failure in the calibration procedure, it is better to adopt a Gauss-Legendre integration with, at least, 20 points.

[3] For more information on the minimization, see Colemand and Li (1994 and 1996), and More (1977).

This second methodology lead to the same results as the trapezoidal rule; however, it drastically reduce the number of equations to be solved by `fsolve`, leading to a lower probability of failure during the calibration.

As a simple example, suppose that someone is trying to calculate the prices of a 10×10 matrix of Swaptions maturities and tenors using the trapezoidal rule. For calculating one price, it is easy to figure out that, just for retrieving this value, Matlab has to evaluate 1'000 different equations. Thus, if we want to price all the swaptions in the matrix, the number of equations that Matlab has to solve for finding the value for all these swaptions is calculated as 100 times the above equation for 1'000 values of x, leading to 100'000 equations. If someone is using the Gauss-Legendre integration with 20 points, instead, the number of equations to be solved is reduced to 2'000, without significant changes in the final swaption prices.

This is the main reasons behind the Gauss-Legendre integration implemented in Matlab by `ES_G2byGL`; with this integration rule, we are able to reduce the overall computational time and the probability of errors during the calibration of the model to the ATM European swaptions volatility surface. However, beware that reduction does not mean complete elimination, given that for some sets of model parameters $(a, b, \sigma, \eta, \rho)$ the Gauss-Legendre integration may also fail.

Therefore, to let the algorithm run despite the presence of errors in the `fsolve` function we should implement a `try – catch` statement. By using this gimmick, we are able to let the algorithm use the following logic:

- it tries to find a solution to the above equation by using `fsolve`;
- if the equation display a negative `exitflag` (i.e. the equation does not have a solution), substitute the missing price value with `NaN` and let the algorithm moves to the next combination of maturity and tenor.

As one can clearly see, with this little trick the loop cycle for calculating all the swaptions prices will not stop running and will automatically substitute missing values with `NaN`s.

Nevertheless, given the computation time required to calculate European swaptions prices also while adopting the latter methodology, we can use a famous approximation for the price of an ATM European swaption under a multi-factor Gaussian term structure model (of which the G2++ is a particular specification). To be more precise, we are referring to a famous approach firstly highlighted by Schrager and Pelsser in 2006.

Skipping the whole demonstration for space reasons, referring to the original paper by Schrager and Pelsser for the derivation of this formula, since it will require an additional chapter on its own, we can find the value of an ATM European swaption by using the following equations:

$$PS^{Schrager}\left(0,T_\alpha,\mathcal{T},N,X=S_{\alpha\beta}(0),\omega=1\right)=BPV_{\alpha\beta}(0)\frac{\sigma_{\alpha\beta}}{\sqrt{2\pi}}N$$

where

$$\sigma_{\alpha\beta}=\sqrt{\sigma^2\left(C_{\alpha\beta}^{(1)}\right)^2\left[\frac{e^{2aT_\alpha}-1}{2a}\right]+\eta^2\left(C_{\alpha\beta}^{(2)}\right)^2\left[\frac{e^{2bT_\alpha}-1}{2b}\right]+2\sigma\eta\rho C_{\alpha\beta}^{(1)}C_{\alpha\beta}^{(2)}\left[\frac{e^{(a+b)T_\alpha}-1}{a+b}\right]},$$

$$C_{\alpha\beta}^{(1)}=\frac{1}{a}\left[e^{-aT_\alpha}P^{BPV}(0,T_\alpha)-e^{-aT_\beta}P^{BPV}\left(0,T_\beta\right)-S_{\alpha\beta}(0)\sum_{i=\alpha}^{\beta}\tau_i e^{-aT_i}P^{BPV}(0,T_i)\right],$$

$$C_{\alpha\beta}^{(2)}=\frac{1}{b}\left[e^{-bT_\alpha}P^{BPV}(0,T_\alpha)-e^{-bT_\beta}P^{BPV}\left(0,T_\beta\right)-S_{\alpha\beta}(0)\sum_{i=\alpha}^{\beta}\tau_i e^{-bT_i}P^{BPV}(0,T_i)\right],$$

$$P^{BPV}(t,T_i)=\frac{P(t,T_i)}{BPV_{\alpha\beta}(t)},$$

$$BPV_{\alpha\beta}(t)=\sum_{i=\alpha+1}^{\beta}\tau_i P(0,t_i).$$

As proved in their paper, despite the computational efficiency of this approximation, which is frequently adopted for calibrating the G2++, the approximation error is small for shorter tenor and maturity options (up to a few basis points) and grows marginally for swaptions with higher tenor and maturity.

This last procedure has also been implemented in Matlab through the `ES_G2bySP` function (see Appendix I for the complete code with commentary). Despite the drawbacks of this approximation in term of errors in the southeast region of the swaptions volatility matrix, the Schrager and Plesser solution for ATM European swaptions prices will be used to calibrate the model given its superior efficiency in computing the swaption price, thus reducing the overall computational time required to calibrate the model.

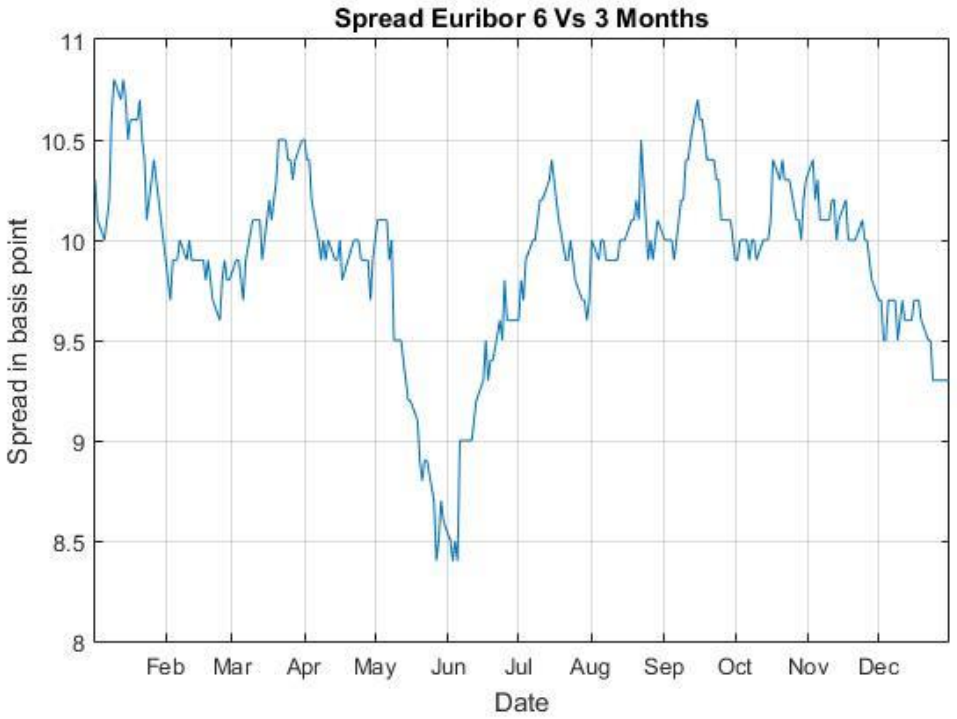
3.5 Remarks

In this third Chapter, we have seen the main results of the G2++ model as proposed by Brigo and Mercurio (2006 page. 137-175) in a mono-curve framework; indeed, the majority of these formulas will be used during the calibration procedure of the model to the market volatility surface. However, it is important to notice how recent developments in the literature has also

detailed the calibration procedure of short-term models under the multi-curve framework; as a matter of facts, the G2++ model under this more modern approach has been described by Bianchetti and Carlicchi (2010), and by Grasselli and Miglietta (2014), just to name a few. Nevertheless, despite these works suggestions, the objective of the current analysis is to calibrate the model under the classical pre-crisis approach that assumes the same curve both for discounting and for forwarding the term structure.

One of the main assumptions of the mono-curve framework is the absence of a relevant spread between different tenors Libor rates; nevertheless, given the current market conditions for the Euribor, represented in Fig. 10, this hypothesis is clearly violated.

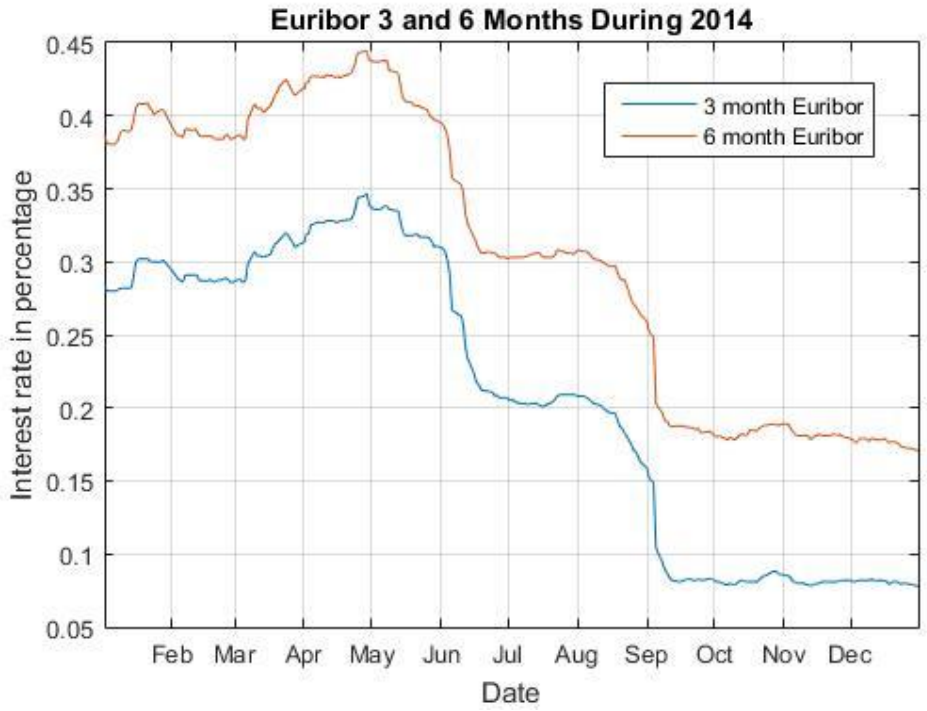
Fig. 10: Euribor 6 vs 3 Months Spread for 2014 (source: EMMI)



Moreover, the present market conditions are unique; indeed, the Euribor deposits quoted by the market have an extremely small interest rate. Thus, when Brigo and Mercurio initially developed the two-factor additive Gaussian interest rate model, they were taking into consideration interest rates that moved around 4.00% values, while now they revolve around levels far below 1.00%, and this will have an impact on the model parameters that is difficult to predict. However, the effect on the volatility surface of swaptions and caps is clear, since the extremely low interest rates, increase market volatilities throughout the whole volatility surface (see Fig. 7, 8 and 9 in the second chapter).

In Fig. 11, the two main Euribor rates used as underling of interest rate derivative contracts are represented. From the figure, one can clearly see the extremely low values throughout the whole 2014.

Fig. 11: Euribor 6 and 3 Months During 2014 (source: EMMI)



The multi-curve framework will require different forward curve for each underling Euribor tenor; nevertheless, our discounting curve has been built taking into consideration the same underling Euribor tenor of the instruments that will be used to calibrate the model. Therefore, the spread among different Euribor curves should not represent an issue for our model specification.

However, the adoption of a multi-curve approach would also imply the usage of the Overnight Indexed Swap curve (OIS from hereafter) as the starting building block for bootstrapping discount factors. The OIS rate of reference for the Euro area is the EONIA (European OverNight Index Average): this quantity refers to a weighted average of all overnight and unsecured lending agreements in the European interbank market. Thus, requiring a different procedure than the one highlighted in the second chapter.

Moreover, following the conclusions of Hull and White (2012) paper on this subject and referring to their proofs, we can state that a true risk-free rate is not observable in financial markets; however, the OIS rate is its best proxy given the ability to divide the value of a portfolio of derivative contracts into its three main components:

- the no-default value of the portfolio;
- the impact of the dealer and counterparty credit risk;
- the effect of non-economic interest rate paid on cash collaterals.

Thus, by using the OIS rate we should be able to capture the real no-default value of any derivative contract. Also various other researchers and practitioners were noticing the

discrepancies between the pre-crisis and the post-crisis interest rate market, among them: Peng *et al.* (2008), Madigan (2008) and Morini (2009); thus, the multi-curve framework was adopted for modelling interest rate derivatives, while questioning the mono-curve framework.

Focusing back to our two-additive-factor Gaussian model, Moreni and Pallavicini (2010), starting from the need to modify the HJM framework to adapt the model to the new multi-curve approach, proposed a weighted G2++ model. This particular model is able to capture for each forward Libor rate a different dynamics, consistently with the hypothesis that each forward rate is considered as a different underlying asset.

Despite all this interesting literature, and many other works not cited in this brief paragraph, we have still adopted the single curve framework given the absence of a fully developed specification of the G2++ model under the multi-curve approach, but now the reader has been warned about the biggest limitation of the present work: the adoption of the mono-curve framework.

4th Chapter

The Model Calibration

4.1 Introduction

Different opportunities arise when calibrating models such as the G2++; indeed, the first decision someone has to make is whatever to calibrate the model to caps (floors) options or to European swaptions. Of course, the final decision of the calibration instrument is not casual: the "best" instrument depends upon the type of claim someone is trying to price with this specific model.

What is important to underline is that: on one hand, if the model is calibrated to the volatility surface of caps (floors) with different strikes, the calibration is faster, but, given the nature of the model, it is not able to capture the skew given by out-of-the-money strikes. On the other hand, caps do not contain any information about the correlation between forward rates; hence, the degeneration of the model to a one-factor model will frequently happen. Indeed, a frequently observed result, when the model is calibrated to caps (floors), is the correlation parameter, ρ, close to -1.

On the contrary, when someone fits the model to the at-the-money European swaptions volatility surface, since no smile effect is present in ATM contracts and swaptions are able to capture negative decorrelation among different forward rates, the correlation parameter is often different from -1.

4.2 The Loss Function and the Implied Volatility Problem

The calibration of a model may be consider more an art rather than a science; as a matter of facts, despite the usual formulation of the problem as a constrained minimization process of some loss function, the practitioner may choose a subset of instruments from the market data matrix of caps or swaptions. This selection may depends on the most traded instruments by the market or by the particular desk that is developing the model, as it may be interested in calibrating the model only to a few points in the swaptions or caps volatility matrix

corresponding to the most traded instruments by the office or to a specific contract that must be priced.

For what concerns the present work, we have decided to avoid caps and swaptions with maturity and tenor, respectively, shorter or equal to 1 year, given that they use as underline the 3 months Euribor, while for longer maturities and tenors the 6 months Euribor is the underline of reference. Indeed, when we have bootstrapped the zero-coupon curve we have relied on forward agreements and interest rate swaps that used as underling the 6 months Euribor; thus, in order to be consistent with our discounting curve, we will only use contracts that have the same Euribor tenor.

Once the calibration instruments have been chosen, it is also possible to assign to each instrument a specific weight; nevertheless, in the present thesis we have decided to give to each instrument the same weight equals to 1.

After all these preliminary passages and remarks, it is necessary to implement a specific loss function. Since the loss function is at the core of the calibration procedure, we will briefly review three of the most used loss functions for adapting interest rate models to markets data:

a) the squared weighted percentage difference between model and market Black implied volatilities

$$\sum_{i=1}^{n} w_i \left(\frac{Volatility_i^{G2++} - Volatility_i^{Market}}{Volatility_i^{Market}} \right)^2$$

b) the squared weighted percentage difference between model and market prices

$$\sum_{i=1}^{n} w_i \left(\frac{Price_i^{G2++} - Price_i^{Market}}{Price_i^{Market}} \right)^2,$$

c) the squared weighted difference between model and market prices standardized by the market vega

$$\sum_{i=1}^{n} w_i \left(\frac{Price_i^{G2++} - Price_i^{Market}}{Vega_i^{Market}} \right)^2.$$

In the present work, we will use the percentage difference between market and model implied volatilities as it represents our best option given that we are trying to fit a matrix of observed volatilities and not prices. Moreover, we do not want to transform market volatilities into Black prices since prices may vary greatly accordingly to different maturities, tenors, strikes and notional values, while the same cannot be said for market-implied volatilities, which usually lie in the interval between $(0,1)$ and are less influenced by the maturity or the tenor of the particular option. However, the main point in favour of the first loss function is also its biggest limit since

it is not always possible to find the model implied volatility, while it is always possible to calculate the price given by the model.

In practice, the calibration procedure using the model implied volatilities might be explained by supposing that we have two functions for calculating the price of the same contract:

I. on one hand, the first function is the Black model formula for the price of a swaption or cap (floor) agreement, which depends on a series of variables, including a specific Black volatility parameter

$$f^{Black}(\cdot, \sigma_{Black}) = price,$$

II. on the other hand, the second function is the G2++ model analytical formula for the price of a swaption or cap (floor) contract, which still depends on a series variables, but not on the Black volatility parameter

$$f^{G2++}(\cdot, a, b, \sigma, \eta, \rho) = price$$

If we are interested in finding model implied volatilities, we should find the value of σ_{Black} given by a specific set of model parameters $(a, b, \sigma, \eta, \rho)$ that will make the difference between the two prices equal to zero

$$f^{G2++}(\cdot, a, b, \sigma, \eta, \rho) - f^{Black}(\cdot, \sigma_{Black}) = 0.$$

This passage requires the solution of a minimization problem given the non-analytical tractability of the Black formula for both caps (floors) and swaptions.

Indeed, the algorithm calculates the price of a contract under the G2++ model for a given set of parameters (in this case for a given set of a, b, σ, η, ρ) and then will try to find the value of σ_{Black} that will make equal to 0 the difference between the two analytical price functions.

Once the implied volatility by the model is known, it is possible to minimize the following quantity:

$$\sum_{i=1}^{n} \left(\frac{\sigma_{Black\ i}^{G2++} - \sigma_{Black\ i}^{Market}}{\sigma_{Black\ i}^{Market}} \right)^2$$

If we take into consideration the calibration to the cap (floor) surface, one can define the cap (floor) model implied volatilities as

$$\sum_{i=1}^{n} P(t, T_i) N \tau_i Bl\left(K, L_i(t), \sigma^{G2++}\sqrt{T_{i-1}}, \omega\right)$$

$$= \sum_{i=1}^{n} [-\omega N(1 + X\tau_i) P(t, T_i) \Phi(\omega d_1) + \omega P(t, T_{i-1}) N \Phi(\omega d_2)].$$

where the left hand of the equation is the Black formula presented in the first chapter, while the right hand of the formula is the G2++ formula presented in third chapter.

For caps (floors) implied volatilities a root-finding algorithm, in the form of `fsolve` in Matlab, has to be implemented. Unfortunately, `fsolve` may fail to converge for some entries in the cap (floor) model implied volatility matrix stopping the algorithm from calculating the value of the overall fitness function.

In these cases, we can opt for two type of solutions in order to let the algorithm runs despite a missing volatility:

I. use a penalty function, which assumes a fixed number for the model volatility if the Black formula is not invertible;

II. use an approximation for the missing value of the volatility.

If we use the second approach, one can find an approximate solution for the implied volatility, by using the following rule of thumb given by Corrado and Miller (1996), which may be adapted to caplet (floorlet) implied volatilities

$$\sigma^{G2++} \approx \frac{1}{n}\sum_{i=1}^{n}\sqrt{\frac{2\pi}{T_{i-1}}}\left\{\frac{CapletFloorlet_i^{G2++}}{\omega N(L_i(0)+K_i)\sum_{i=1}^{n}\tau_i P(0,T_i)} - \frac{1}{2}\frac{L_i(0)-K_i}{L_i(0)+K_i}\right.$$
$$\left. + \left[\left(\frac{CapletFloorlet_i^{G2++}}{\omega N(L_i(0)+K_i)\sum_{i=1}^{n}\tau_i P(0,T_i)} - \frac{1}{2}\frac{L_i(0)-K_i}{L_i(0)+K_i}\right)^2 - \frac{1}{\pi}\left(\frac{L_i(0)-K_i}{L_i(0)+K_i}\right)^2\right]^{\frac{1}{2}}\right\}.$$

The above approximation has as a main drawback, the possibility that the term under the root may be negative, leading to an approximated implied volatility that is composed also by an imaginary part. Therefore, we will simply substitute missing model volatilities with a constant times the market volatility, leading to a high percentage error for missing entries.

On the other hand, if we use the At-The-Money (ATM) European swaptions volatility data to calibrate the G2++ model, the value of σ^{G2++} has to be recovered from the below expression:

$$N\omega\left[P(0,T_\alpha) - P\left(0,T_\beta\right)\right]\left[2\Phi\left(\omega\frac{1}{2}\sigma^{G2++}\sqrt{T_\alpha}\right) - 1\right]$$
$$= N\omega P(0,T_\alpha)\int_{-\infty}^{+\infty}\frac{e^{-\frac{1}{2}\left(\frac{x-\mu_x}{\sigma_x}\right)}}{\sigma_x\sqrt{2\pi}}\left[\Phi\left(-\omega h_1(x)\right)\right.$$
$$\left. - \sum_{i=1}^{n}\lambda_i(x)e^{\kappa_i(x)}\Phi\left(-\omega h_2(x)\right)\right]dx,$$

where on the right hand of the equation we have the ATM European swaption price under the Black Model, while on the right side we have the usual formula for European swaptions given by the G2++ model (as presented in the third chapter).

Nevertheless, in this case we can avoid the implementation of a root-finding algorithm for retrieving the value of the model-implied volatilities. Indeed, by simply observing that from the above Black formula for ATM European swaptions we can isolate the volatility term as a function of the inverse normal cumulative function and the price of the swaption under the G2++ model:

$$\sigma^{G2++} = \frac{2}{\omega\sqrt{T_\alpha}}\Phi^{-1}\left(\frac{EuropeanSwaption^{G2++}}{2\omega N\left[P(0,T_\alpha) - P\left(0,T_\beta\right)\right]} - \frac{1}{2}\right),$$

where $\Phi^{-1}(x)$ represents the inverse normal cumulative distribution (`invnorm` in Matlab). However, despite the savings in computational time, the inverse normal cumulative function may not be invertible if:

$$\frac{EuropeanSwaption^{G2++}}{2\omega N\left[P(0,T_\alpha) - P\left(0,T_\beta\right)\right]} - \frac{1}{2} > 1.$$

In these cases, we can still adopt one of the two solutions also available for caps (floors) volatilities: using an approximation or substitute the missing value.

By adapting the formula proposed by Brenner and Subrahmanyam (1988) for the implied volatility of an ATM call option, we may give a reasonable approximation for the missing values of implied volatilities:

$$\sigma^{G2++} \approx \frac{EuropeanSwaption^{G2++}}{\omega^2 N\left[P(0,T_\alpha) - P\left(0,T_\beta\right)\right]}\sqrt{\frac{2\pi}{T_\alpha}}.$$

However, the above formula is more a rule of thumb than a real approximation on the value of the implied volatility. Thus, it is more advisable to use the following rational approximation, adapted from the one given by Li (2006), of the inverse error function (i.e. the function from which the inverse normal distribution is calculated):

$$\sigma^{G2++} \approx \frac{1}{\omega\sqrt{T_\alpha}}\left(\frac{2.506297c - 0.686461c^2}{1 - 0.277069c - 0.237552c^2}\right),$$

where c is the normalized option price, more precisely,

$$c = \frac{EuropeanSwaption^{G2++}}{\omega N\left[P(0,T_\alpha) - P\left(0,T_\beta\right)\right]}.$$

Only if the rational approximation does not converge to a solution, the model implied volatility is replaced by a fixed number that will generate an error in the final loss function sufficient for the minimization algorithm to ignore combinations of model parameters $(a, b, \sigma, \eta, \rho)$ that will make the model implied volatilities not retrievable.

Moreover, if we want to add ATM caplets (or floorlets data) to our swaptions model calibration, we should find the root of the following equation

$$wNP(0, T+\tau)F(0; T; T+\tau)\left[2\Phi\left(\omega\frac{\sigma^{G2++}\sqrt{T}}{2}\right)-1\right]$$
$$= -\omega N(1+X\tau)P(0, T+\tau)\Phi(\omega d_1) + \omega P(0, T)N\Phi(\omega d_2).$$

Analogously to the ATM swaptions case, it is possible to use the normal inverse function, instead of `fzero`, in order to obtain the model implied volatility; additionally, also for the ATM caplets (floorlets), the above rational approximation for swaptions can be adopted.

Going back to our calibration problem, Blanchard (2012) proposed the following boundaries for the model parameters:

$$0 < b < a < 2.50,$$
$$0 < \eta < \sigma < 0.75,$$
$$-1 \leq \rho \leq 1.$$

However, traders and practitioners usually set the correlation parameter, ρ, equals to -0.70. With this simple trick they are able to force a true two-factor model, despite some papers suggesting a lower value for the correlation parameter, almost equals to -1, as reported by Moreni and Pallavicini (2010).

Thus, since we want to find the real value of the model parameters we will adopt these slightly different boundaries from Blanchard proposed parameters, without imposing $a > b$ or $\sigma > \eta$ and reducing the overall domain of the model parameters:

$$0 < b \leq 1.00, \qquad 0 < a \leq 1.00$$
$$0 \leq \eta \leq 0.50, \qquad 0 \leq \sigma \leq 0.50$$
$$-1.00 \leq \rho \leq 1.00.$$

4.3 The Minimization of the Loss Function with Mention to Coding

After that we have decided a loss function and a set of boundaries for our parameters, we should also choose the algorithm for minimizing the differences between model and market data. For what concerns the present work, each different minimization procedures could be included within two larger group of methodologies: deterministic or stochastic optimizations. Therefore,

even in this passage, we find another trade-off between time and efficiency. Indeed, on one hand deterministic algorithms are faster, but do not guarantee a global optimum; on the other hand, stochastic algorithms are slower, but have higher likelihood of convergence to the real problem solution.

If we decide to use a deterministic optimization algorithm, an additional preliminary step is required, since these algorithms must start their routines from a specific initial point, the user has to decide a starting point a priori. Thereof, another question arise for deterministic procedures: how could we decide this initial seed point? To make a long story short, it is just a guess made by the user. Indeed, if someone has already calibrated the model to previous day market data, the initial point could be set equal to the previous day optimal parameters, but there is still a high probability that the algorithm will converge to a local minimum rather than to a true global minimum.

After these more general observations, it is time to describe each methodology and their implementations in Matlab, focusing in particular on different minimization algorithms that have been adopted for minimizing the sum of the square percentage difference between model and market implied volatilities.

4.3.1 Deterministic Minimization Algorithms

The two major common aspects of numerous deterministic minimization algorithms have been already discussed in the previous section, however, we will now focus on the two main algorithms used in Matlab to minimize our percentage volatility loss function: `lsqnonlin` and `fmincon`.

Starting from the former, `lsqnonlin` is used to solve problems, including nonlinear data fitting, without equation constraints, but this limitation is not an issue for the calibration of the G2++ model. Basically, the function minimizes the following quantity

$$\min_x \|f(x)\|_2^2 = \min_x (f_1(x)^2 + f_2(x)^2 + f_3(x)^2 + \cdots + f_n(x)^2)\,,$$

in our case, the function $f(x)$ is expressed as a percentage difference between market and model volatilities (prices) and n is the number of instruments used to calibrate the model; this implies that the algorithm will automatically compute the sum of the squares between different function values once a suitable objective function has been built by the user.

However, Matlab does not allow passing extra parameters and in such cases the two most efficient methodologies to pass enough, or any, parameters throughout the minimization algorithm are:

a) the extensive use of `global` variables, since, once a quantity gains the status of global variable, it is memorized in Matlab for any future function calls (i.e. we do not need to write the variable as a function input despite the fact that it is still used inside the function);
b) the use of nested functions that are able to call any variable from the original function (i.e. we have to build another function that accepts as input only the variable to be optimized inside a function that accepts also other parameters).

The first methodology is easier to adopt and make the whole code more elegant, since we could make any function depending on just few input variables, recalling all the other less important ones through the global variable syntax. Nevertheless, using nested functions is the only viable alternative when someone wants to implement parallel computing (i.e. enable more computer cores to do the same calculation in parallel) in order to save computational time or to debug the code. Indeed, global variables may be overwritten during a specific function without the user noticing the change, which is hidden within the global variable itself. Thus, despite the fact that we will not use parallel computing, we have preferred to use nested functions to feed extra parameters to the optimization algorithm.

Once the user has defined the nested function, `lsqnonlin` minimizes the function using the trust-region approach. According to Matlab documentation, the algorithm uses a suitable approximation of the original function $f_{app}(x)$ to research a point, P_1, nearby the starting point, P_0, that generates a lower value for the approximated function:

$$f_{app}(P_1) < f_{app}(P_0).$$

This procedure is repeated until Matlab is not able to find a nearby point P_i that generates a lower value for the approximated function. The function approximation should be calculated using a second degree Taylor expansion, but this would require the calculation of the Hessian matrix. Thereof, Matlab uses another methodology presented in Coleman and Li (1996) and in Byrd, Schnabel and Shultz (1988), which consist in reducing the trust-region sub problem into a two-dimensional subspace. This shift into a two dimensional subspace ease the whole problem, but now a suitable subspace has to be determined. Without going into further details, the objective of this whole approach is to obtain a global convergence or achieve at least a local convergence.

Consequently, as one can clearly imagine, the result of `lsqnonlin` is highly linked to the initial starting point; thus, the likelihood of convergence of this specific algorithm to a global minimum is extremely low, suggesting the need of more complex minimization procedures.

Switching to `fmincon`, the same also applies for this more flexible minimization algorithm: it still requires a specific nested function and still applies the same minimization methodology; however, the function does not automatically calculate the sum of the squared function values, so it is necessary to code manually this calculation inside the nested function. Actually, the objective function must have a number as input, unlike the previous algorithm, `lsqnonlin`, which accepted as input also a vector containing the results from the objective function.
Consequently, apart from this small difference, both `lsqnonlin` and `fmincon` leads to the same result as both uses the trust-region minimization procedure; hence, we have to implement more powerful and non-deterministic techniques to increase our odds of finding a combination of model parameters $(a, b, \sigma, \eta, \rho)$ that represents a global and not a local minimum.

4.3.2 Stochastic Minimization Algorithms

Stochastic minimization encompasses a set of optimization methodologies that have higher probability of convergence to a global optimum; indeed, despite the presence of a deterministic set of variables, the introduction of some randomness in the optimum research may lead to a better convergence.
For our specific task of calibrating the G2++ model, the adoption of one of these approaches appears particularly useful since the minimization problem is not well-posed mainly because:

i. the loss function is not monotonic, suggesting the presence of many strong local minimums;
ii. the problem is highly sensitive to small changes in input data, implying that small changes in European swaptions, caps or discount factors data generate different final model parameters.

In Matlab, the following algorithms were used for calibrating the G2++ model using a non-deterministic function: simulated annealing, genetic minimization and differential evolutionary.
The Global Optimization toolbox in Matlab already includes simulated annealing optimization for solving bound-constrained minimization problems, the so-called `sa` function. This function solves the minimization by adopting a method used to minimize the energy of a system in physic; indeed, by heating a given material and then slowly reducing the temperature, `sa` is able to minimize the energy present in the system.
A new point is randomly generated at each algorithm iteration, the space between the previous point and the newly generated point is given by a probability distribution that is related to the temperature. The algorithm converges to new points that lower the value of the objective function; nevertheless, with a specific probability, the algorithm also accepts point that raise the value of the objective function. The acceptance of these points should lower the likelihood

of being stuck into a local minimum, since the algorithm does not stop its research too soon and is able to explore more combinations of parameters.

However, despite the flexibility of this algorithm, simulated annealing still requires a starting point at the beginning of the optimization routine; thus, other options should be considered to eliminate the initial guess.

To avoid the need of an initial point, a genetic algorithm minimization has also been implemented to calibrate the model (`ga` in Matlab). A genetic algorithm tries to mimic the process of natural selection in order to find solutions to optimization and search problems; indeed, this type of optimization algorithms is included in the larger class of evolutionary algorithms, which follows techniques always inspired by natural evolution theories.

The `ga` function is already included in the Global Optimization toolbox and its implementation still requires a nested function to pass different parameters to the objective function; however, we should focus on a series of options that are necessary for increasing the likelihood of convergence toward a global optimum. Indeed, `ga` requires some adjustment to the pre-set options:

- since we are trying to minimize a function with five variables, our G2++ model parameters (a, b, σ, η, ρ), we must have an initial population higher than 25, which is the Matlab standard population size; hence, the population has been raised to 50, ten times the number of variables;
- we must also ensure enough diversity in the initial population given that Matlab set the initial population range between $(0,1)$; thus, we have to modify the initial population range in order to include all the numbers between our parameters boundaries;
- Matlab automatically ends the optimization procedure after 100 iterations, but calibrating the two-factor Gaussian model may require more than 100 iterations; so the maximum number of iterations has been increased to 500;
- for debugging reasons we must also let the algorithm show the result of each iteration, so that it is possible to notice if the algorithm is able to explore enough deeply the solution domain of the model paramters.

All these options can be changed by suing a statement similar to the following one:

```
options = gaoptimset('Generations',500,'PopInitRange',[lb;ub],...
            'PopulationSize',50,'Display','iter','TolFun',1e-6);
```

where all the above purple words are expression that Matlab uses to identify specific options classes for the genetic algorithm optimization toolbox.

Once the algorithm properties have been modified, we can start the optimization routine. `ga`

randomly generates an initial population of 50 members, with each member consisting of five model parameters included between the problem boundaries. The algorithm will assign to each member a set of genetic characteristic and will proceed with the calculation, for each population member, of the G2++ prices of those instruments that have been selected to calibrate the model. The loss function is calculated from these model prices; however, since we have decided to calibrate the model using volatilities instead of prices, the algorithm will invert the G2++ prices in order to retrieve the model implied Black volatilities by using the function `ES_ImpVol`, or `CF_ImpVol`, accordingly to which instruments have been considered.

At each iteration, `ga` saves the fittest member (i.e. the parameters combination with the lowest loss function value) of the population and create children taking into consideration genes from parents individual. Indeed, Matlab allows for the percentage of elite children (i.e. the number of population members that will remain also for the next iteration) to be selected by the user, but we believes that the default Matlab set is sufficient. Thus, by improving the population fitness at each iteration, the algorithm is able to converge to a solution and, at the same time, to explore regions of the problem that other deterministic algorithms will not take into consideration.

However, despite its desirable features, the `ga` toolbox has its limitations since:

1. the algorithm is extremely computationally intensive; thus, the model requires more than half an hour to be calibrated to the ATM European swaptions surface;
2. `ga` may not be able to converge to a global minimum and, more importantly, the final solution may not be even a local minimum; hence, it is necessary to refine the solution by calling another optimization algorithm, such as `lsqnonlin` or `fmincon`;
3. given its random nature, the final solution can change depending on the initial population and its later evolution.

Taking into consideration these limitations, another approach has been implemented based on an algorithm initially developed by Price, Storn and Lampinen. Differential evolution is another population based function minimizer, falling in the same category of `ga`, and it is still considered among the best genetic optimizer for solving real-valued test functions. Leaving the mathematical basis of the algorithm to the book "Differential Evolution – A Practical Approach to Global Optimization", the main development from `ga` is the scheme for generating the parameter vectors, as this procedure is able to reduce the computational time required for `differentialevo` (i.e. the name of Matlab function) to converge to an efficient solution.

As for the afore mentioned genetic algorithm, also differential evolutionary algorithms require a suitable nested function in order to pass extra parameters to our objective function; however,

the main difference between other built in Matlab algorithms is the objective based language needed to change the optimization default settings. As a matter of facts, algorithm requires some tweaks to its standard options since: the initial population range has to be adjusted to reflect the problem boundaries; secondly, we should increase the parents number to 50, following a suggestion made by Price, Storn and Lampien, who required the population to be at least ten times the number of parameters to be optimized; lastly, we must switch off the option of starting from a specific point.

After these small changes, the algorithm is able to search through the problem domain as efficiently as `ga`, but in less time; hence, this algorithm is preferred to Matlab built in genetic algorithm. Moreover, to better refine the results from the differential evolutionary algorithm, we should also run a deterministic algorithm. Despite this complicated calibration procedure, we could not assure that the final result is a global minimum, but the likelihood of being stuck in a local minimum is lower than when adopting only deterministic calibration techniques.

4.4 Calibration Results

Before we report the calibration results to the whole swaption surface represented in Fig. 12, it is important to define what type of swaptions are displayed by Bloomberg VCUB. VCUB displays a set of receiver swaptions with yearly cash flow, using as underlying the 6 months Euribor.

After this remark, we calculate the zero-coupon bond value from market data on deposits, FRAs and IRSs by bootstrapping the zero-curve, following the procedure already explained in the second chapter. Indeed, despite the desirable features of the Svensson model, only through bootstrapping the interest rate curve fully reflects the market one.

By minimizing the sum of the percentage squared difference between model and market implied volatilities through a differential algorithm and refining the intermediate results using a deterministic optimization algorithm, we arrive at the solution in Tab. 5.

Tab. 5: Model Parameters on the 18th of August 2014

a	b	σ	η	ρ
0.4954	0.0460	0.0206	0.0124	-0.9985

The first thing to notice is the difference between present day model parameters and the value initially given by Brigo and Mercurio (2006) on page 169. This change between model parameters is due to differences in the volatility surface level. Indeed, while in 2001 the highest and lowest swaption volatility presents in the 10×10 market data matrix were 16.40% and

8.40%, respectively; it is seen from Tab. 3 in the second chapter, that today the highest and lowest market volatility are 66.05% and 30.10%, respectively. The current level of the market volatility surface is partially explained by the extremely low levels of the 6 months Euribor rate, which greatly influences the discount factor curve and the lognormal volatility.

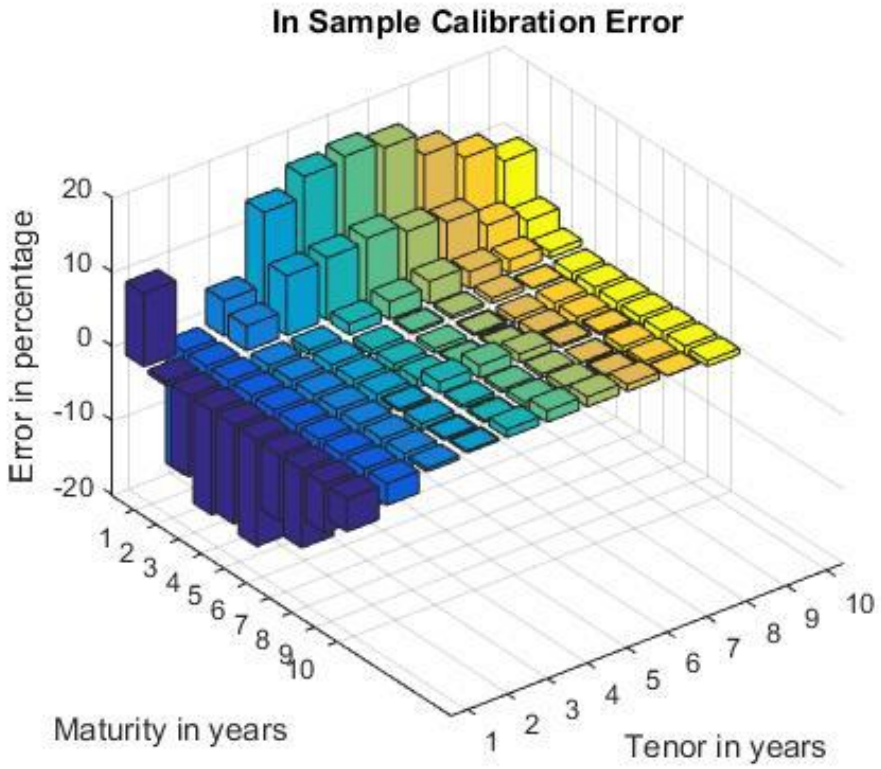
Another important observation is that the model correlation parameter, ρ, lies nearby the lower boundary, implying a degeneration of the G2++ model to a simpler one-factor model. The shape of the volatility surface may also explain the degeneration to a one-factor model. As a matter of facts, in spite of higher volatilities across the whole market data matrix, by focusing on only one row or column, the section of the volatility surface is rather flat, resembling almost a straight line; hence, the algorithm is not able to generate enough convexity in the volatility surface.

The drift parameter of the second process, b, which represents the strength of the mean reversion in the interest rate, is lower than the drift of the first process, a. This is also true for the standard deviations of the two processes, with η lower than σ. Thereof, this result is still similar to the one obtained by Brigo and Mercurio (2006) pag. 168.

It is important to make some remarks before going on with the discussion on the calibration methodology adopted in the present work. Firstly, the likelihood of convergence to a global minimum is high; nevertheless, given the random nature of the differential evolutionary algorithm and the need to ensure a reasonable trade-off between efficacy and computational efficiency, there is no absolute certainty that the resulting parameters represents a global minimum. Secondly, given the daily nature of the inputs, the model is useful only for one day: the date of the data used to calibrate it. Therefore, the today calibrated model parameters are not consistent with the optimal parameters for tomorrow or yesterday and nothing ensure that these two nearby days parameters are close to the ones observed today.

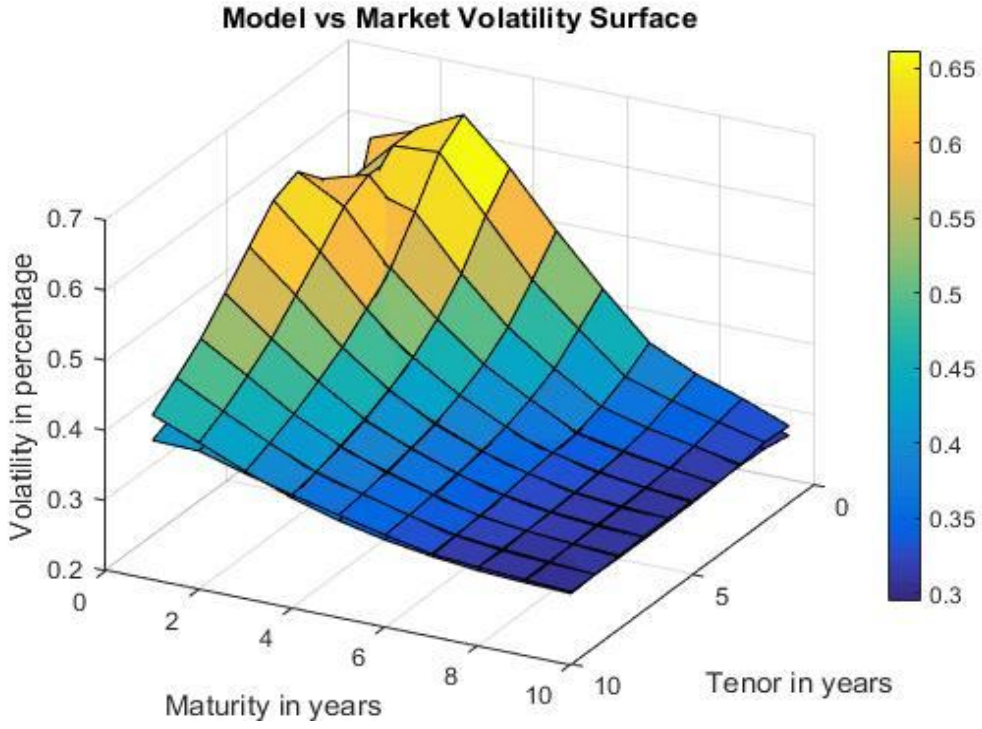
In Fig. 12, the error in percentage between model and market volatilities is shown for the instruments used to calibrate the model. It appears that the model parameters are able to give a discrete fit to the input data, resulting in an average absolute percentage error of 4.21%; however, by looking to swaptions with maturity and tenor combinations equal or lower than two years, the percentage error is higher than for other instruments. This phenomenon is due to the shape of the market volatility surface (as displayed in Fig. 7, second chapter, and in the following Fig. 13). Indeed, the volatility surface presents an atypical humped shape for swaption contracts with tenor equals to two years while, as one can clearly see in the example of Brigo and Mercurio (2006) on page 21, the hump is less pronounced for swaptions with the same maturities.

Fig. 12: In Sample Model Calibration Errors in Percentage



In the below figure, both the volatility surface resulting from the calibrated G2++ model and the one observed in the market are depicted.

Fig. 13: Model vs Market Volatility Surface



From the figure, one can also notice that the model is able to give a reasonable fit to the most liquid contracts (i.e. northeast region of the surface), having high errors in upper left corner of

the surface. This result is somehow expected for European swaptions lying in the northwest region of the matrix; as a matter of facts, the adoption of the percentage difference of volatilities as our loss function gives precedence to longer maturity and tenor contracts.

Another possible explanation of this result is the particular method used to calculate the model implied volatilities. As already discussed in the previous chapter, the algorithm tries to calculate the model implied volatilities by using the `norminv` function in Matlab; nevertheless, it is not always possible to invert the normal distribution and, in substitution of missing `NaN` results, a rational approximation has been adopted. The quality of this particular approximation may be questionable, but the substitution of missing volatilities with a penalty function (i.e. the market volatility times a constant) is seen as a far worst solution; furthermore, the rational approximation is extremely fast in computing missing volatilities. A better solution may be the adoption of a bisection algorithm, which will always be able to find a solution for the model-implied volatility; however, this approach will slower the algorithm by using a while statement for cutting the domain of the function at each minimization function iteration.

Despite the calibration procedure followed in the present work has its limitations, it is nonetheless able to generate a volatility surface that represents the one observed in the market with relatively small errors.

In Tab. 6, we also report the percentage error between model and market volatilities and our model volatility surface seems to lie slightly below the market one on the northwest region and on the southeast region, leading to small errors between model and market volatilities, while it rests above the market surface for the central region.

Tab. 6: Model vs Market Volatilities Error in Percentage for Swaptions

		Tenor									
	Year	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Maturity	**1**	9.88	-16.59	4.79	14.59	17.38	18.00	17.12	14.07	11.60	9.10
	2	-0.58	-2.86	3.32	8.02	8.37	8.93	7.76	6.99	4.48	3.03
	3	-10.86	-11.11	-4.84	-0.71	1.49	2.37	3.03	2.27	1.79	0.84
	4	-14.01	-11.03	-5.69	-2.08	-0.76	0.36	0.44	0.71	-0.12	-1.20
	5	-13.28	-10.46	-5.06	-2.69	-1.01	-0.60	-0.31	-0.75	-1.47	-2.25
	6	-14.30	-10.03	-5.74	-2.69	-1.30	-0.34	-0.24	-1.02	-1.51	-2.10
	7	-8.95	-6.32	-2.19	-0.34	1.51	1.68	1.16	0.33	-0.34	-1.04
	8	-10.55	-6.55	-3.34	-0.88	0.45	0.46	0.47	-0.06	-0.79	-1.53
	9	-6.60	-5.54	-1.94	-0.28	0.34	0.73	0.76	0.28	-0.52	-1.33
	10	-4.42	-2.91	-0.47	0.30	1.07	1.52	1.32	0.91	0.07	-0.76

As a further check, in Tab. 7, the percentage error between model and market caps volatilities is presented. The error is high for maturities lower than five years, but this is somehow expected,

given the characteristic of a cap contract, which is a summation of caplets, and the absence of calibration instruments with a combination of maturity and tenor lower than two year (i.e. the lowest sum of maturity and tenor combination used to calibrate the model).

Tab. 7: Model vs Market Volatilities Error in Percentage for Caps

Maturity	1	2	3	4	5	6	7	8	9	10
Error in percentage	211.93	89.38	46.37	14.32	-1.53	-7.31	-9.82	-10.07	-10.18	-10.02

The model calibration to ATM or OTM caps using the analytical price equations given in the third chapter will generate similar errors; however, as already stated in the previous chapter, this type of contract does not include any information on the correlation factor, leading to a correlation parameter, ρ, equals to minus one.

5th Chapter

The Monte Carlo Simulation Implementation

5.1 Introduction

As already highlighted at the end of the previous chapter, the degeneration of the G2++ to a one-factor model has been frequently observed after the financial crisis and the development of the multi-curve framework. Indeed, the model correlation parameters, ρ, resulting from the calibration to the swaption surface, is close to -1, thus confirming the results of Di Francesco (2012) and of Moreni and Pallavicini (2010).

Once the model has been calibrated to the analytical prices, it is possible to simulate interest rate paths and use these paths in order to price derivatives with more complex payoffs; nevertheless, before going on with an example, it is necessary to verify if our Monte Carlo simulation is able to approximate the analytical formulas with small errors.

Moreover, one can calibrate directly the Monte Carlo simulation without implementing the analytical pricing formulas for the model; however, the result may not be satisfactory given the increase in computational time required by calibrating the Monte Carlo directly. Hence, our discussion from now on assumes that the model has been already calibrated to a given set of instruments by following the procedure highlighted in the previous chapter.

5.2 The Monte Carlo Simulation and Its Matlab Implementation

Initially developed in 1946[4] for investigating distances of neutrons trough different materials, after the introduction in 1964 by Hertz and its first application for valuing derivatives by Boyle (1977), the Monte Carlo method has become over the years one of the most used tools for pricing options today. Indeed, while only few vanilla options have closed form solutions in many equity and interest rate models, the majority of instruments traded in OTC markets have to be evaluated by recurring to Monte Carlo simulations or Trees.

The two model slightly differ, since:

[4] See Eckhardt (1987) for more information on the origin of the Monte Carlo method.

a) on one hand, trees are able to capture the benefit of early exercise typical of American options, since the payoff is calculated from last tree branches backward in time until the starting point is reached;
b) on the other hand, Monte Carlo simulations show its true potential for pricing path dependent derivatives, such as Asian options (i.e. their payoffs are calculated as an average of the underlying price over a given time).

Focusing on the Monte Carlo simulation of the G2++ model, we should be aware that many different discretization schemes are feasible when approximating the distribution of the interest rate process, $r(t)$; however, we will focus on one of the most frequently used one: the Euler discretization scheme.

Recalling the G2++ interest rate dynamic

$$r(t) = x(t) + y(t) + \varphi(t), \quad r(0) = r_0,$$

where the two processes $x(t)$ and $y(t)$ follows the two stochastic differential equations

$$dx(t) = -ax(t)dt + \sigma dW_1(t), \quad x(0) = 0,$$

$$dy(t) = -by(t)dt + \eta dW_2(t), \quad y(0) = 0,$$

the two terms W_1 and W_2 refers to two-dimensional Brownian motions defined by an instantaneous correlation equals to

$$dW_1(t)\, dW_2(t) = \rho dt.$$

The discretization is quite straightforward for the two stochastic factors, but we cannot say so for the deterministic fitting function $\varphi(t)$, given its dependence upon the instantaneous forward rate

$$\varphi(T) = f^M(0,T) + \frac{\sigma^2}{2a^2}(1 - e^{-aT})^2 + \frac{\eta^2}{2b^2}(1 - e^{-bT})^2 + \rho\frac{\sigma\eta}{ab}(1 - e^{-aT})(1 - e^{-bT}).$$

However, it is possible to run an exact simulation for the two processes under the T-forward measure, as already stated before, indeed, we already know that under the Q^T measure the solutions for the two processes Stochastic Differential Equation (SDE) are, respectively:

$$x(t) = x(s)e^{-a(t-s)} - M_x^T(s,t) + \sigma\int_s^t e^{-a(t-u)}dW_1^T(u),$$

$$y(t) = y(s)e^{-b(t-s)} - M_y^T(s,t) + \eta\int_s^t e^{-b(t-u)}dW_2^T(u).$$

Thus, as suggested by Brigo and Mercurio (2006), we can arrive at the following discretization scheme under the T-forward measure by using the known transition density of the two processes:

$$x(t) = x(s)e^{-a(t-s)} - M_x^T(s,t) + N(t-s),$$

$$y(t) = y(s)e^{-b(t-s)} - M_y^T(s,t) + N(t-s),$$

with:

$$M_x^T(s,t) = \left(\frac{\sigma^2}{a^2} + \rho\frac{\sigma\eta}{ab}\right)\left[1 - e^{-a(t-s)}\right] - \frac{\sigma^2}{2a^2}\left[e^{-a(T-t)} - e^{-a(T+t-2s)}\right] - \frac{\rho\sigma\eta}{b(a+b)}\left[e^{-b(T-t)} - e^{-bT-at+(a+b)s}\right],$$

$$M_y^T(s,t) = \left(\frac{\eta^2}{b^2} + \rho\frac{\sigma\eta}{ab}\right)\left[1 - e^{-b(t-s)}\right] - \frac{\eta^2}{2b^2}\left[e^{-b(T-t)} - e^{-b(T+t-2s)}\right] - \frac{\rho\sigma\eta}{a(a+b)}\left[e^{-a(T-t)} - e^{-aT-bt+(a+b)s}\right],$$

$${\sigma_x}^2 = \sigma^2\frac{1 - e^{-2a(t-s)}}{2a},$$

$${\sigma_y}^2 = \eta^2\frac{1 - e^{-2b(t-s)}}{2b},$$

$$\rho_{xy}\sigma_x\sigma_y = \frac{\rho\sigma\eta}{(a+b)}\left[1 - e^{-(a+b)(t-s)}\right].$$

The term $N(t-s)$ refers to random numbers extracted from a multivariate normal distribution with means equal to 0 and variance covariance matrix given by

$$\Sigma = \begin{bmatrix} {\sigma_x}^2 & \rho_{xy}\sigma_x\sigma_y \\ \rho_{xy}\sigma_x\sigma_y & {\sigma_y}^2 \end{bmatrix}$$

This particular implementation of the Monte Carlo will make the final price much easier and faster to calculate.

As a matter of facts, a Monte Carlo simulation usually involves a payoff that is related to the instantaneous interest rate $r(t_i)$ calculated at different times or simply at the maturity of the contract, with $i = 0 < t_1 < t_2 < \cdots < t_m = T$, with the last time being the maturity of the agreement. We can denote the discounted payoff of this contract as:

$$\sum_{j=1}^{m} exp\left[-\int_0^{t_j} r(s)ds\right] H\left(r(t_1), \dots, r(t_j)\right).$$

However, the payoff usually depends from the simple spot rate rather than from the instantaneous interest rate, hence

$$L\big(t_i, t_i+\tau, r(t_i)\big) \coloneqq \frac{1}{\tau}\left[\frac{1}{P(t_i, t_i+\tau)} - 1\right],$$

which is a function of the zero-coupon bond price. Thankfully, the G2++ model has a closed solution formula for the zero-coupon price, which depends on the value of $x(t)$ and $y(t)$; thus, it does not require a simulation of the short process, $r(t)$, but only of the two processes. In the most common Monte Carlo discretization schemes, the process has to be divided in small intervals following a discretization model chosen among the many available (including the widely used Euler scheme). Therefore, if we have to simulate n paths for our interest rate process, we need to divide each of these paths in a series of small time intervals that are defined as sampling times, Δs_i. Moreover, the intervals must contain intermediate payoffs generated by the option that we are trying to price; therefore, limiting even further our flexibility on the suitable Δs_i choice.

Additionally, once a fair sampling time has been chosen, the discount factors for each simulation path has to be calculated as an integral of the short rate $r(t)$

$$exp\left[-\int_0^{t_j} r(s)ds\right],$$

In practice, this integral can only be approximated as a series of trapezoidal areas:

$$exp\left(-\sum_{i=k}^{j-1} r(s_i)\Delta s_i\right).$$

One can clearly imagine the computational burden required for calculating and storing the values of the discount factors along all simulation paths at each sampling time, especially since the latter should be chosen small enough in order to ensure the accuracy of the above approximation for the discount factors.

Nevertheless, as suggested by Brigo and Mercurio (2006, page. 115), since the SDE solution is known for the G2++ instantaneous interest rate, the whole simulation could be implemented under the T-forward measure. In other words, instead of discounting each cash flow at time 0 from time t_i using the integral of the interest rate, the forward zero-coupon bonds is calculated for time t_i to T and all the payoffs are divided by these quantities in order to capitalize all cash flows values to time T, by using the model analytical formula for the zero-coupon bond value.

After the mean of all the cash flows at time T has been calculated, its value could be discounted back to time 0 by using the corresponding value of the zero-coupon bond observed in the market

$$E\left\{exp\left[-\int_0^{t_j} r(s)ds\right] H\left(r(t_1),\dots,r(t_j)\right)\right\} = P^M(0,T)E^{Q_T}\left\{\frac{H\left(r(t_1),\dots,r(t_j)\right)}{P(t_j,T)}\right\}.$$

Thus, by implementing the T-forward simulation scheme, we do not have to calculate the value of the discount factor from the simulation of $r(t)$ by integrating the asset paths. The time space between each s_i can be increased, reducing the computational burden required by the simulation, since the discount factors are not approximated.

The Monte Carlo simulation for pricing European swaptions has been implemented in Matlab by following these steps (the implementation for caps and floors is analogous).

Firstly, we must generate a matrix of random number extracted from a multivariate normal distribution. The size of the matrix is given by the number of paths to be simulated and by the combination of Δs and T, the terminal forward measure, the interval between each sampling time and the forward time used to capitalize all cash flows.

However, not all paths have to be simulated directly; as a matter of facts, by implementing antithetic variates[5], as suggested by Hammersley and Morton (1956), not only the number of random extractions is halved, but also the standard error of the simulation is reduced. An important remark when calibrating the Monte Carlo simulation directly is that these random extractions should be fixed throughout the calibration procedure; thus, they must be calculated outside of the optimization algorithm used to calibrate the simulation.

The matrix and both y and x under the T-forward measure are generated by the `XY_Sim` function. The simulation of the two process is implemented through a `for` loop that calculates the values for both processes at each sampling times by using the equations highlighted at the beginning of this paragraph. As suggested by Matlab, before starting the loop, all the relevant variables must be initialized and, more importantly, the first row must be equal to zero for both x and y in order to fulfil the initial condition suggested by Brigo and Mercurio (2006 page 143).

The next step is to calculate the swaptions prices with the `ES_MC` function. This function is composed by a loop that calculates the price of all the swaptions by sampling the value of the payoff at each payment time. The payoff is given by the following formula

[5] See also Glasserman (2003) for a fully explanation of anithetic variates.

$$ND(t,T_\alpha)\left(S_{\alpha\beta}(T_\alpha)\ -K\right)^+\sum_{i=\alpha+1}^{\beta}P(T_\alpha,T_i)\tau_i,$$

where $S_{\alpha\beta}(T_\alpha)$ is the forward swap rate at time T_α for the period that goes from T_α to T_β (see the first chapter), τ_i is the year fraction between T_{i-1} and T_i (i.e. the distance between each cash flow), K is the strike rate of the contract, N is the option notional value, and $D(t,T_\alpha)$ is the discount factor that has to be applied to the payoff.

Given that, each contract will have a different T_α, which represents the maturity of the option on the underlying IRS, and a different number of cash flows depending on the tenor $T_\beta$of the option, another nested loop has to be built in order to calculate the payoff for each intermediate T_i. To store the undiscounted payoff value for all the swaptions in a single matrix, one can create an empty matrix with the following dimensions:

1. number of rows equals to the number of simulated interest rate paths;
2. number of columns equals to the time grid of our simulation (i.e. for our case this number is given by the sum of maximum maturity, 10, and maximum tenor, 10, times the number of cash flows per year, 2, leading to 200);
3. the number of "pages" must match the number of instruments with different maturities, or other parameters, that are priced at the same time.

However, it is not necessary to store these values mainly because, for large number of asset paths or contracts, Matlab may run out of memory (i.e. the code may results in a Java out of memory error). Thus, to avoid this error we will let the matrix change in size at each iteration and, to obtain this result, we will not store the value for each contract throughout the simulation, but we will rather overwrite the same matrix at each loop iteration.

On one hand, the first loop switches among contracts characteristic: thus, as far as we are concerned, the algorithm moves vertically across all the combinations of maturities and tenors. On the other hand, the second loop moves through specific contract cash flows to calculate the appropriate value for the zero-coupon bond, including the one needed to move each payoff at the final time $T = T_{\max(\alpha+\beta)}$. The zero-coupon bond is calculated by the function `P_MC` using the analytical formula for the zero-coupon bond in the G2++ model (see the third chapter).

After this, the `ES_MC` function proceeds with calculating the capitalized value at time T (i.e. the T in the T-forward measure) for all the simulated paths. Mathematically speaking, the value at time T of a generic payoff is given by

$$h_i \coloneqq \sum_{j=1}^{m} \frac{H\left(r(t_1), \dots, r(t_j)\right)}{P(t_j, T)},$$

where $H\left(r(t_1), \dots, r(t_j)\right)$ are the payoffs of the contract, $P(t_j, T)$ is the value of a zero-coupon bond using the G2++ formula and m is the number of simulation steps. Notice how, in our swaption example, we do not need to calculate the value of the short rate process, $r(t)$, since the option payoff depends on the simple forward Libor, which is calculated from the value of zero-coupon bond using the G2++ analytical formula.

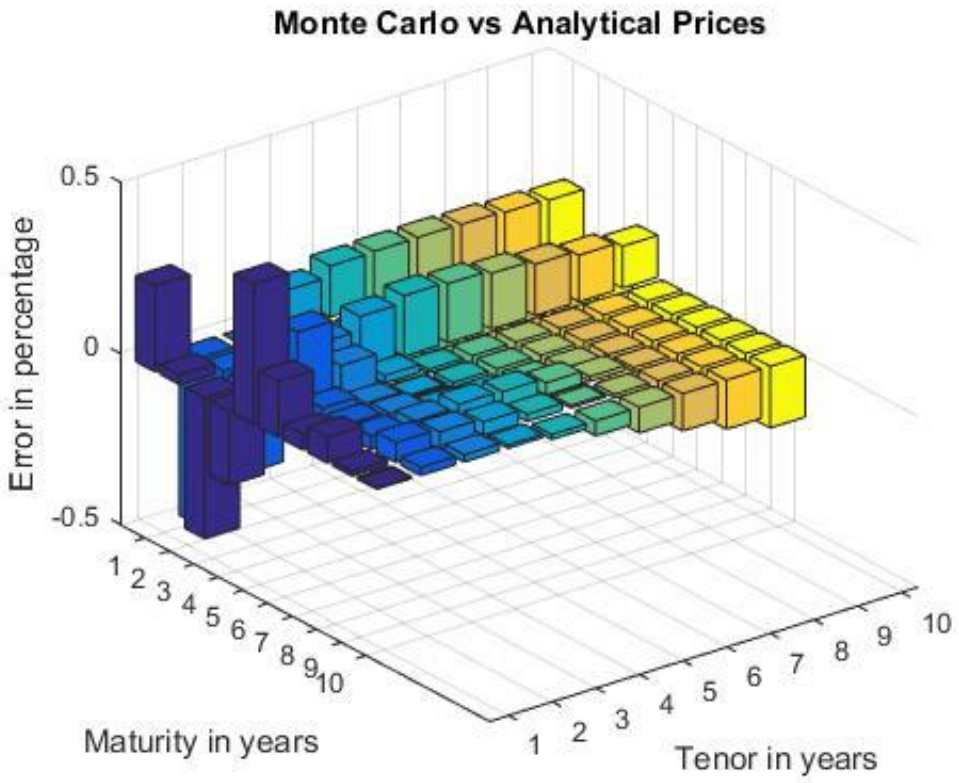
Before the algorithm moves to the next contract, all the values from the previous passage are averaged and discounted back at time 0 using the discount factor currently observed in the market, $P^M(0, T)$,

$$P^M(0, T) \frac{\sum_{i=1}^{n} h_i}{n},$$

obtaining the price of one swaption in this particular example.

Given the parameters resulting from the analytical calibration, in Fig. 14 we report the simulated percentage errors between analytical and Monte Carlo simulation swaption prices for the instruments used to calibrate the model in the previous chapter. The number of simulation has been set to 50'000. The difference between the two prices is mainly due to our analytical

Fig. 14: Analytical vs Monte Carlo Prices for Swaptions



swaption price approximation used to calculate the analytical price; nevertheless, the average absolute percentage error is equal to 0.10%.

In the following page table, the Monte Carlo error for caps is reported in percentage, the average absolute percentage error is equal to 0.20%. Thus, suggesting that our simulation scheme is enough robust to be used also for pricing more complicated contracts composed of both caps and swaptions.

Tab. 8: Analytical vs Monte Carlo Prices for Caps

Maturity	1	2	3	4	5	6	7	8	9	10
Error in percentage	-0.50	-0.19	-0.30	14.32	-0.15	0.04	0.00	0.16	0.19	0.21

Before moving on, it is not necessary to calculate the value of the interest rate process, $r(t)$, because only the results from the two stochastic processes $x(t)$ and $y(t)$ are needed to calculate the value of the zero-coupon bond in the G2++ model, which is required to calculate the 6 months Euribor rate from the simulation scheme. However, the value of the interest rate may be necessary to calculate payoffs of more complex instruments; hence, in what follows, we will explain how to build the $\varphi(t)$ function (i.e. the deterministic shift for the simulation), even if it is not required for including in our simulated short rate process the market observed discount factor curve.

The deterministic shift in the G2++ model is given by the following formula:

$$\varphi(T) = f^M(0,T) + \frac{\sigma^2}{2a^2}(1-e^{-aT})^2 + \frac{\eta^2}{2b^2}(1-e^{-bT})^2 + \rho\frac{\sigma\eta}{ab}(1-e^{-aT})(1-e^{-bT})$$

Thankfully, has already explained in the third chapter, the G2++ analytical price for the zero-coupon bond is built in order to reflect the value of this deterministic shift; therefore, by calibrating the model to swaptions prices or caps prices, the resulting parameters are able to capture also to discount factor curve observed in the market.

Nevertheless, we still have to define the value of the instantaneous forward rate, $f^M(0,T)$ to build the model deterministic shift. As already explained in the second chapter, if the Svensson model is adopted to build the discount factor curve, the instantaneous forward will be given by:

$$f(t) = \alpha_1 + \alpha_2 e^{-\frac{t}{\beta_1}} + \alpha_3 \frac{t}{\beta_1} e^{-\frac{t}{\beta_1}} + \alpha_4 \frac{t}{\beta_2} e^{-\frac{t}{\beta_2}},$$

thus, no further efforts are required to implement the deterministic shift, apart from adding to $f(t)$ the expression with the remaining elements of the deterministic shift.

In this example, however, we have built the discount factor curve using the classical mono-curve bootstrapping procedure, also highlighted in the second chapter. Unfortunately, in this case, we do not have a parametrical form for our instantaneous forward, but Matlab can calculate the instantaneous forward from the following formula

$$f^M(0,t) = -\frac{\partial ln P^M(0,t)}{\partial t},$$

while taking into account also the shape of the discount factor curve resulting from the shape preserving piecewise cubic interpolation used to interpolate discount factors for missing intermediate dates.

The polynomial form of the zero-curve is retrieved in Matlab by using the following function

```
ppPchip = interp1(CT,log(DF),'pchip','pp');.
```

Once we have our polynomial form, Matlab can also calculate the first derivative of a known polynomial function through the following code

```
dPM = fnder(ppPchip,1);.
```

Now, as already done for the bootstrapped zero-coupon curve, it is possible to define an anonymous function to retrieve the value of the instantaneous forward rate at each time T, by running this code

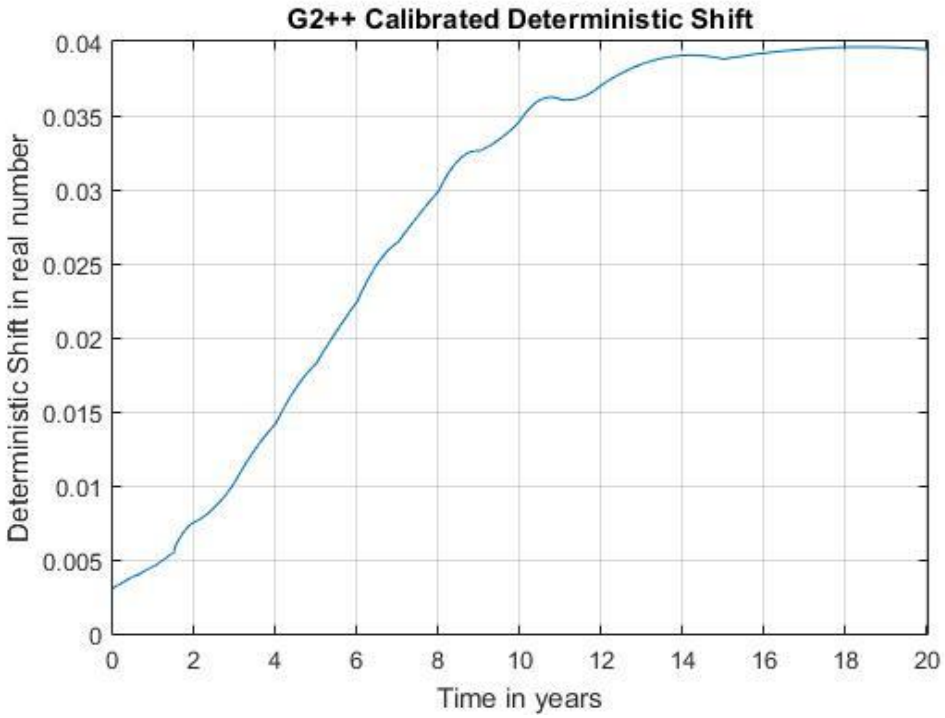
```
FM = @(t) -fnval(dPM,t);.
```

Once we have defined the instantaneous forward as an anonymous function, we can also completely define the deterministic shift as an anonymous function that returns the value of the model calibrated deterministic shift at each time T,

```
phit = @(t) FM(t) + sigma.^2./(2.*a.^2).*(1 - exp(-a.*t)) + ...
    eta.^2/(2.*b.^2).*(1 - exp(-b.*t)) + rho.*sigma.*eta./(a.*b) ...
    .*(1 - exp(-a.*t)).*(1 - exp(-b.*t));
```

In Fig. 15, the value of the calibrated deterministic shift is represented. It is easy to see that the value of the deterministic shift depends more from the value of the instantaneous forward rate in the left section of the curve, while it is governed more from the model parameters for the far right part of the curve.

Fig. 15: Calibrated Deterministic Shift

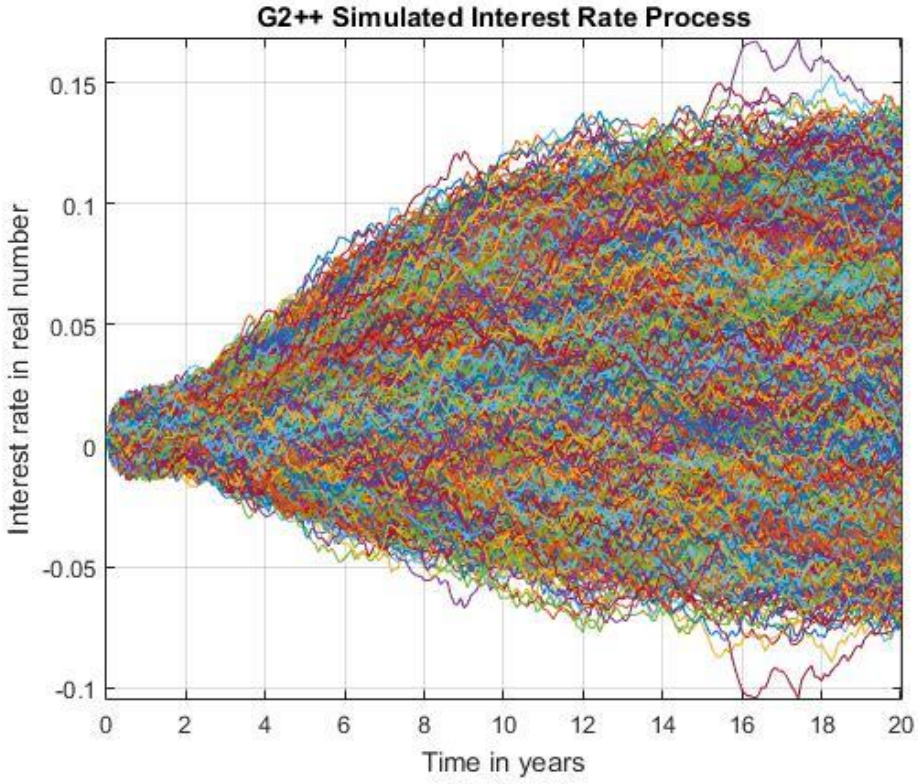


The short rate process, $r(t)$, is given by the sum of the two stochastic processes and the deterministic shift, namely:

$$r(t) = \varphi(t) + x(t) + y(t).$$

Fig. 16 depicts the 50'000 simulated interest rate paths used to calculate swaptions prices, each simulation step is approximately equal to one month, since each step has been adjusted in order

Fig. 16: Simulated Interest Rate Process



to take into account business holidays. However, the calculation of the deterministic shift may be avoided given that the model analytical solution for the zero-coupon bond does already include an adjustment to its value that takes into account the integral of the $\varphi(t)$ function.

5.3 A Practical Example

In order to see how the Monte Carlo simulation perform relative to a benchmark, which, in this particular case, is the price given by Bloomberg SWPM (SWaP Manager); a practical example is described in this paragraph. Thus, suppose that a contract between two counterparties (i.e. A is the client, while B is the bank) has to be priced, the contract has the following characteristics:

- The contract starts on the 28th February 2008; hence, data from Bloomberg on the discount factor curve and the volatility of both at-the-money caps and swaptions have been retrieved.
- The underlying asset of the contract is the 6 months Euribor and payments are made by using the rate observed two working days before the beginning of the reference period (i.e. "in advance").
- The day count convention used throughout the whole contract is act/360, thus the real number of days between cash flows and Euribor measurements has to be calculated.
- The second counterparty of the contract, the bank in this example, has to pay the 6 months Euribor.
- Until the 31 December 2011 (i.e. if the reference interest rate is measured before this date), the first counterparty has to pay the quantities contained in the table below, depending on the value of the 6 months Euribor rate.

Tab. 9: Payments Structure for Part A Until the 31 December 2011

Different cases	*Euribor value at measurement*	*Payment*
Case A	If the Euribor is less than 4.65%	Part A has to pay 4.05%
Case B	If the Euribor is between 4.65% and 5.00%	Part A has to pay the Euribor
Case C	If the Euribor is higher or equal to 5.00%	Part A has to pay 5.00%

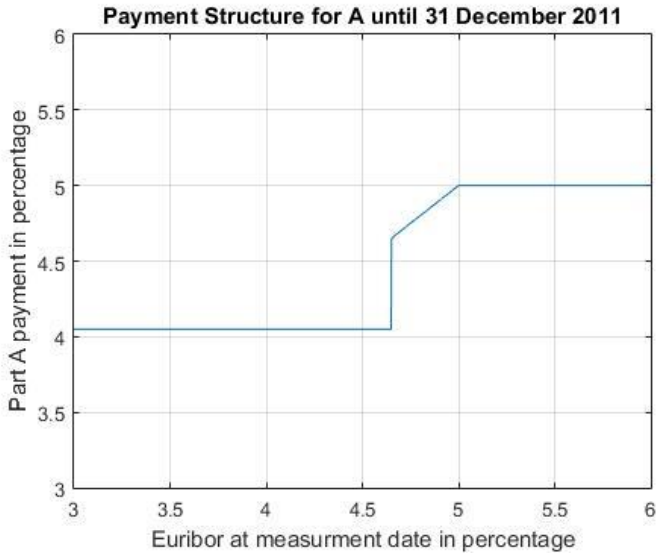
This part of the contract can be divided into a sum of vanilla contracts:

$$Payments_t^A = 4.05\% + (4.65\% - 4.05\%)Digital_{Cap}(4.65\%) + max(Libor_t - 4.65\%, 0) - \max(Libor_t - 5.00\%, 0).$$

Where a $Digital_{Cap}(K)$ is a contract that pays 0 if the interest rate is lower than the strike K, otherwise it pays 1.

In the below figure a plot of the payments structure of A for the first part of the contract is represented.

Fig. 17: Payments Structure for Part A Until 31 December 2011



- After the 31 December 2011, the contract payments for the first part change to the ones contained in the following table until the 31 December 2024, (i.e. the maturity date).

Tab. 10: Payments Structure for Part A Until Maturity

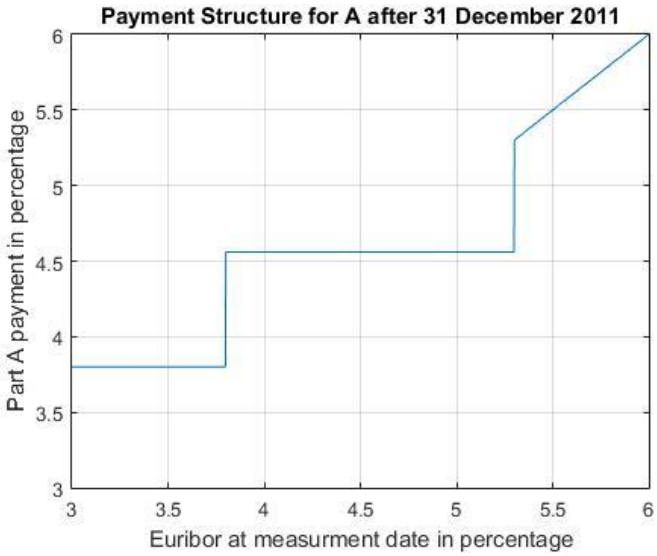
Different cases	Euribor value at measurement	Payment
Case D	If the Euribor is between 3.80% and 5.30%	Part A has to pay 4.56%
Case E	If the Euribor is higher or equal to 5.30%	Part A has to pay the Euribor
Case F	If the Euribor is lower or equal to 3.80%	Part A has to pay 3.80%

In addition, the second part of the contract for A can be divided into a series of vanilla contract:

$$Payments_t^A = 3.80\% + (4.56\% - 3.80\%)Digital_{Cap}(3.80\%) + (5.30\% - 4.56\%)Digital_{Cap}(5.30\%) + \max(Libor_t - 5.30\%, 0).$$

In the figure in the following page, also the payments structure of A for the second part of the contract is reported.

Fig. 18: Payments Structure for Part A Until Maturity



- Finally yet importantly, the contract has an amortized payments schedule (i.e. the cash flows notional tends to decrease as time goes on).

Once the model is calibrated to the at-the-money caps prices and to the discount factors resulting from the `DF_Bootstrapping` function and the interest rate is simulated by following the procedures highlighted in the previous chapter, it is possible to mark to market the value of the contract for Part A, which is given by:

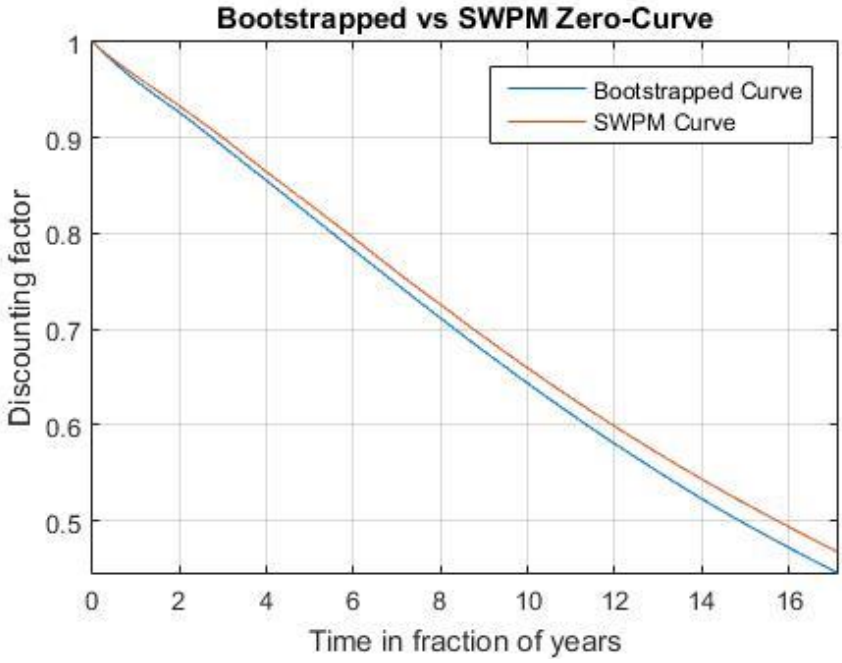
$$Value_A = Payments_t^B - Payments_t^A = -255{,}823.28€$$

The value of the contract is close enough to the one obtained by Bloomberg, which is equal to $-260{,}327.15€$. However, by looking more carefully into the Bloomberg cash flows, it is possible to notice that SWPM wrongly computes the first cash flow value by taking into consideration the value of the Euribor on the 18th February 2008, while the correct first measurement date is the 27 December 2007. Thereof, by adjusting the value of the first cash flow, which should be equal to zero, the SWPM price changes to $-272{,}390.03€$; thus, increasing the difference between the two results.

This 6.08% difference between the two prices can be explained by comparing the discount factors used by Bloomberg with the ones resulting from `DF_Bootstrapping`. In the next figure, the two zero-coupon curves are plotted and it can be clearly see that the curve used by SWPM is above the one resulting from our bootstrapping procedure, suggesting that Bloomberg is using the multi-curve approach to discount and simulate the interest rate.

Thus, the difference between the Bloomberg price and our model price can be attributed partially to the different curve used for discounting the cash flows and calibrating the G2++ model, given the impact on the model resulting parameters.

Fig. 19: Bootstrapped Zero-Curve vs SWPM Zero-Curve on the 28th February 2008



Another justification for the difference between the two prices is the different methodology used to calibrate the model by Bloomberg. Indeed, Bloomberg does not add the deterministic shifts, $\varphi(t)$, to the two random processes; instead, the shift is included by using piecewise constant volatility factors (i.e. σ and η) for the two model stochastic processes (i.e. $x(t)$ and $y(t)$). In order to implement the piecewise constant volatilities Bloomberg bootstrap the parameters along different instruments maturities, but this generate a fundamental difference between our model and the one implemented by Bloomberg.

On one hand, the distribution of the two processes does not change across the Monte Carlo simulation in our simulation, since the model parameters are assumed constant in our model. On the other hand, the Bloomberg parameters for the volatilities do change, effecting the covariance and variances of the two processes in the matrix used to generate random numbers in our simulation scheme (i.e. the Σ matrix on page 81); therefore, changing also the distribution of the two processes in the Bloomberg simulation scheme.

To conclude this example, the distribution of the simulated payoffs and the cash flows resulting from our Monte Carlo simulation are reported in the figures next page.

From the distribution of the payoffs, it is possible to see how the distribution is asymmetric and the client (Part A) is facing a substantial tail risk, while the bank does not face any substantial tail risk.

While the discounted cash flows figure highlights the effect of the amortization on the value of each cash flow.

Fig. 20: Discounted Payoff Distribution for A

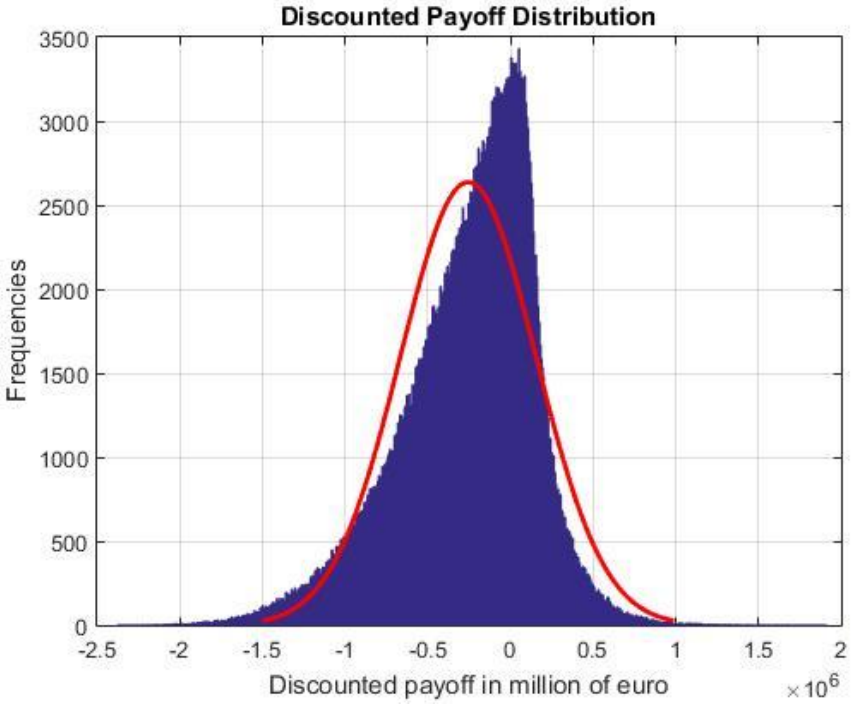
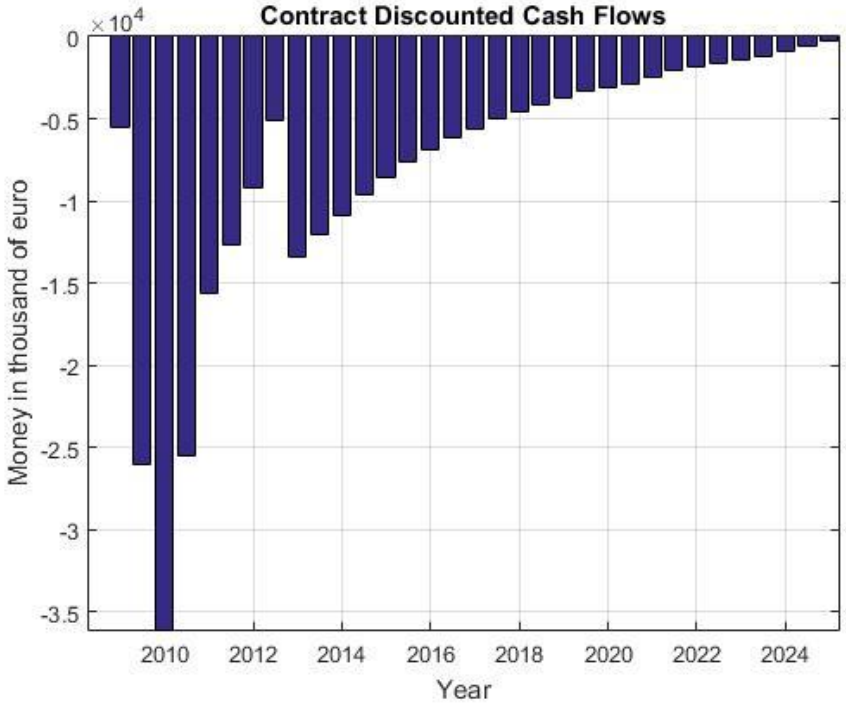


Fig. 21: Discounted Cash Flows for A



5.4 A Recap

In this chapter, an implementation of a Monte Carlo scheme for simulating the interest rate has been explained in details. Nevertheless, it is not a true simulation, given that the interest rate process transition density is known in full under the forward measure; thus, it is possible to extract different future realizations of the interest rate and to use discretional time steps in the simulation. Another great advantage of this procedure, is to avoid the calculation of the discount factors from the simulated trajectories of the interest process, since an analytical formula for the value of the zero coupon bond is also present in the model; therefore, only the values of the two stochastic processes, $x(t)$ and $y(t)$, are required in order to compute the zero coupon value, which already include the integral of the deterministic shift, $\varphi(t)$.

To check if our simulation scheme is correct, a contract between two counterparties has been priced. The resulting value is close to the one displayed by Bloomberg swap manager, but the implementation of the multi-curve framework and the particular calibration procedure used by Bloomberg, which assumes piecewise constant volatilities for both $x(t)$ and $y(t)$, generate a 6.08% difference between our simulation result and the one obtained by Bloomberg. However, despite this difference, our model is coherent to the market input data of the ATM caps volatilities, ATM swaptions and the zero-coupon curve built from SWDF by following the procedure highlighted in the second chapter of the present work.

To conclude, the value resulting in the present work diverges from the Bloomberg one, but this is not due to mistakes. Indeed, Bloomberg follows a different approach when building the G2++ model, as it uses the more updated multi-curve framework and bootstraps piecewise constant volatilities both from ATM caps and swaptions. However, this signals that the model should be updated to the more modern multicurve framework.

General Conclusion

The main objective of this thesis is to calibrate the Gaussian two-additive-factor interest rate model to the market observed volatility surface, and to generate a Monte Carlo simulation scheme in order to price contracts with more complex payoffs. However, the calibration procedure requires a series of trade-offs which can be summarized in the typical coding dilemma of having a computationally efficient code, or building a true global minimization algorithm.

In the present work, the trade-off between efficiency and efficacy has been solved by using a differential evolutionary algorithm, followed by a deterministic minimization algorithm that uses an interior-point solution. To avoid excessive computational time, the differential algorithm running time has been limited to twenty minutes; thus, limiting the likelihood of convergence to a global minimum.

The final model parameters obtained in the fourth chapter are able to fit the swaption market observed volatility surface with small errors across all the observations, with the exception of a few swaptions with maturities and tenors combinations equal or lower than two years. Nevertheless, as has been observed frequently after the 2007 crisis (see Moreni and Pallavicini (2010) and by Di Francesco (2012)) the model correlation parameter lies nearby the lower boundary, implying a degeneration of the G2++ model into a Gaussian one-factor interest rate model.

Before calibrating the G2++ model to the volatility surface, it is necessary to build a discount factor curve consistent with the one observed in the market. Therefore, in the second chapter, different methodologies for building the discount factor curve were discussed, more precisely: a classical mono-curve bootstrapping procedure and a Nelson-Siegel-Svensson model fitting were explained in details. Despite the no-arbitrage properties of the Nelson-Siegel model have been proven by Coroneo *et al.* (2011), the faster mono-curve bootstrapping methodology has been preferred, mainly because this procedure generates smaller errors between the resulting discount factors and the ones calculated by Bloomberg SWDF proprietary algorithm.

Once the model-calibrated parameters are known, an exact Monte Carlo simulation is implemented under the forward measure. As already discussed in the fifth chapter, simulating

the G2++ under the terminal measure is faster and easier than simulate the process under the risk neutral measure. Indeed, given that G2++ model has an analytical solution for the zero-coupon bond, and the transition density of the interest rate process is known under the forward measure, the time steps in the simulation can be chosen freely without generating consequences on the discount factors used to find the present values of future payoffs.

In the end, despite the degeneration of the model, the algorithm presented in this thesis is able to price correctly the majority of the swaptions contained in the volatility matrix on the 18th August 2014; however, in order to fully capture present market conditions, a multi-curve framework may result more efficient and may not degenerate to a one-factor model.

At the end of the fifth chapter, the model has been used to price an agreement between to counterparties, after calibrating the G2++ model to ATM caps data on the 21st February 2008. The resulting correlation parameter is not close to -1, thus the G2++ does not degenerate into a one-factor model; however, the price obtained shows a small difference in relation to the one calculated by Bloomberg. The main reasons behind this difference in the two prices have been briefly explained in the fifth chapter, which confirmed that our model is consistent with the market data, but has the strong limitation of being consistent only under the mono-curve framework, while Bloomberg uses the multi-curve framework in its proprietary algorithm to price contracts (i.e. SWPM).

Moreover, Bloomberg uses piecewise constant volatility parameters and, rather than minimizing the difference between all the market data in a single function, it implements a bootstrapping procedure along different maturity dates for both caps and swaptions, generating a higher fit to the market data.

In conclusion, the G2++ model presented in this thesis is a great tool for pricing interest rate derivatives under the mono-curve framework, and is consistent with the market data. Nevertheless, the current adoption of the multi-curve framework by the market has to be addressed by also adapting the G2++ model to this new framework. Grasselli and Miglietta (2014) have made some progress in this direction. However, in order to calibrate the model to the multi-curve framework, the procedure highlighted in the present work has to be partially updated, requiring further studies.

List of Tables and Figures

Figures

Tables

Appendix

To better understand the implementation of the G2++ model and its calibration, the main functions used in this thesis are reported in this Appendix as written in Matlab, including their respective commentary.

0. The Main Matlab Function

In this part, the function used to run the whole code is reported.

```
%% Clear, load data and set global variable.

% Clear.
clear all
clc

% Upload data from excel.
run DataLoader.m

%% Calculate discounting function.

% Set minimization to differential evolutionary for calibrating the
% Svensson model if, Curve is set to Boot.
Opt = 'de';

% Switch among bootstrapping and Svesson model.
switch Curve
    case 'Boot'
        % Bootstrap the discounting factor curve from market data.
        [CT,DF,R] = DF_Bootstrapping(Mar_Rate,Mat_Date,Set,Type,Res_IRS);
        % Interpolate the zero-coupon curve for plotting.
        PM = @(t) interp1(CT,DF,t,'pchip','extrap');
        % Return the piece-wise polynomial form corresponding to the method
        % used to interpolate (i.e. it is used to generate the first
        % derivative of the log of the discounting factors).
        pp = pchip(CT,-log(DF));
        % Find the derivatives.
        dPM = fnder(pp,1);
        % Find the value of the instantaneous forward from the derivative
        % structure having the polynomial form implied by the interpolation
        % of the discounting factors.
        FM = @(t) fnval(dPM,t);
        % Set x as empty variable to avoid errors due to missing variable
        % in future functions.
        sx = [];
    case 'Svensson'
        % Calibrate the Svesson model to discounting factors observed in
        % the market.
        [sx,fvalSvensson] = DF_Fitting(Set,Mar_Rate,Mat_Date,Type,Opt);
        % Calculate an anonymous function that, using the calibrated
        % parameters, will give the value of the discounting factor at each
        % time "t".
        PM = @(t) exp(-t.*sx(1) - sx(5).*(sx(2) + sx(3)).*(1 - ...
            exp(-t./sx(5))) + sx(3).*t.*exp(-t./sx(5)) - ...
            sx(4).*sx(6).*(1 - exp(-t./sx(6))) + ...
            sx(4).*t.*exp(-t./sx(6)));
```

```
        % Generate another anonymous function for the instantaneous forward
        % rate, using the Svensson specification (i.e. it is the
        % derivative with respect to "t" of the minus log of the above
        % function "-log(PM)").
        FM = @(t) sx(1) + sx(2).*exp(-t./(sx(5))) + sx(3).*(t./sx(5)) ...
            .*exp(-t./sx(5)) + sx(4).*(t./sx(6)).*exp(-t./sx(6));
        % Set CT and DF as empty variables to avoid errors due to missing
        % variables in the functions belows.
        CT = [];
        DF = [];
    otherwise
        % In case of wrong input.
        display('Check Curve variable')
        return
end

% Generate a set of times between 0 and 50 (i.e. used to plot the above
% discounting factors curve or forward curve, write
% plot(Ct,PM(Ct),Ct,FM(Ct)) to see a plot of both curves.
Ct = linspace(0,50,100000);


%% Calculate strikes and market Black prices for caps and swaptions.

% Calculate Cap strikes.
X_Cap = CF_Strk(Mat_Cap,Res_Cap,Set,CT,DF,sx,Curve);

% Calculate Cap Black market prices.
Mar_Price_Cap = CapFloor_BL(X_Cap,Mat_Cap,Res_Cap,Not_Cap,Mar_Vol_Cap,...
    OptSpec_Cap,Set,CT,DF,sx,Curve);

% Calculate ATM strikes for European swaptions.
X_Swap = ES_Strike(Ten_Swap,Mat_Swap,Res_Swap,Set,CT,DF,sx,Curve);

% Calculate Black prices from market prices.
Mar_Price_Swap = ES_Black(Not_Swap,Mat_Swap,Ten_Swap,OptSpec_Swap,...
    Mar_Vol_Swap,Set,CT,DF,sx,Curve);

%% Bootstrap caplet volatilities

% Bootstrapping caplet by minimizing the difference between caps
% prices calculated using a flat vol and the sum of underlying caplets
% calculated with different spot volatilities.
[Mar_Vol_Caplet,Mat_Caplet,X_Caplet,Not_Caplet,OptSpec_Caplet,AdjTi] ...
    = Caplet_MissingData(Mar_Vol_Cap,Res_Cap,Not_Cap,Mat_Cap,...
    OptSpec_Cap,Set,CT,DF,sx,Curve);

%% Calculate the forward rates.

% Find the forward rate for each caplet.
tau = diff(AdjTi);
Forward = 1./tau.*(PM(AdjTi(1:(end-1)))./PM(AdjTi(2:end)) - 1);

% Find the market prices for the caplets.
Mar_Price_Caplet = Caplet_BL(Forward,Forward,Mar_Vol_Caplet,AdjTi,...
    OptSpec_Caplet,DF,CT,sx,Curve,Not_Caplet);

%% Calibrate the G2++ model using volatilities.

% Use differential evolutionary.
Cal_Criteria = 'DifferentialMin';
```

```
% Calibrate the model.
[P, fval] = G2_Calibration(Cal_Criteria,X_Swap,X_Cap,X_Caplet,...
    Mat_Swap,Mat_Cap,Mat_Caplet,Ten_Swap,Res_Swap,Res_Cap,Not_Swap,...
    Not_Cap,Not_Caplet,OptSpec_Swap,OptSpec_Cap,OptSpec_Caplet,Set,CT,...
    DF,sx,Curve,Mar_Vol_Swap,Mar_Vol_Cap,Mar_Vol_Caplet,Mar_Price_Swap,...
    Mar_Price_Cap,Mar_Price_Caplet,AdjTi,P0,Obj,Method,InstrumentSet);

% Avoid a random shift in the optimization parameters if one lies on the
% boundaries.
lb = [eps eps 0 0 -1];
ub = [1 1 0.5 0.5 1];
for j = 1:length(P)
    if P(j) == lb(j)
        P(j) = P(j) + 1/1000;
    elseif P(j) == ub(j)
        P(j) = P(j) - 1/1000;
    end
end

%% Refine results.

% Use gradient minimization to refine the previous result.
Cal_Criteria = 'GradientMin';

% Refine the results.
[PRef, fvalRef] = G2_Calibration(Cal_Criteria,X_Swap,X_Cap,X_Caplet,...
    Mat_Swap,Mat_Cap,Mat_Caplet,Ten_Swap,Res_Swap,Res_Cap,Not_Swap,...
    Not_Cap,Not_Caplet,OptSpec_Swap,OptSpec_Cap,OptSpec_Caplet,Set,CT,...
    DF,sx,Curve,Mar_Vol_Swap,Mar_Vol_Cap,Mar_Vol_Caplet,Mar_Price_Swap,...
    Mar_Price_Cap,Mar_Price_Caplet,AdjTi,P,Obj,Method,InstrumentSet);

%% Generate random numbers.

% Set the time horizon of the simulation.
THor = Mat_Cap(end);

% Set the time of year of the simulation.
PYear = 2;

% Set the number of paths.
PathN = 50000;

%% Simulate the interest rate.

% Build the simulation of the interest rate using the calibrated
% parameters.
[r,xx,yy] = XY_Sim(PRef(1),PRef(2),PRef(3),PRef(4),PRef(5),THor,PYear,...
    PathN,CT,DF,sx,Curve,Set);

% Redefine results for the phit function.
a = PRef(1);
b = PRef(2);
sigma = PRef(3);
eta = PRef(4);
rho = PRef(5);

% Build the phit function.
phit = @(t) FM(t) + sigma.^2./(2.*a.^2).*(1 - exp(-a.*t)) + ...
        eta.^2/(2.*b.^2).*(1 - exp(-b.*t)) + rho.*sigma.*eta./(a.*b) ...
        .*(1 - exp(-a.*t)).*(1 - exp(-b.*t));
```

```
%% Test for Monte Carlo errors in the Simulation of European Swaptions.

% Calculate the Monte Carlo prices.
ES_Sim = ES_MC(PRef(1),PRef(2),PRef(3),PRef(4),PRef(5),X_Swap,Mat_Swap,...
    Ten_Swap,Res_Swap,OptSpec_Swap,xx,yy,PYear,THor,CT,DF,sx,Curve,Set);

% Calculate the analytical prices.
ES_Ana = ES_G2(PRef(1),PRef(2),PRef(3),PRef(4),PRef(5),X_Swap,Mat_Swap,...
    Ten_Swap,Res_Swap,Not_Swap,OptSpec_Swap,Set,CT,DF,sx,Curve,Method);

%% Calculate the calibration error in sample.

% Calculate the model implied volatility.
Mod_Vol_Swap = ES_ImpVol(Mat_Swap,Ten_Swap,Not_Swap,OptSpec_Swap,...
    ES_Ana,Set,CT,DF,sx,Curve);

% Calculate the percentage error.
CalErrorPer_Swap = 100.*(Mod_Vol_Swap - Mar_Vol_Swap)./Mar_Vol_Swap;

%% Calculate the Monte Carlo error in sample.

% Calculate the percentage difference.
MonErrorPer_Swap = 100.*(ES_Sim - ES_Ana)./ES_Ana;

%% Cap out of sample.

% Calculate the model prices.
Cap_Ana = CapFloor_G2(PRef(1),PRef(2),PRef(3),PRef(4),PRef(5),X_Cap,...
    Mat_Cap,Res_Cap,Not_Cap,OptSpec_Cap,Set,CT,DF,sx,Curve);

Mod_Vol_Cap = CF_ImpVol(Cap_Ana,OptSpec_Cap,Mat_Cap,X_Cap,Res_Cap,...
    Not_Cap,Set,CT,DF,sx,Curve);

% Error Cap.
CalErrorPer_Cap = 100.*(Mod_Vol_Cap - Mar_Vol_Cap)./Mar_Vol_Cap;

%% Compare Monte Carlo to analytical prices for caps.

% Calculate the Monte Carlo Price for caps.
Cap_Mon = CF_MC(PRef(1),PRef(2),PRef(3),PRef(4),PRef(5),X_Cap,Mat_Cap,...
    Res_Cap,OptSpec_Cap,Not_Cap,xx,yy,PYear,THor,Set,CT,DF,sx,Curve);

% Calculate the Monte Carlo error.
MonErrorPer_Cap = 100.*(Cap_Mon - Cap_Ana)./Cap_Ana;

%% Simulate x, y and r for calculating the value of the contract.

PathN = 400000;

% Generate the interest rate process for pricing the contract.
[r_P,x_P,y_P] = XY_SimPri(PRef(1),PRef(2),PRef(3),PRef(4),PRef(5),...
    PathN,CT,DF,sx,Curve,Sim_Dates,Set_Con);

%% Calculate the price.

% Calculate the price.
[MatrUnd_A,MatrUnd_B,MatrDis_A,MatrDis_B,Matr_A,Matr_B,Price] = Price1(...
    PRef(1),PRef(2),PRef(3),PRef(4),PRef(5),Not_Con,x_P,y_P,Sim_Dates,...
```

```
    Lib_Dates,CF_Dates,Set_Con,CT,DF,sx,Curve);
```

1. Loading the Data from Excel

This function loads the data form a set of Excel files, containing data on the volatility surface of caps and swaptions, the amortization schedule of the contract and the interest rate data from SWDF.

```
%% ATM Caps data.

% Load ATM Caps volatilities.
Mar_Vol_Cap = xlsread('CapVols.xlsx','Foglio1','B2:B16')./100;

% Load ATM Caps maturity dates.
[~,AdjTi] = xlsread('CapVols.xlsx','Foglio1','A2:A16');
Mat_Cap = str2double(strrep(AdjTi,'Yr',''));

% Build the ATM Caps notional matrix.
Not_Cap = ones(size(Mar_Vol_Cap));

% Build the ATM Caps reset matrix.
Res_Cap = 2.*ones(size(Mar_Vol_Cap));

% Build the contract specification matrix for ATM Caps.
OptSpec_Cap = repmat({'cap'},size(Mar_Vol_Cap,1),1);

%% OTM Caps data.

% Load OTM Caps volatilities.
Mar_Vol_OTM = xlsread('CapVols.xlsx','Foglio1','C2:S16')./100;

% Load OTM Caps strikes.
[~,AdjTi] = xlsread('CapVols.xlsx','Foglio1','C1:S1');
X_OTM = str2double(strrep(AdjTi,'%',''))./100;
X_OTM = repmat(X_OTM,size(Mar_Vol_OTM,1),1);

% Build the OTM Caps maturity matrix and transform the matrixes into
% vectors.
Mat_OTM = repmat(Mat_Cap,1,size(Mar_Vol_OTM,2));

% Build the OTM Caps notional matrix.
Not_OTM = ones(size(Mar_Vol_OTM));

% Build the OTM Caps reset matrix.
Res_OTM = 2.*ones(size(Mar_Vol_OTM));

% Build the contract specification matrix for OTM Caps.
OptSpec_OTM = repmat({'cap'},size(Mar_Vol_OTM));

% Transform matrixes into vectors for OTM Caps.
Mar_Vol_OTM = deal(Mar_Vol_OTM(:));
X_OTM = deal(X_OTM(:));
Mat_OTM = deal(Mat_OTM(:));
Not_OTM = deal(Not_OTM(:));
Res_OTM = deal(Res_OTM(:));
OptSpec_OTM = deal(OptSpec_OTM(:));
```

```
%% ATM Swaptions data.

% Load ATM Swaptions volatilities.
Mar_Vol_Swap = xlsread('DatiSwap.xlsx','Foglio1','B2:K15')./100;

% Load ATM Swaptions maturities.
[~,Raw_Maturity] = xlsread('DatiSwap.xlsx','Foglio1','A2:A15');
Mat_Swap = zeros(size(Raw_Maturity));
for j = 1:length(Raw_Maturity)
    if ne(cell2mat(strfind(Raw_Maturity(j),'Mo')),0)
        Mat_Swap(j) = str2double(strrep(Raw_Maturity(j),'Mo',''))./12;
    else
        Mat_Swap(j) = str2double(strrep(Raw_Maturity(j),'Yr',''));
    end
end
Mat_Swap = repmat(Mat_Swap,1,size(Mar_Vol_Swap,2));

% Load ATM Swaptions tenors.
[~,AdjTi] = xlsread('DatiSwap.xlsx','Foglio1','B1:K1');
Ten_Swap = str2double(strrep(AdjTi,'Yr',''));
Ten_Swap = repmat(Ten_Swap,size(Mar_Vol_Swap,1),1);

% Build the ATM Swaptions notional matrix.
Not_Swap = ones(size(Mar_Vol_Swap));

% Build the ATM Swaptions reset matrix.
Res_Swap = ones(size(Mar_Vol_Swap));

% Build the contract specification matrix for ATM Swaptions.
OptSpec_Swap = repmat({'receiver'},size(Mar_Vol_Swap));

% Transform ATM Swaptions matrices into vectors.
Mar_Vol_Swap = deal(Mar_Vol_Swap(:));
Mat_Swap = deal(Mat_Swap(:));
Ten_Swap = deal(Ten_Swap(:));
Not_Swap = deal(Not_Swap(:));
Res_Swap = deal(Res_Swap(:));
OptSpec_Swap = deal(OptSpec_Swap(:));

%% Load discounting factors.

% Set the settlement date.
Set = cellstr('21-Feb-2008');

% Load discounting factors maturity in fraction of year and transform them
% into maturity dates.
[Term,Unit] = xlsread('DatiSwap.xlsx','Foglio1','A21:B57');
Year_Fraction = zeros(size(Term));
for j = 1:length(Term)
    if strcmp(Unit(j),'DY')
        Year_Fraction(j) = Term(j)/360;
    elseif strcmp(Unit(j),'MO')
        Year_Fraction(j) = Term(j)/12;
    else
        Year_Fraction(j) = Term(j);
    end
end

% Load day count convention.
[~,Daycount] = xlsread('DatiSwap.xlsx','Foglio1','L21:L57');
Conv = zeros(size(Daycount));
```

```
for j = 1:length(Daycount)
    if strcmp(Daycount(j),'ACT/360')
        Conv(j) = 2;
    else
        Conv(j) = 5;
    end
end

% Transform year fractions into serial date.
Mat_Date = repmat(cell(0),size(Year_Fraction));
for j = 1:length(Year_Fraction)
    AdjTi = datestr(daysadd(datenum(Set),Year_Fraction(j)*360,Conv(j)));
    Busday = isbusday(AdjTi);
    while Busday == 0
        AdjTi = daysadd(datenum(AdjTi),1,Conv(j));
        Busday = isbusday(AdjTi);
    end
    Mat_Date(j) = cellstr(datestr(AdjTi));
end
Mat_Date = Mat_Date';

% Load Bid-Ask spread and calculate the mid rate.
BidAsk = xlsread('DatiSwap.xlsx','Foglio1','D21:E57')./100;
Mar_Rate = (BidAsk(:,2) + BidAsk(:,1))./2;

% Load instrument type and transform into instrument type recognized by
% DF_Bootstrapping.
[~,Raw_Type] = xlsread('DatiSwap.xlsx','Foglio1','K21:K57');
Type = repmat(cell(0),size(Raw_Type));
for j = 1:length(Raw_Type)
    if strcmp(Raw_Type(j),'Cash Rates') || strcmp(Raw_Type(j),'Deposits')
        Type(j) = cellstr('Deposit');
    elseif strcmp(Raw_Type(j),'Serial FRAs')
        Type(j) = cellstr('FRA');
    elseif strcmp(Raw_Type(j),'Swap Rates')
        Type(j) = cellstr('IRS');
    else
        display('Type of instrument for bootstrapping not recognized')
        break
    end
end
Type = Type';

% Build the Interest Rate Swaps reset matrix.
AdjTi = double(strcmp(Type,'IRS'));
AdjTi(AdjTi == 0) = [];
Res_IRS = AdjTi;

%% Clear useless stuff.

clear BidAsk
clear Busday
clear Conv
clear Daycount
clear j
clear Raw_Type
clear Raw_Maturity
clear Temp
clear Term
clear Unit
clear Year_Fraction
```

```
%% Add P0, Curve, Method and InstrumentSet.

% Starting point for the gradient minimization if adopted as the first
% algorithm.
P0 = [0.773511777 0.082013014 0.022284644 0.010382461 -0.701985206];

% Curve bootstrapping method.
Curve = 'Boot';

% Method for calculating the ATM Swaptions prices.
Method = 'SP';

% Instrument set.
InstrumentSet = 'Cap';

% Set objective.
Obj = 'Price';

%% Build the notional matrix for the contract under analysis.

% Set the beginning of the contract.
Set_Con = cellstr('21-Feb-2008');

% Find the date of beginning and end of each amortization period.
Dates = xlsread('ContractAmm.xlsx','Foglio1','A2:B36');
Beg_Date = cellstr(datestr(x2mdate(Dates(:,1))));
End_Date = cellstr(datestr(x2mdate(Dates(:,2))));

% Load the notional amount from the ammortizing table.
Not_Amm = xlsread('ContractAmm.xlsx','Foglio1','C2:C36');

% Calculate the time intervals of the simulation adjusting each cash flow
% time.
AdjTi = datenum([Beg_Date(2:end);End_Date(end)]);
for j = 1:length(AdjTi)
    Busdate1 = isbusday(AdjTi(j));
    while Busdate1 == 0
        AdjTi(j) = daysadd(AdjTi(j),-1,2);
        Busdate1 = isbusday(AdjTi(j));
    end
    count = 0;
    while count < 2
        AdjTi(j) = daysadd(AdjTi(j),-1,2);
        Busdate2 = isbusday(AdjTi(j));
        if Busdate2 == 1
            count = count + 1;
        end
    end
end

% Transform date number into strings.
Lib_Dates = cellstr(datestr(AdjTi));
CF_Dates = End_Date(2:end);

%% Calculate each cash flow notional.

% Calculate the corresponding cash flow.
Not_Con = zeros(size(CF_Dates));
for j = 1:length(CF_Dates)
    k = 1;
    Temp = isbetween(datetime(CF_Dates(j)),datestr(Beg_Date(k)),...
```

```
        datestr(End_Date(k)));
    Not_Con(j) = Not_Amm(k);
    while Temp ~= 1
        k = k + 1;
        Temp = isbetween(datetime(CF_Dates(j)),datestr(Beg_Date(k)),...
             datestr(End_Date(k)));
        Not_Con(j) = Not_Amm(k);
    end
end

% Combine cash flows and simulation times.
Temp = [Lib_Dates; CF_Dates];
Sim_Dates = sort(datenum(Temp),'ascend');
Sim_Dates(1) = [];

%% Clear useless stuff.

clear temp
clear Busdate1
clear Busdate2
clear Beg
clear j
clear k
clear Dates
clear count
clear Temp
```

2. Bootstrapping the Discount Factors

This function bootstraps the value of the discount factors following the procedure highlighted in the second Chapter.

```
function [CT,DF,R] = DF_Bootstrapping(Mar_Rate,Mat_Date,...
    Set,Type,Res_IRS)

% DF_BOOTSTRAPPING bootstrap the zero coupon value from market data.
%  Description: takes data from Bloomberg SWDF and find the discounting
%   factor  curve, taking into account each instrument used for
%   bootstrapping the curve in Bloomberg.
%  Input:
%   - Mar_Rate: middle rate from Bloomberg SWDF;
%   - Mat_Date: maturity date of the instrument from Bloomberg SWDF;
%   - Set: settlement date from Bloomberg SWDF (i.e. today plus two working
%     days);
%   - Type: instruments' type (i.e. 'Deposit', 'FRA', 'IRS');
%   - Res: payments in a year by the Interest Rate Swap (set equal to one,
%     i.e. one per year).
% OutPut:
%   - CT: year fraction from Set to Mat_Date (calculated using the
%     instrument specific day count convention);
%   - DF: zero-coupon bond value at time CT;
%   - R: zero-coupon rate value at time CT.

% Transform each maturity date and the settlement date string into a
% number.
DateNum = datenum(Mat_Date);
SettleNum = datenum(Set);
```

```
% Generate empty matrix for the year fractions between cash flows and the
% discounting factors.
YearFraction = zeros(length(DateNum),1);
DF = zeros(length(DateNum),1);

% Add at the beginning of reset times matrix for Interest Rate Swaps (IRS)
% a number of rows equals to the number of Deposits and Forward Rate
% Agreements (FRA). This passage is needed during the loop for
% bootstrapping, so we can use the new matrix with the same index of the
% loop.
Temp = zeros((length(DateNum) - length(Res_IRS)),1);
FullReset = [Temp; Res_IRS];

% Start the bootstrapping loop.
for j = 1:length(DateNum)
    % The if loop divides instruments in categories and apply the right
    % formula for retrieving the value of the zero coupon bond.
    % If the instrument is a deposit (Deposit).
    if strcmp(Type(j),'Deposit')
        % Calculate the year fraction using actual/360 convention between
        % the settlement date and the maturity of the deposit.
        YearFraction(j) = daysdif(SettleNum,DateNum(j),2)./360;
        % Inverted formula n. 1.8 from Brigo and Mercurio, pag. 7, to find
        % the value of the zero coupon bond.
        DF(j) = 1/(1 + YearFraction(j)*Mar_Rate(j));
    % In case the contract is a Forward Rate Agreement (FRA).
    elseif strcmp(Type(j),'FRA')
        % Calculate the year fraction using actual/360 convention between
        % the settlement date and the maturity of the FRA.
        YearFraction(j) = daysdif(SettleNum,DateNum(j),2)./360;
        % Calculate the year fraction in between the maturities of two
        % consecutive zero-coupon bond (i.e. in this case is equal to 6
        % months approximately).
        tau = daysdif(DateNum(j-1),DateNum(j),2)./360;
        % Invert formula n. 1.20 from Brigo and Mercurio, pag. 12, to find
        % the value of the forward zero coupon bond at the previous
        % zero-coupon bond maturity. Hence, the value is multiplied by the
        % zero-coupon value at the previous point in the curve in order to
        % find the value of the zero-coupon bond from the forward
        % zero-coupon bond.
        DF(j) = (1/(1 + tau*Mar_Rate(j)))*DF(j-1);
    % In case the contract is a Interest Rate Swap (IRS).
    elseif strcmp(Type(j),'IRS')
        % Calculate the year fraction between today and the maturity of the
        % IRS using the 30/360 SIA convention.
        YearFraction(j) = daysdif(SettleNum,DateNum(j),5)./360;
        % Round the year fraction to an integer number.
        T = round(YearFraction(j));
        % Calculate the time grid of cash flows in months, using the
        % FullReset matrix.
        ti = (0:(1/FullReset(j)):T)*360;
        % Generate an empty matrix to be filled with the adjusted value for
        % each ti taking unto account holidays.
        AdjTi = zeros(size(ti));
        % Loop to adjust each ti.
        for jj = 1:length(ti);
            % Add to the settlement date ti, which is expressed in months.
            AdjTi(jj) = daysadd(SettleNum,ti(jj),5);
            % Check if AdjTi is a business day.
            BusdateAdjTi = isbusday(AdjTi(jj));
            % While loop for adjusting AdjTi until the next working day.
            while BusdateAdjTi == 0
                AdjTi(jj) = daysadd(AdjTi(jj),1,5);
```

```
                BusdateAdjTi = isbusday(AdjTi(jj));
            % End the while loop, given that, if AdjTi is a business day
            % from the start, there is no need for further operations.
            end
        % End the loop for moving through the ti vector.
        end
        % Set ti equals to its adjusted value.
        ti = AdjTi;
        % Calculate the year fractions between each ti using 30/360 (SIA)
        % convention.
        ti = daysdif(SettleNum,ti,5)./360;
        % Set the last ti equal to the Bloomberg SWDF maturity.
        ti(end) = YearFraction(j);
        % Calculate the adjusted year fraction between cash flows.
        tau = diff(ti);
        % Eliminate the first column of ti to match tau dimensions.
        ti(1) = [];
        % Transpose both ti and tau.
        ti = ti';
        tau = tau';
        % Temporary discounting factor interpolation in order to find the
        % value of the zero coupon bond for all cash flows dates.
        TemporaryDF = @(t) interp1(YearFraction(1:(j-1)),DF(1:(j-1)),t,...
            'pchip','extrap');
        % Inverted formula n. 1.25 from Brigo and Mercurio, pag. 15, to
        % find the value of the zero coupon bond.
        DF(j) = (1 - Mar_Rate(j)*(sum(bsxfun(@times,tau(1:(end-1)),...
            TemporaryDF(ti(1:(end-1)))))))/(1 + tau(end)*Mar_Rate(j));
    % Check for wrong Type entries.
    else
        % Display an error message in the command window.
        display('Error this type of instrument is not recognized')
        % Stop the function from continuing the loop.
        return
    % End the if statement.
    end
% End the loop that moves through different data points.
end

% Find the value of the continuous interest rate from the zero-coupon
% curve.
R = -bsxfun(@rdivide,log(DF),YearFraction);
% Add the value at time zero for the continuous interest rate (not
% necessary, but graphically pleasing when plotting the curve).
R = [ 0; R];

% Add the value at time zero for the time grid CT and for the zero-coupon
% curve.
CT = [ 0; YearFraction];
DF = [ 1; DF];

% End the function.
end
```

3. Calibrating the Svensson Model

The following algorithm calibrates the Svensson model to the market data, following the procedure explained in the second Chapter.

```
function [sx,fval] = DF_Fitting(Set,Mar_Rate,Mat_Date,Type,Opt)

% DF_FITTING fit the Svensson model to market rate data
% Description: minimize the error function (i.e. the squared difference
%  between model and market bootstrapped discounting factors) in order
%  to fit a Nelson-Siegel-Svensson model to the observed market interest
%  rates.
% Inputs:
%  - Set: settlement date;
%  - Mar_Rate: market rate from Bloomberg in real number;
%  - Mat_Date: maturity date from Bloomberg in real number;
%  - Type: type of market instruments (i.e. 'Deposits', 'FRA' or 'IRS');
%  - Opt: type of minimization algorithm (i.e. 'ga' or 'de').
% Outputs:
%  - sx: parameters.

% Set global variables.
global TT
global P
global AA

% Add missing swaps between date by interpolating linearly.
[Mar_R,Mat_D,Ty] = missing_Swap(Mar_Rate,Mat_Date,Type,Set);

% From raw adjusted data to adjusted matrixes and vectors.
[P,T,A] = matrix_vector_LS(Set,Mar_R,Mat_D,Ty);

% Create matrices.
[TT,AA,~] = matrix_vector(T,A);

% Set options for fminsearch.
options = optimset('TolFun',1e-9,'TolX',1e-9,'MaxIter',1e9,...
    'MaxFunEvals',1e9);

switch Opt
    % Use ga and then fminsearch.
    case 'ga'

        % Set options for ga. This algorithm already includes a refinement
        % of the result from ga, as one can notice from the 'HybridFcn'
        % options.
        optionsX = gaoptimset('Generation',600,'PopulationSize',1800,...
            'TolFun',1e-9,'HybridFcn',{@fminsearch,options},'Display',...
            'iter');

        % Run ga.
        [sx,fval] = ga(@SvenssonObj,6,[],[],[],[],[],[],[],optionsX);

    % Use de and then fminsearch.
    case 'de'

        % Define the title of the optimization.
        optimInfo.title = 'Svesson Calibration';

        % Define the function to optiCurveTimesmize.
        objFctHandle = @SvenssonObj;

        % Define parameters and range.
        paramDefCell = {'',[0 10; -10 10; -10 10; -10 10; 0 10; 0 10], ...
            [0; 0; 0; 0; 0; 0], [0.020328908307512; -0.014832363988668; ...
            -2.876787243252884; 2.862163497180667; 4.005463294399057; ...
```

```
            4.042529049072074]};

        % 1. column: parameter names
        % 2. column: parameter ranges
        % 3. column: parameter quantizations (i.e. 0 if not present)
        % 4. column: initial values (optional)

        % No other setting needed.
        objFctSettings = {};

        % No parameter vector needed.
        objFctParams = [];

        % Get default DE parameters.
        DEParams = getdefaultparams;

        % Set number of population members (often 10*D is suggested).
        DEParams.NP = 1800;

        % Do not use slave process here.
        DEParams.feedSlaveProc = 0;

        % Set times parameters for the optimization.
        DEParams.maxiter       = 1e9; % maximum number of iterations.
        DEParams.maxtime       = 60; % maximum running time in seconds.
        DEParams.maxclock      = [];

        % Set display options during the optimization.
        DEParams.refreshiter   = 0;
        DEParams.refreshtime   = 5;  % in seconds
        DEParams.refreshtime2  = 10;  % in seconds
        DEParams.refreshtime3  = 25;  % in seconds

        % Do not display final parameters or sound.
        DEParams.displayResults = 0;
        DEParams.playSound = 0;

        % Do not send E-mails with results.
        emailParams = [];

        % Start differential evolution using the algorithm developed by
        % Marku Buehren, freely available at Matlab Central.
        [x0] = differentialevolution(DEParams,paramDefCell,objFctHandle,...
            objFctSettings,objFctParams,emailParams,optimInfo);

        % Start fminsearch to refine the results, as in this case the
        % equation does not allow the inclusion of a hybrid function
        % inside the optimization algorithm.
        [sx,fval] = fminsearch(@(x) SvenssonObj(x),x0,options);

    % In case of wrong Opt value.
    otherwise

        % Dispaly an error message.
        display('Check Opt variable')

        % Stop the algorithm.
        return

% End of the switch statement.
```

```
end

% End of the function.
end
```

4. Svensson Model Objective Function

Minimizes the difference between the market and the model resulting discount factors.

```
function [f] = SvenssonObj(x)

% SVENSON objective function to minimize
% Description: calculate the value of the function to minimize in order to
%  calibrate the Svensson model.

global AA
global P
global TT

T = TT;
A = AA;

% Penalize the function if it does not satisfy parameters constraints.
if x(1) <= 0
    f = 10^15;
elseif (x(1) + x(2)) <= 0
    f = 10^15;
elseif x(5) <= 0
    f = 10^15;
elseif x(6) <= 0
    f = 10^15;
else
    % If the function satisfies all the constraints, calculate the error
    % function.
    for j = 1:length(T)
        t(j,1) = exp(-x(1)*T(j) - x(5)*(x(2) + x(3))*(1 - ...
            exp(-T(j)/x(5))) + x(3)*T(j)*exp(-T(j)/x(5)) - ...
            x(4)*x(6)*(1 - exp(-T(j)/x(6))) + x(4)*T(j)*exp(-T(j)/x(6)));
    end
    temp = P - A*t;
    f = 0;
    for j = 1:length(temp)
        f = f + temp(j)^2;
    end
end
```

5. Other Functions Used During the Svensson Calibration

Interpolates linearly between missing IRS data.

```
[Mar_R,Mat_D,Ty] = missing_Swap(Mar_Rate,Mat_Date,Type,Set)
```

Finds the zero coupon value, the date of payment and the face value for each instrument and stores the values in a matrix.

```
[P,T,A] = matrix_vector_LS(Set,Mar_R,Mat_D,Ty)
```

Computes the full matrix of payment dates and cash flows for each instrument.

```
[TT,AA,~] = matrix_vector(T,A);
```

6. G2++ Analytical Cap Price

Finds the price of a cap in the G2++ model.

```
function CF_G2 = CapFloor_G2(a,b,sigma,eta,rho,X,Mat,Res,Not,OptSpec,...
    Set,CT,DF,sx,Curve)

% CAPFLOOR_G2 Compute caps (floors) prices using the G2++ model analytical
%  formula.
% Description: compute caps prices for two-additive-factor Gaussian short
%  rate model given a zero-coupon bond curve and the model parameters (i.e.
%  a, b, sigma, eta and rho) using the analytical formula by Brigo and
%  Mercurio pag. 157/158 n. 4.29/4.30.
% Inputs:
%  - a: mean reversion of the first process;
%  - b: mean reversion of the second process;
%  - sigma: standard deviation of the first process;
%  - eta: standard deviation of the second process;
%  - rho: correlation among the two processes;
%  - X: strike of the option;
%  - Mat: maturity of the option;
%  - Res: number of caplets (floorlets) per year (i.e. 2, except for the
%    first year);
%  - Not: notional values of the option;
%  - OptSpec: type of option (i.e. 'cap' or 'floor');
%  - Set: settlement date of the contract;
%  - CT: time grid for our discounting factors;
%  - DF: value of our discounting factors for each time CT;
%  - sx: parameters of the Svensson model;
%  - Curve: set 'Svensson' for the Svensson model or 'Boot' for
%    bootstrapping.
% Output:
%  - CF_G2: caps (floors) prices for the G2 model.

% Create an interpolation between the discounting factors and their time
% grid as an anonymous function that gives values for zero-coupon bond
% depending on the value of t in PM(t).
switch Curve
    case 'Boot'
        PM = @(t) interp1(CT,DF,t,'pchip','extrap');
    case 'Svensson'
        PM = @(t) exp(-t.*sx(1) - sx(5).*(sx(2) + sx(3)).*(1 - ...
            exp(-t./sx(5))) + sx(3).*t.*exp(-t./sx(5)) - ...
            sx(4).*sx(6).*(1 - exp(-t./sx(6))) + sx(4).*t.*exp(-t./sx(6)));
    otherwise
        % In case of wrong input.
        display('Check Curve variable')
        return
end

% Function handle that generate the sigma function given by Brigo and
% Mercurio at pag. 155, as a function of the model parameters and of T and
% S, with S > T.
```

```
Sigma = @(T,S,a,b,sigma,eta,rho) sqrt((sigma^2/(2*a^3)).*(1 - exp(-a.*(S
...
    - T))).^2.*(1 - exp(-2*a*T)) + (eta^2/(2*b^3)).*(1 - exp(-b.*(S - ...
    T))).^2.*(1 - exp(-2*b*T)) + (2*rho*sigma*eta/(a*b*(a + b))).*(1 - ...
    exp(-a.*(S - T))).*(1 - exp(-b.*(S - T))).*(1 - exp(-(a + b).*T)));

% Create an empty matrix to be filled with model prices.
CF_G2 = zeros(size(Not));

% Start of the loop to find the model prices.
for j = 1:length(Not)
    % The if statement find the value of omega depending on the type of
    % contract.
    if strcmp(OptSpec(j),'cap')
        % If the contract is a cap omega is equal to 1.
        w = 1;
    elseif strcmp(OptSpec(j),'floor')
        % If the contract is a floor omega is equal to -1.
        w = -1;
    else
        % If the OptSpec value is different from 'payer' or 'receiver',
        % display a message and stop the loop.
        display('Input not recognized.')
        return;
    % End the if statement.
    end
    % Calculate days between each cash flows.
    ti = (0:(1/Res(j)):Mat(j))*360;
    % Generate an empty matrix of zeros with the same size of ti to be
    % filled with the business day adjusted cash flow dates.
    AdjTi = zeros(size(ti));
    % Start the loop for adjusting each cash flow with adjusting cash flows
    % date to match as much as possible the strikes rate present in
    % Bloomberg VCUB.
    for jj = 1:length(ti);
        % Add to the settlement date the number of months for each cash
        % flow.
        AdjTi(jj) = daysadd(datenum(Set),ti(jj),5);
        % Check if the final day is a working day or not (i.e. if
        % BusdateAdjTi = 1, then the day is a working day).
        BusdateAdjTi = isbusday(AdjTi(jj));
        % While loop for adjusting the cash flow date, until a working day
        % is reached.
        while BusdateAdjTi == 0
            AdjTi(jj) = daysadd(AdjTi(jj),1,5);
            BusdateAdjTi = isbusday(AdjTi(jj));
        end
    % Terminate the loop for adjusting cash flows dates.
    end
    % Set ti equal to its adjusted value.
    ti = AdjTi;
    % Calculate the year fractions between the settlement date and each
    % cash flow.
    ti = daysdif(datenum(Set),ti,5)./360;
    ti(1) = [];
    % Calculate the difference between cash flows dates.
    tau = diff(ti);
    % Calculate the d1 and d2 of the formula as defined in Chapter 3,
    % Paragraph 3 of the thesis.
    d1 = w.*(log(PM(ti(1:(end-1)))./bsxfun(@times,(ones(size(tau))...
        + X(j).*tau),PM(ti(2:end))))./Sigma(ti(1:(end-1)),...
        ti(2:end),a,b,sigma,eta,rho) - .5.*Sigma(ti(1:(end-1)),...
        ti(2:end),a,b,sigma,eta,rho));
```

```
    d2 = w.*(d1 + Sigma(ti(1:(end-1)),ti(2:end),a,b,sigma,eta,...
        rho));
    % Calculate the value of each caplet or floorlet.
    CapletFloorletG2 = -Not(j).*w.*(bsxfun(@times,(ones(size(tau)) + ...
        X(j).*tau),PM(ti(2:end))).*normcdf(d1) - PM(ti(1:(end-1)))...
        .*normcdf(d2));
    % Sum all the caplet or floorlet to obtain the cap or floor value.
    CF_G2(j) = nansum(CapletFloorletG2);
% End of the loop.
end
% End of the function.
end
```

7. G2++ Monte Carlo Cap Price

Finds the Monte Carlo simulation price of cap using the simulated G2++ model.

```
function CF_G2 = CF_MC(a,b,sigma,eta,rho,X,Mat,Res,OptSpec,Not,x,y,...
    PYear,THor,Set,CT,DF,sx,Curve)

% CF_MC Monte Carlo simulation for finding the price of a series of caps
%  (floors).
% Definition: the function calculate the price of a series of caps
%  contracts starting from x and y
% Inputs:
%  - a: drift for the first process;
%  - b: drift for the second process;
%  - sigma: standard deviation for the first process;
%  - eta: standard deviation for the second process;
%  - rho: correlation between the two processes;
%  - X: strike for the caps (floors);
%  - Mat: maturity of the contract;
%  - Res: number of caplets per year (i.e. 2);
%  - OptSpec: type of option (i.e. 'cap' or 'floor');
%  - Not: notional value of the contract;
%  - x: matrix of values for the x process;
%  - y: matrix of values for the y process;
%  - PYear: number of caplets per year;
%  - THor: the T in the T-forward measure;
%  - CT: time grid for the discounting factors;
%  - DF: values of the discounting factor at times CT;
%  - sx: Svensson model parameters;
%  - Curve: choose between 'Boot' or 'Svensson'.
% Outputs:
%  - CF_G2: caps (floors) prices using Monte Carlo simulation.

% Create an interpolation between the discounting factors and their time
% grid as an anonymous function that gives values for zero-coupon bond
% depending on the value of t in PM(t).
switch Curve
    case 'Boot'
        PM = @(t) interp1(CT,DF,t,'pchip','extrap');
    case 'Svensson'
        PM = @(t) exp(-t.*sx(1) - sx(5).*(sx(2) + sx(3)).*(1 - ...
            exp(-t./sx(5))) + sx(3).*t.*exp(-t./sx(5)) - ...
            sx(4).*sx(6).*(1 - exp(-t./sx(6))) + sx(4).*t.*exp(-t./sx(6)));
    otherwise
        % In case of wrong input.
        display('Check Curve variable')
        return
end
```

```
% Adjust simulation time horizon to match business days.
S = daysadd(datenum(Set),THor*360,5);
BusdateS = isbusday(S);
while BusdateS == 0
    S = daysadd(S,1,5);
    BusdateS = isbusday(S);
end
S = daysdif(datenum(Set),S,5)/360;

% Generate an empty matrix to be filled with caps (floors) prices.
CF_G2 = zeros(size(Mat));

% Loop for the Monte Carlo simulation that moves through each contract.
for k = 1:length(Mat)
    % The if statement find the value of omega depending on the type of
    % swaption contract.
    if strcmp(OptSpec(k),'cap')
        % If the contract is a cap omega is equal to 1.
        w = 1;
    elseif strcmp(OptSpec(k),'floor')
        % If the contract is a floor omega is equal to -1.
        w = -1;
    else
        display('Input not recognized.')
        return
    end
    % Calculate the time between the payoffs (i.e. adjusted for the
    % business days).
    ti = (0:(1/Res(k)):Mat(k))*360;
    % Generate an empty matrix of zeros with the same size of ti to be
    % filled with the business day adjusted cash flow dates.
    AdjTi = zeros(size(ti));
    % Start the loop for adjusting each cash flow with adjusting cash flows
    % date to match as much as possible the strikes rate present in
    % Bloomber VCUB.
    for jj = 1:length(ti);
        % Add to the settlement date the number of months for each cash
        % flow.
        AdjTi(jj) = daysadd(datenum(Set),ti(jj),5);
        % Check if the final day is a working day or not (i.e. if
        % BusdateAdjTi = 1, then the day is a working day).
        BusdateAdjTi = isbusday(AdjTi(jj));
        % While loop for adjusting the cash flow date, until a working day
        % is reached.
        while BusdateAdjTi == 0
            AdjTi(jj) = daysadd(AdjTi(jj),1,5);
            BusdateAdjTi = isbusday(AdjTi(jj));
        end
    % Terminate the loop for adjusting cash flows dates.
    end
    % Set ti equal to its adjusted value.
    ti = AdjTi;
    % Calculate the year fractions between the settlement date and each
    % cash flow.
    ti = daysdif(datenum(Set),ti,5)./360;
    % Calculate the difference between cash flows dates.
    tau = diff(ti);
    % Set to zero the first entry in ti since it is not relevant.
    ti(1) = [];
    % Set the initial payoff value to zero.
    Pay = 0;
    % Calculate the payoff for each caplet.
```

```
    for j = 2:length(ti)
        % Calculate the value of the zero coupon bond from the simulation.
        P_T = P_MC(ti(j-1),ti(j),a,b,sigma,eta,rho,x,y,PYear,CT,DF,sx,...
            Curve);
        % Calculate the zero-coupon bond to move forward all the cash
        % flows.
        P_F = P_MC(ti(j),S,a,b,sigma,eta,rho,x,y,PYear,CT,DF,sx,Curve);
        % Calcualte the spot Libor rate.
        Lib = 1/tau(j-1).*(1./P_T - 1);
        % Add to the before calculated caplet value the last one.
        Pay = Pay + Not(k)*tau(j-1)*max(w*(Lib - X(k)),0)./P_F;
    % Terminate the loop at the last caplet.
    end
    % Find the average of the payoff and discount back at time zero using
    % the market observed discounting factor.
    CF_G2(k) = PM(S)*mean(Pay);
% End of the loop.
end
% End of the function.
end
```

8. Find the Model Implied Volatility from Cap Model Prices

Computes the implied volatility from the G2++ model prices by minimizing the difference between the model and the Black price.

```
function [IV_CF] = CF_ImpVol(CF_G2,OptSpec,Mat,X,Res,Not,Set,CT,DF,sx,...
    Curve)

% CF_IV implied volatility from caps (floors prices).
% Description: this function calculate the implied volatility from the
%  G2++ prices under a Black like formula.
% Inputs:
%  - CF_G2: caps (floors) prices;
%  - OptSpec: type of contract (i.e. 'cap' or 'floor');
%  - Mat: time to maturity of the option;
%  - X: strike rate of the option;
%  - Res: number of caplets per year (i.e. 2, excluding the first year);
%  - Not: notional value of the contract;
%  - Set: settlement date of the contract;
%  - CT: time grid for our discounting factors;
%  - DF: value of our discounting factors for each time CT;
%  - sx: Svensson model parameters;
%  - Curve: set 'Svensson' for the Svensson model or 'Boot' for
%    bootstrapping.
% Outputs:
%  - IV_CF: Black volatility from caps (floors) prices.

% Empty matrix for omegas having the same size as the number of contracts.
w = zeros(length(Mat),1);
% Loop for calculating the value of omega depending on the contract type.
for j = 1:length(OptSpec)
    % If the contract is a cap set omega equal to 1.
    if strcmp(OptSpec(j),'cap')
        w(j) = 1;
    % If the contract is a floor set omega equal to -1.
    elseif strcmp(OptSpec(j),'floor')
        w(j) = -1;
    % In case of wrong OptSpec entry.
```

```
    else
        % Display an error message in the command window.
        display('Input not recognized.')
        % Stop the function.
        return;
    % End of the if statement.
    end
% End of the for loop needed for finding omega values.
end

% Empty matrix to be filled with implied volatilities.
IV_CF = zeros(size(Mat));

% Loop for finding the implied volatility for every option.
for j = 1:length(Mat)
    % Set the type of optimizer for the delta between the option price and
    % the Black formula calculated with the unknown implied volatility;
    % indeed, if it is possible to find a volatility for the Black formula
    % that makes the delta equal to zero, we have found our implied
    % volatility.
    options = optimset('fzero');
    % Set the minimum function tolerance to 10^(-6) and do not display the
    % result after each iteration.
    options = optimset(options, 'TolX', 1e-6, 'Display', 'off');
    % Try - catch statement in order to let the algorithm run if the fzero
    % function is not able to retrive the implied volatility.
    try
        % Minimize the delta in the local objective function at the end of
        % this function.
        [IV_CF(j), ~, exitFlag] = fzero(@objfcn,[0, 10], options,X(j),...
            Mat(j),Res(j),Not(j),CF_G2(j),OptSpec(j),Set,CT,DF,sx,Curve);
        % If the exitflag is negative (i.e. the above fzero fails to find a
        % solution for the implied volatility), substitute the missing
        % value with a NaN.
        if exitFlag < 0
            IV_CF(j) = NaN;
        % Otherwise do nothing.
        else
        % End of the if statment.
        end
    % If fzero results in an error during the minimization procedure, which
    % is different from having a negative exit flag, substitute the missing
    % value with a NaN.
    catch
        IV_CF(j) = NaN;
    % End of the catch statement.
    end
% End of the for loop.
end
% End of the function once all the contract implied volatilities have been
% calculated.
end

function  delta = objfcn(IV_CF,X,Mat,Res,Not,CF_G2,OptSpec,Set,CT,DF,sx,...
    Curve)
% OBJFCN local objective function.
% The objective function is simply the difference between the specified
% model value, or price, of the swaption and the theoretical value derived
% from the Black model, which is a function of the unknown implied
% volatility.
% Calculate Black price using the unknown implied volatility.
CF_BLK = CapFloor_BL(X,Mat,Res,Not,IV_CF,OptSpec,Set,CT,DF,sx,Curve);
% Find the delta between the price and the Black price using the unknown
```

```
% volatility.
delta = CF_G2 - CF_BLK;
% End of the local function
end
```

9. Others caps functions

Finds the strike of an ATM cap.

```
X_Cap = CF_Strk(Mat_Cap,Res_Cap,Set,CT,DF,sx,Curve)
```

Calculates the Black price of a cap.

```
Mar_Price_Cap = CapFloor_BL(X_Cap,Mat_Cap,Res_Cap,Not_Cap,Mar_Vol_Cap,...
    OptSpec_Cap,Set,CT,DF,sx,Curve);
```

10. G2++ ATM European Swaption Price Hub

Switchs between different analytical prices computational methodologies.

```
function Swap_G2 = ES_G2(a,b,sigma,eta,rho,X,Mat,Ten,Res,Not,OptSpec,...
   Set,CT,DF,x,Curve,Method)

% ES_G2 Switch between different calculation method.
% Description: uses three methodologies to calculate the price of an
%  European Swaption under the G2++ model.
% Input:
%  - a: mean reversion of the first process;
%  - b: mean reversion of the second process;
%  - sigma: standard deviation of the first process;
%  - eta: standard deviation of the second process;
%  - rho: correlation among the two processes;
%  - X: strike of the option;
%  - Mat: maturity of the swaption;
%  - Ten: tenor of the swaption;
%  - Res: payment for years (i.e. Res = 2 for 6 months);
%  - Not: notional value of the contract;
%  - OptSpec: type of swaption, equal to 'payer' or 'receiver';
%  - Set: settlement date of the contract;
%  - CT: time grid for our discounting factors;
%  - DF: value of our discounting factors for each time CT;
%  - x: Svensson model parameters;
%  - Curve: method used to build the zero-coupon curve;
%  - Method: methodology used to calculate the swaption price in the G2++
%    model (i.e. equal to: 'GL' for Gauss-Legendre integration, 'TR' for
%    trapezoidal iteration, 'SP' for approximating the price using
%    Schrager and Plesser approximation).
% Output:
%  - Swap_G2: swaption price using the methodology set in Method.

% Switch statement among different methodologies used to calculate the G2++
% swaption price. The methodology is set by the user through the Method
% variable.
switch Method
    % Gauss-Legendre to approximate the value of the integral.
    case 'GL'
```

```
        % Use the parameters passed through the ES_G2 function to calculate
        % the value of a European swaption in the G2++ model using
        % Gauss-Legendre integration.
        Swap_G2 = ES_G2byGL(a,b,sigma,eta,rho,X,Mat,Ten,Res,Not,OptSpec,...
            Set,CT,DF,x,Curve);
    % Trapezoidal integration to approximate the value of the integral.
    case 'TR'
        % Use the parameters passed through the ES_G2 function to calculate
        % the value of a European swaption in the G2++ model using
        % trapezoidal integration.
        Swap_G2 = ES_G2byTR(a,b,sigma,eta,rho,X,Mat,Ten,Res,Not,OptSpec,...
            Set,CT,DF,x,Curve);
    % Approximation of the swaption formula under an n-affine interest
    % rate model proposed by Scharager and Pelsser.
    case 'SP'
        % Use the parameters passed through the ES_G2 function to calculate
        % the value of a European swaption in the G2++ model using
        % the Schrager and Pelsser approximation.
        Swap_G2 = ES_G2bySP(a,b,sigma,eta,rho,X,Mat,Ten,Res,Not,Set,CT,...
            DF,x,Curve);
    % For wrong entries.
    otherwise
        % In case of wrong values of the Method variable, display an error
        % and stop the function.
        display('Check Method');
        return;
% End of the switch statement.
end
% End of the function (can be omitted in this case).
end
```

11. G2++ ATM European Swaption Price from Schrager and Plesser

Calculates the at-the-money European swaption price by using the approximation of Schrager and Plesser.

```
function Swap_G2 = ES_G2bySP(a,b,sigma,eta,rho,X,Mat,Ten,...
    Res,Not,Set,CT,DF,x,Curve)

% ES_G2BYSP analytical approximation.
% Description: uses the approximation of Schrager and Plesser to find the
%  analytical price of an European Swaption contract, as detailed in the
%  Interest Model - Theory and Practice by Brigo and Mercurio.
% Input:
%  - a: mean reversion of the first process;
%  - b: mean reversion of the second process;
%  - sigma: standard deviation of the first process;
%  - eta: standard deviation of the second process;
%  - rho: correlation among the two processes;
%  - X: strike of the option;
%  - Mat: maturity of the swaption;
%  - Ten: tenor of the swaption;
%  - Res: payment for years (i.e. Res = 2 for 6 months);
%  - Not: notional value of the contract;
%  - Set: settlement date of the contract;
%  - CT: time grid for our discounting factors;
%  - DF: value of our discounting factors for each time CT;
```

```
%  - x: Svensson model parameters;
%  - Curve: method used to build the zero-coupon curve.
% Output:
%  - Swap_G2: swaption price using the Gauss-Legendre integration for
%    approximating the value of the analytical G2++ swaption formula.

% Create an interpolation between the discounting factors and their time
% grid as an anonymous function that gives values for zero-coupon bond
% depending on the value of t in PM(t).
switch Curve
    case 'Boot'
        PM = @(t) interp1(CT,DF,t,'pchip','extrap');
    case 'Svensson'
        PM = @(t) exp(-t.*x(1) - x(5).*(x(2) + x(3)).*(1 - ...
            exp(-t./x(5))) + x(3).*t.*exp(-t./x(5)) - x(4).*x(6).*(1 - ...
            exp(-t./x(6))) + x(4).*t.*exp(-t./x(6)));
    otherwise
        % In case of wrong input.
        display('Check Curve variable')
        return
end

% Calculate the number of swaptions contract to set the last loop
% iteration.
nSwaptions = length(Mat);

% Create an empty matrix to be filled with the resulting model prices,
% having one column and the number of rows set to be equal to the size of
% the loop.
Swap_G2 = zeros(nSwaptions,1);

% Swaption Price loop.
for j = 1:nSwaptions
    % Calculate days between each cash flows.
    ti = round((Mat(j):(1/Res(j)):(Mat(j) + Ten(j)))*360);
    % Generate an empty matrix of zeros with the same size of ti to be
    % filled with the business day adjusted cash flow dates.
    AdjTi = zeros(size(ti));
    % Start the loop for adjusting each cash flow with adjusting cash flows
    % date to match as much as possible the strikes rate present in
    % Bloomber VCUB.
    for jj = 1:length(ti);
        % Add to the settlement date the number of months for each cash
        % flow.
        AdjTi(jj) = daysadd(datenum(Set),ti(jj),5);
        % Check if the final day is a working day or not (i.e. if
        % BusdateAdjTi = 1, then the day is a working day).
        BusdateAdjTi = isbusday(AdjTi(jj));
        % While loop for adjusting the cash flow date, until a working day
        % is reached.
        while BusdateAdjTi == 0
            AdjTi(jj) = daysadd(AdjTi(jj),1,5);
            BusdateAdjTi = isbusday(AdjTi(jj));
        end
    % Terminate the loop for adjusting cash flows dates.
    end
    % Set ti equal to its adjusted value.
    ti = AdjTi;
    % Calculate the year fractions between the settlement date and each
    % cash flow.
    ti = daysdif(datenum(Set),ti,5)./360;
    % Calculate the difference between cash flows dates.
    tau = diff(ti);
```

```
    % Set the maturity to the first ti value, since it is now adjusted for
    % holidays.
    T = ti(1);
    % Set the tenor to the last ti value, since it is now adjusted for
    % holidays.
    S = ti(end);
    % Eliminate the first column from ti to match its dimension with the
    % tau matrix of differences.
    ti(1) = [];
    % Calculate the denominator for each factor.
    BPV = sum(bsxfun(@times,tau,PM(ti)));
    % Calculate Cx, Cy and VOL for calculating the price an European
    % swaption by using the approximation by Schrager and Plesser, see Di
    % Francesco (2012).
    Cx = 1/a*(exp(-a*T)*PM(T)/BPV - exp(-a*S)*PM(S)/BPV - ...
        X(j)*sum(bsxfun(@times,exp(-a*ti).*tau,PM(ti))./BPV));
    Cy = 1/b*(exp(-b*T)*PM(T)/BPV - exp(-b*S)*PM(S)/BPV - ...
        X(j)*sum(bsxfun(@times,exp(-b*ti).*tau,PM(ti))./BPV));
    VOL = sqrt(sigma^2*(Cx^2)*((exp(2*a*T) - 1)/(2*a)) + ...
        eta^2*(Cy^2)*((exp(2*b*T) - 1)/(2*b)) + 2*rho*eta*sigma...
        *Cx*Cy*((exp((a + b)*T) - 1)/(a + b)));
    % Value of the European swaption.
    TempVal = BPV*(VOL/sqrt(2*pi));
    % Obtain the final price by multiplying the unitary value by the
    % notional of the contract.
    Swap_G2(j) = Not(j)*TempVal;
% End of the loop for finding the price of each contract.
end
% End of the function once all the prices have been calculated.
end
```

12. ATM European Swaption Model Implied Volatility

Computes the G2++ model implied volatility by using the normal inverse function in order to reduce the computational time required to calculate the swaption prices.

```
function IV_Swap = ES_ImpVol(Mat,Ten,Not,OptSpec,Swap_G2,Set,CT,DF,x,...
    Curve)

% ES_IMPVOL calculate the model implied volatility for At-The-Money
%  European swaptions
% Description: calculate the implied volatility by inverting the normal
%  distribution
% Inputs:
%  - Mat: maturity of the swaption;
%  - Ten: tenor of the swaption;
%  - Not: notional value of the contract;
%  - OptSpec: type of swaption, equal to 'payer' or 'receiver';
%  - Swap_G2: G2++ European swaption price from ES_G2 function;
%  - Set: settlement date of the contract;
%  - CT: time grid for our discounting factors;
%  - DF: value of our discounting factors for each time CT;
%  - x: Svensson model paramters;
%  - Curve: method used to build the zero-coupon curve.
% Outputs:
%  - IV_Swap: G2++ model implied volatility.

% Create an interpolation between the discounting factors and their time
% grid as an anonymous function that gives values for zero-coupon bond
% depending on the value of t in PM(t).
```

```
switch Curve
    case 'Boot'
        PM = @(t) interp1(CT,DF,t,'pchip','extrap');
    case 'Svensson'
        PM = @(t) exp(-t.*x(1) - x(5).*(x(2) + x(3)).*(1 - ...
            exp(-t./x(5))) + x(3).*t.*exp(-t./x(5)) - x(4).*x(6).*(1 - ...
            exp(-t./x(6))) + x(4).*t.*exp(-t./x(6)));
    otherwise
        % In case of wrong input.
        display('Check Curve variable')
        return
end

% Calculate the number of swaptions contract to set the last loop
% iteration.
nSwaptions = length(Mat);

% Create an empty matrix to be filled with the resulting model volatilities
% having one column and the number of rows set to be equal to the size of
% the loop.
IV_Swap = zeros(nSwaptions,1);
for j = 1:nSwaptions
    % The if statement find the value of omega depending on the type of
    % swaption contract.
    if strcmp(OptSpec(j),'payer')
        % If the contract is a payer swaption omega is equal to 1.
        w = 1;
    elseif strcmp(OptSpec(j),'receiver')
        % If the contract is a receiver swaption omega is equal to -1.
        w = -1;
    else
        % If the OptSpec value is diffrent from 'payer' or 'receiver',
        % display a message and stop the loop.
        display('Input not recognized.')
        return;
    end
    % Set T equal to the maturity of the contract.
    temp = daysadd(datenum(Set),round(Mat(j)*360),5);
    % Adjust for business day.
    BusdateAdjTi = isbusday(temp);
    while BusdateAdjTi == 0
        temp = daysadd(temp,1,5);
        BusdateAdjTi = isbusday(temp);
    end
    T = daysdif(datenum(Set),temp,5)/360;
    % Set S equal to the tenor of teh contract.
    temp = daysadd(datenum(Set),round((Mat(j) + Ten(j))*360),5);
    % Adjust for business day.
    BusdateAdjTi = isbusday(temp);
    while BusdateAdjTi == 0
        temp = daysadd(temp,1,5);
        BusdateAdjTi = isbusday(temp);
    end
    S = daysdif(datenum(Set),temp,5)/360;
    % Adjust the G2++ price from ES_G2 before using the normal inverse
    % function.
    AdjPrice = Swap_G2(j)/(Not(j)*w*(PM(T) - PM(S))*2)+.5;
    % The try statement is necessary, since norminve may not be able to
    % find the value of x; however, by using this trick, the algorithm is
    % able to substitute these missing values with NaNs in the IV_Swap
    % matrix (i.e. the output of the function).
    try
        % Find the value of d1 by inverting the adjusted model swaption
```

```
        % price.
        xx = norminv(AdjPrice);
        % Transform d1 in the model implied volatility.
        IV_Swap(j) = 2/(w*sqrt(T))*xx;
    % Avoid failure in the inverse error function.
    catch
        c = Swap_G2(j)./(w*Not(j)*(PM(T) - PM(S)));
        IV_Swap(j) = 1/(w*sqrt(T))*(2.506297*c - 0.686461*c^2)/(1 - ...
            0.277069*c - 0.237552*c^2);
    % End of the catch statement.
    end
% End of the loop.
end
% End of the function once the last implied volatility has been calculated.
end
```

13. ATM European Swaption Monte Carlo Price

Calculates the price of an ATM European Swaption by using the Monte Carlo simulation of the G2++ mode detailed by Brigo and Mercurio (2006)

```
function ES_G2 = ES_MC(a,b,sigma,eta,rho,X,Mat,Ten,Res,OptSpec,x,y,...
    PYear,THor,CT,DF,sx,Curve,Set)

% ES_MC Monte Carlo simulation for finding the price of a series of
%  European swaptions.
% Definition: the function calculate the price of a series of European
%  swaptoins starting from x and y values.
% Inputs:
%  - a: drift for the first process;
%  - b: drift for the second process;
%  - sigma: standard deviation for the first process;
%  - eta: standrd deviation for the second process;
%  - rho: correlation between the two processes;
%  - X: strike for the option;
%  - Mat: maturity of the contract;
%  - Ten: tenor of the contract;
%  - Res: reset times per year;
%  - OptSpec: type of option (i.e. 'payer' or 'receiver');
%  - sx: matrix of values for the x process;
%  - y: matrix of values for the y process;
%  - PYear: number of simulation per year;
%  - THor: the T in the T-forward measure;
%  - CT: time grid for our discounting factors;
%  - DF: value of our discounting factors for each time CT;
%  - sx: parameters for the svensson model;
%  - Curve: type of methodology used;
%  - Set: settlement date.
% Outputs:
%  - ES_G2: caps (floors) prices using Monte Carlo simulation.

% Create an interpolation between the discounting factors and their time
% grid as an anonymous function that gives values for zero-coupon bond
% depending on the value of t in PM(t).
switch Curve
    case 'Boot'
        PM = @(t) interp1(CT,DF,t,'pchip','extrap');
    case 'Svensson'
        PM = @(t) exp(-t.*sx(1) - sx(5).*(sx(2) + sx(3)).*(1 - ...
            exp(-t./sx(5))) + sx(3).*t.*exp(-t./sx(5)) - ...
```

```
            sx(4).*sx(6).*(1 - exp(-t./sx(6))) + sx(4).*t.*exp(-t./sx(6)));
    otherwise
        % In case of wrong input.
        display('Check Curve variable')
        return
end

% Initialize vairables.
ES_G2 = zeros(size(Mat));

% Adjust simulation time horizon to match business days.
S = daysadd(datenum(Set),THor*360,5);
BusdateS = isbusday(S);
while BusdateS == 0
    S = daysadd(S,1,5);
    BusdateS = isbusday(S);
end
S = daysdif(datenum(Set),S,5)/360;

% Loop that moves each swaption that has to be priced.
for k = 1:length(Ten)
    % The if statement find the value of omega depending on the type of
    % swaption contract.
    if strcmp(OptSpec(k),'payer')
        % If the contract is a payer swaption omega is equal to 1.
        w = 1;
    elseif strcmp(OptSpec(k),'receiver')
        % If the contract is a receiver swaption omega is equal to -1.
        w = -1;
    else
        display('Input not recognized.')
        break
    end
    % Calculate days between each cash flows.
    ti = round((Mat(k):(1/Res(k)):(Mat(k) + Ten(k)))*360);
    % Generate an empty matrix of zeros with the same size of ti to be
    % filled with the business day adjusted cash flow dates.
    AdjTi = zeros(size(ti));
    % Start the loop for adjusting each cash flow with adjusting cash flows
    % date to match as much as possible the strikes rate present in
    % Bloomber VCUB.
    for jj = 1:length(ti);
        % Add to the settlement date the number of months for each cash
        % flow.
        AdjTi(jj) = daysadd(datenum(Set),ti(jj),5);
        % Check if the final day is a working day or not (i.e. if
        % BusdateAdjTi = 1, then the day is a working day).
        BusdateAdjTi = isbusday(AdjTi(jj));
        % While loop for adjusting the cash flow date, until a working day
        % is reached.
        while BusdateAdjTi == 0
            AdjTi(jj) = daysadd(AdjTi(jj),1,5);
            BusdateAdjTi = isbusday(AdjTi(jj));
        end
    % Terminate the loop for adjusting cash flows dates.
    end
    % Set ti equal to its adjusted value.
    ti = AdjTi;
    % Calculate the year fractions between the settlement date and each
    % cash flow.
    ti = daysdif(datenum(Set),ti,5)./360;
    % Calculate the difference between cash flows dates.
    tau = diff(ti);
```

```
    % Set T_a, the maturity of the option.
    T_a = ti(1);
    % Eliminate the first column from ti to match its dimension with the
    % tau matrix of differences.
    ti(1) = [];
    % Calcualate the value of the zero-coupon bond to capitalize the
    % payoff value until time T (the T in T-forward measure).
    P_F = P_MC(T_a,S,a,b,sigma,eta,rho,x,y,PYear,CT,DF,sx,Curve);
    % Start the loop to calculate the payoff value at each simulation time
    % step. However, we do not have to consider the maturity of the option,
    % since it is just the beginning of the IRS, and not the first payoff
    % time.
    P_Ti = zeros(size(x,1),length(ti));
    for j = 1:(length(ti))
        % Calculate the zero coupon bond at each reset date.
        P_Ti(:,j) = P_MC(T_a,ti(j),a,b,sigma,eta,rho,x,y,PYear,CT,DF,sx,...
            Curve);
    % End of the loop that moves thorugh the cash flows of the underling
    % interest rate swap.
    end
    % Calculate the value of the swap rate.
    IRS = (1 - P_Ti(:,end))./sum(bsxfun(@times,repmat(tau,size(P_Ti,...
        1),1),P_Ti),2);
    % Calculate the value of the payoff
    Payoff = max(w.*(IRS - X(k)),0).*sum(bsxfun(@times,repmat(tau,size(...
        P_Ti,1),1),P_Ti),2);
    % Capitalize the payoff.
    CapPayoff = Payoff./P_F;
    % Find the mean of all payoffs at time T.
    Pri_F = mean(CapPayoff);
    % Discount from the forward time using the market discounting factor
    % relative to time T.
    ES_G2(k) = PM(S)*Pri_F;
% End of the loop that moves through different contracts.
end
% End of the function.
End
```

14. Others ATM European Swaption Functions

Finds the at-the-money strike of an European swaption.

```
X_Swap = ES_Strike(Ten_Swap,Mat_Swap,Res_Swap,Set,CT,DF,sx,Curve)
```

Calculates the Euopean swaption price using the Gauss-Legendre approximation of the improper integral in the G2++ formula provided by Brigo and Mercurio (2006).

```
Swap_G2 =
ES_G2byGL(a,b,sigma,eta,rho,X,Mat,Ten,Res,Not,OptSpec,Set,CT,DF,x,Curve)
```

Computes the European swaption price by approximating the improper integral in the G2++ formula provided by Brigo and Mercurio (2006).

```
Swap_G2 =
ES_G2byTR(a,b,sigma,eta,rho,X,Mat,Ten,Res,Not,OptSpec,Set,CT,DF,x,Curve)
```

15. Simulating the G2++ Interest Rate Process

This function simulates the interest rate process by following the procedure highlighted in the fifth Chapter.

```
function [r,x,y] = XY_Sim(a,b,sigma,eta,rho,THor,PYear,PathN,CT,DF,sx,...
    Curve,Set)

% XY_SIM simulate x and y under the T-forward measure
% Description: uses the solution under the T-forward measure for the two
%  process dynamics in order to simulate the two process. Using this
%  methodology, we do not need to compute the integral of the interest rate
%  to discount cash flows.
% Inputs:
%  - a: mean reversion of the x(t) process;
%  - b: mean reversion of the y(t) process;
%  - sigma: standard deviation of the x(t) process;
%  - eta: standard deviation of the y(t) process;
%  - rho: correlation between the two processes;
%  - THor: time horizon of the simulation (the T of the T-forward measure);
%  - PYear: number of steps per year in the simulation;
%  - PathN: number of path to generate;
%  - CT: time grid for our discounting factors;
%  - DF: value of our discounting factors for each time CT;
%  - sx: parameters for the Svensson model;
%  - Curve: type of methodology used;
%  - Set: settlement date.
% Outputs:
%   - r: interest rate from Monte Carlo;
%   - x: x process evolution;
%   - y: y process evolution.

% Switch among bootstrapping and Svensson model.
switch Curve
    case 'Boot'
        % Return the piece-wise polynomial form corresponding to the method
        % used to interpolate.
        pp = pchip(CT,-log(DF));
        % Find the derivatives.
        dPM = fnder(pp,1);
        FM = @(t) fnval(dPM,t);
    case 'Svensson'
        % Set instantaneous forward curve.
        FM = @(t) sx(1) + sx(2).*exp(-t./(sx(5))) + sx(3).*(t./sx(5)) ...
            .*exp(-t./sx(5)) + sx(4).*(t./sx(6)).*exp(-t./sx(6));
    otherwise
        % In case of wrong input.
        display('Check Curve variable')
        return
end

% Time grid for the simulation (i.e. 6 months for European swaptions, in
% this example
Ct = round((0:1/PYear:THor)*360);
% Generate an empty matrix of zeros with the same size of ti to be
% filled with the business day adjusted cash flow dates.
AdjCt = zeros(size(Ct));
% Start the loop for adjusting each cash flow with adjusting cash flows
% date to match as much as possible the strikes rate present in
% Bloomberg VCUB.
```

```
for jj = 1:length(Ct);
    % Add to the settlement date the number of months for each cash
    % flow.
    AdjCt(jj) = daysadd(datenum(Set),Ct(jj),5);
    % Check if the final day is a working day or not (i.e. if
    % BusdateAdjTi = 1, then the day is a working day).
    BusdateAdjCt = isbusday(AdjCt(jj));
    % While loop for adjusting the cash flow date, until a working day
    % is reached.
    while BusdateAdjCt == 0
        AdjCt(jj) = daysadd(AdjCt(jj),1,5);
        BusdateAdjCt = isbusday(AdjCt(jj));
    end
% Terminate the loop for adjusting cash flows dates.
end
% Set ti equal to its adjusted value.
Ct = AdjCt;
% Calculate the year fractions between the settlement date and each
% cash flow.
Ct= daysdif(datenum(Set),Ct,5)./360;

% Initialize the two variables.
x = zeros(PathN,size(Ct,2));
y = zeros(PathN,size(Ct,2));
Ft = zeros(PathN,size(Ct,2));
Phit = zeros(PathN,size(Ct,2));

% Add an additional row of zero at the beginning of the matrix to match the
% initial value for x and y as proposed by Brigo and Mercurio.
x(:,1) = 0;
y(:,1) = 0;
Ft(:,1) = FM(0);
Phit(:,1) = Ft(:,1);

% Loop for the simulation of the two processes.
for j = 1:(size(Ct,2) - 1)
    % Calculate the variance of the two processes.
    Var_x = sigma^2*(1 - exp(-2*a*(Ct(j+1) - Ct(j))))/2/a;
    Var_y = eta^2*(1 - exp(-2*b*(Ct(j+1) - Ct(j))))/2/b;
    % Calculate the covariance.
    Cov_xy = sigma*eta*rho/(a + b)*(1 - exp(-(a + b)*(Ct(j+1) - ...
        Ct(j))));
    % Set the mean of the two process equal to zero.
    Mean_x = 0;
    Mean_y = 0;
    % Calculate inputs for the multivariate normal distribution.
    Mu = [Mean_x; Mean_y];
    Sigma = [Var_x Cov_xy; Cov_xy Var_y];
    % Extract the random numbers.
    Temp = mvnrnd(Mu,Sigma,PathN/2);
    RandNum = [Temp; -Temp];
    % Calculate the drift change in the T-forward measure, Mu_x is defined
    % by Brigo and Mercurio at pag. 154.
    Mu_x = (sigma^2/a^2 + rho*sigma*eta/a/b)*(1 - exp(-a*(Ct(j+1) - ...
        Ct(j)))) - sigma^2/2/a^2*(exp(-a*(Ct(end) - Ct(j+1))) - ...
        exp(-a*(Ct(end) + Ct(j+1) - 2*Ct(j)))) - rho*sigma*eta/b/(a+b)...
        *(exp(-b*(Ct(end) - Ct(j+1))) - exp(-b*Ct(end) - a*Ct(j+1)...
        + (a + b)*Ct(j)));
    % Calculate the drift change in the T Forward measure for the second
    % process.
    Mu_y = (eta^2/b^2 + rho*sigma*eta/a/b)*(1 - exp(-b*(Ct(j+1) - ...
        Ct(j)))) - eta^2/2/b^2*(exp(-b*(Ct(end) - Ct(j+1))) - ...
        exp(-b*(Ct(end) + Ct(j+1) - 2*Ct(j)))) - rho*sigma*eta/a/(a+b)...
```

```
        *(exp(-a*(Ct(end) - Ct(j+1))) - exp(-a*Ct(end) - b*Ct(j+1)...
        + (a + b)*Ct(j)));
    % Calculate the first process.
    x(:,j+1) = x(:,j)*exp(-a*(Ct(j+1) - Ct(j))) - Mu_x + RandNum(:,1);
    y(:,j+1) = y(:,j)*exp(-b*(Ct(j+1) - Ct(j))) - Mu_y + RandNum(:,2);

    % Calculate the instantaneous forward.
    Ft(:,j+1) = FM(Ct(j+1));

    % Calculate the deterministic shift.
    Phit(:,j+1) = Ft(:,j+1) + sigma^2/(2*a^2)*(1 - exp(-a*Ct(j))) + ...
        eta^2/(2*b^2)*(1 - exp(-b*Ct(j))) + rho*sigma*eta/(a*b)*(1 - ...
        exp(-a*Ct(j)))*(1 - exp(-b*Ct(j)));

% End of the loop.
end

% Calculate the value of the interest rate.
r = Phit + x + y;

% End of the function.
end
```

16. Zero Coupon Bond Value from the Monte Carlo Simulation

This algorithm computes the value of the zero coupon bond resulting from the Monte Carlo simulation of the G2++ model.

```
function ZCB = P_MC(ti,Ti,a,b,sigma,eta,rho,x,y,PYear,CT,DF,sx,Curve)

% P_MC gives the value of a zero-coupon bond in the G2++ model for the
%  Monte Carlo simulation.
% Description: calculate the value at time t of a zero bond with maturity
%  T, taking into consideration the value at time t of the two process
%  x(t) and y(t), for one step of the simulation in all processes paths.
% Inputs:
%  - ti: t in the analytical formula by Brigo and Mercurio;
%  - Ti: T in the analytical formula by Brigo and Mercurio;
%  - a: mean constant of the process x(t);
%  - b: mean constant of the process y(t);
%  - sigma: standard deviation of x(t);
%  - eta: standard deviation of y(t);
%  - rho: correlation between x(t) and y(t);
%  - PYear: number of simulaton per year;
%  - x: matrix with the value of the x process;
%  - y: matrix with the value of the y process;
%  - CT: time grid for our discounting factors;
%  - DF: value of our discounting factors for each time CT;
%  - sx: parameters for the svensson model;
%  - Curve: type of methodology used.
% Output:
%  - ZCB: value of a zero bond at time t at with maturity T in the G2++
%    model.

% Create an interpolation between the discounting factors and their time
% grid as an anonymous function that gives values for zero-coupon bond
% depending on the value of t in PM(t).
switch Curve
```

```
    case 'Boot'
        PM = @(t) interp1(CT,DF,t,'pchip','extrap');
    case 'Svensson'
        PM = @(t) exp(-t.*sx(1) - sx(5).*(sx(2) + sx(3)).*(1 - ...
            exp(-t./sx(5))) + sx(3).*t.*exp(-t./sx(5)) - ...
            sx(4).*sx(6).*(1 - exp(-t./sx(6))) + sx(4).*t.*exp(-t./sx(6)));
    otherwise
        % In case of wrong input.
        display('Check Curve variable')
        return
end

% Define V(t,T) as defined by Brigo and Mercurio at pag. 145, formula n.
% 4.10.
V = @(t,T) sigma^2/a^2*(T - t + 2/a*exp(-a*(T-t)) - ...
    1/(2*a)*exp(-2*a*(T-t)) - 3/2/a) + eta^2/b^2*(T - t + ...
    2/b*exp(-b*(T-t)) - 1/(2*b)*exp(-2*b*(T-t)) - 3/2/b) + ...
    2*rho*sigma*eta/(a*b)*(T - t + (exp(-a*(T-t)) - 1)/a + ...
    (exp(-b*(T-t)) - 1)/b - (exp(-(a + b)*(T-t)) - 1)/(a + b));

% Calculate the value of the zero-coupon bond using the G2++ analytical
% formula.
ZCB = PM(Ti)./PM(ti).*exp(1/2*(V(ti,Ti) - V(0,Ti) + V(0,ti)) - (1 - ...
    exp(-a.*(Ti - ti)))./a.*x(:,round(ti*PYear) + 1) - (1 - ...
    exp(-b.*(Ti - ti)))./b.*y(:,round(ti*PYear) + 1));

% End of the function.
end
```

17. Model Calibration Function

This function switches between different calibrations algorithms; each one has an objective function that, in the majority of the cases, is a nested function as explained in the fourth chapter.

```
function [P, fval] = G2_Calibration(Cal_Criteria,X_Swap,X_Cap,X_Caplet,...
    Mat_Swap,Mat_Cap,Mat_Caplet,Ten_Swap,Res_Swap,Res_Cap,Not_Swap,...
    Not_Cap,Not_Caplet,OptSpec_Swap,OptSpec_Cap,OptSpec_Caplet,Set,CT,...
    DF,sx,Curve,Mar_Vol_Swap,Mar_Vol_Cap,Mar_Vol_Caplet,Mar_Price_Swap,...
    Mar_Price_Cap,Mar_Price_Caplet,AdjTi,P0,Obj,Method,InstrumentSet)

% G2_CALIBRATION calibrate the G2++ model to market data.
% Description: in order to implement the model is necessary to calibrate
%  the parameters of the model; indeed, this function uses four
%  algorithms in order to minimize the user selected error function.
% Inputs:
%  - Cal_Criteria: choosing between different calibration algorithms
%    (i.e.'GradientMin' for deterministic minimization, 'GeneticMin' for
%    genetic algorithm minimization, 'DifferentialMin' for differential
%    evolutionary minimization, 'SimulatedAnnealing' for simulated
%    annealing);
%  - X: ATM strike price for swaptions, caps and caplets + swaptions;
%  - Mat: maturity of swaptions, caps and caplets + swaptions;
%  - Ten: tenor of swaptions, caps and caplets + swaptions;
%  - Res: payment for years (i.e. Res = 2 for 6 months);
%  - Not: notional value;
%  - OptSpec: type of swaptions, caps and caplets + swaptions;
%  - Set: settlement date of the contract;
%  - CT: time grid for our discounting factors;
```

```
%  - DF: value of our discounting factors for each time CT;
%  - x: parameters for the Svensson model;
%  - Curve: method used to build the zero-coupon curve;
%  - Mar_Vol: instrument volatilities from VCUB;
%  - Mar_Price: Black prices from Mar_Vol calculated using Black formula;
%  - P0: starting point for the minimization algorithm (when required);
%  - Obj: error function to be minimized (i.e. 'Vol' for the percentage
%    difference between model and market volatilitis, 'Price'  for the
%    percentage difference between model and market prices);
%  - Method: methodology used to calculate the swaption price in the G2++
%    model (i.e. equal to: 'GL' for Gauss-Legendre integration, 'TR' for
%    trapezoidal iteration, 'SP' for approximating the price using
%    Schrager and Plesser approximation) or 'MC' for Monte Carlo
%    calibration.
% Output:
%  - P: vector containing the parameters of the model (i.e. P(1) = a,
%    P(2) = b, P(3) = sigma, P(4) = eta, P(5) = rho);
%  - fval: value of the error function for the model parameters in P.

% Parameters boundaries for the problem (the reult for a, b, sigma and
% eta has to be positive and the correlation lies between -1 and +1).
lb = [eps eps 0.0 0.0 -1.0];
ub = [1.0 1.0 0.5 0.5 1.0];

if strcmp(InstrumentSet,'Swaption')
    switch Cal_Criteria
        % If the user uses a deterministic minimization algorithm.
        case 'GradientMin'
            % Options for fmincon that will be used inside
            % RunNesFunGM_Swaption.
            % These options refer to the 'fmincon' minimizer: the maximum
            % of functions that the algorithm will evaluate is set to
            % 20000, we let the algorithm show the value of fval for each
            % iteration, we set the minimum change in function tolerance to
            % 10^(-6), the same value has been chosen for the minimum
            % tolerance in changes on our variables, the maximum number of
            % iterations has been set to 500, the solution algorithm is the
            % classical interior point solution.
            options = optimoptions('fmincon','MaxFunEvals',20000,...
                'Display','iter','TolFun',1e-6,'TolX',1e-6,'MaxIter',...
                500,'Algorithm','interior-point');
            % Optimization by suing a nested function in order to pass
            % extra parameters to the objective function, including the
            % afore mentioned options.
            [ P, fval] = RunNesFunGM_Swaption(X_Swap,Mat_Swap,Ten_Swap,...
                Res_Swap,Not_Swap,OptSpec_Swap,Set,CT,DF,sx,Curve,...
                Mar_Vol_Swap,Mar_Price_Swap,P0,Obj,Method,lb,ub,options);
        % If the user uses a genetic minimization algorithm.
        case 'GeneticMin'
            % Options for ga that will be used inside RunNesFunGA_Swaption.
            % These options refer to the 'ga' minimizer: the maximum number
            % of generations that the algorithm will calculate is set to
            % 500, the initial population range has been set equal to
            % parameters boundaries, we let the algorithm show the value
            % of the best population member and the average of the
            % population for each iteration, we set the minimum change in
            % our objective function to 10^(-6).
            options = gaoptimset('Generations',100,'PopInitRange',...
                [lb;ub],'PopulationSize',50,'Display','iter','TolFun',...
                1e-6);
            % Optimization by using a nested function in order to pass
            % extra parameters to the objective function, including the
            % afore mentioned options.
```

```
            [ P, fval] = RunNesFunGA_Swaption(X_Swap,Mat_Swap,Ten_Swap,...
                Res_Swap,Not_Swap,OptSpec_Swap,Set,CT,DF,sx,Curve,...
                Mar_Vol_Swap,Mar_Price_Swap,Obj,Method,options,lb,ub);
        % If the user uses a differential evolutionary algorithm.
        case 'DifferentialMin'
            % Differential evolutionary minimization using devec3 through a
            % nested function, all the options are setted inside the nested
            % function.
            [ P, fval] = RunNesFunDF_Swaption(X_Swap,Mat_Swap,Ten_Swap,...
                Res_Swap,Not_Swap,OptSpec_Swap,Set,CT,DF,sx,Curve,...
                Mar_Vol_Swap,Mar_Price_Swap,P0,Obj,Method,lb,ub);
        % If the user uses simulated annealing to minimize the objective
        % function.
        case 'SimulatedAnnealing'
            % Objective function of the simulated annealing.
            G2PPobjfun = @(P) ObjFunSA_Swaption(P(1),P(2),P(3),P(4),...
                P(5),X_Swap,Mat_Swap,Ten_Swap,Res_Swap,Not_Swap,...
                OptSpec_Swap,Set,CT,DF,sx,Curve,Mar_Vol_Swap,...
                Mar_Price_Swap,Obj,Method);
            % Options for sa: minimum tolerance function set to 10^(-6),
            % display the results for each iteration.
            options = saoptimset('TolFun',1e-6,'Display','iter');
            % Start simulated annealing from the point P0 (initial guess
            % equals to Brigo and Mercurio parameters.
            [ P, fval] = simulannealbnd(G2PPobjfun,P0,lb,ub,options);
        case 'ParticleSwarm'
            % Set options for particle swarm
            options = optimoptions('particleswarm','Display','iter',...
                'SwarmSize',50);
            % Minimze the error function using pattern swarm.
            [P, fval] = RunNesFunPS_Swaption(X_Swap,Mat_Swap,Ten_Swap,...
                Res_Swap,Not_Swap,OptSpec_Swap,Set,CT,DF,sx,Curve,...
                Mar_Vol_Swap,Mar_Price_Swap,Obj,Method,lb,ub,options);
        % In case the users insert a wrong value in the 'Cal_Criteria'
        % variable.
        otherwise
            % Display an error message.
            display('Calibration criteria is not recognized.')
            % Stop the function.
            return
    % End of the switch statement.
    end
elseif strcmp(InstrumentSet,'Cap')
    switch Cal_Criteria
        % If the user uses a deterministic minimization algorithm.
        case 'GradientMin'
            % Objective function for lsqnonlin, which is a function of the
            % five model parameters.
            ObjFun = @(P) ObjFun_CapFloor(P(1),P(2),P(3),P(4),P(5),...
                X_Cap,Mat_Cap,Res_Cap,Not_Cap,OptSpec_Cap,CT,DF,sx,...
                Curve,Mar_Vol_Cap,Obj,Mar_Price_Cap,Set);
            % These options refer to the 'fmincon' minimizer: the maximum
            % of functions that the algorithm will evaluate is set to 5000,
            % we let the algorithm show the value of fval for each
            % iteration, we set the minimum change in function tolerance to
            % 10^(-6), the same value has been chosen for the minimum
            % tolerance in changes on our variables.
            options = optimset('MaxFunEvals',5000,'Display','iter',...
                'TolX',1e-6,'TolFun',1e-6);
            % Optimization using least square non linear (lsqnonlin) for
            % minimizing the error function defined in the Obj variable.
            [ P, fval] = lsqnonlin(ObjFun,P0,lb,ub,options);
        % If the user uses a genetic minimization algorithm.
```

```
        case 'GeneticMin'
            % Options for ga that will be used inside RunNesFunGA_Cap.
            % These options refer to the 'ga' minimizer: the maximum number
            % of generations that the algorithm will calculate is set to
            % 500, the initial population range has been set equal to
            % parameters boundaries, we let the algorithm show the value of
            % the best population member and the average of the population
            % for each iteration, we set the minimum change in our
            % objective function to 10^(-6).
            options = gaoptimset('Generations',500,'PopulationSize',50,...
                'PopInitRange',[lb; ub],'Display','iter','TolFun',1e-6);
            % Optimization by using a nested function in order to pass
            % extra parameters to the objective function, including the
            % afore mentioned options.
            [ P, fval] = RunNesFunGA_CapFloor(X_Cap,Mat_Cap,Res_Cap,...
                Not_Cap,OptSpec_Cap,CT,DF,sx,Curve,Mar_Vol_Cap,options,...
                lb,ub,Obj,Mar_Price_Cap,Set);
        % If the user uses a differential evolutionary algorithm.
        case 'DifferentialMin'
            % Differential evolutionary minimization using devec3 through a
            % nested function, all the options are set inside the nested
            % function.
            [ P, fval] = RunNesFunDF_CapFloor(X_Cap,Mat_Cap,Res_Cap,...
                Not_Cap,OptSpec_Cap,CT,DF,sx,Curve,Mar_Vol_Cap,lb,ub,P0,...
                Obj,Mar_Price_Cap,Set);
            % In case the users insert a wrong value in the 'Cal_Criteria'
            % variable.
        otherwise
            % Display an error message.
            display('Calibration criteria is not recognized.')
            % Stop the function.
            return
    % End of the switch statement.
    end
elseif strcmp(InstrumentSet,'SwaptionCaplet')
    switch Cal_Criteria
        % If the user uses a deterministic minimization algorithm.
        case 'GradientMin'
            % Options for fmincon that will be used inside
            % RunNesFunGM_Swaption.
            % These options refer to the 'fmincon' minimizer: the maximum
            % of functions that the algorithm will evaluate is set to
            % 20000, we let the algorithm show the value of fval for each
            % iteration, we set the minimum change in function tolerance to
            % 10^(-6), the same value has been chosen for the minimum
            % tolerance in changes on our variables, the maximum number of
            % iterations has been set to 500, the solution algorithm is the
            % classical interior point solution.
            options = optimoptions('fmincon','MaxFunEvals',20000,...
                'Display','iter','TolFun',1e-6,'TolX',1e-6,'MaxIter',...
                500,'Algorithm','interior-point');
            % Optimization by suing a nested function in order to pass
            % extra parameters to the objective function, indcluding the
            % afore mentioned options.
            [ P, fval] = RunNesFunGM_SC(X_Swap,X_Caplet,Mat_Swap,...
                Mat_Caplet,Ten_Swap,Res_Swap,Not_Swap,Not_Caplet,...
                OptSpec_Swap,OptSpec_Caplet,Set,CT,DF,sx,Curve,...
                Mar_Vol_Swap,Mar_Vol_Caplet,Mar_Price_Swap,...
                Mar_Price_Caplet,AdjTi,Obj,Method,P0,options,lb,ub);
        % If the user uses a genetic minimization algorithm.
        case 'GeneticMin'
            % Options for ga that will be used inside RunNesFunGA_Swaption.
            % These options refer to the 'ga' minimizer: the maximum number
```

```
            % of generations that the algorithm will calculate is set to
            % 500, the initial population range has been set equal to
            % parameters boundaries, we let the algorithm show the value
            % of the best population member and the average of the
            % population for each iteration, we set the minimum change in
            % our objective function to 10^(-6).
            options = gaoptimset('Generations',100,'PopInitRange',...
                [lb;ub],'PopulationSize',50,'Display','iter','TolFun',...
                1e-6);
            % Optimization by using a nested function in order to pass
            % extra parameters to the objective function, including the
            % afore mentioned options.
            [ P, fval] = RunNesFunGA_SC(X_Swap,X_Caplet,Mat_Swap,...
                Mat_Caplet,Ten_Swap,Res_Swap,Not_Swap,Not_Caplet,...
                OptSpec_Swap,OptSpec_Caplet,Set,CT,DF,sx,Curve,...
                Mar_Vol_Swap,Mar_Vol_Caplet,Mar_Price_Swap,...
                Mar_Price_Caplet,AdjTi,Obj,Method,options,lb,ub);
        % If the user uses a differential evolutionary algorithm.
        case 'DifferentialMin'
            % Differential evolutionary minimization using devec3 through a
            % nested function, all the options are setted inside the nested
            % function.
            [ P, fval] = RunNesFunDF_SC(X_Swap,X_Caplet,Mat_Swap,...
                Mat_Caplet,Ten_Swap,Res_Swap,Not_Swap,Not_Caplet,...
                OptSpec_Swap,OptSpec_Caplet,Set,CT,DF,sx,Curve,...
                Mar_Vol_Swap,Mar_Vol_Caplet,Mar_Price_Swap,...
                Mar_Price_Caplet,AdjTi,P0,Obj,Method,lb,ub);
            otherwise
            % Display an error message.
            display('Calibration criteria is not recognized.')
            % Stop the function.
            return
    % End of the switch statement.
    end
% In case of wrong instrument set.
else
    % Display an error message.
    display('Instrument set is not recognized.')
    % Stop the function.
    return
end
```

18. Objective Functions for Calibrating the Model

Calibrates the model to the European swaptions volatilities or prices by using a deterministic algorithm.

```
[P, fval] = RunNesFunGM_Swaption(X_Swap,Mat_Swap,Ten_Swap,...
    Res_Swap,Not_Swap,OptSpec_Swap,Set,CT,DF,x,Curve,Mar_Vol_Swap,...
    Mar_Price_Swap,P0,Obj,Method,lb,ub,options)
```

Calibrates the model to the European swaptions volatilities or prices by using a genetic algorithm.

```
[P, fval] = RunNesFunGA_Swaption(X,Mat,Ten,Res,Not,OptSpec,...
    Set,CT,DF,x,Curve,Mar_Vol,Mar_Price,Obj,Method,options,lb,ub)
```

Calibrates the model to the European swaptions volatilities or prices by using the differential evolutionary algorithm.

```
[P, fval] = RunNesFunDF_Swaption(X,Mat,Ten,Res,Not,OptSpec,...
            Set,CT,DF,sx,Curve,Mar_Vol,Mar_Price,P0,Obj,Method,lb,ub)
```

Calibrates the model to the European swaptions volatilities or prices by using simulated annealing.

```
Error_Swap = ObjFunSA_Swaption(a,b,sigma,eta,rho,X,Mat,Ten,...
    Res,Not,OptSpec,Set,CT,DF,sx,Curve,Mar_Vol,Mar_Price,Obj,Method)
```

Calibrates the model to the European swaptions volatilities or prices by using particle swarm.

```
[P, fval] = RunNesFunPS_Swaption(X_Swap,Mat_Swap,Ten_Swap,...
    Res_Swap,Not_Swap,OptSpec_Swap,Set,CT,DF,x,Curve,Mar_Vol_Swap,...
    Mar_Price_Swap,Obj,Method,lb,ub,options)
```

Calibrates the model to the ATM caps volatilities or prices by using a deterministic algorithm.

```
CF_Error = ObjFun_CapFloor(a,b,sigma,eta,rho,X,...
    Mat,Res,Not,OptSpec,CT,DF,sx,Curve,Mar_Vol,Obj,Mar_Price,Set)
```

Calibrates the model to the ATM caps volatilities or prices by using a genetic algorithm.

```
[P, fval] = RunNesFunGA_CapFloor(X_Cap,Mat_Cap,Res_Cap,Not_Cap,...
    OptSpec_Cap,CT,DF,sx,Curve,Mar_Vol_Cap,options,lb,ub,Obj,...
    Mar_Price_Cap,Set)
```

Calibrates the model to the ATM caps volatilities or prices by using differential evolutionary.

```
[P,fval] = RunNesFunDF_CapFloor(X_Cap,Mat_Cap,Res_Cap,Not_Cap,...
    OptSpec_Cap,CT,DF,sx,Curve,Mar_Vol_Cap,lb,ub,P0,Obj,Mar_Price_Cap,Set)
```

Calibrates the model to the ATM caplets and European swaptions volatilities or prices by using a deterministic algorithm.

```
[ P, fval] = RunNesFunGM_SC(X_Swap,X_Caplet,Mat_Swap,...
                Mat_Caplet,Ten_Swap,Res_Swap,Not_Swap,Not_Caplet,...
                OptSpec_Swap,OptSpec_Caplet,Set,CT,DF,sx,Curve,...
                Mar_Vol_Swap,Mar_Vol_Caplet,Mar_Price_Swap,...
                Mar_Price_Caplet,AdjTi,Obj,Method,P0,options,lb,ub)
```

Calibrates the model to the ATM caplets and European swaptions volatilities or prices by using a genetic algorithm.

```
[ P, fval] = RunNesFunGA_SC(X_Swap,X_Caplet,Mat_Swap,...
                Mat_Caplet,Ten_Swap,Res_Swap,Not_Swap,Not_Caplet,...
                OptSpec_Swap,OptSpec_Caplet,Set,CT,DF,sx,Curve,...
                Mar_Vol_Swap,Mar_Vol_Caplet,Mar_Price_Swap,...
                Mar_Price_Caplet,AdjTi,Obj,Method,options,lb,ub)
```

Calibrates the model to the ATM caplets and European swaptions volatilities or prices by using differential evolutionary.

```
[ P, fval] = RunNesFunDF_SC(X_Swap,X_Caplet,Mat_Swap,...
          Mat_Caplet,Ten_Swap,Res_Swap,Not_Swap,Not_Caplet,...
          OptSpec_Swap,OptSpec_Caplet,Set,CT,DF,sx,Curve,...
          Mar_Vol_Swap,Mar_Vol_Caplet,Mar_Price_Swap,...
          Mar_Price_Caplet,AdjTi,P0,Obj,Method,lb,ub)
```

19. Functions for Bootstrapping the ATM Caplets from Caps

The following functions are used to bootstrap the ATM caplets volatilities from ATM caps by following the methodology presented by Schoenmakers (2005, pages 58-59)

```
y = Caplet_Bootstrapping(Mar_Vol_Caplet,Forward,X_CapAdj,...
    OptSpec_Caplet,Vol_Cap,Vol_Caplet,Not_Caplet,AdjTi,DF,CT,sx,...
    Curve,K)
```

Finds the other ATM caplets data.

```
[Mar_Vol_Caplet,Mat_Caplet,X_Caplet,Not_Caplet,OptSpec_Caplet,...
    AdjTi] = Caplet_MissingData(Mar_Vol_Cap,Res_Cap,Not_Cap,Mat_Cap,...
    OptSpec_Cap,Set,CT,DF,sx,Curve)
```

Computes the ATM caplets Black price.

```
BL_Caplet = Caplet_BL(Forward,X_Caplet,Caplet_Vol,AdjTi,...
    OptSpec_Caplet,DF,CT,sx,Curve,Not_Caplet)
```

Calculates the ATM caplets G2++ analytical price.

```
G2_Caplet = Caplet_G2(a,b,sigma,eta,rho,Not_Caplet,X_Caplet,...
    OptSpec_Caplet,AdjTi,CT,DF,sx,Curve)
```

Inverts the ATM caplets G2++ analytical price in order to compute the model implied volatility.

```
IV_Caplet = Caplet_ImpVol(Not_Caplet,G2_Caplet,OptSpec_Caplet,...
    AdjTi,CT,DF,sx,Curve)
```

20. Pricing the Contract of the Fifth Chapter

Finds the price of the contract in the fifth chapter by dividing the option in a series of vanillas building blocks.

```
function [MatrUnd_A,MatrUnd_B,MatrDis_A,MatrDis_B,Matr_A,Matr_B,Price] ...
    = Price2(a,b,sigma,eta,rho,Not_Con,x,y,Sim_Dates,Lib_Dates,CF_Dates,...
    Set_Con,CT,DF,sx,Curve)

% Create an interpolation between the discounting factors and their time
% grid as an anonymous function that gives values for zero-coupon bond
% depending on the value of t in PM(t).
```

```
switch Curve
    case 'Boot'
        PM = @(t) interp1(CT,DF,t,'pchip','extrap');
    case 'Svensson'
        PM = @(t) exp(-t.*x(1) - x(5).*(x(2) + x(3)).*(1 - ...
            exp(-t./x(5))) + x(3).*t.*exp(-t./x(5)) - x(4).*x(6).*(1 - ...
            exp(-t./x(6))) + x(4).*t.*exp(-t./x(6)));
    otherwise
        % In case of wrong input.
        display('Check Curve variable')
        return
end

% Set ti equals to cash flows dates ratios.
Sim_ti = yearfrac(datenum(Set_Con),datenum(Sim_Dates),2);
CF_ti = yearfrac(datenum(Set_Con),datenum(CF_Dates),2);
Lib_ti = yearfrac(datenum(Set_Con),datenum(Lib_Dates));
CF_tau = [CF_ti(1); diff(CF_ti)];
Lib_tau = abs(diff(Lib_ti));

S = Sim_ti(end);

% Set the initial value of the payoff matrix.
MatrUnd_A = [];
MatrUnd_B = [];
MatrDis_A = [];
MatrDis_B = [];
Matr_A = [];
Matr_B = [];
% Calculate the payoff for each caplet.
for j = 1:(length(Lib_ti)-1)
    % Calculate the value of the zero coupon bond from the simulation.
    if Lib_ti(j) <= 0
        % Use the historical Libor rate, which is known.
        Lib = repmat(4.768,size(x(:,1)));
        % Calculate the value of the zero coupon bond for moving forward
        % the cash flows to the terminal measure.
        P_F = P_Con(CF_ti(j),Sim_ti(end),a,b,sigma,eta,rho,x,y,j*2+1,CT,...
            DF,sx,Curve);
        % Calculate the payoff.
        Temp = (4.05 + (4.65 - 4.05)*ge(Lib,4.65.*ones(size(Lib)))...
            + max(Lib - 4.65,0) - max(Lib - 5,0))./100;
        % Find the unidscounted payoff.
        PayUnd_A = Not_Con(j).*(CF_tau(j).*Temp);
        PayUnd_B = Not_Con(j).*(CF_tau(j).*Lib./100);
        % Calculate the undiscounted payoff matrix.
        MatrUnd_A = [MatrUnd_A PayUnd_A];
        MatrUnd_B = [MatrUnd_B PayUnd_B];
        % Discount the payoff using the forward measure.
        PayDis_A = PayUnd_A.*PM(S)./P_F;
        PayDis_B = PayUnd_B.*PM(S)./P_F;
        % Find the mean discounted value for each cash flows.
        PayMean_A = mean(PayDis_A);
        PayMean_B = mean(PayDis_B);
        % Build a matrix containing the discounted value for each
        % simulation at each cash flow dates.
        MatrDis_A = [MatrDis_A PayDis_A];
        MatrDis_B = [MatrDis_B PayDis_B];
        % Build a matrix with all the payoffs.
        Matr_A = [Matr_A; PayMean_A];
        Matr_B = [Matr_B; PayMean_B];
    else
        P_T = P_Con(Lib_ti(j),Lib_ti(j+1),a,b,sigma,eta,rho,x,y,j*2,CT,...
```

```
            DF,sx,Curve);
        % Calculate the zero-coupon bond to move forward all the cash
        % flows.
        P_F = P_Con(CF_ti(j),S,a,b,sigma,eta,rho,x,y,j*2+1,CT,DF,sx,Curve);
        % Calcualte the spot Libor rate.
        Lib = 1/Lib_tau(j).*(1./P_T - 1).*100;
        % Calculate the payoff depending on the Euribor rate.
        if Lib_ti(j) <= yearfrac(Set_Con,'31-Dec-2011',2)
            Temp = (4.05 + (4.65 - 4.05)*ge(Lib,4.65.*ones(size(Lib))) ...
                + max(Lib - 4.65,0) - max(Lib - 5,0))./100;
        else
            Temp = (3.80 + (4.56 - 3.80)*gt(Lib,3.80.*ones(size(Lib))) ...
                + (5.30 - 4.56)*ge(Lib,5.30.*ones(size(Lib))) ...
                + max(Lib - 5.30,0))./100;
        end
        % Calculate the undiscounted payoff.
        PayUnd_A = Not_Con(j).*(CF_tau(j).*Temp);
        PayUnd_B = Not_Con(j).*(CF_tau(j).*Lib./100);
        % Calculate the undiscounted payoff matrix.
        MatrUnd_A = [MatrUnd_A PayUnd_A];
        MatrUnd_B = [MatrUnd_B PayUnd_B];
        % Discount the payoff using the forward measure.
        PayDis_A = PayUnd_A.*PM(S)./P_F;
        PayDis_B = PayUnd_B.*PM(S)./P_F;
        % Find the mean discounted value for each cash flows.
        PayMean_A = mean(PayDis_A);
        PayMean_B = mean(PayDis_B);
        % Build a matrix containing the discounted value for each
        % simulation at each cash flow dates.
        MatrDis_A = [MatrDis_A PayDis_A];
        MatrDis_B = [MatrDis_B PayDis_B];
        % Build a matrix with all the payoffs.
        Matr_A = [Matr_A; PayMean_A];
        Matr_B = [Matr_B; PayMean_B];
    % Terminate the if statement.
    end
% Terminate the loop.
end

Price =  - sum(Matr_A) + sum(Matr_B);

end
```

20. Other Functions Used During the Pricing of the Contract

Computes the price of the contracts by using if statements for each case highlighted in the fifth Chapter instead of decomposing the contract into a series of vanilla options.

```
[MatrUnd_A,MatrUnd_B,MatrDis_A,MatrDis_B,Matr_A,Matr_B,Price] ...
    = Price1(a,b,sigma,eta,rho,Not_Con,x,y,Sim_Dates,Lib_Dates,CF_Dates,...
    Set_Con,CT,DF,sx,Curve)
```

Simulates the interest process for specific dates instead of simulating the interest rate at a regular time intervals of n° months.

```
[r,x,y] = XY_SimPri(a,b,sigma,eta,rho,PathN,CT,DF,sx,Curve,...
    Sim_Dates,Set_Con)
```

Calculates the value of the zero-coupon bond from the interest rate simulation by using a counter to identify the $x(t)$ and $y(t)$ column.

```
ZCB = P_Con(ti,Ti,a,b,sigma,eta,rho,x,y,count,CT,DF,sx,Curve)
```

Bibliography

BIANCHETTI, M. and CARLICCHI, M., 2010. *Interest Rates After the Credit Crunch: Markets and Models Evolution*. SSRN Working Paper. Freely available at: http://ssrn.com/abstract=1783070

BLANCHARD, A., 2012. *The Two-Factor Hull-White Model: Pricing and Calibration of Interest Rate Derivatives*. Stockholm, Royal Institute of Technolgy. Available at: http://www.math.kth.se/matstat/seminarier/120220b.htm

BOYLE, P. P., 1977. Options: A Monte Carlo Approach. *Journal of Financial Economics*, 4 (3), pp. 323-338.

BYRD, R.H., SCHNABEL, R.B., and SHULTZ, G.A., 1988. Approximate Solution of the Trust Region Problem by Minimization over Two-Dimensional Subspaces. *Mathematical Programming*, 40, pp 247–263.

COLEMAN, T.F. and LI, Y., 1996. An Interior, Trust Region Approach for Nonlinear Minimization Subject to Bounds. *SIAM Journal on Optimization*, 6, pp. 418–445.

BRENNAN, M. J. and SCHWARTZ, E. S., 1979. A Continuous Time Approach to Pricing Bonds. *Journal of Banking and Finance*, 3, pp. 133-155.

BRENNAN, M. J. and SCHWARTZ, E. S., 1982. An Equilibrium Model of Bond Pricing and a Test of Market Efficiency. *Journal of Financial and Quantitative Analysis*, 21 (3), pp. 301-329.

BRENNER, M. and SUBRAHMANYAM, M. G., 1988. A Simple Formula to Compute the Implied Standard Deviation. *Financial Analyst Journal*, 5, pp. 80-83.

BRIGO, D. and MERCURIO, F., 2006. *Interest Rate Models - Theory and Practice: White Smile, Inflation and Credit*. 2nd edition, Berlin: Springer-Verlag.

COLEMAN, T. F. and Li, Y., 1996. An Interior, Trust Region Approach for Nonlinear Minimization Subject to Bounds. *SIAM Journal on Optimization*, 6, pp. 418-445.

COLEMAN, T. F. and Li., Y., 1994. On the Convergence of Reflective Newton Methods for Large-Scale Nonlinear Minimization Subject to Bounds. Mathematical Programming. *Mathematical Programming*, 67 (2), pp. 189-224.

CORONERO, L., NYHOLM, K. and VIDOVA-KOLEVA, R., 2011. How Arbitrage-Free is the Nelson-Siegel Model?. *Journal of Empirical Finance*, 18 (3), pp. 393-407.

CORRADO, C. J. and MILLER, Jr. T. W., 1996. A Note on a Simple, Accurate Formula to Compute Implied Standard Deviations. *Journal of Banking and Finance*, 20, pp. 595-603.

COX, J. C., INGERSOLL, J. E. and ROSS, S. A., 1985. A Theory of the Term Structure of Interest Rates. *Econometrica*, 53, pp. 385-407.

DI FRANCESCO, M., 2012. A General Gaussian Interest Rate Model Consistent With the Current Term Structure. ISRN Probability and Statistics, Volume 2012.

ECKHARDT, R., 1987. Stan Ulam, John von Neumann, and the Monte Carlo Method. *Los Alamos Science, Special Issue*, 15, pp. 131-137.

FRITSCH, F. N. and CARLSON, R. E., 1980. Monotone Piecewise Cubic Interpolation. *SIAM Journal of Numerical Analysis*, 17, pp. 238-246.

GLASSERMAN, P., 2003. *Monte Carlo Methods in Financial Engineering*. 1st edition, New York: Springer-Verlag, pp. 205-208.

GRASSELLI, M. and MIGLIETTA, G., 2014. *A Flexible Spot Multiple-Curve Model. SSRN Working Paper*. Freely available at: http://ssrn.com/abstract=2424242

HAMMERSLAY, J. M. and MORTON, K.W., 1956. A New Monte Carlo Technique: Antithetic Variates. *Mathematical Proceedings of the Cambridge Philosophical Society*, 52 (3), pp 449-475.

HERTZ, D. B., 1964. Risk Analysis in Capital Investment. *Harvard Business Review*, 42, pp. 95-106.

HULL, J. C. and WHITE, A., 1990. Pricing Interest-Rate Derivative Securities. *Review of Financial Studies*, 3 (4), pp. 573-592.

HULL, J. C. and WHITE, A., 1994. Numerical Procedures for Implementing Term Structure Models II: Two-Factor Models. *Journal of Derivatives*, 2 (2), pp. 37-48.

HULL, J. C., 2009. *Options, Futures and Other Derivatives*. 7th edition, New Jersey: Pearson.

HULL, J. C. and WHITE, A., 2012. CVA and Wrong-Way Risk. *Financial Analysts Journal*, 68 (5). Freely available at: http://ssrn.com/abstract=2151507

Li, M., 2006. *You Don't Have to Bother Newton for Implied Volatility*. SSRN Working Paper. Freely available at: http://ssrn.com/abstract=952727

MADIGAN, P., 2008. Libor Under Attack. *Risk*, June 2008. Available at: http://www.risk.net/risk-magazine/feature/1497684/libor-attack

MORÉ, J. J., 1977. *The Levenberg-Marquardt Algorithm: Implementation and Theory. Numerical Analysis*. Lecture Notes in Mathematics 630. Springer-Verlag. pp. 105-116.

MORENI, N. and PALLAVICINI, A., 2010. *Parsimonious HJM Modelling for Multiple Yield-Curve Dynamics*. SSRN Working Paper. Freely available at: http://ssrn.com/abstract=1699300

MORINI, M., 2009. *Solving the Puzzle in the Interest Rate Market*. SSRN Working Paper. Freely available at: http://ssrn.com/abstract=1506046

NELSON, C. R. and SIEGEL, A. F., 1987. Parsimonious Modelling of Yield Curves. *Journal of Business*, 60 (4), pp. 473-489.

PENG, Z. *et al.*, 2008. *Is Libor Broken?*. Fixed Income Strategies, Citigroup.

PRICE, K., STORN, R. M. and LAMPIEN, J. A., 2005. *Differential Evolution: A Practical Approach to Global Optimization (Natural Computing Series)*. 1st edition, Berlin: Springer.

RENDLEMAN, R. and BARTTER, B., 1980. *The Pricing of Options on Debt Securities*. Journal of Financial and Quantitative Analysis, 15, pp 11-24.

SCHOENMAKERS, J., 2005. *Robust Libor Modelling and Pricing of Derivative Products*. 1st edition: Chapman and Hall/CRC Financial Mathematics Series.

SCHRAGER, D. F. and PELSSER, A., 2006. Pricing Swaptions and Coupon Bond Options in Affine Term Structure Models. *Mathematical Finance*, 16 (4), pp. 673-694.

SIVANANDAM, S. N. and DEEPA, S. N., 2008. *Introduction to Genetic Algorithms*. 1st edition, Berlin: Springer-Verlag.

SVENSSON, L. E. O., 1994. *Estimating and Interpreting Forward Interest Rates: Sweden 1992-4*. International Monetary Fund, IMF Working Paper, 1994/114.

VASICEK, O., 1977. An Equilibrium Characterization of the Term Structure. *Journal of Financial Economics*, 5, pp. 177-188.

Sitography

http://www.emmi-benchmarks.eu/euribor-org/about-euribor.html - European Money Market Institute (EMMI) site, with information about Euribor interest rates.

http://www1.icsi.berkeley.edu/~storn/code.html - Differential Evolution (DE) site, with information about differential evolutionary minimization and its history.

http://www.mathworks.com/matlabcentral/fileexchange/18593-differential-evolution - Differential Evolution implementation in Matlab by Markus Buehren, used to minimize loss functions in the present work.

http://www.mathworks.com/matlabcentral/fileexchange/4540-legendre-gauss-quadrature-weights-and-nodes/content/lgwt.m - Gauss-Legendre integration implementation in Matlab by Greg von Winckel, used to calculate the G2++ European swaption price in the present work.

http://it.mathworks.com/help/matlab/index.html - Matlab online help documentation database (updated to the latest Matlab r2015a).