Politecnico di Milano
School of Industrial and Information Engineering
Master of Science in Mathematical Engineering

Neural Network calibration of the two-additive factor Gaussian model. A Machine Learning approach to Swaption pricing

Relatore: Prof. Marcello RESTELLI

Candidate:
Luca SABBIONI
Matr. 853764

Academic Year 2017-2018

Abstract

Pricing financial derivatives is one of the most important tasks for an investment bank: it is often relying on the choice of a mathematical model that describes the dynamics of interest rates. Several models depend on a set of parameters to be calibrated in order to fit market data as close as possible. It is fundamental for a calibration method to be accurate and fast, and for this purpose Machine Learning techniques are gathering increasing attention in the last years.
The aim of this project, developed with the collaboration of Banca IMI, is to propose a black-box calibration of interest rate models in a multicurve framework for swaption prices using Machine Learning techniques. In particular this thesis is focused on an Artificial Neural Network trained to calibrate the parameters of the two-additive factor Gaussian Model (G2++).
The implementation of the calibrator follows a wide analysis of the principal theorems providing pricing formulas, the main characteristics of Neural Networks and of the features provided in the dataset used. The calibration procedure is then optimized thanks to several techniques and algorithms, as well as the execution on GPU, providing a remarkable speedup.
The results of the batch and online calibrations are finally compared with the model and procedure currently used.

Keywords: Artificial Neural Network, Machine Learning, Black-Box Calibration, Pricing Model Calibration

Abstract

La procedura di pricing di derivati finanziari è una delle funzioni più importanti per una banca d'investimento: spesso si basa sulla scelta di un modello matematico che descrive la dinamica dei tassi d'interesse. Diversi modelli dipendono da un set di parametri che devono essere calibrati per poter replicare al meglio i dati di mercato. Per un metodo di calibrazione è fondamentale essere veloce e accurato, e a questo scopo negli ultimi anni stanno guadagnano attenzione crescente le tecniche di Machine Learning.
L'obiettivo di questo progetto, sviluppato con la collaborazione di Banca IMI, consiste nel proporre tramite tecnihce di Machine Learning una calibrazione black-box di un modello di tassi d'interesse in un contesto muticurve per i prezzi di swaptions. In particolare, questa tesi si focalizza su una Rete Neurale, istruita per calibrare i parametri del modello Gaussiano bifattoriale ($G2++$).
Lo sviluppo del calibratore segue un'ampia analisi dei principali teoremi da cui si ricavano le formule di pricing, delle caratteristiche salienti delle Reti Neurali e delle proprietà del dataset utilizzato. L'implementazione del calibratore è successivamente ottimizzata grazie a diverse tecniche e algoritmi, così come all'esecuzione su GPU, risultante in un notevole aumento della velocità computazionale.
I risultati delle calibrazioni batch e online sono infine comparati con il modello e le tecniche attualmente utilizzate.

Parole Chiave: Artificial Neural Network, Machine Learning, Calibrazione Black-Box, Calibrazione del Modello di Pricing

Ringraziamenti

Finalmente é giunto il momento di compiere l'ultimo passo di questo incredibile e faticoso percorso, fatto non solo di studio e duro lavoro, ma anche pieno di gioia e momenti indimenticabili. Per questo, vorrei ringraziare tutti coloro che ne hanno fatto parte e che mi hanno permesso di raggiungere questo obiettivo e diventare quello che sono ora.

Il primo ringraziamento é per il Professor Restelli, per avermi guidato in questi mesi con grande disponibilitá e energia nonostante i suoi mille progetti e impegni, e per aver sempre stimolato il mio interesse e la mia curiositÃ in un campo per me nuovo. Queste due righe certamente non bastano per ringraziarlo in modo adeguato.

Vorrei anche ringraziare tutti coloro che nel team di Financial Engineering e nell'XVA desk di BANCA IMI mi hanno seguito e consigliato, in particolare Roberto Da Luiso e Giorgio Facchinetti, i quali hanno sempre risolto tutti i miei dubbi, anche i piú banali.

Inoltre, un ringraziamento speciale va fatto a Edoardo Vittori e Andrea Donati, che mi hanno accompagnato e aiutato in tutto questo percorso con una pazienza incredibile.

Un grazie di cuore alla mia famiglia per essere sempre stata il mio punto di riferimento. Mi avete sempre sostenuto in ogni situazione, spronandomi a imparare e a diventare la persona che sono oggi.

Grazie ai miei amici di sempre, che mi hanno sempre supportato e sopportato: non posso elencarvi tutti (cito solo Filo, senza il quale questa tesi sarebbe piena di errori), ma sappiate che, anche se le nostre strade dovessero dividersi, vi porteró sempre tutti con me.

Grazie a Simon, per aver portato gioia in tutte queste settimane in biblioteca.

Vorrei anche ringraziare i miei compagni del Poli, a partire da Gra, Braga, Rot, Febo, Peri e tutti gli amici che ho conosciuto fin dal primo giorno e che hanno reso tutte le lezioni, i progetti, i viaggi in treno e le giornate di studio piú divertenti e mai banali.

Grazie a Lone per esserci sempre, per non rifiutare mai di darmi consigli e opinioni. Sappi che vali piú di quello che credi.

Un immenso ringraziamento a Pibe: non credo sia possibile trovare un compagno migliore; con la tua simpatia e intelligenza sei sempre riuscito a spronarmi ad andare avanti col sorriso. Non hai la minima idea di quanto sia felice della possibilitá di continuare a lavorare insieme.

Infine grazie a tutte le persone che non ho citato, ma che mi hanno permesso, anche con poco, di raggiungere questo traguardo.

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

ANN Artificial Neural Network.

ATM At-the-Money.

BACK Backpropagation Algorithm.

BFGS Broyden-Fletcher-Goldfarb-Shanno.

CE CrossEntropy.

CUDA Compute Unified Device Architecture.

EUR Euro.

FFNN Feed-Forward Neural Network.

FWD Forward Curve.

G2++ Two-additive Factor Gaussian Model.

GPU Graphics Processing Unit.

IRS Interest-rate swap.

ML Machine Learning.

MLP Multilayer Perceptron.

MP Multiprocessor.

MRS Mean Reversion Speed.

NPV Net Present Value.

OIS Overnight Indexed Swap.

PC Principal Component.

PCA Principal Component Analysis.

ReLu Rectified Linear Unit.

RNN Recurrent Neural Network.

SFDM Stochastic Finite Difference Method.

SL Supervised Learning.

SP Stream Processor.

TANH hyperbolic tangent.

VOL Volatility.

Chapter 1

Introduction

> In the last 30 years, derivatives have become increasingly important in Finance. Futures and options are actively traded on many exchanges throughout the world. [...]
> They play a key role in transferring a wide range of risks in the economy.
>
> ["Options, Futures and other Derivatives", J.C.Hull]

Every day a huge amount of derivatives is traded on the markets. It is consequently vital the ability of pricing financial instruments with a high degree of accuracy in a short time, especially regarding the most liquid options such as swaptions.
Speed is the curse of traders, because only through fast computations the derivatives market can be profitable.
Pricing usually relies on the choice of a mathematical model representing the dynamics of interest rates. The most common models are "parametric", since they depend on a set of parameters.
The problem arising with such models is that these parameters are, in general, not known; the only way to choose them properly consists in the direct evaluation of their influence on real market prices. This procedure is called calibration and it is the main goal of this project.
Given the high volumes of datasets regarding both present and historical records, and the quantitative nature of the subject, this field is becoming one of the best suited for the research and development of Computer Science applications: in the last years, investment banks are giving more and more attention to out-of-the-box projects involving Data Science, Artificial Intelligence and Machine Learning (ML).

1.1 Research target

The present work is the result of an intense cooperation of Politecnico di Milano with BANCA IMI. The main goal of the project is to develop a calibrator of a model using European swaptions. The data available on a daily basis, which can be used as input to the calibrator, are the market quotes of a set of swaptions, i.e., prices and volatilities of the swaptions, with the addition of information about the discounting and forward curves. Hence, starting from the current market data, the calibrator must provide the tuned set of parameters that can best describe market data.

The model currently used for swaption pricing is the Vasicek model, in which interest rates are described as stochastic processes defined by the pair (k, σ), where k is the mean reversion speed and σ is the volatility (as explained in Chapter 2).
The calibration used nowadays is a compromise between accuracy and computational speed: since the calibration on the overall set of swaptions is heavy in terms of computational times, it is performed on a subset of the most liquid options.
For this reason, there is a strong interest in the development of a new calibrator capable of using all contracts available in a short time and with suitable accuracy. The idea is to consider an Artificial Neural Network (ANN), which should be flexible enough to be applied to different interest rate models, preferably more complex than Vasicek's, such as the Two-additive Factor Gaussian Model (G2++).

1.2 Previous works and current objectives

The present work is based upon the results of the cooperation between Politecnico di Milano and BANCA IMI which started in 2016, and this is currently the third master thesis based on this project. The achievements of such a collaboration were manifold and handled slightly different topics:

- In the first step, the primarily focus went on the evaluation of the analytical model used and on a deep analysis of the multicurve context. Moreover, a Neural Network was built in a Supervised Learning (SL) manner: in fact, it was trained to fit the daily couple (k, σ) provided with the dataset in order to analyze the approximation properties of the network, without actually using any pricing formula [Cella, 2016].

- The second step is the real starting point for this thesis: the black-box calibration through an ANN was introduced and developed. The calibrator is used not to fit the given parameters, but to predict their values from market data generating a set of prices as close as possible to the real ones. The training procedure was still made on the same subset of swaptions of the original calibrator in order to be able

to compare the results and for reasons of computational times [Donati, 2018].

The above-mentioned work highlighted that there is a wide range of possible pairs (k, σ) whose error on the generated prices is almost identical to the minimal one. The problem lies in the chosen model, which is not complex enough and therefore has low expressivity. This suggested the introduction of the G2++ model as one of the main objectives of this work. This new model is based on 5 parameters instead of 2, therefore introducing an additional difficulty due to the fact that there is no more an analytic formula to price swaptions. For this reason, its implementation requires the use of numerical approximations, which in turn generate a small bias in the calibration procedure (there is not only the error generated by a non-optimal choice of the parameters, but also a small approximation error in the prices).
Another main goal is to keep computational times low enough to allow the calibration on the entire set of swaptions. Eventually, there is also the possibility to extend the project in a multicurrency scenario, considering European swaptions written on different currencies.

1.3 Outline of Contents and Contributions

This section outlines the structure of this document, explaining the original contributions that were made in this thesis.

In Chapter 2 the financial aspects regarding the project are introduced. In a first instance, it provides an explanation of the interest rate models involved: the Extended Vasicek model, used in previous works, and the G2++, introduced in this thesis project: this model increases the complexity of the problem from a bi-dimensional to a penta-dimensional framework. The latter model is then used to price some basic contracts (as Bond or Swaps), necessary for the computations of swaption prices, whose formulas are derived at the end of the Chapter.

Chapter 3, after the definition of the error measure adopted to evaluate the calibrations, provides an overview of the principal Machine Learning techniques used, starting with the Neural Network. It is followed by an overview of the optimization algorithms used, namely the CrossEntropy and the BFGS, adopted also in the previous steps of this project with some differences. Finally, the potentialities and limitations of the current calibrator are presented.

Chapter 4 contains some analysis made on the dataset used as input for the calibrator. It starts with the exploration of the features provided in the dataset, which is then manipulated through a procedure of dimensionality reduction (PCA), presented with

some further observations. This procedure was introduced in [Cella, 2016]; however, its application and related analysis are different.

In order to reduce computational times, some particular algorithms and technologies in the calibration had to be considered: they are presented in Chapter 5. As shown in the first section, the greatest speedup is gained thanks to the implementation of the pricing procedure on the GPU using CUDA architecture, in a similar manner of the previous works.
The last part shows different algorithms developed for the computation of the gradient of the error measure with respect to the parameters of the neural network: the need of these contributions emerged only in this last step of the project because of the increased model complexity and brought another remarkable gain in terms of computational times.

Chapter 6 is dedicated to the results of the experiments (batch and online calibrations on the EURO dataset) and their comparison with the bank's calibrator. Moreover, some multicurrency tests are also performed and presented.

Finally, Chapter 7 provides some comments and conclusions about the overall project and some perspectives for future works.

Chapter 2

Swaption pricing

2.1 An overview

As introduced in the previous Chapter, the main task of this thesis is the calibration of an interest rate model through the pricing of European Swaptions; but before considering all the technicalities and equations needed to find the pricing formula, there is the need to understand the basics of these derivatives.

An Interest-rate swap (IRS) is a contract between two parties agreeing to exchange payments between two differently indexed legs, starting from a future time instant. The payment made by one party (the swap payer) is based on a fixed interest rate, known as the swap rate, and in return it receives (from the receiver) a payment stream based on a floating rate. The floating rate is defined by the so-called Forward Curve (FWD), which usually corresponds to the LIBOR or EURIBOR curves. The streams of payments are called *floating* and *fixed legs*.

Swaptions are options on interest rate swaps, and are the most popular types of interest rate options. These derivatives, also called swap options, are options on an IRS. There are two main types of swaptions, a payer version and a receiver version. A European payer swaption is an option that gives the holder the right to enter a payer IRS at a specified strike price on a specified date (called expiry). At the expiry, if the swap rate of the underlying IRS is higher than the strike rate, the contract owner can exercise the option. On the contrary, in a Europen receiver swaption, the underlying IRS is a receiver one.
The length of the underlying IRS is called tenor. A representation of the cash flows involved in a swaption is depicted in Figure 2.1.

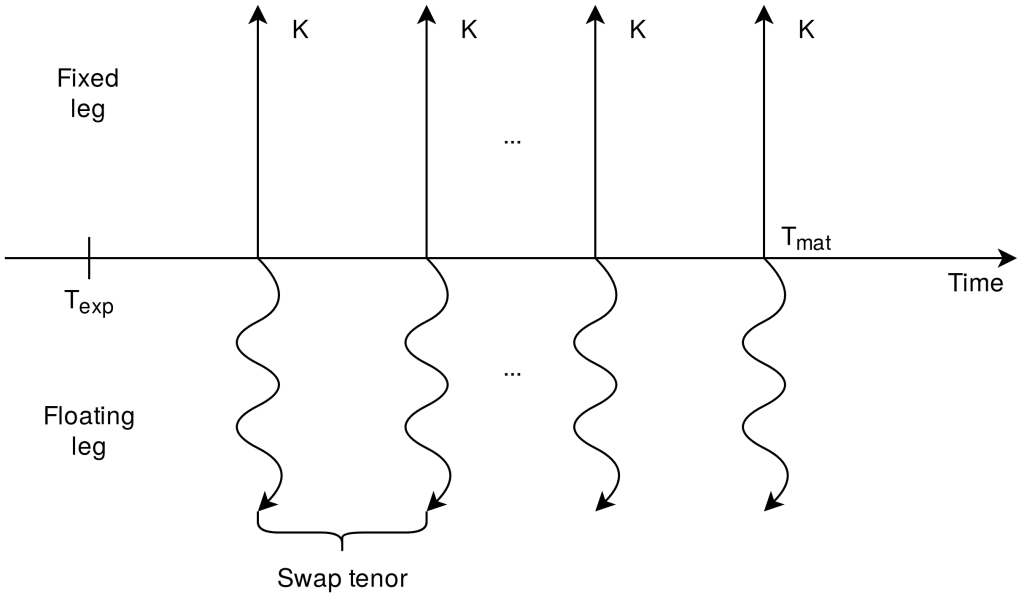


Figure 2.1: Representation of an example of Interest Rate Swap with fixed and floating payments on the same dates

As it will be shown later, swaps and swaptions pricing can be performed once the mathematical model describing the evolution of interest rates is defined; hence, the next sections deal with the definition of the short-rate model used, and it is then applied step by step in order to find the pricing formulas.

2.2 The model

2.2.1 Vasicek model

The first model considered for this project is a slight generalization of one the first models devised to simulate the behavior of the short rate: The Multicurve-Shifted Vasicek Model. In this model the short rate $r(t)$ follows the dynamics described by the following definition:

Definition 2.2.1. Let $\mathbb{Q}$ be the Risk Neutral probability. A stochastic process defined on a probability space $(\Omega, \mathscr{F}, \mathbb{Q})$ is said to be a Vasicek process assuming values in $\mathbb{R}$ if its dynamics follows the SDE:

$$dr(t) = k(\theta - r(t))dt + \sigma dW(t) + \varphi(t)$$

Where $k, \sigma, \theta > 0$; $\varphi \colon \mathbb{R} \to \mathbb{R}$ and W is a $\mathbb{R}$ Brownian motion under $\mathbb{Q}$.

In this case $W(t)$ is a Wiener process modelling the random market risk factor, and the parameters k, σ and θ characterize the dynamics of the model as follows:

- "long-term mean" θ: the value to which the rate r will converge in the long term;
- "Mean Reversion Speed (MRS)" k: the speed rate at which r will converge to the long-term mean θ after a perturbation;
- "Volatility (VOL)" σ: the factor that controls the variance of the Brownian motion entering in the system.
- "shift" $\varphi(t)$: it is a deterministic shift, depending on the curve taken into account. For every currency, when discounting, the curve considered is the Overnight Indexed Swap (OIS)[1]; in this case the shift will be referred as $\varphi_d(t)$. On the contrary, when forwarding, the one used is the forward curve with specific tenor (carrying its related risk); it will be referred as $\varphi_F(t)$.

The first goal of the project was the calibration of k and σ (once fixed θ) in order to minimize the error on predicted swaption prices. The pricing procedure using this model is very easy and fast to compute since closed-form formulas are available, as explained in [Brigo and El-Bachir, 2007] and [Brigo and Mercurio, 1998]. The main property used to find the closed-form solution is derived in [Christensen, 2007].

The main problem is that this interest rate model is "too poor" as not very sensitive to its parameters, in the sense that there is a wide range in the parameter space where the error remains almost the same. This suggested to use a more complex model, capable of providing more accurate swaption prices, thus reducing the feedback error.

2.2.2 The G2++ Model

In order to have a better approximation of interest rates and of swaption prices there is the need to add complexity with respect to Vasicek model. This goal is achieved by adding another source of randomness, while keeping the multicurve context. Hence, the model used is called Two-Additive-Factor Gaussian Model, or G2++.

In this model the short rate $r(t)$ follows the dynamics described in the following Definition:

Definition 2.2.2. Let $\mathbb{Q}$ be the Risk Neutral probability. A stochastic process defined on a probability space $(\Omega, \mathscr{F}, \mathbb{Q})$ is said to be a G2++ process assuming values in $\mathbb{R}$ if:

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

[1]The Overnight Indexed Swap (OIS) curve is currently the most common choice for the risk-free discount rate.

where $\varphi(t) : \mathbb{R} \to \mathbb{R}$ such that $\varphi(0) = r_0$;
the processes $\{x(t) : t \geq 0\}$ and $\{y(t) : t \geq 0\}$ satisfy

$$\begin{aligned} dx(t) &= -ax(t)dt + \sigma dW_1(t) \quad x(0) = 0 \\ dy(t) &= -by(t)dt + \eta dW_2(t) \quad y(0) = 0 \end{aligned} \tag{2.1}$$

and (W_1, W_2) is a two-dimensional $\mathbb{Q}$-Brownian motion with instantaneous correlation ρ:

$$dW_1(t)dW_2(t) = \rho dt. \tag{2.2}$$

$\varphi(t)$ is the same deterministic shift as the previous case, still diversified in discount and forward shift.

The parameters to be calibrated are a, b, σ, η, ρ.

This model presents two sources of randomness: one is usually meant to lead the dynamics on the short term, the other is more important on the long term (for this reason the "long-term mean" θ is no longer used).
The dynamics of the processes x and y can be also expressed in terms of two independent Brownian motions $\widetilde{W_1}$ and $\widetilde{W_2}$:

$$\begin{aligned} dx(t) &= -ax(t)dt + \sigma d\widetilde{W_1}(t) \\ dy(t) &= -by(t)dt + \eta\rho d\widetilde{W_1}(t) + \eta\sqrt{1-\rho^2}d\widetilde{W_2}, \end{aligned} \tag{2.3}$$

where the independent Brownian motions are related to the previous ones through Cholesky decomposition

$$\begin{aligned} dW_1(t) &= d\widetilde{W_1}(t) \\ dW_2(t) &= \rho d\widetilde{W_1}(t) + \sqrt{1-\rho^2}d\widetilde{W_2}(t). \end{aligned} \tag{2.4}$$

Using Ito's formula it is easy to obtain $x(t)$ and $y(t)$:

Proposition 2.2.1.

$$\begin{aligned} x(t) &= x(s)e^{-a(t-s)} + \sigma \int_s^t e^{-a(t-u)}dW_1(u) \quad \forall s < t \\ y(t) &= y(s)e^{-b(t-s)} + \eta \int_s^t e^{-b(t-u)}dW_2(u) \quad \forall s < t \end{aligned} \tag{2.5}$$

in particular, for $s = 0$:

$$\begin{aligned} x(t) &= \sigma \int_0^t e^{-a(t-u)}dW_1(u) \\ y(t) &= \eta \int_0^t e^{-b(t-u)}dW_2(u). \end{aligned} \tag{2.6}$$

From this Proposition, immediately follows the next one:

Proposition 2.2.2. *$r(t)$, conditional on $\mathscr{F}_s$ is a normal random variable, with mean and variance given by*

$$\begin{aligned}\mathbb{E}[r(t)|\mathscr{F}_s] &= x(s)e^{-a(t-s)} + y(s)e^{-b(t-s)} + \varphi(t)\\ Var[r(t)|\mathscr{F}_s] &= \frac{\sigma^2}{2a}\big[1-e^{-2a(t-s)}\big] + \frac{\eta^2}{2b}\big[1-e^{-2b(t-s)}\big] + 2\rho\frac{\sigma\eta}{a+b}\big[1-e^{-(a+b)(t-s)}\big].\end{aligned} \tag{2.7}$$

2.3 Bond pricing

The most basic contract that must be priced is without any doubt the Zero Coupon Bond. The pricing formulas that we are going to see will always start from the computation of bond prices. Arbitrage Free arguments let us compute easily bond prices as the expectation of a functional of the stochastic process r under the Risk-Neutral measure. Hence, let us introduce the general Risk-Neutral Valuation Formula for every contingent claim:

Theorem 2.3.1 (Risk-Neutral Valuation Formula)**.** *Let χ be a contingent T-claim of the form $\chi = X(T)$. In an arbitrage-free market the price $X(t)$ is given by*

$$X(t) = \mathbb{E}\big[D(t,T)X(T)|\mathscr{F}_t\big] = \mathbb{E}\big[exp(-\int_t^T r(s)ds)X(T)|\mathscr{F}_t\big] \tag{2.8}$$

where the expectation is made with respect to the Risk-Neutral probability.

Considering that a zero-coupon bond is a contingent claim where $X(T) = 1$, the following proposition holds.

Proposition 2.3.2.

$$\begin{aligned}P(t,T) :&= \mathbb{E}[exp\{-\int_t^T r(s)ds\}|\mathscr{F}_t]\\ &= e^{-\int_t^T \varphi(s)ds}\ \ \mathbb{E}[exp\{-\int_t^T (x(u)+y(u))du\}|\mathscr{F}_t]\\ &= \Phi(t,T)\ \ \mathbb{E}[exp\{-I(t,T)\}|\mathscr{F}_t]\end{aligned}$$

where

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

and

$$\Phi(t,T) := e^{-\int_t^T \varphi(s)ds}$$

.

The following Lemma allows us to obtain an analytic formula for bond prices.

Lemma 2.3.3. *The random variable $I(t,T)$, conditional to the sigma-field $\mathscr{F}_t$, is normally distributed $\forall t, T > 0$.*
Specifically:

$$I(t,T) \sim N(M(t,T), V(t,T))$$

where

$$\begin{aligned}
M(t,T) &= \frac{1-e^{-a(T-t)}}{a}x(t) + \frac{1-e^{-b(T-t)}}{b}y(t) = B(a,t,T)x(t) + B(b,t,T)y(t) \\
V(t,T) = &\quad \frac{\sigma^2}{a^2}[T-t+\frac{2}{a}e^{-a(T-t)} - \frac{1}{2a}e^{-2a(T-t)} - \frac{3}{2a}] \\
&+ \frac{\eta^2}{b^2}[T-t+\frac{2}{b}e^{-b(T-t)} - \frac{1}{2b}e^{-2b(T-t)} - \frac{3}{2b}] \\
&+ 2\rho\frac{\sigma\eta}{ab}[T-t+\frac{e^{-a(T-t)}-1}{a} + \frac{e^{-b(T-t)}-1}{b} - \frac{e^{-(a+b)(T-t)}-1}{a+b}] \\
B(z,t,T) &:= \frac{1-e^{-z(T-t)}}{z}
\end{aligned}$$

Proof. ([Brigo and Mercurio, 2001], Lemma 4.2.1).
Using stochastic integration by part

$$\begin{aligned}
\int_t^T x(u)du &= Tx(T) - tx(t) - \int_t^T u dx(u) \\
&= Tx(t) + T\int_t^T dx(u) - tx(t) - \int_t^T u dx(u) \\
&= \int_t^T (T-u)dx(u) + (T-t)x(t)
\end{aligned}$$

If we focus on the last integral, we can use the definition of x and Proposition 2.2.1 to obtain

$$\begin{aligned}
\int_t^T (T-u)dx(u) &= -a\int_t^T (T-u)x(u)du + \sigma\int_t^T (T-u)dW_1(u) \\
&= -a\int_t^T (T-u)\big[x(t)e^{-a(u-t)} + \sigma\int_t^u e^{-a(u-s)}dW_1(s)\big]du \\
&\quad + \sigma\int_t^T (T-u)dW_1(u)
\end{aligned}$$

We can compute separately the integrals:

$$-ax(t)\int_t^T (T-u)e^{-a(u-t)}du = -x(t)(T-t) - \frac{e^{-a(T-t)-1}}{a}x(t)$$

and, using integration by parts,

$$\begin{aligned}
&-a\sigma\int_t^T (T-u)\int_t^u e^{-a(u-s)}dW_1(s)du\\
&\quad=-a\sigma\int_t^T \big(\int_t^u e^{as}dW_1(s)\big)\partial_u\big(\int_t^u (T-v)e^{-av}dv\big)\\
&\quad=-a\sigma\Big[\big(\int_t^T e^{au}dW_1(u)\big)\big(\int_t^T (T-v)e^{-av}dv\big)\\
&\qquad\qquad-\int_t^T\Big(\int_t^u (T-v)e^{-av}dv\Big)e^{au}dW_1(u)\Big]\\
&\quad=-a\sigma\Big[\int_t^T\big(\int_u^T (T-v)e^{-av}dv\big)e^{au}dW_1(u)\Big]\\
&\quad=-a\sigma\Big[\int_t^T\Big(\frac{(T-u)e^{-au}}{a}+\frac{e^{-aT}-e^{-au}}{a^2}\Big)e^{au}dW_1(u)\Big]\\
&\quad=-\sigma\int_t^T\Big[(T-u)+\frac{e^{-a(T-u)}-1}{a}\Big]dW_1(u)
\end{aligned}$$

Adding all the previous terms:

$$\begin{aligned}
\int_t^T x(u)du =&(T-t)x(t)+\sigma\int_t^T (T-u)dW_1(u)-x(t)(T-t)-\frac{e^{-a(T-t)}-1}{a}x(t)\\
&-\sigma\int_t^T (T-u)dW_1(u)-\sigma\int_t^T\frac{e^{-a(T-u)}-1}{a}dW_1(u)\\
=&\frac{1-e^{-a(T-t)}}{a}x(t)+\frac{\sigma}{a}\int_t^T\Big[1-e^{-a(T-u)}\Big]dW_1(u)
\end{aligned}$$

The same argument can be done for y, so that

$$\begin{aligned}
\int_t^T x(u)du =&\frac{1-e^{-a(T-t)}}{a}x(t)+\frac{\sigma}{a}\int_t^T\Big[1-e^{-a(T-u)}\Big]dW_1(u)\\
\int_t^T y(u)du =&\frac{1-e^{-b(T-t)}}{b}y(t)+\frac{\eta}{b}\int_t^T\Big[1-e^{-b(T-u)}\Big]dW_2(u)
\end{aligned}$$

So it is clear that $I(t,T)$ is normally distributed, and $M(t,T)$ is directly verified, too. For what concerns the conditional variance $V(t,T)$, we have

$$\begin{aligned}
&Var\Big\{I(t,T)|\mathscr{F}_t\Big\}\\
=&Var\Big\{\frac{\sigma}{a}\int_t^T\big[1-e^{-a(T-u)}\big]dW_1(u)+\frac{\eta}{b}\int_t^T\big[1-e^{-b(T-u)}\big]dW_2(u)\mid\mathscr{F}_t\Big\}\\
=&\frac{\sigma^2}{a^2}\int_t^T\big[1-e^{-a(T-u)}\big]^2du+\frac{\eta^2}{b^2}\int_t^T\big[1-e^{-b(T-u)}\big]du\\
&+\qquad 2\rho\frac{\sigma\eta}{ab}\int_t^T\big[1-e^{-a(T-u)}\big]\big[1-e^{-b(T-u)}\big]du
\end{aligned}$$

Simple integration proves the hypothesis. □

Thanks to this lemma we are now able to prove the following theorem:

Theorem 2.3.4.

$$\begin{aligned} P(t,T) &= \Phi(t,T) \cdot exp\{-M(t,T) + \frac{1}{2}V(t,T)\} \\ &= \Phi(t,T)A(t,T) \cdot exp\{-B(a,t,T)x(t) - B(b,t,T)y(t)\} \end{aligned} \tag{2.9}$$

Where $A(t,T) = exp(V(t,T)/2)$.

Proof. Starting from Proposition 2.3.2, it is enough to recall that the moment-generating function of a normal random variable $X \sim N(\mu, \sigma^2)$ is

$$\mathbb{E}[\exp(uX)] = \exp(\mu u + \sigma^2 u^2/2).$$

By using Lemma 2.3.3, the theorem is easily proven. □

Remark. Since the deterministic shift φ varies through discounting and forwarding, we need to distinguish also between $\Phi_d(t,T)$ for discounting and $\Phi_F(t,T)$ for forwarding. Obviously, the same notation will be used on $P(t,T)$. This shift is not directly available, but must be derived from market curves. For example, if the goal is to derive $\Phi_d(t,T)$, we will consider the OIS discount curve $P_{OIS}(t_0,T)$ starting from the settlement date t_0. The other curve to be considered is the risk-free one coming from the model, i.e., without the shift.

$$\overline{P}(t,T) = A(t,T) \cdot exp\{-B(a,t,T)x(t) - B(b,t,T)y(t)\}. \tag{2.10}$$

Now, the shift $\Phi_d(t,T)$ can be found easily.

$$\begin{aligned} \Phi_d(t,T) &= \frac{P_{OIS}(t,T)}{\overline{P}(t,T)} \\ &= \frac{P_{OIS}(t_0,T)}{P_{OIS}(t_0,t)} \frac{\overline{P}(t_0,t)}{\overline{P}(t_0,T)} \end{aligned} \tag{2.11}$$

Analogously, The forward curve $P_{FWD}(t_0,T)$ allows the computation of $\Phi_F(t,T)$.

2.4 Change of probability and T-Forward measure

2.4.1 Generalities

In order to find the pricing formula for swaptions, we need to change the probability measure in the so-called T-forward measure. Hence, we will give a brief (and not complete nor technical) introduction of the mathematical instruments and theorems used. The notations and propositions for this section are referenced to "Aribtrage Theory in continuous time" [Björk, 2009].
The fundamentals of the theory of changes of probability measures for stochastic processes are Girsanov's theorem and the change of numeraire.

Theorem 2.4.1 (Girsanov's theorem)**.** *Let $W^{\mathbb{P}}$ be a $\mathbb{P}$-standard d-dimensional Brownian motion on the filtered probability space $(\Omega, \mathscr{F}, \mathbb{P}, \{\mathscr{F}_t\}_t)$;*
Let ϕ_t be a d-dimensional column process (called Girsanov kernel), adapted with respect to $\mathscr{F}_t$.
Fixed T, define a new process L on $[0,T]$ such that:

$$dL_t = \phi_t' L_t dW_t^{\mathbb{P}} \quad L_0 = 1 \tag{2.12}$$

i.e.,

$$L_t = exp\left(\int_0^t \phi_s dW_s^{\mathbb{P}} - \frac{1}{2}\int_0^t \|\phi_s\|^2 ds\right). \tag{2.13}$$

Under the hypothesis that L_t is a martingale w.r.t. $\mathscr{F}_t$ and that $\mathbb{E}^{\mathbb{P}}[L_T] = 1$, it can be possible to define a new probability measure $\mathbb{Q}$ on $\mathscr{F}_T$ where L_T is the Radon-Nikodym derivative (or likelihood process):

$$L_T = \frac{d\mathbb{Q}_T}{d\mathbb{P}|_{\mathscr{F}_T}}.$$

In this way, the stochastic process $W_t^{\mathbb{Q}}$ defined with dynamics

$$dW_t^{\mathbb{Q}} = dW_t^{\mathbb{P}} - \phi_t dt$$

is a Brownian motion on the new probability space $(\Omega, \mathscr{F}, \mathbb{Q}, \{\mathscr{F}_t\}_t)$.

Corollary. *Under the same hypotheses of Girsanov's theorem, let X_t be a Ito process defined on $(\Omega, \mathscr{F}, \mathbb{P}, \{\mathscr{F}_t\}_t)$ such that*

$$dX_t = F_t dt + G_t dW_t^{\mathbb{P}}.$$

Hence, X_t is a Ito process also in $(\Omega, \mathscr{F}, \mathbb{Q}, \{\mathscr{F}_t\}_t)$ with dynamics

$$dX_t = (F_t + G_t\phi_t)dt + G_t dW_t^{\mathbb{Q}}.$$

The main application of these properties is the change of numeraire:

Definition 2.4.1. A *numeraire* is any positive non-dividend-paying asset.

In the general pricing formula in Theorem 2.3.1 we are considering as numeraire the "money market account" $B(t)$, which is a price process of a risk-free asset, with dynamics

$$dB(t) = r(t)B(t)dt, \qquad B(0) = 1. \tag{2.14}$$

As stressed in [Björk, 2009], the problem with the general pricing formula in Theorem 2.3.1 from a computational point of view is that "in order to compute the expected value we have to get hold of the joint distribution (under $\mathbb{Q}$) of the two stochastic variables $\int_0^T r(s)ds$ and $X(T)$, and finally we have to integrate with respect to that distribution. Thus we have to compute a double integral, and in most cases this turns out to be rather hard work".

The goal of the change of numeraire is to consider a new asset as numeraire, so to obtain a more suitable pricing formula.

Proposition 2.4.2 ([Björk, 2009], proposition 26.4)**.** *Assume that $\mathbb{Q}_0$ is a martingale measure for the numeraire S_0 (on $\mathscr{F}_T$), and assume that S_1 is a positive asset price process such that $S_1(t)/S_0(t)$ is a martingale in $\mathbb{Q}_0$. Define $\mathbb{Q}_1$ on $\mathscr{F}_T$ by the likelihood process*

$$L_0^1(t) = \frac{S_0(0)}{S_1(0)} \frac{S_1(t)}{S_0(t)}.$$

Then $\mathbb{Q}_1$ is a martingale measure for S_1.

This result is very convenient, because it is possible to consider as new numeraire the bond price $p(t,T)$. In fact, our models were specified under the Risk-Neutral martingale measure $\mathbb{Q}$ with the money account B as the numeraire. Zero-Coupon Bonds are positive asset price processes such that $p(t,T)/B(t)$ is a martingale. Hence the theorem holds, and it is possible to adopt a new martingale measure for $p(t,T)$, which is called T-forward measure $\mathbb{Q}^T$.

The most convenient property of this new measure is expressed in the following Proposition:

Proposition 2.4.3 (Pricing formula under T-forward measure)**.** *For any T-claim χ:*

$$X(t) = p(t,T)\mathbb{E}^T\big[X(T)|\mathscr{F}_t\big],$$

where $\mathbb{E}^T$ denotes the expected value under $\mathbb{Q}^T$.

This is exactly what we were looking for, a pricing formula that simplifies the general one. The goal now is to express the dynamics under the new probability measure; again there is a helpful property.

Proposition 2.4.4 ([Björk, 2009], Proposition 26.7)**.** *If $\mathbb{Q}$ denotes the Risk-Neutral martingale measure, then the likelihood process*

$$L^T(t) = \frac{d\mathbb{Q}^T}{d\mathbb{Q}}, \qquad \text{on } \mathscr{F}_t,\ 0 \le t \le T$$

is given by

$$L^T(t) = \frac{p(t,T)}{B(t)p(0,T)}. \tag{2.15}$$

In particular, if the $\mathbb{Q}$-dynamics of the T-bond are Wiener driven, i.e. of the form

$$dp(t,T) = r(t)p(t,T)dt + p(t,T)v(t,T)dW(t), \tag{2.16}$$

where W is a $\mathbb{Q}$-Wiener process, then L^T dynamics are given by

$$dL^T(t) = L^T(t)v(t,T)dW(t). \tag{2.17}$$

i.e., the Girsanov kernel for the transition from $\mathbb{Q}$ to $\mathbb{Q}^T$ is given by the T-bond volatility $v(t,T)$.

2.4.2 Application on G2++ model

Now it is time to apply the last propositions in order to find the dynamics of $r(t)$ under the new forward measure; in order to do that we need to derive an explicit formula for the kernel function $v(t,T)$. So, if we manage to write the dynamics $p(t,T)$ as in Equation 2.16, then Proposition 2.4.4 gives us what is needed.

Proposition 2.4.5. *In G2++ model, the price of a zero-coupon bond with maturity T satisfies the stochastic differential equation*

$$dP(t,T) = r(t)P(t,T)dt - \sigma B(a,t,T)P(t,T)dW_1(t) - \eta B(b,t,T)P(t,T)dW_2(t). \tag{2.18}$$

Proof. Let us start with some simple computations of the derivatives of $B(z,t,T)$ and $V(t,T)$:

$$\begin{aligned} B(z,t,T) &:= \frac{1-e^{-z(T-t)}}{z} \\ dB(z,t,T) &= -e^{-z(T-t)}dt = \big[zB(z,t,T) - 1\big]dt \end{aligned}$$

$$
\begin{aligned}
dV(t,T) =& \Big\{ \frac{\sigma^2}{a^2}\big[-1 + 2e^{-a(T-t)} - e^{-2a(T-t)}\big] \\
&+ \frac{\eta^2}{b^2}\big[-1 + 2e^{-b(T-t)} - e^{-2b(T-t)}\big] \\
&+ 2\rho\frac{\sigma\eta}{ab}\big[-1 + e^{-a(T-t)} + e^{-b(T-t)} - e^{-(a+b)(T-t)}\big] \Big\} dt \\
=& \big[-\sigma^2 B^2(a,t,T) - \eta^2 B^2(b,t,T) - 2\rho\sigma\eta B(a,t,T)B(b,t,T)\big] dt
\end{aligned}
$$

Now, consider

$$
R(t,T) = -B(a,t,T)x(t) - B(b,t,T)y(t) + V(t,T)/2.
$$

We want to compute its dynamics. For simplicity we will use a shorter notation $B(a) := B(a,t,T)$

$$
\begin{aligned}
dR(t,T) =& - dB(a)x(t) - B(a)dx(t) - dB(b)y(t) - B(b)dy(t) + dV(t,T)/2 \\
=& - aB(a)x(t)dt + x(t)dt + aB(a)x(t) - \sigma B(a)dW_1(t) \\
& - bB(b)y(t)dt + y(t)dt + bB(b)y(t) - \eta B(b)dW_2(t) + dV(t,T)/2 \\
=& (r(t) - \varphi(t))dt + dV(t,T)/2 - \sigma B(a)dW_1(t) - \eta B(b)dW_2(t)
\end{aligned}
$$

Now, since $P(t,T) = \Phi(t,T)exp(R(t,T))$, using Ito's Lemma:

$$
\begin{aligned}
dP(t,T) =& \varphi(t)P(t,T) + P(t,T)dR(t,T) \\
&+ \frac{1}{2}P(t,T)\big[\sigma^2 B^2(a) + \eta^2 B^2(b) + 2\rho\sigma\eta B(a)B(b)\big] dt.
\end{aligned}
$$

Replacing the dynamics of R and V we obtain the result. □

Rewriting this last result in terms of independent Brownian motions, Equation 2.4 leads to:

$$
\begin{aligned}
dP(t,T) =& r(t)P(t,T)dt - P(t,T)\begin{bmatrix}\sigma B(a) & \eta B(b)\end{bmatrix}\begin{bmatrix} dW_1(t) \\ dW_2(t)\end{bmatrix} \\
=& r(t)P(t,T)dt - P(t,T)\begin{bmatrix}\sigma B(a) & \eta B(b)\end{bmatrix}\begin{bmatrix} 1 & 0 \\ \rho & \sqrt{1-\rho^2}\end{bmatrix}\begin{bmatrix} d\widetilde{W}_1(t) \\ d\widetilde{W}_2(t)\end{bmatrix}.
\end{aligned} \tag{2.19}
$$

As a consequence, it is possible to apply Girsanov's Theorem using as kernel:

$$
v(t,T) = \begin{bmatrix}\sigma B(a,t,T) + \eta B(b,t,T)\rho & \eta\sqrt{1-\rho^2}B(b,t,T)\end{bmatrix}. \tag{2.20}
$$

This let us define a new couple of independent Brownian motions under the T-forward measure

$$
\begin{aligned}
d\widetilde{W}_1^T =& d\widetilde{W}_1 + \big[\frac{\sigma}{a}(1 - e^{-a(T-t)}) + \rho\frac{\eta}{b}(1 - e^{-b(T-t)})\big] dt \\
d\widetilde{W}_2^T =& d\widetilde{W}_2 + \sqrt{1-\rho^2}\frac{\eta}{b}(1 - e^{-b(T-t)})dt.
\end{aligned} \tag{2.21}
$$

Now Equation 2.4 can be used back to consider again a two-dimensional T-Brownian motion with the same correlation matrix as the initial one.

Finally, combining this with Equation 2.1 we obtain the new dynamics of the processes x and y under the new T-forward measure

$$\begin{aligned} dx(t) &= [-ax(t) - \frac{\sigma^2}{a}(1 - e^{-a(T-t)}) - \rho\frac{\sigma\eta}{b}(1 - e^{-b(T-t)})] + \sigma dW_1^T(t) \\ dy(t) &= [-by(t) - \frac{\eta^2}{b}(1 - e^{-b(T-t)}) - \rho\frac{\sigma\eta}{a}(1 - e^{-a(T-t)})] + \sigma dW_2^T(t), \end{aligned} \tag{2.22}$$

where $W_1^T(t)$ e $W_2^T(t)$ are such that $dW_1^T(t)dW_2^T(t) = \rho dt$.

As a consequence, using simple integration, the following holds:

Proposition 2.4.6. *r(t), conditional on $\mathscr{F}_0$, is a normal random variable, with mean and variance given by*

$$\begin{aligned} \nu(0,t) &= \mathbb{E}^T\left[r(t)|\mathscr{F}_0\right] = -M_x^T(0,t) - M_y^T(0,t) + \varphi(t) \\ \xi^2(0,t) &= \mathbb{V}ar^T\left[r(t)|\mathscr{F}_0\right] = \sigma_x^2 + \sigma_y^2 + 2\rho_{xy}\sigma_x\sigma_y \end{aligned} \tag{2.23}$$

where

$$\begin{aligned} M_x^T(0,t) &= (\frac{\sigma^2}{a^2} + \rho\frac{\sigma\eta}{ab})[1 - e^{-at}] - \frac{\sigma^2}{2a^2}[e^{-a(T-t)} - e^{-a(T+t)}] - \rho\frac{\sigma\eta}{b(a+b)}[e^{-b(T-t)} - e^{-bT-at}] \\ M_y^T(0,t) &= (\frac{\eta^2}{b^2} + \rho\frac{\sigma\eta}{ab})[1 - e^{-bt}] - \frac{\eta^2}{2b^2}[e^{-b(T-t)} - e^{-b(T+t)}] - \rho\frac{\sigma\eta}{a(a+b)}[e^{-a(T-t)} - e^{-aT-bt}] \\ \sigma_x &= \sigma\sqrt{\frac{1 - e^{-2at}}{2a}} \\ \sigma_y &= \eta\sqrt{\frac{1 - e^{-2bt}}{2b}} \\ \rho_{xy} &= \rho\frac{\sigma\eta}{(a+b)\sigma_x\sigma_y}[1 - e^{-(a+b)t}]. \end{aligned} \tag{2.24}$$

2.5 Swap Pricing

Let's see now the pricing formula for an European swaption when discount and forward rates are described by G2++ model, with a different shift Φ. First of all, we need to understand how to price a Swap.

2.5.1 Swap

We want to price in t a payer IRS contract, whose cash flows start in T_α. The notional is considered equal to 1. At every instant T_i in $\Upsilon = \{T_{\alpha+1}, \dots, T_\beta\}$ the fixed leg pays out the amount $K\tau_i$, where K is the fixed rate and τ_i is the year fraction between T_{i-1} and T_i (convention 30/360). From the other hand, at every instant S_i in $\overline{\Upsilon} = \{S_{\overline{\alpha}+1}, \dots, S_{\overline{\beta}}\}$, the floating leg pays the amount $\tau_i L(S_{i-1}, S_i)$, where L is the simply-compounded spot interest rate, which is the interest rate resetting at the previous instant S_{i-1} for the maturity given by the current payment instant S_i (convention Act/360).

The price of this contract is really simple: it is equal to the difference of the value in t of the discounted cash flows of the floating leg (called Net Present Value (NPV)) and the NPV of the fixed leg.

In particular, the simply-compounded spot interest rate with tenor F is computed as follows

$$
\begin{aligned}
L(S_{i-1}, S_i) =& \frac{1}{\tau_i} \frac{1 - P_F(S_{i-1}, S_i)}{P_F(S_{i-1}, S_i)} \\
\tau_i L(S_{i-1}, S_i) =& \mathbb{E}\big[D(t,T) \frac{1 - P_F(S_{i-1}, S_i)}{P_F(S_{i-1}, S_i)} | \mathscr{F}_T\big] \\
=& \mathbb{E}\big[\Phi_d(t, S_i) exp\{-\int_t^{S_i} r(s)ds\}\big(\frac{1}{P_F(S_{i-1}, S_i)} - 1\big) | \mathscr{F}_T\big] \\
=& \mathbb{E}\big[\frac{\Phi_d(t, S_i)}{\Phi_F(S_{i-1}, S_i)} exp\{-\int_t^{S_{i-1}} r(s)ds\} \frac{\mathbb{E}[exp\{-\int_{S_{i-1}}^{S_i} r(s)ds\} | \mathscr{F}_{S_{i-1}}]}{\mathbb{E}[exp\{-\int_{S_{i-1}}^{S_i} r(s)ds\} | \mathscr{F}_{S_{i-1}}]} | \mathscr{F}_t\big] - P_d(t, S_i) \\
=& \mathbb{E}\big[\frac{\Phi_d(t, S_i)}{\Phi_F(S_{i-1}, S_i)} exp\{-\int_t^{S_{i-1}} r(s)ds\} | \mathscr{F}_t\big] - P_d(t, S_i) \\
=& \mathbb{E}\big[\frac{\Phi_d(t, S_{i-1}) \Phi_d(S_{i-1}, S_i)}{\Phi_F(S_{i-1}, S_i)} exp\{-\int_t^{S_{i-1}} r(s)ds\} | \mathscr{F}_t\big] - P_d(t, S_i) \\
=& \left[\frac{\Phi_d(S_{i-1}, S_i)}{\Phi_F(S_{i-1}, S_i)} P_d(t, S_{i-1}) - P_d(t, S_i)\right]
\end{aligned}
\tag{2.25}
$$

In this way, we are able to compute the NPV of the swap in t just as the difference between the legs' discounted cash flows.

$$
\begin{aligned}
SWAP(t,\Upsilon,\overline{\Upsilon},K) &= \sum_{i=\overline{\alpha}+1}^{\overline{\beta}} \left[\frac{\Phi_d(S_{i-1},S_i)}{\Phi_F(S_{i-1},S_i)} P_d(t,S_{i-1}) - P_d(t,S_i)\right] - K\sum_{i=\alpha+1}^{\beta}\tau_i P_d(t,T_i) \\
&= \sum_{i=\overline{\alpha}+1}^{\overline{\beta}} \frac{\Phi_d(S_{i-1},S_i)}{\Phi_F(S_{i-1},S_i)} \Phi_d(t,S_{i-1})A(t,S_{i-1})exp\Big\{-M(t,S_{i-1})\Big\} \\
&\quad - \sum_{i=\overline{\alpha}+1}^{\overline{\beta}} \Phi_d(t,S_i)A(t,S_i)exp\Big\{-M(t,S_i)\Big\} \\
&\quad - K\sum_{i=\alpha+1}^{\beta}\tau_i\Phi_d(t,T_i)A(t,T_i)exp\Big\{-M(t,T_i)\Big\}.
\end{aligned}
\tag{2.26}
$$

ATM swaps

A particular mention must be given to At-the-Money (ATM) swaps: they are swaps whose fixed rate is such that the total NPV is equal to zero. In order to compute the par swap rate in t we just need to put $SWAP(r_t,t,\Upsilon,\overline{\Upsilon},\widetilde{K})=0$, so that

$$
\widetilde{K} = \frac{\sum_{i=\overline{\alpha}+1}^{\overline{\beta}}\left[\frac{\Phi_d(S_{i-1},S_i)}{\Phi_F(S_{i-1},S_i)}P_d(t,S_{i-1}) - P_d(t,S_i)\right]}{\sum_{i=\alpha+1}^{\beta}\tau_i P_d(t,T_i)}. \tag{2.27}
$$

Useful notation

In the next section, we will see how the Swaps' pricing Formula (2.26) will be used with $x(t)$ and $y(t)$ as variables. For this reason, we will adopt a shorter notation, so as to express the pricing formula as a unique sum of exponential terms. .
Let's begin defining:

$$
\begin{aligned}
d_i^1 &= \frac{\Phi_d(S_{i-1},S_i)}{\Phi_F(S_{i-1},S_i)}\Phi_d(t,S_{i-1})A(t,S_{i-1}) && i=\overline{\alpha}+1,...,\overline{\beta} \\
d_i^2 &= -\Phi_d(t,S_i)A(t,S_i) && i=\overline{\alpha}+1,...,\overline{\beta} \\
d_i^3 &= -K\tau_i\Phi_d(t,T_i)A(t,T_i). && i=\alpha+1,...,\beta
\end{aligned}
\tag{2.28}
$$

Now we operate an union among all the dates ($\Gamma=\Upsilon\cup\overline{\Upsilon}$) and define a unique sequence of coefficients c_j (that sums up all the d_i^n) and times $t_j\in\Gamma$ in order to obtain:

$$
\begin{aligned}
SWAP(t,\Gamma,K) &= \sum_{j\in\Gamma} c_j exp\Big\{-M(t,t_j)\Big\} \\
&= \sum_j c_j exp\Big\{-B(a,t,t_j)x(t) - B(b,t,t_j)y(t)\Big\}.
\end{aligned}
\tag{2.29}
$$

2.6 Swaption Pricing

All the building blocks needed to compute the price of a European swaption have been introduced. Swaptions are options written on a swap starting at the expiry T_α with swap rate equal to the strike K; hence the following pricing formula holds:

$$SO(t, T_\alpha, \Gamma, K) = \mathbb{E}\Big[D(t, T_\alpha)(SWAP(T_\alpha, \Gamma, K))^+ | \mathscr{F}_t\Big] \tag{2.30}$$

where

$$\begin{aligned} SWAP(T_\alpha, \Gamma, K) &= \sum_{j\in\Gamma} c_j exp\Big\{-M(T_\alpha, t_j)\Big\} \\ &= \sum_{j\in\Gamma} c_j exp\Big\{-B(a, T_\alpha, t_j)x(T_\alpha) - B(b, T_\alpha, t_j)y(T_\alpha)\Big\}. \end{aligned}$$

The focus in this project will go only on ATM swaptions, in which the strike rate K is equal to the par swap rate $\widetilde{K}$.

We will use a slight modification of Theorem 4.2.3 of Brigo-Mercurio to find the final pricing formula that we are going to use.

Theorem 2.6.1.

$$SO(t, T_\alpha, \Gamma, K) = P_d(t, T_\alpha) \int_{-\infty}^{+\infty} \frac{e^{-\frac{1}{2}(\frac{x-\mu_x}{\sigma_x})^2}}{\sigma_x\sqrt{2\pi}} \Big[\sum_{j\in\Gamma} \lambda_j(x) e^{\kappa_j(x)} \Phi[-h_j(x)]\Big] dx \tag{2.31}$$

Where

$$\begin{aligned} h_j(x) &= \overline{h} + B(b, T, t_j)\sigma_y\sqrt{1-\rho_{xy}^2} \\ \overline{h} &= \frac{\overline{y}(x) - \mu_y}{\sigma_y\sqrt{1-\rho_{xy}^2}} - \frac{\rho_{xy}(x-\mu_x)}{\sigma_x\sqrt{1-\rho_{xy}^2}} \\ \lambda_j(x) &= c_j e^{-B(a,T,T_j)x} \\ \kappa_j(x) &= -B(b, T, t_j)\Big[\mu_y - \frac{1}{2}(1-\rho_{xy}^2)\sigma_y^2 B(b, T, t_j) + \rho_{xy}\sigma_y\frac{x-\mu_x}{\sigma_x}\Big] \\ \mu_x &= -M_x^{T_\alpha}(0, T_\alpha) \\ \mu_y &= -M_y^{T_\alpha}(0, T_\alpha) \end{aligned}$$

and $\overline{y}(x)$ is the only solution of

$$\sum_{j\in\Gamma} c_j e^{-B(a,T_\alpha,t_j)x - B(b,T_\alpha,t_j)\overline{y}(x)} = 0.$$

Proof. By the general pricing Formula (2.4.3) we know that

$$\begin{aligned} SO(t,T_\alpha,\Gamma,K) =& P_d(t,T_\alpha)\ \mathbb{E}^{T_\alpha}\Big[(SWAP(T_\alpha,\Gamma,K))^+|\mathscr{F}_t\Big] \\ =& P_d(t,T_\alpha)\int_{\mathbb{R}^2}\left[\sum_j c_j e^{-B(a,T_\alpha,t_j)x-B(b,T_\alpha,t_j)y}\right]^+ f(x,y)dxdy. \end{aligned}$$

where f is the density of the random vector $(x(T_\alpha), y(T_\alpha))$, i.e.,

$$f(x,y) := \frac{exp\Big\{-\frac{1}{2(1-\rho_{xy}^2)}\Big[(\frac{x-\mu_x}{\sigma_x})^2 - 2\rho_{xy}\frac{(x-\mu_x)(y-\mu_y)}{\sigma_x\sigma_y} + (\frac{y-\mu_y}{\sigma_y})^2\Big]\Big\}}{2\pi\sigma_x\sigma_y\sqrt{1-\rho_{xy}^2}}.$$

Now we observe that $SWAP(T_\alpha,\Gamma,K)$ is monotonically increasing in y;[2] indeed:

$$SWAP(T_\alpha) = \sum_{i=\overline{\alpha}+1}^{\overline{\beta}}\left[\frac{\Phi_d(S_{i-1},S_i)}{\Phi_F(S_{i-1},S_i)}P_d(T_\alpha,S_{i-1}) - P_d(T_\alpha,S_i)\right] - K\sum_{i=\alpha+1}^{\beta}\tau_i P_d(T_\alpha,T_i)$$

if we compute the partial derivative with respect to y:

$$\begin{aligned} \frac{\partial}{\partial y}SWAP(T_\alpha) = &\sum_{i=\overline{\alpha}+1}^{\overline{\beta}}\left[-\frac{\Phi_d(S_{i-1},S_i)}{\Phi_F(S_{i-1},S_i)}B(b,T_\alpha,S_{i-1})P_d(T_\alpha,S_{i-1}) + B(b,T_\alpha,S_i)P_d(T_\alpha,S_i)\right] \\ &+ K\sum_{i=\alpha+1}^{\beta}\tau_i B(b,T_\alpha,T_i)P_d(T_\alpha,T_i). \end{aligned}$$

Since the second term is already positive, we focus on the first sum, recalling that $S_{\overline{\alpha}} = T_\alpha$:

$$\begin{aligned} &\sum_{i=\overline{\alpha}+1}^{\overline{\beta}}\left[-\frac{\Phi_d(S_{i-1},S_i)}{\Phi_F(S_{i-1},S_i)}B(b,t,S_{i-1})P_d(t,S_{i-1}) + B(b,t,S_i)P_d(t,S_i)\right] \\ &= -\frac{\Phi_d(T_\alpha,S_{\overline{\alpha}+1})}{\Phi_F(T_\alpha,S_{\overline{\alpha}+1})}B(b,T_\alpha,T_\alpha)P_d(T_\alpha,T_\alpha) + B(b,T_\alpha,S_{\overline{\beta}})P_d(T_\alpha,S_{\overline{\beta}}) \\ &\qquad + \sum_{i=\overline{\alpha}+1}^{\overline{\beta}-1}B(T_\alpha,S_i)P_d(T_\alpha,S_i)\left[1-\frac{\Phi_d(S_i,S_{i+1})}{\Phi_F(S_i,S_{i+1})}\right]. \end{aligned}$$

If we consider that $B(b,t,t) = 0$ and that $\Phi_F(a,b) > \Phi_d(a,b)$, we have proven that $\partial_y SWAP(T_\alpha) > 0$, so the NPV is monotonically increasing in y.

[2] from now on, we will use as notation just $SWAP(T_\alpha)$, but we have to keep in mind that also $\Gamma, K, x_{T_\alpha}, y_{T_\alpha}$ are to be considered.

This fact allows us to freeze x in the integrand and to consider

$$\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty}\Big[\sum_j \lambda_j e^{-B(b,T_\alpha,t_j)y}\Big]^+ f(x,y)dydx$$
$$=\int_{-\infty}^{+\infty}\int_{\overline{y}(x)}^{+\infty}\Big[\sum_j \lambda_j e^{-B(b,T_\alpha,t_j)y}\Big]\gamma e^{E+F(y-\mu_y)-G(y-\mu_y)^2}dydx,$$

where

$$\gamma := \frac{1}{2\pi\sigma_x\sigma_y\sqrt{1-\rho_{xy}^2}}$$
$$E := -\frac{1}{2(1-\rho_{xy}^2)}(\frac{x-\mu_x}{\sigma_x})^2$$
$$F := \frac{\rho_{xy}}{1-\rho_{xy}^2}\frac{x-\mu_x}{\sigma_x\sigma_y}$$
$$G := \frac{1}{2(1-\rho_{xy}^2)\sigma_y^2}.$$

Using the general formula

$$\int_a^b e^{-Ax^2+Bx}dx = \frac{\sqrt{\pi}}{\sqrt{A}}e^{\frac{B^2}{4A}}\Big[\Phi(b\sqrt{2A}-\frac{B}{\sqrt{2A}})-\Phi(a\sqrt{2A}-\frac{B}{\sqrt{2A}})\Big],$$

with $A>0, a, b, B$ real constants, we obtain
(omitting the integral in x and indicating $B(t,T_\alpha,t_j)$ as B_j)

$$\int_{\overline{y}(x)}^{+\infty}\Big[\sum_j \lambda_j e^{-B(b,T_\alpha,t_j)y}\Big]\gamma e^{E+F(y-\mu_y)-G(y-\mu_y)^2}dy$$
$$=\gamma e^E\sum_j\Big[\lambda_j e^{-B_j\mu_y}\int_{\overline{y}}^{+\infty} exp\big\{(F-B_j))(y-\mu_y)-G(y-\mu_y)^2\big\}dy\Big]$$
$$=\gamma e^E\frac{\sqrt{\pi}}{\sqrt{G}}\sum_j\Big\{\lambda_j e^{-B_j\mu_y+\frac{(F-B_j)^2}{4G}}\Big[\Phi[+\infty]-\Phi[(\overline{y}-\mu_y)\sqrt{2G}-\frac{F-B_j}{\sqrt{2G}}]\Big]\Big\}$$
$$=\gamma\frac{\sqrt{\pi}}{\sqrt{G}}\sum_j\Big[\lambda_j exp\Big\{E+\frac{F^2}{4G}-B_j(\mu_y+\frac{F}{2G}-\frac{B_j}{4G})\Big\}\Phi\big[\frac{F-B_j}{\sqrt{2G}}-(\overline{y}-\mu_y)\sqrt{2G}\big].\Big]$$

Finally, noting that:

$$\begin{aligned}
\gamma\frac{\sqrt{\pi}}{\sqrt{G}} &= \frac{1}{\sigma_x\sqrt{2\pi}} \\
E + \frac{F^2}{4G} &= -\frac{1}{2}\left(\frac{x-\mu_x}{\sigma_x}\right)^2 \\
\frac{F}{\sqrt{2G}} &= \frac{\rho_{xy}}{\sqrt{1-\rho_{xy}^2}}\frac{x-\mu_x}{\sigma_x} \\
\sqrt{2G} &= \frac{1}{\sigma_y\sqrt{1-\rho_{xy}^2}},
\end{aligned}$$

it is possible to rearrange the terms to obtain Formula 2.31. □

The final pricing formula is quite complicated: in fact it consists of a numerical integration against a Gaussian distribution $x \sim \mathbb{N}(\mu_x, \sigma_x^2)$. Moreover, the function to be integrated $\Psi(x) = \left[\sum_j \lambda_j(x) e^{\kappa_j(x)} \Phi[-h_j(x)]\right]$ is a sum of terms depending on a parameter $\overline{y}(x)$, which must be obtained through a root-finding algorithm. In the following subsections, there is a short introduction of the methods used to integrate numerically and to find the roots, but first of all there is the need to underline that this will be the main reason for which the overall computational time is quite big. During the main calibration of the neural network, this will change step-by-step the weights in order to find the optimal parameters to price all the swaptions considered. This means that the pricing procedure will be performed a lot of times (238 swaptions per day). For every swaption there will be a numerical integration, and every node x considered will require an iterative method to find the root $\overline{y}(x)$. It is easy to understand that the main task is to find suitable methods to compute everything with a sufficient level of approximation and reasonable computation time.

2.6.1 Gauss-Hermite quadrature

In order to compute the integral in an efficient way, the numerical method chosen is Gauss-Hermite integration. First of all, this quadrature method is used primarily to approximate integrals of the following kind:

$$\int_{-\infty}^{\infty} e^{-x^2} f(x)dx \approx \sum_{k=1}^{n} \omega_k f(x_k), \tag{2.32}$$

where ω_k and x_k are called Hermite weights and nodes. However, this method can be really helpful for our case: indeed, given y $\sim \mathbb{N}(\mu, \sigma^2)$, the expected value of $\psi(y)$ is equal to

$$\mathbb{E}[\psi(y)] = \int_{-\infty}^{\infty} \frac{1}{\sigma\sqrt{2\pi}} exp\left\{-\frac{(y-\mu)^2}{2\sigma^2}\right\}\psi(y)dy. \tag{2.33}$$

Using a simple change of variables

$$x = \frac{y - \mu}{\sqrt{2\sigma}}$$

we can find the integral in the desired form

$$\begin{aligned}\mathbb{E}[\psi(y)] &= \int_{-\infty}^{\infty} \frac{1}{\sqrt{\pi}} exp(-x^2)\psi(\sqrt{2}\sigma x + \mu)dx \\ &\approx \frac{1}{\sqrt{\pi}} \sum_{k=1}^{n} \omega_k \psi(\sqrt{2}\sigma x_k + \mu)\end{aligned}$$

Hence, considering also that swaptions are priced at the Reference Date ($t = 0$), the final formula adopted is the following:

$$SO(0, T_\alpha, \Gamma, K) = \frac{P_d(0, T_\alpha)}{\sqrt{\pi}} \sum_{k=1}^{n} \omega_k \left[\sum_{j \in \Gamma} \lambda_j(x_k) e^{\kappa_j(x_k)} \Phi[-h_j(x_k)] \right] \tag{2.34}$$

In the code, the number of nodes chosen is 20, more than sufficient to compute swaption prices with an acceptable level of approximation.

2.6.2 Root finding

Another, important bottleneck of the pricing process is the search of the root $y = \overline{y}$ of the implicit function

$$F(a, b, \sigma, \eta, \rho, x_k, y) = \sum_j \lambda_j e^{-B(b, T_\alpha, t_j)y}.$$

This means that we are looking for $\overline{y}(x)$ such that

$$F(x, \overline{y}(x)) = 0.$$

The algorithm considered in this case is the irrational Halley's method, which guarantees cubic convergence for C^2 functions [Shloof and Salmi Noorani, 2012].

Algorithm 1 Irrational Halley's algorithm

Input:

- Definition of $F(x,\cdot)$, $F'(x,\cdot)$, $F''(x,\cdot)$,
- initial guess y_0,
- Max number of iterations M,
- Stopping Criterium.

Output: Approximation of the root $\overline{y}$ such that $F(x,\overline{y}) = 0$

repeat

  Evaluate $f = F(x,y_k)$, $f_1 = F'(x,y_k)$, $f_2 = F''(x,y_k)$ **if** $f_2 = 0$ **then**

    $y_{k+1} = y_k - f/f_1$

  else

    $\Delta = f_1^2 - 2ff_2$

  end

  if $\Delta \leq 0$ **then**

    $y_{k+1} = y_k - f_1/f_2$

  else

    $y_{k+1} = y_k - 2f/(f_1 + sign(f_1)\sqrt{\Delta})$

  end

until *stopping criterium is satisfied or* $k == M$;

As it is possible to see in Figure 2.2, in many cases an approximation of the root is found with a small number of iterations, usually below 10.

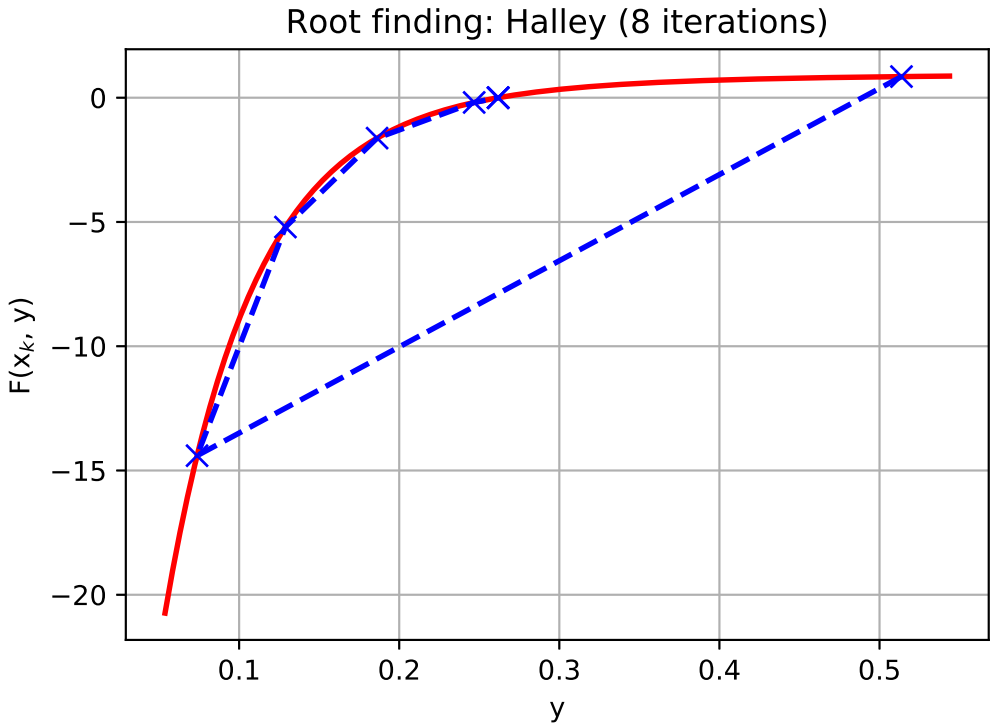


Figure 2.2: Converging iterations of Halley's method

However, in some particular cases (especially if one of the MRSs is lower than its relative VOL) it might happen that the function F shows a narrow curve, almost similar to a step. This slows down the algorithm, which can try hundreds of iterations before finding the root. As a consequence, in this cases, the simple bisection method becomes a more efficient way to find the root, as shown in Figure 2.3.

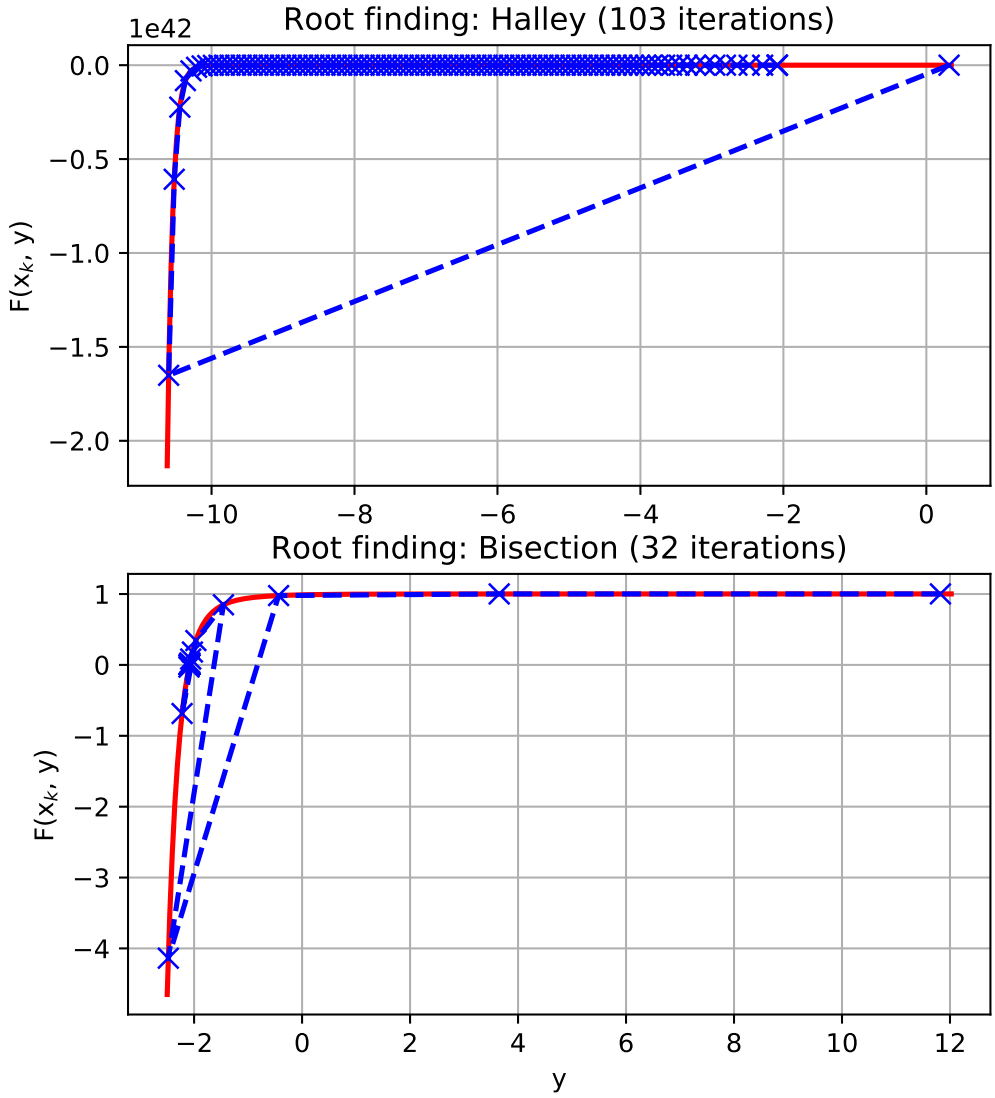


Figure 2.3: Comparison of the methods: in this case Halley is less efficient than Bisection.

Chapter 3

Calibration and Neural Network

The presented project is the result of the combination of two different subjects: Finance and Machine Learning. The financial framework of the project has been defined in the previous Chapter; here the Machine Learning framework is presented, starting with the introduction of the main task of this work, which is calibration.
Calibration is the process that looks for optimal model parameters given market data (e.g., swaption prices and volatilities), by minimizing the error obtained through a specific feedback function. ML methods are perfect for this purpose. In a first instance, a description of the feedback function used will be given, as well as its importance in the supervised learning procedure. Afterwards, the neural network structure and related algorithms are presented, along with an analysis of the calibrator limits.

3.1 Calibration and supervised learning

The main task in the implementation of a calibrator is to achieve a suitable trade-off between accuracy and computational speed. The calibration process must be done before any pricing procedure in order to fit the mathematical model to the real market. In our case, for a specific day d, the inputs of the calibrator are the market quotes of a set of swaptions, i.e. prices, volatilities and vegas of the swaptions, with the addition of the information about the discounting and forwarding curves. This set of market quotes will be referred as Q_d, whereas the output is the set of parameters $\Theta = \{a, b, \sigma, \eta, \rho\}$ of the G2++ model. This calibration procedure produces a representation of the dynamics of interest rates in the future, that could be used to price several instrument, not only swaptions.

As a metric of the goodness of the model, the measure considered is the error given by the difference between swaption prices (given as input) and the ones predicted from

the model through Θ. In particular, in a single-currency framework, for each day the calibrator must price 238 swaptions, which is every combination of 17 expiries and 14 tenors. Their importance is not the same, so the feedback function must consider some sort of weight assigned to each of them.

For each market price y_i, let $\widehat{y}_i(\Theta, Q_d)$ be the price given by the G2++ model[1]: the error is computed as

$$\epsilon_i(\Theta, Q_d) = \big(\widehat{y}_i(\Theta, Q_d) - y_i\big)^2. \tag{3.1}$$

Then, the feedback function on the whole set of swaptions available for a specific day d is given by

$$E(\Theta, Q_d) = \sqrt{\sum_i \frac{1}{\nu_i^2}\epsilon_i(\Theta, Q_d)}, \tag{3.2}$$

where the ν_i are the Vegas of the swaptions, provided as inputs to the problem. This "vega weighting" has a normalization purpose: the most difficult swaptions to price are the ones with largest time span; in the same time, they are the least liquid and they have largest vegas. With the Vega normalization, less importance is given to the error related to these derivatives, giving more focus on the most liquid ones[2].

Let us call $\widetilde{\Theta}$ the result of a complete calibration, i.e., the one capable of finding the set of parameters that minimizes the feedback function.
Unfortunately, a 5-dimensional optimization process becomes really slow in terms of computational time. This leads to the necessity to define a more efficient calibrator, which may possibly remember similar solutions found in the past. Indeed, in finance, the ability of pricing financial instruments in a short time has become of vital importance, and this is the main reason that led to the exploration of ML methods.
In particular, this is the typical scenario where SL methods come to help. Supervised Learning is a branch of Machine Learning in which the task is to estimate the unknown model that maps known inputs to known outputs. Actually, our problem is not one of the standard SL ones; indeed, the output of the calibrator is to find the not-known parameters of the G2++ model. The only known data are the prices, which is why the feedback function is so important for our case.

The typical procedure is the following:

[1]The notation here is slightly different from the one used in Chapter 2, but is still consistent:

$$\widehat{y}_i(\Theta, Q_d) = SO(0, (T_\alpha)_i, \Gamma_i, K_i)$$

where the variables on which the $i-th$ price depends are summarized in Θ and Q_d.

[2] Vega is defined as the derivative of the price of the contract with respect to its volatility.

1. At the beginning, a large part of the past dataset makes part of the *training phase*, where the calibrator tunes its parameters. For this purpose, there is the need to define a measure of the error implied by the parameters used, which is the *feedback function*.

2. Successively, the fitted model is run to predict the responses for the observations in a second part of the dataset called *validation phase*; This validation procedure provides an unbiased evaluation of a model fit on the training dataset while tuning the model's hyper-parameters, which for example determine the complexity of the model used.
   In the project this phase will not be adopted, because it would require the calibrator to perform multiple optimization procedures with high computational times.

3. Once the model is fixed, it is run with unseen input in the *test phase*, used to provide an unbiased evaluation of a final model fit on the training dataset.

It is clear that now, if the model has shown an adequate accuracy, it can predict new targets as soon as new data is available, without the need to restart the global calibration. The output should be approximately near the optimal one, so that just a local calibration may eventually be needed.

In brief, instead of tuning on a single day the G2++ model parameters, the calibrator will try to use past data to find a realistic pattern to fit all of them simultaneously. Once this pattern is found, it will be used to predict the parameters in future times. Of course, this overall calibration will be a lot more time consuming than the single-day-calibration, but it would not be performed every day but just once in a while. Moreover, the "fitting" procedure would be done in advance with respect to the prediction, which is almost immediate.

In the particular framework considered in this study, it is necessary to build a unique global feedback function that must take into account all the D days considered:

- the parameters of the G2++ model are a set $\Theta = \{\Theta_d\}_{d=1}^{D}$,
  where $\Theta_d = \{a_d, b_d, \sigma_d, \eta_d, \rho_d\}$ for $d = 1, 2, \ldots, D$;

- the global feedback function F is the algebraic mean of the single-day- feedback E, depending on the set of daily market quotes $Q = \{Q_d\}_{d=1}^{D}$:

$$F(\Theta, Q) = \frac{1}{D} \sum_{d=1}^{D} E(\Theta_d, Q_d). \tag{3.3}$$

3.1.1 Offline and online calibration

One of the major dichotomies in ML algorithms is the distinction between *online* and *offline* methods. The classification is usually defined on the availability of data, but actually it is based on how the algorithm considers them.

- *Batch* or offline learning techniques process the entire training set in one go. This can be heavy in terms of computational times if applied to large datasets.
- *Sequential* or online methods make use of available data one at a time, or in small batches, and the model parameters are updated after each new observation is presented. They are usually used for large datasets, or in real-time scenarios where there is a stream of incoming data.

In this project, the first type of algorithm used is a batch one: the data is split into training and test sets, maintaining the chronological order by reference date. Most of the time is spent during the training phase, but once performed, the testing phase is almost immediate. This kind of approach presents one major issue: historical data may present trends based on the evolution of the underlying and, if there are changes in market behavior, the model may not capture their effect in the test data.

For this reason, a second type of algorithm will be considered, using an online approach only in the test phase. In a first instance, a global calibration will be performed over the training set exactly as in the previous case, then every time just one new sample of the test set will be considered: after a quick evaluation of the model, it is added to the training set to perform a local calibration with the increased dataset. If the predicted parameters are close enough to the optimal ones, this latter procedure should be quite fast.

For sure this last method is slower than the previous one, but offers some advantages since the model would be able to capture market variations. Moreover, online evaluation is closer to the real application of the method, where new data arrive on a daily basis and the model, after the prediction, can be re-trained to fit the new data available.

Clearly, since the training set is sequentially increased, the local calibration will get slower and slower, and may become unpractical for real applications. This problem can be addressed in two possible ways. The first one is to simply forget the oldest data used, so to fix the training size.
The second one still fixes the train size, but the data considered are randomly sampled with replacement. This procedure, known as *bootstrap*, allows to keep track of all historical scenarios, in case they reappear in the future.

3.2 Artificial Neural Network

One of the most popular models used in SL is the Artificial Neural Network (ANN). The simplest definition of a neural network is provided by the inventor of one of the first neurocomputers, Dr. Robert Hecht-Nielsen. He defines a neural network as:

> "... a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs.".

They constitute a class of flexible nonlinear models designed to mimic biological neural systems, elaborating signals through several layers, each with a large number of neural units (neurons) that can process the information in a parallel manner. So an ANN has a multi-layer structure such that every layer is built upon many simple nonlinear functions, playing the role of neurons in a biological system. By allowing the complexity of the structure to increase indefinitely, multi-layered ANNs are able to approximate any continuous function with any desired degree of accuracy. Thanks to their representation power, they are said to be *universal approximators*, and became very popular in the fields relating to Machine Learning ([Bishop, 2009], [Hotelling, 1933], [Witten et al., 2011], [Haykin, 2009]).
The number of applications of neural networks grew larger and larger in the last decade along with the evolution of GPUs and distributed computed systems, capable of supporting the computational power required to perform tasks in a short time.

3.2.1 Model definition

A feed-forward neural network is a generalization in multiple layers of one of the simplest models user for regression, the *perceptron*. For this reason, it is called also *Multilayer Perceptron (MLP)*.

Building blocks: Neurons

Every layer is made of several *neurons*, the building blocks of the overall structure. Each neuron receives as input a linear combination of the data elaborated in the previous layer, and then transforms it to generate a neural signal to be forwarded to the neurons in the next layer.

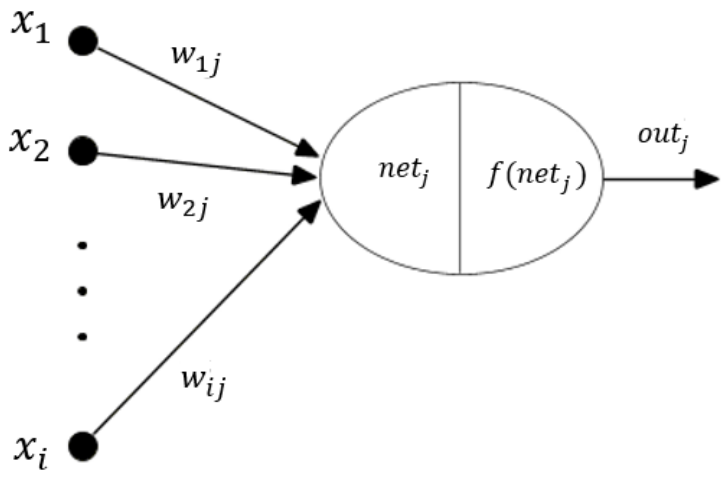


Figure 3.1: Composition scheme of a neuron.

For example, consider the j^{th} neuron in one of the layers; suppose that it is connected to N neurons of the previous layer with values called $\{x_i\}_{i=1}^{N}$. Then, the j^{th} neuron will have as input (also known as *net value*):

$$net_j = \sum_{i=1}^{N} w_{ij}x_i + w_{0j}. \tag{3.4}$$

In this formula, the coefficient w_{ij} corresponds to the weight of the connection between the input i and the neuron j. The last weight w_{0j} is called *bias*, and it behaves like a connection with a fictitious input always equal to 1. The value that this neuron returns as output, also called *activation value*, is simply

$$out_j = \Psi(net_j), \tag{3.5}$$

where $\Psi(\cdot)$ is called *activation function*.

Final architecture: the network

A neural network is built by hooking together many simple neurons in several layers, so that the output of a neuron can be the input of another. For example, in Figure 3.2 a simple representation of a neural network is given:

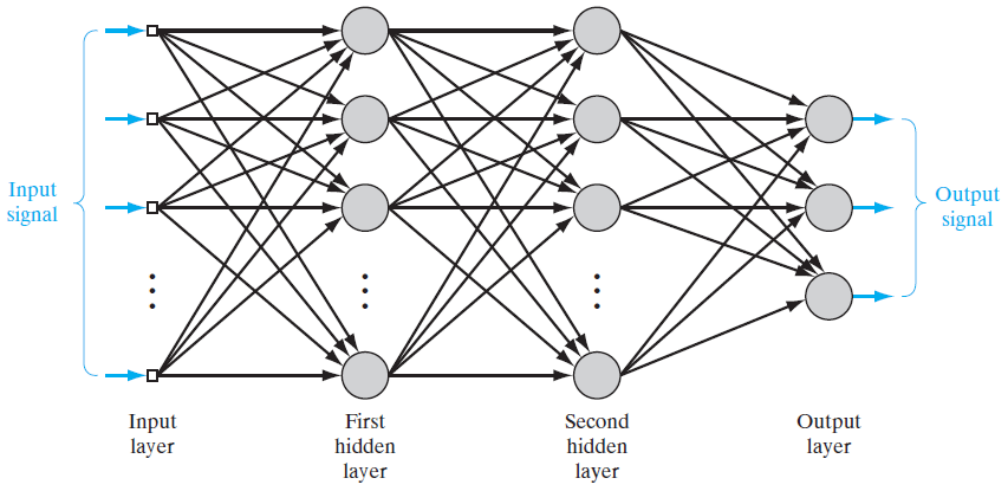


Figure 3.2: Scheme of a fully connected neural network

In this figure, it is possible to see the different roles of the layers. The leftmost one is called the *input layer*, and it is the one that directly considers the input data (eventually elaborated through a preprocessing procedure). The rightmost is the *output layer*; which is the final step where the results can be observed. The middle layers of the network are called *hidden layers*, because their values are not directly accessible. In general, their number may vary, but in most cases only one or two hidden layers are used. It is also possible to have no hidden layers, and in this case the NN degenerates into a simple perceptron).

Neural networks usually exhibit a high degree of connectivity, whose extension is determined by the presence of weights in the network. In this example, there is a connection between any couple of neurons in consecutive layers, hence this neural network is said to be *fully connected*. Moreover, the sample information here is passed only forward from the input layer to the output one. This is defined as a *Feed-Forward Neural Network (FFNN)*. Another commonly used neural architecture is the Elman *Recurrent Neural Network (RNN)*. This class of networks allows neurons to depend not only on input variables, but also on their own lagged values through connections that form a directed cycle.
In any case, this project is focused specifically on FFNN.

The architecture considered in the project is made of four layers (two of them are hidden). With the exception of the biases, every couple of consecutive layers is fully connected.

For each day considered, the goal is to price 238 swaptions; the input features that can be used to fit the interest short rate curve are their prices and their volatilities[3]. This means that for each day there is a set of 576 features available, but not all of them are significant, since most of them are higly correlated. This suggests considering a preliminary process of dimensionality reduction, called *Principal Component Analysis (PCA)*, which will be presented in the next chapter. The result of this process is a matrix of a selected number of features for each day considered, which is the real input dataset of the neural network.

The output layer is the one from which the feedback function is computed. In particular, from this layer we obtain the vectors of the G2++ model parameters, so there is a fixed dimension, equal to 5 neurons. [4] In the calibration phase, they will be used to compute the feedback with respect to the real prices.

The final output of the neural network are the parameters of the financial model used to price swaptions, and will be called *targets*. This definition is used to not misunderstand the network parameters: the weights. In fact, the calibrator will sequentially change the weights of the ANN in order to transform the fixed input data into the targets that best fit market prices.

Choosing Activation Functions

The choice of the activation function is important when building a neural network, because a different choice results in a different model, leading to different results. As cited in [LeCun et al., 1998]:

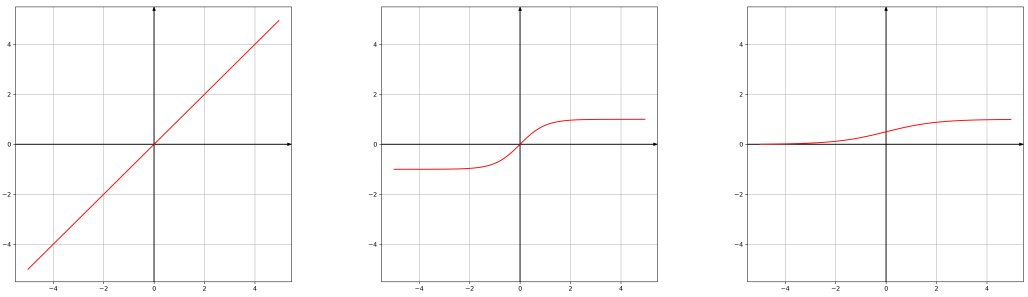
> "These choices can be critical, yet there is no foolproof recipe for deciding them because they are largely problem and data dependent."

As it will be explained in the next chapter, the presence of a simple derivative is often a key parameter, since that significantly simplifies the calibration phase. Some of the most common activation functions, and the ones that will be used in this project are briefly presented.

The choice of the function in the output layer depends mostly on the task that the network has to perform and on the target space. Usually, for classification and regression,

[3] see chapter 5 to see the list of available features in the dataset

[4] Remark: the network is used to predict the G2++ parameters for all the dataset made of D days; hence, each output neuron, as well as all the other neurons in the network, will not have as output a single real number but a vector of D real numbers. For example, the first neuron in the final layer will have as output $a = \{a^d\}_{d=1}^{D}$, and all the a^d are computed using the same weights of the network applied on different input data.



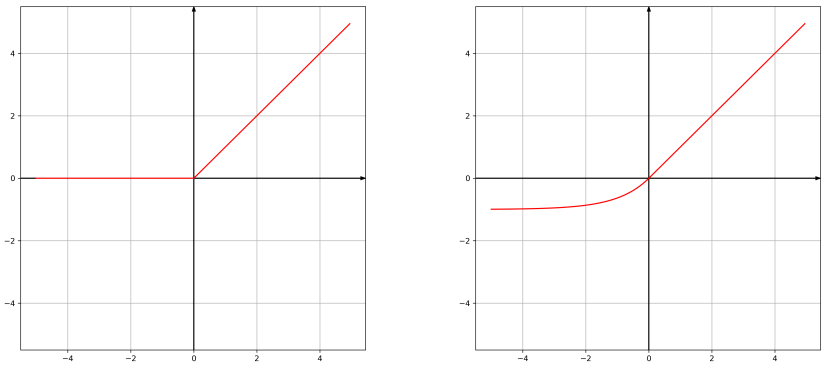
(a) Linear *(b) TanH* *(c) Sigmoid*

Figure 3.3: Few examples of activation functions that can be applied to the neurons in the output layer.

the most used output functions are a logistic sigmoid or hyperbolic tangent (TANH) for the former and a linear function for the latter. In this case, the activation function used is an affine transformation of the tanh:

- for the neurons related to the two MRSs, the result of the tanh is shifted to cover values in $[-1, 3]$: even if in theory they should be always positive, from a practical point of view it is convenient to let them assume also negative values. Too low values, as well as too high ones, might lead to numerical issues;
- the activation functions of the VOLs neurons must be positive. Also in this case values over 2 may lead to numerical problems, so the volatility space is set to $[0, 2]$;
- the correlation must be in a range $[-1, 1]$, hence the related neuron adopts a simple tanh function without transformations.

For what concerns the hidden layers, the choice is wider, and each function has different properties that can be better for different classes of tasks. For example, if the neuron is just required to have on/off response, a *heaviside* function is common practice. On the other hand, the neurons may assume any value between zero and one or just positive values, and this leads to consider again a sigmoid function or the positive part function (also called Rectified Linear Unit (ReLu)). In other cases, it may be convenient to adopt smooth versions of the previous ones, like Elu (Exponential Linear Unit).
In this project, the first function considered for the hidden layers of our model was ReLu, but this led to a problem known as "dying ReLu": when updating the weights incoming in a ReLu neuron it may happen that it outputs zero. But from that point on, the neuron will never activate again (an example of this particular scenario can be seen in



(a) ReLU *(b) Elu*

Figure 3.4: Few examples of activation functions that can be applied to the neurons in the hidden layers.

Section 3.5).
For this reason, the activation function chosen was the sigmoid.

3.3 Calibration algorithm

The main structure of the calibrator has been introduced, but its functioning still needs to be explained.
Thanks to the ANN, the set of estimates Θ^{ω} is provided as a function g of the set of market quotes Q and of the set of weights $\boldsymbol{\omega}$, as in (3.6).

$$\Theta^{\omega} = g(Q, \boldsymbol{\omega}). \tag{3.6}$$

Remark. The set of weights $\boldsymbol{\omega}$ does not depend on the time interval of the input given; indeed the restricted version of equation 3.6 holds:

$$\Theta_d^{\omega} = g(Q_d, \boldsymbol{\omega}). \tag{3.7}$$

The goal of the calibrator is to find the best subset $\overline{\boldsymbol{\omega}}$ with fixed cardinality W (defined with the number of neurons chosen) with respect to the feedback function resulting when Θ^d is applied. In other terms, the optimization problem to be solved is to find $\overline{\Theta}$:

$$\overline{\Theta} = g(Q, \overline{\boldsymbol{\omega}}), \tag{3.8}$$

where[5]

$$\overline{\boldsymbol{\omega}} = \underset{\boldsymbol{\omega} \in \mathbb{R}^W}{\arg\min}\ F(g(Q, \boldsymbol{\omega}), Q). \tag{3.9}$$

This W dimensional minimization problem is solved by using the combination of two different algorithms:

- The first algorithm is a slight modification of the *CrossEntropy* methodology. It performs a stochastic exploration of the parameter space to find a neighborhood of the global optimum.
- Once a neighborhood of the solution is found, it is possible to perform a local minimization procedure; namely, the *BFGS* algorithm, a gradient-based iterative process belonging to the class of quasi-Newton methods.

3.3.1 Cross-Entropy Optimization

The CrossEntropy (CE) method is an optimization algorithm, based on Monte Carlo simulations. It is usually used for finding the global minimum of noisy functions. The algorithm considered in this project is a slightly customized version of the original method, available in [Rubinstein, 2004] and [Alon et al., 2005].

The custom part of the algorithm consists in considering not only the data sampled in the current iteration to update the normal distribution parameters, but also in using the best B points found during the full run of the method.
The idea is that for each iteration the area where the algorithm chooses to generate new samples becomes narrower, focusing on the best results obtained so far. The stopping condition considered, other than the maximum number of iterations, is a threshold on the variance of the best samples: if the best scenarios discovered are sufficiently "close" to each other, it means that the algorithm has found a candidate space where look for the global optimum. An example of how the algorithm focuses iteratively its search on the best samples is shown in Figure 3.5.

[5] later, the objective function $F(g(Q, \boldsymbol{\omega}), Q)$ is shortly denoted as $F(\boldsymbol{\omega})$

Algorithm 2 Cross-Entropy optimization algorithm (CE)

Input:

- Definition of the objective function F($\cdot$),
- Normal mean $\boldsymbol{\mu}$ and variance $\boldsymbol{\Sigma}$ for the first iteration,
- Number of samples N to be generated,
- Number of best samples B to be kept for the next iteration,
- Stopping condition: maximum number of iterations M and variance threshold ϵ.

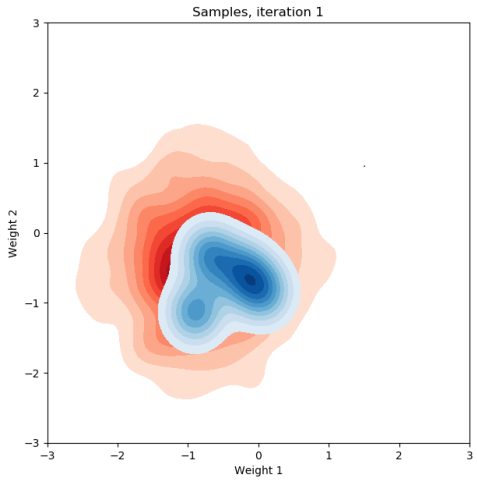
Output: Estimation of $\overline{\boldsymbol{\omega}}$ minimizing the objective function

1: *Simulation* step: sample N data vectors $\boldsymbol{\omega}^{(1)}, \ldots, \boldsymbol{\omega}^{(N)} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$,

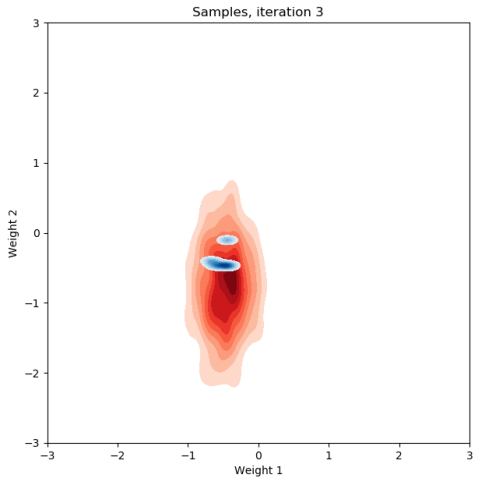
2: *Sorting* step: Compute the feedbacks $F(\boldsymbol{\omega}^{(i)})$ for all i and order them from the smallest to the biggest. Select only the first B best samples (of the overall simulation, considering also previous steps),

3: *Update* step: compute the new distribution parameters $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ as sample mean and variance of the best point kept in order to produce better samples in the next iteration,

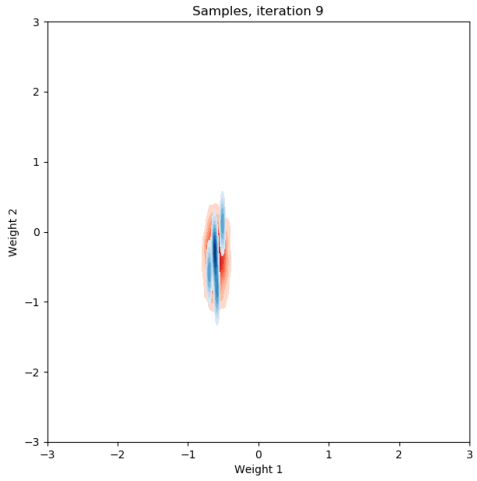
4: Iterate 1, 2 and 3 until stopping condition is met, i.e., the maximum number of iterations M is met or $||\boldsymbol{\Sigma}||_\infty \leq \epsilon$.



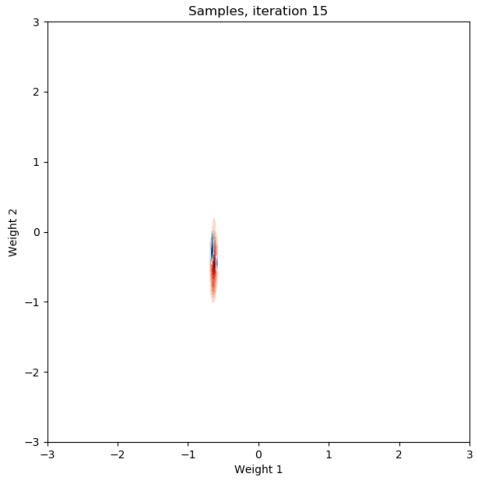
(a) Iteration 1



(b) Iteration 3



(c) Iteration 9



(d) Iteration 10

Figure 3.5: Four iterations of the cross-entropy method. The red area shows the distribution of two weights of the $N = 1000$ data sampled from the normal distribution, while the blue area shows their distribution in the best 8 data points ever found. It is possible to see that the search region becomes smaller with the iterations.

3.3.2 BFGS algorithm

Thanks to the CE algorithm, the calibrator is able to find a suitable starting point where to look for the global minimum; Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm performs a local minimization to find a stationary point of the objective function [J.M. Rondinelli and Marks, 2007]; hence, the necessary condition for the optimality, as well as one of the stopping criteria, is that the gradient is equal to zero.

The algorithm starts with at an initial estimate for the optimal value $\boldsymbol{\omega}_0$ and proceeds iteratively to get a better estimate at each stage. Since it is a gradient-based algorithm, for each iteration it is necessary to compute the gradient of the feedback function with respect to the current set of optimal weights, so that it is possible to determine the best search direction accordingly; the gradient will be denoted as $\nabla_W F$ or $\nabla F(\boldsymbol{\omega})$. Several methods for computing the gradient have been considered, each with its own strengths and weaknesses. They are presented along with their results in the next chapter.
This quasi-Newton method also considers the inverse of the Hessian Matrix of the feedback, called H; indeed, at the iteration k, the search direction $\boldsymbol{p}_k$ is given by the solution of the analogue of the Newton equation:

$$\boldsymbol{p}_k = -H_k^{-1} \nabla F(\boldsymbol{\omega}_k). \tag{3.10}$$

From a practical point of view, the direct computation of H_k^{-1} is not a good choice; the best practice consists in its iterative approximation: H_0 is initialized with the identity matrix (in this case, the first step is equivalent to a gradient-descent) and in further steps H_k is updated through Sherman-Morrison formula (as shown in Algorithm 3) in such a way that the secant equation and curvature condition are satisfied:

$$\boldsymbol{\omega}_{k+1} - \boldsymbol{\omega}_k = H_{k+1}\big[\nabla F(\boldsymbol{\omega}_{k+1}) - \nabla F(\boldsymbol{\omega}_k)\big] \qquad \text{(secant equation)}$$

$$\big[\nabla F(\boldsymbol{\omega}_{k+1}) - \nabla F(\boldsymbol{\omega}_k)\big]^T \big[\boldsymbol{\omega}_{k+1} - \boldsymbol{\omega}_k\big] > 0. \qquad \text{(curvature condition)}$$

Once the search direction is found, a *line search algorithm* is used; its purpose consists in finding the best step α_k minimizing the feedback error along this direction; i.e.:

$$\alpha_k = \underset{\alpha>0}{\arg\min}\ F(\boldsymbol{\omega}_k + \alpha \boldsymbol{p}_k) \tag{3.11}$$

Finally, the algorithm ends when the gradient of the feedback function is sufficiently small or when consecutive iterations do not change the solution guessed, as summarized in Algorithm 3.

Algorithm 3 Broyden-Fletcher-Goldfarb-Shanno algorithm (BFGS)

Input:

- Definition of objective function $F(\cdot)$, and its gradient $\nabla F(\cdot)$
- Initial guess $\boldsymbol{\omega}_0$
- Gradient threshold ϵ and step threshold R.

Output: Estimation of $\overline{\boldsymbol{\omega}}$ minimizing the objective function.

$$k = 0$$

$$H_0 = \mathbb{I}$$

$$s_0 = \infty$$

while $\|\nabla F(\boldsymbol{\omega}_k)\| \geq \epsilon$ *and* $\|\boldsymbol{s}_k\| \geq R$ **do**

Search direction and step size

$$\boldsymbol{p}_k = -H_k \nabla f(\boldsymbol{\omega}_k)$$

$$\alpha_k = \underset{\alpha>0}{\arg\min}\ F(\boldsymbol{\omega}_k + \alpha \boldsymbol{p}_k)$$

Update optimal guess

$$\boldsymbol{s}_k = \alpha_k \boldsymbol{p}_k$$

$$\boldsymbol{\omega}_{k+1} = \boldsymbol{\omega}_k + \boldsymbol{s}_k$$

$$\boldsymbol{y}_k = \nabla F(\boldsymbol{\omega}_{k+1}) - \nabla F(\boldsymbol{\omega}_k)$$

Update Hessian approximation

$$H_{k+1} = \left(\mathbb{I} - \frac{\boldsymbol{s}_k \boldsymbol{y}_k^T}{\boldsymbol{y}_k^T \boldsymbol{s}_k}\right) H_k \left(\mathbb{I} - \frac{\boldsymbol{y}_k \boldsymbol{s}_k^T}{\boldsymbol{y}_k^T \boldsymbol{s}_k}\right) + \frac{\boldsymbol{s}_k \boldsymbol{s}_k^T}{\boldsymbol{y}_k^T \boldsymbol{s}_k}$$

$$k = k + 1$$

end

$$\overline{\boldsymbol{\omega}} = \boldsymbol{\omega}_k$$

3.4 Potentiality and limits: supervised learning

The complexity of the network is equal to the total number of weights, which grows exponentially if the number of neurons in the hidden layers increases. Obviously, as the complexity increases, the calibration error will get lower. This is due to the fact that the parameter space Θ, which can be represented via Equation 3.6, by some set of weights $\boldsymbol{\omega}$ increases with W. There is a minimum limit of the calibration error, and it is the one due to the choice of the financial model. The questions arising from these considerations are:

1. How good is the pricing model considered, especially if compared to the simpler Vasicek? What is the minimal error that can be achieved using this model?

2. Is it possible to fit the optimal parameters for all the time interval considered? What is the complexity that the neural network must have in order to reach this limit?

Let us call

$$\Theta_d^* := \arg\min_{\Theta_d} E(\Theta_d, Q_d)$$

this is the set of optimal G2++ parameters for the day d.
The maximum potentiality of the pricing model can be achieved if it is able to find the optimum set for all the days taken into account. Recalling that

$$F(\Theta, Q) = \frac{1}{D}\sum_{d=1}^{D} E(\Theta_d, Q_d),$$

it is possible to define $F(\Theta^*, Q)$ as the " G2++ error", where $\Theta^* = \{\Theta_d^*\}_{d=1}^D$. This is the task of the optimal calibrator that, unfortunately, is not available.
However, it is possible to get a close approximation of this result by training a neural network focused only on one date at a time. Since this single-day calibration is done only once, in order to avoid complexity issues the number of neurons per hidden layer has been fixed to 30 (more than enough for this purpose).
The feedback obtained is then compared to the one resulting from the optimal calibrator for the Vasicek's model. The result is shown in Figure 3.6:

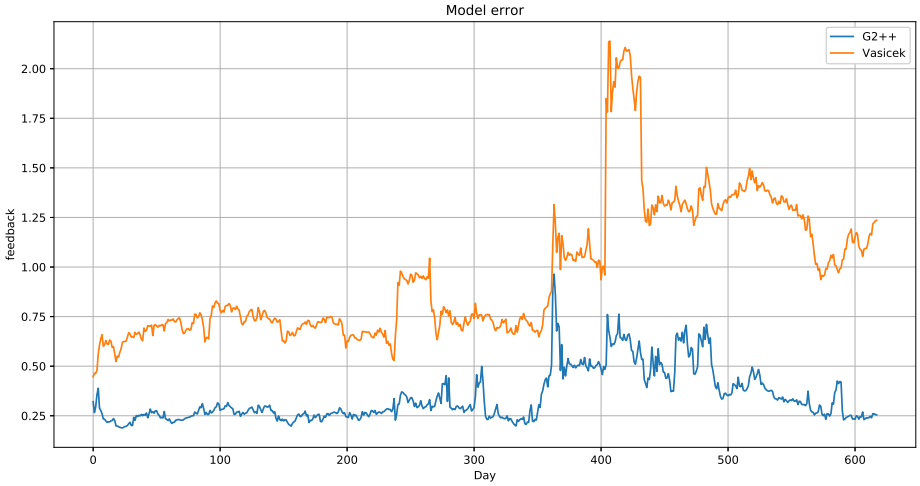


Figure 3.6: Feedback comparison between Vasicek and G2++ models. The improvement of the new model is about 65%.

Now the focus goes on the complexity of the network: of course, as the size of the set weights W increases, the calibrator gets closer to the optimal one: by calling $\hat{\Theta}^W$ the set of targets predicted by the calibrator (fixed W), the following holds:

$$\lim_{W\to\infty} \hat{\Theta}^W = \Theta^*.$$

Since the computational time of the calibration is directly proportional to the number of weights used in the network, there is a trade-off between the network expressivity and the computational burden. Namely, the purpose is to choose a small number of neurons (and consequently of weights), but big enough to approximate the optimal calibrator in an adequate manner.
Moreover, it is preferable not to have high complexity, because it may lead to overfitting, which is the capability to reach an excellent fit over the training data, but a terrible performance when new samples are introduced.
For this reason, the next step for the ANN is trying to fit the curves of the parameters resulting from the previous single-day calibration, in order to see if it has the capability to replicate those suboptimal results for different complexities. This is the typical framework of SL.

The procedure is the following:

- in the first phase the feedback function involving model pricing is not considered; instead the error will be computed simply as the Euclidean distance between the

target and the prediction. The optimization algorithm used to perform the fit is known as Adam.

- Once the fit is done, the original feedback function will be applied to the predicted parameters. The curve of the feedback functions of the single-day calibration and the one resulting from this fitting procedure are compared, also in terms of relative error.

- The same procedure is made by increasing the complexity of the network: for simplicity, the number of neurons of the two hidden layers was kept equal.

The results are clearly depicted in Figures 3.7 and 3.8:

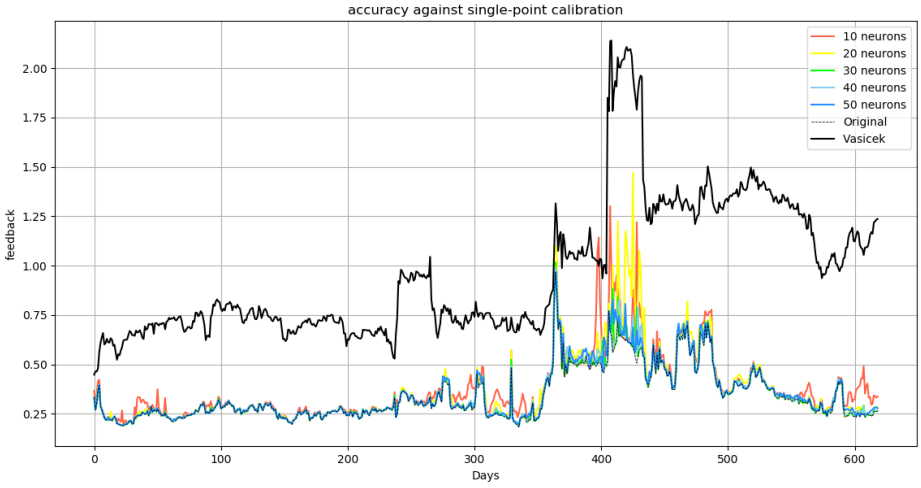


Figure 3.7: Comparison of the feedback error between the single-day-calibration and the fit of the ANN with increasing complexity. Still there is the comparison with the optimal feedback of Vasicek's model.

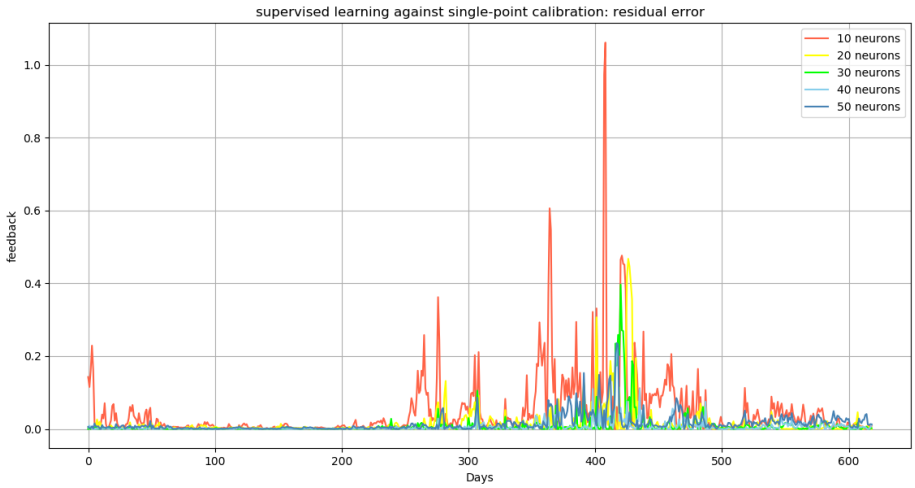


Figure 3.8: Comparison of the relative error between the single-day-calibration and the fit of the ANN with increasing complexity.

As predicted by the theory, it is clear that as complexity increases, the network becomes more capable of fitting the targets and replicating the feedback curve. There is also a set of dates in which the model seems to be very sensitive: even with complex networks, capable of replicating the parameters with a low error, the feedback can be negatively affected in a remarkable way.
In general, from approximately 30 neurons the curve is sufficiently replicated, with few spikes. In order to reduce the risk of overfitting, the complexity will be set to 40 neurons per hidden layer.

3.5 Dying ReLu

Now it is clear that the calibration of the ANN follows the computation of the gradient in the BFGS algorithm. In this section it is briefly shown what is the main issue in the choice of the ReLu hidden activation function, namely the "Dying ReLu".
A simple calibration has been performed on a reduced dataset, with only 4 components selected by the PCA and a network based on 6 neurons per hidden layer. In the first case, the hidden activation function considered is the "ReLu", in the latter the Sigmoid function is taken into account.
Once the procedure is done, a graph representing the network is drawn, as depicted in figure 3.9 and 3.10. on the bottom the input layer is represented; going upwards, it is followed by the hidden layers and finally by the output layer. The positive weights connecting the neurons are shown in blue, while the red lines correspond to negative weights. The width of the lines is corresponding to the absolute value of the weights represented.
The bias neurons are filled in red, while the neurons in the hidden layer are blue if their activation value is different from zero.
The same graph is then used to show the values of the gradient of the feedback function with respect to the weights.

The result of the choice of the ReLu activation function is clearly depicted in figure 3.9: the majority of the hidden neurons have negative net values, resulting in null activation values. As a consequence, also the derivative of the feedback with respect to the weights related to this neurons is zero. This has three main effects:

- BFGS algorithm will not update the weights leading to all the "dead" neurons; hence the majority of the network is not considered.
- One of the stopping criteria of BFGS consists in the norm of the gradient to be under a certain threshold: since many of the derivatives are zero, it becomes easier for the algorithm to stop after just a few iterations, even if the current set of weights is far from the optimal one.
- Great importance is given to the weights connecting the output neurons to the last bias: this means that the network is almost neglecting the input features.

The behavior changes completely if a different activation function is chosen: for example, as shown in figure 3.10, the sigmoid activation function leads the calibrator to consider and update always all the weights.

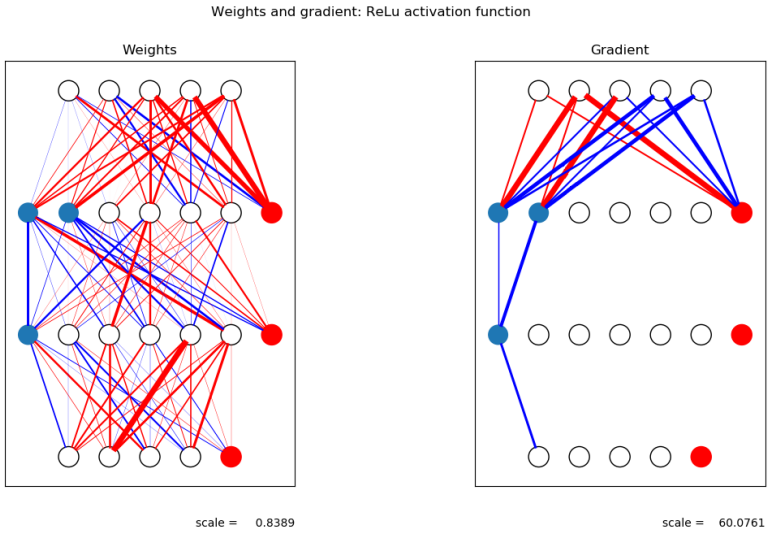


Figure 3.9: Set of weights and relative gradient with ReLu hidden activation function.

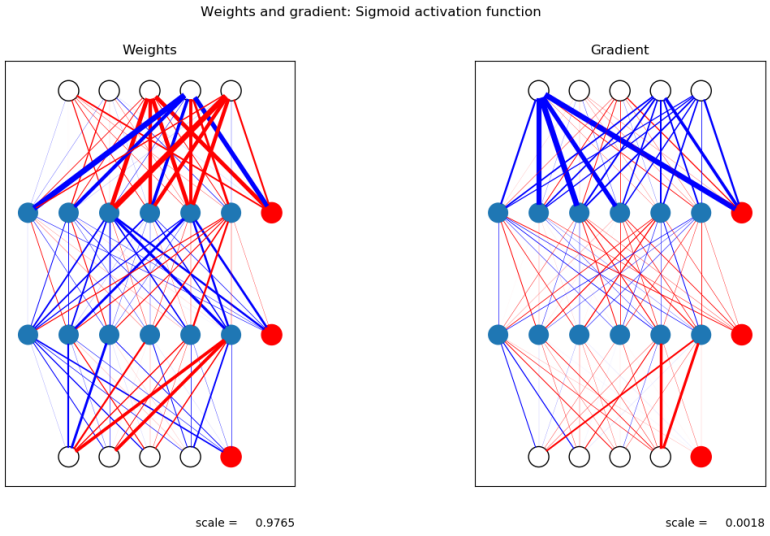


Figure 3.10: Set of weights and relative gradient with Sigmoid hidden activation function.

Chapter 4

Data Analysis

In the previous chapter, the general idea of the calibrator and of the related algorithm was introduced; this chapter provides an overview of the procedure followed to implement the solution of the calibration problem, as well as some essential technologies. The first section contains a closer look at the dataset provided. The composition of several of the available features is analyzed, as well as their usage. The massive dimension of the input data must be reduced in order to be effectively used as input of the ANN: this manipulation is performed through the PCA algorithm, described in the second section.

4.1 Dataset exploration

4.1.1 An overview

The main analysis of how a solution to the calibration problem can be built must start from the exploration of the available dataset. The preliminary focus lays on the dataset relating to the Euro (EUR) currency; in a second time, other currencies will be introduced. The time span covered by the EUR dataset goes from 2013-06-28 to 2017-02-21, for a total of 901 reference dates (904 considering also the days with anomalous or missing data). The relevant fields of each daily sample are described in Table 4.1.

Table 4.1: Descriptions of the features present in the dataset.

Feature	**Description**
Reference date T_{ref}	The date on which the sample is taken
Swaption expiries	The list of expiries T_{exp} of the swaptions traded. For the *EUR* dataset there are 17 different daily expiries, ranging from 1 month since the reference date to 30 years.
Swaption tenors	The list of tenor intervals τ of the swaptions traded. For the *EUR* dataset there are 14 different tenors for every sample, ranging from 1 to 30 years.
Swaption prices	Matrix containing the swaption prices for each combination of expiry T_{exp} and tenor τ. In *EUR* dataset there are 17x14 prices per day.
Swaption volatilities	Matrix containing the swaption volatilities for each combination of expiry T_{exp} and tenor τ. The volatility is a sensitivity index of the uncertainty about the returns provided by the underlying asset [Hull]. In *EUR* dataset there are 17x14 prices per day.
Swaption vegas	Matrix containing the swaption vegas for each combination of expiry T_{exp} and tenor τ. Vega is defined as "the rate of change of the value of the portfolio with respect to the volatility of the underlying asset" [Hull]. In *EUR* dataset there are 17x14 prices per day.
Forward dates	The set of dates for which the values of the forward curve are bootstrapped. For every daily sample there are about 60 dates, covering a time span of 60 years after the reference date.
Forward values	The set of values of the bootstrapped forward curve (corresponding to the forward dates).
Discount dates	The set of dates for which the values of the discount curve are bootstrapped. For every daily sample there are about 60 dates, covering a time span of 60 years after the reference date
Discount values	The set of values of the bootstrapped discount curve (corresponding to the discount dates).
Calibration error	The feedback error made by the bank daily calibrator adopting Vasicek model.

4.1.2 Feature exploration

Multi-curve bootstrap

The credit and liquidity crisis, started in the second half of 2007, triggered, among many other consequences, a general reflection about some of the standard methods and assumptions used to price and hedge interest rate derivatives. In particular, it has been shown that using a single risk-free curve to forecast and discount cash flows is no longer valid. Standard market practice has evolved into a multicurve approach, taking properly into account different curves to forecast and discount cash flows ([Bianchetti, 2008], [Sender, 2017]).

In this multicurve framework, the dataset provides for each currency the values of two distinct curves, one used to discount cash flows (the *OIS* curve), and one for forwarding, with a specified tenor (in the EUR dataset the *EURIBOR 6M* curve). The curves are bootstrapped on a daily basis, i.e., they are built every day from the market prices of liquid, simple instruments. Figure 4.1 shows the behavior of the daily curves with respect to the distance in time (in days) from the reference date.

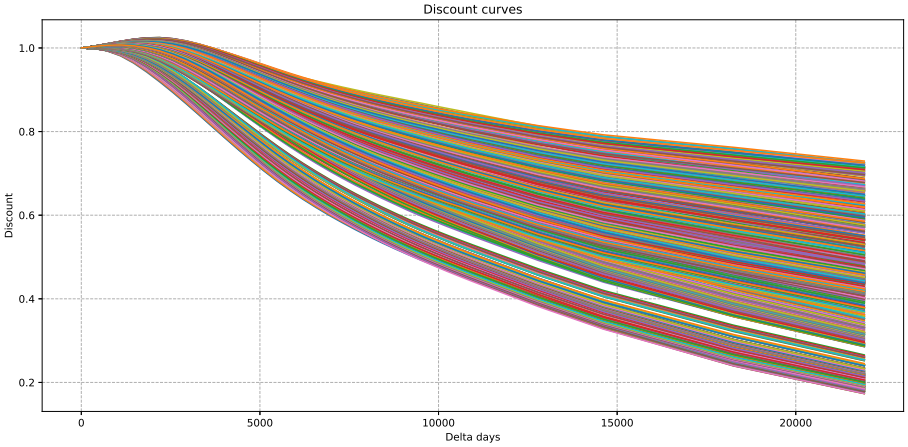
The discount curve is the one closest to the risk-free curve thanks to its overnight tenor; on the other hand, the forward curve takes into account the credit risk related to the underlying tenor: for this reason, it is possible to see in Figure 4.2 that the discounting curve dominates the forwarding curve.

These features are not directly used as input of the calibrator, but their manipulation is fundamental for the computation of the feedback: the values of the discount and forward curves relating to the swaptions' cash flow dates are obtained through interpolation on the set of dates given; although, this interpolation is not performed directly on the values on the curves, but on their zero rate: given the reference date t_0, and the value $B(t_0, t_i)$ relative to the date t_i, the zero rate $Z(t_0, t_i)$ is defined as:

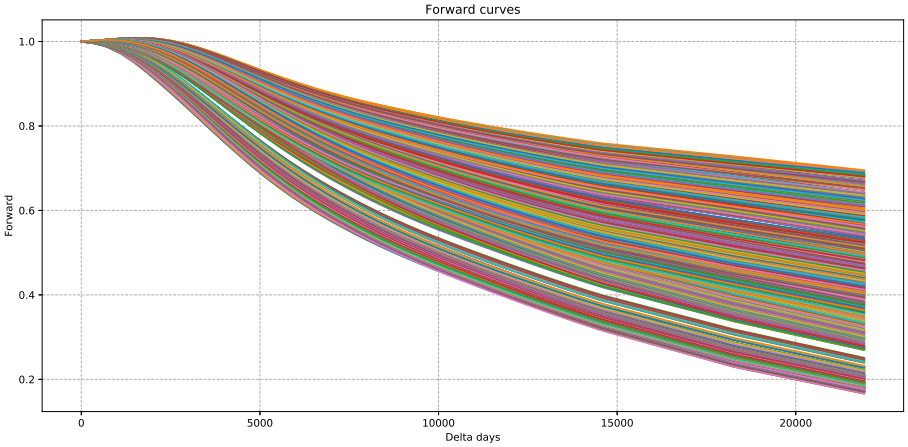
$$Z(t_0, t_i) := -\frac{ln(B(t_0, t_i))}{\delta(t_0, t_i)}, \tag{4.1}$$

where $\delta(t_0, t_i)$ is the year fraction between t_0 and t_i.[1]

[1] Using *ACT/365* daycount convention

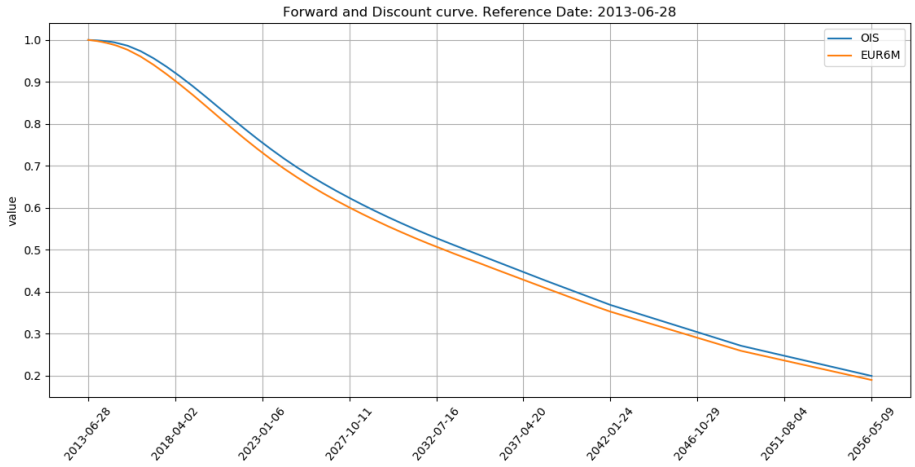


(a) Discount (OIS) curves

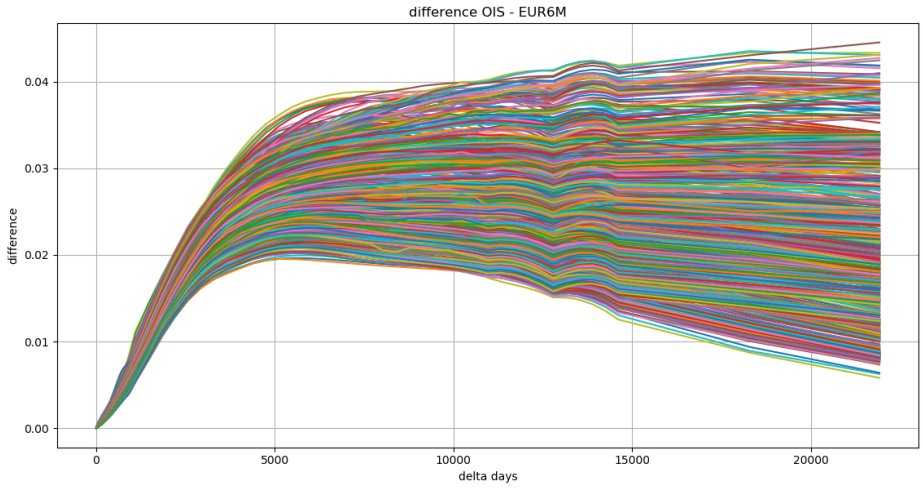


(b) Forward (EUR6M) curves

Figure 4.1: Discount OIS curves and forward EUR6M curves plot for the EUR dataset. The time interval is expressed in difference of days from the relative reference date.



(a) Comparison of the discount and forward rate curve on a specified reference date. It is clearly depicted that the discount curve dominates the forward one. The time interval is expressed in difference of days from the relative reference date.



(b) Plot of the difference of the discount and forward curve for the overall EUR dataset. The time interval is expressed in difference of days from the relative reference date.

Figure 4.2: Comparison OIS/EUR6M curves

Swaption Matrices

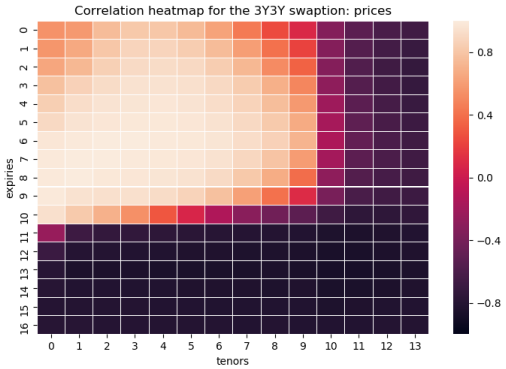
The real input of the calibrator is given by the matrices related to the set of swaptions. In a daily sample in the *EUR* dataset, there are quotations of prices, volatility and vega for every couple expiry-tenor. This means that there are three different 17x14 daily matrices, for a total amount of 714 features per day.
It is easy to see that the dimension of the input features is very high if compared to the number of samples (900). This is the first reason that led to the introduction of a dimensionality reduction.

The other main reason is that sometimes the features are highly correlated, hence some data may not add useful information for the model while it increases the complexity of the calibrator; indeed, it is possible to verify that the price with a specified tenor and expiry, corresponding to a specific value in the matrix, is highly correlated to the prices of the closest cells. This is due to the fact that close swaptions have similar tenors and expiries, and the related cash flows exchanged between the counterparties are located in similar time intervals. While moving far from the considered swaption the correlation decreases, assuming also strong negative correlations. This behavior is clearly depicted in Figure 4.3a, where the correlation matrix is the one related to the price of the swaption with 3 years as expiry, and with a tenor of 3 years.

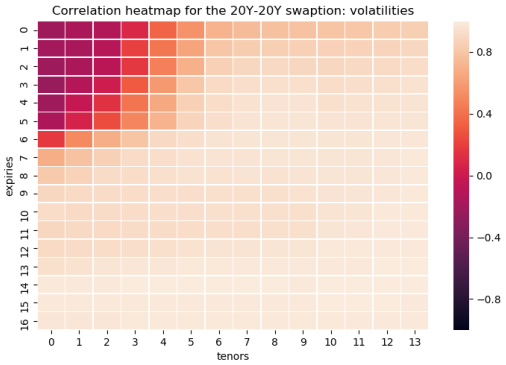
As far as the volatilities are concerned, it is possible to detect two groups of swaptions: the first collects the swaptions with lowest expiries and tenors (top-left corner in Figure 4.3b); the other swaptions are in the latter group. The swaptions in the same group are highly correlated with each other; on the other hand, the two groups have a poor, negative correlation.

Finally, the correlation matrix of the vegas presented in Figure 4.3c shows that all swaptions have generally high correlations. As a consequence, the exclusion of these features from the set of input parameters for the neural network does not affect in a remarkable way the calibration procedure.

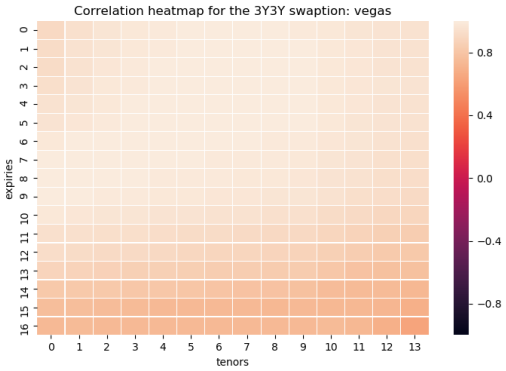
The correlation matrices of prices, volatilities, and vegas for every possible combination expiry/tenor (flattened as a single vector) are presented in Figure 4.4.



(a) price correlations for the 3Y-3Y swaption (cell 7x2)

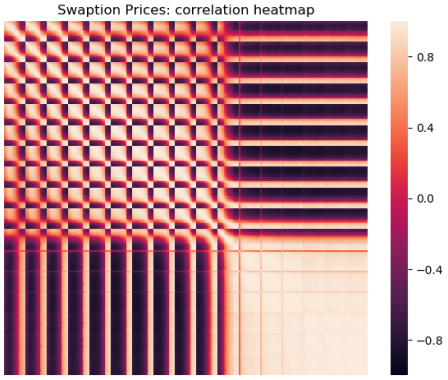


(b) volatility correlations for the 20Y-20Y swaption (cell 14x11)

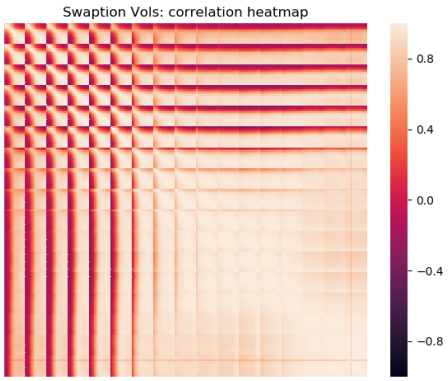


(c) vega correlations for the 3Y-3Y swaption (cell 7x2)

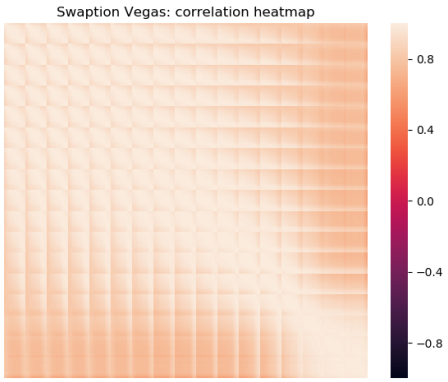
Figure 4.3: Some examples of correlation matrices.



(a) prices



(b) volatilities



(c) vegas

Figure 4.4: Correlation matrices of the flattened swaptions features.

4.1.3 Multicurrency Dataset

The list of features presented for the *EUR* dataset is the same for every other currency; although there are some differences to be remarked.

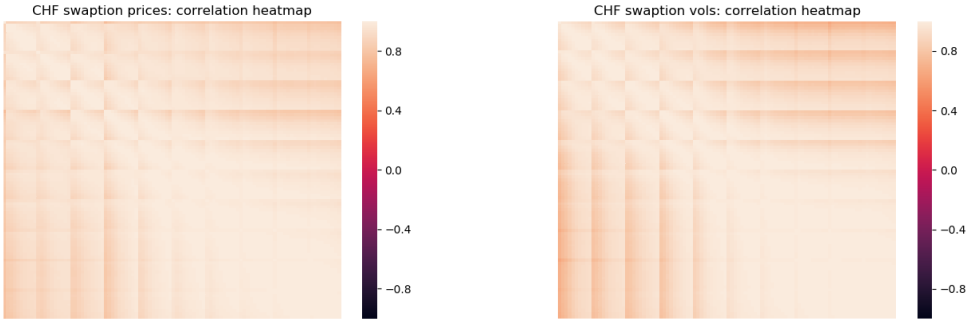
The first difference consists in the time span covered by the reference dates for each currency. Table 4.2 shows also that in the same interval there may also be a different number of samples for different currencies.
For this reason, the starting date for the calibration among a multicurrency framework was set to the first reference date in which all the currencies were available. From that date on, all the next business dates within the calibration range were considered, without paying attention to select only the samples relating to common dates. This is due to the fact that, on some specific days, the samples for a currency might be missing because of a local holiday, but the calibration and pricing procedure for all other currencies must still be active.

Currency	first date	last date	samples
EUR	2013-06-28	2017-02-21	900
USD	2013-06-28	2017-05-24	929
CHF	2014-07-28	2017-05-24	665
CAD	2014-07-28	2017-05-24	729

Table 4.2: Samples interval ranges divided by currency

The second difference is perhaps the most important one: the set of swaption priced and traded on a daily base can vary. For example, in the *CHF* framework the swaptions available have only 10 different expiries and tenors (100 total swaptions against 238 for the *EUR* dataset). The solution adopted consists in considering only the common combinations swaption, which are the most liquid, discarding the other ones; in this way the most liquid contracts are given the same importance for every set.

The analysis regarding the correlation for the features for the *CHF* and *USD* currencies is similar to the one made for the *EUR* dataset; although, the correlation matrices for prices and volatilities in the *CHF* dataset show a different behavior: all features are highly correlated, without the distinction between short-term and long-term swaptions.



(a) prices *(b) volatilities*

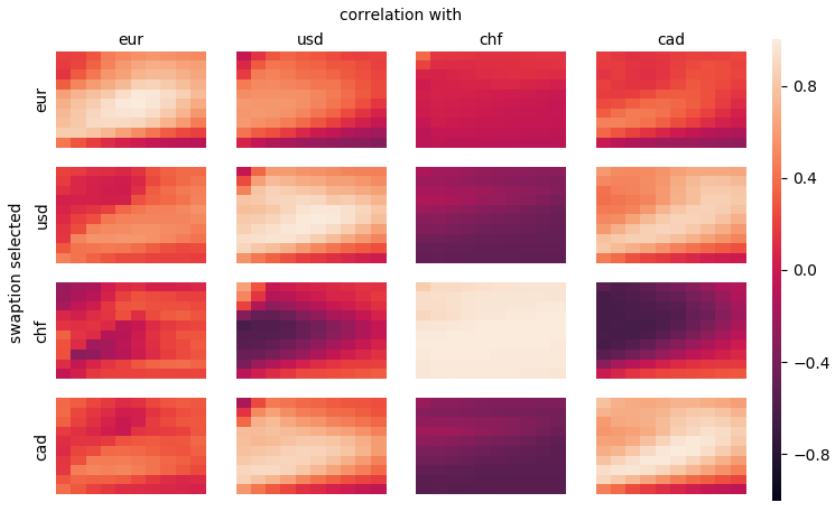
Figure 4.5: Correlation matrices of the flattened swaptions features for CHF dataset

Some brief observations can be done also with respect to the cross-currency correlations. Since the sizes of the various datasets are different, only the dates having all the possible features available have been selected. It is unfeasible to show the correlation matrix of the overall dataset, hence some particular features have been selected as example. In particular, the swaption with couple expiry/tenor of 3 and 6 years has been chosen for every currency.
In Figure 4.6 it is possible to see the correlation of the prices and of the volatilities of this swaption with the correspondent feature in every dataset.

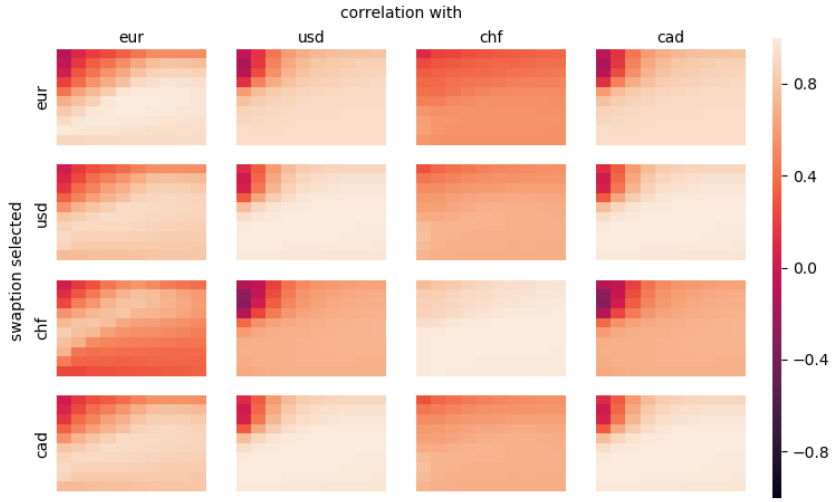
Focusing on the diagonal matrices on the figure related to prices (Figure 4.6a), it is possible to verify what was already underlined: within the features of the same currency, there is a high correlation rate with the swaptions with similar tenors and expiries which decreases as we move towards the most different ones (the 3x6 swaption is placed in the middle of the matrices). This does not hold for the *CHF* dataset where, as mentioned, the 3Y x 6Y swaption's price depends almost linearly on all other prices of the same dataset.
The patterns in the cross correlations *USD/CAD* and *CAD/USD* are close to the ones within the same currency (namely, the *CAD/CAD* and *USD/USD*). This can be an indication of strong correlations between the two markets. A similar, but weakened behavior can be seen when comparing these currencies with the *EUR* prices. It is noticeable that the correlation of the 3x6 EUR swaption price with all the ones in the *CHF* market is almost null, where it becomes strongly negative when taking the *CAD* and *USD* swaptions into account.

Easier is the analysis for the volatilities, where Figure 4.6b shows not only that the division into two groups (short-term and long-term swaptions) still holds, but also that it is

extended to a multicurrency framework. Again, the *CHF* market behaves in a different manner.



(a) prices



(b) volatilities

Figure 4.6: Correlation matrices of the 3Yx6Y swaptions of four different currencies. Every row corresponds to the correlations of a specific swaption with respect to the others. Every column specifies the currency of the dataset with which the swaptions are being compared.

4.2 Dimensionality reduction: PCA

4.2.1 Single currency: EURO

Section 4.1.2 underlined an important issue: the dimension of the input features (238 swaptions per sample, for a total amount of 714 or 576 features if Vegas are considered or excluded) is too high if compared to the number of samples (about 900); moreover, many of the features are highly correlated. This is the typical case in which "the data points all lie close to a manifold of much lower dimensionality than that of the original data space" [Bishop]; leading to the need to perform a dimensionality reduction technique, namely the PCA.
This unsupervised learning technique looks for the orthonormal basis which build the projection of the data onto a lower dimensional linear space in such a way that the loss of information is minimized, i.e., maximizing the variance of the projected data [Hotelling, Bishop].

The components of this basis are known as Principal Component (PC). The first principal component is the dimension that maximizes the variance of the projected data. In an iterative approach, given the set of the first $k-1$ Principal Components, the k-th PC can be seen as the dimension, orthogonal to the subspace spanned by the former components, which accounts for the maximum projected variance. The total number of components is the minimum that captures a proportion of the variance over a fixed threshold.

Before performing the PCA, the dataset is subject to a normalization preprocess: this is due to the fact that the features considered have different ranges, and they must be scaled in order not to bias the PCA. When all the features have the same range the Analysis can be performed (as explained in Algorithm 4).

This process has been applied in two different cases. In the first scenario the considered dataset consists only of prices and volatilities; in the second one, the matrices of the vegas were added.
In both cases the PCA with a 99% threshold selected 10 components; their ratio of total variance captured is shown in Tables 4.3 and 4.4. It is possible to see that, if the vegas are added to the input dataset, only the first component has a gain in terms of variance, but generally the inclusion of vegas does not add a significant amount of information.

Algorithm 4 Principal Component Analysis (PCA)

Input:

- Normalized dataset of D-dimensional samples $\{\boldsymbol{x}_i\}_{i=1}^N$;
  In matrix form it is denoted as $\boldsymbol{X} \in \mathbb{R}^{NxD}$,
- Variance threshold ϵ.

Output: M-dimensional dataset $\{\widehat{\boldsymbol{x}}_{\boldsymbol{i}}\}_{i=1}^N$, $M < D$;
In matrix form it is denoted as $\widehat{\boldsymbol{X}} \in \mathbb{R}^{NxM}$

1: Compute the sample mean $\overline{\boldsymbol{x}}$ and Covariance matrix S:

$$\overline{\boldsymbol{x}} = \frac{1}{N}\sum_{n=1}^{N} \boldsymbol{x}_i$$

$$S = \frac{1}{N-1}\sum_{n=1}^{N} (\boldsymbol{x}_n - \overline{\boldsymbol{x}})(\boldsymbol{x}_n - \overline{\boldsymbol{x}})^T$$

2: Calculate eigenvalues and eigenvectors of S, ordering the eigenvalues in decreasing order; i.e. $\{(\boldsymbol{e}_i, \lambda_i)\}_{i=1}^D$, s.t. $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_D$;[2]

3: Define the dimension M as the minimal dimension of the principal subspace maintaining a proportion of explained variance greater or equal than ϵ, i.e. such that:

$$\frac{\sum_{k=1}^{M} \lambda_k}{\sum_{i=1}^{D} \lambda_i} > \epsilon$$

4: Define the projection matrix on the subspace, whose orthogonal basis is defined by the first M eigenvectors:

$$\boldsymbol{E}_M = (\boldsymbol{e}_1, \ldots, \boldsymbol{e}_M)$$

5: Project the original dataset into the principal subspace:

$$\widehat{\boldsymbol{X}} = \boldsymbol{X}\boldsymbol{E}_M$$

Component rank	variance ratio	cumulative variance ratio
1	0.71697	0.71697
2	0.11391	0.83088
3	0.06639	0.89727
4	0.04302	0.94029
5	0.01806	0.95835
6	0.01435	0.97270
7	0.00593	0.97863
8	0.00474	0.98337
9	0.00425	0.98762
10	0.00314	0.99076

Table 4.3: Variance captured by the first ten principal components excluding vegas.

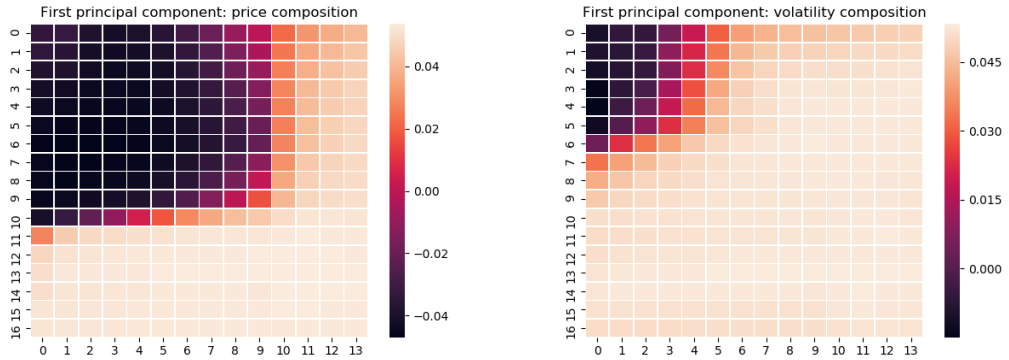
Component rank	variance ratio	cumulative variance ratio
1	0.77779	0.77779
2	0.08636	0.86415
3	0.04752	0.91167
4	0.03458	0.94625
5	0.01798	0.96423
6	0.01415	0.97838
7	0.00440	0.98278
8	0.00390	0.98668
9	0.00312	0.98980
10	0.00252	0.99232

Table 4.4: Variance captured by the first ten principal component including vegas

It is possible also to visualize the contributions of the original dimensions to the first PC to better understand what are the most important features selected in both cases. The visualization of the contributions to the first principal component is provided in Figures 4.7 and 4.8. The most important aspect to notice is that the contributions of prices and volatilities are almost the same.
As far as the prices are concerned, there is a negative contribution of a large part

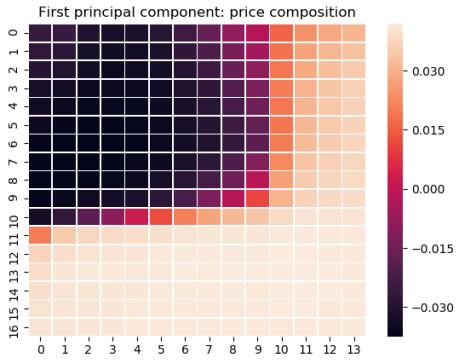
[2] $\boldsymbol{e}_1$ with eigenvalue λ_1 is the first PC; $\boldsymbol{e}_k$ related to λ_k is the k-th one: it captures a proportion of variance equal to $\frac{\lambda_k}{\sum_{i=1}^{D}\lambda_i}$

of the top-left prices (swaptions with the lowest expiries and tenors), and a positive contribution for the other ones. With the exception of a few cells (mostly located on the boundary delimiting the two sub-groups), the absolute value of the contributions has a small range. This effect can be seen as if the PCA were capable of capturing a large part of the information by exploiting the difference of the prices of the long-term swaptions and the shortest ones.
Taking a closer look at the volatility matrix, it is clear that the swaptions at the top-left corner make a small contribution to the first component: the knowledge of these values is not so useful when compared with the other ones. The dichotomy in the importance of the volatilities depicted here is the same underlined in Figure 4.3b.
Observing the vega matrix, almost every swaption has a similar, negative contribution (with the exception of the swaptions with largest expiries).

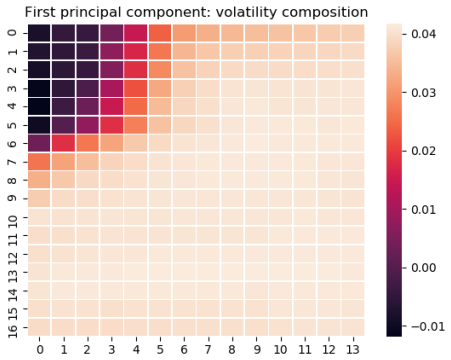


(a) prices *(b) volatilities*

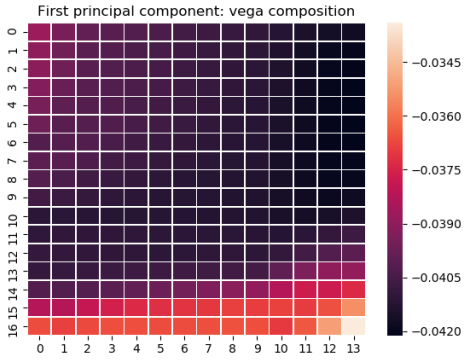
Figure 4.7: Composition matrix of the first component excluding vegas



(a) prices



(b) volatilities



(c) vegas

Figure 4.8: Composition matrix of the first component including vegas.

Chapter 5

Practical Methodologies

One of the most important problems faced in this project is the overall time required to perform the whole calibration procedure.
In order to reduce the computational time, the feedback function is then implemented on a GPU using Compute Unified Device Architecture (CUDA), as described in the first section. Finally, some different methods for computing the gradient in the BFGS algorithm have been considered: the algorithms and their results are presented in the last section.

5.1 Parallel computation: GPU and CUDA

In the previous chapters, the major issue of the overall project was introduced: a large part of the overall computational time is involved in the pricing procedure; indeed, the feedback function is evaluated thousands of times, and every time it needs the computation of 238 swaptions per daily sample.
This is the main reason why there is a need for parallel computing, which is a type of computation in which the various instructions can be split into different sub-tasks that can be executed simultaneously on different devices. In this case, parallel processing consists of performing the same set of tasks (namely, the pricing procedure) applied to different values of the input variables.

In this project a great effort was made to implement the pricing procedure involving parallel computing on Graphics Processing Unit (GPU), a specialized computer processor built to execute massively parallelized tasks. In particular, the CUDA architecture developed by NVIDIA was used to move the tasks from CPU to the GPU, giving an incredible performance boost.

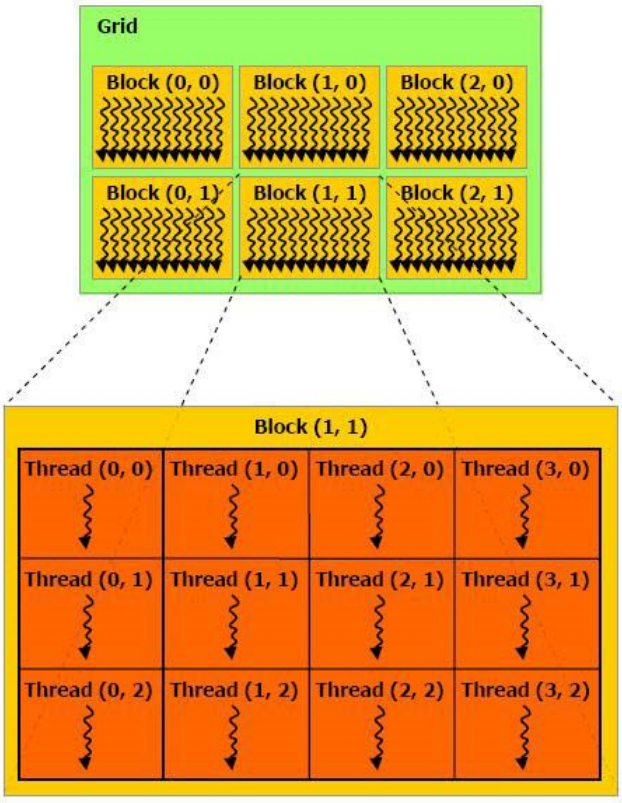


Figure 5.1: Example of code structuring in GPU.

In CUDA, the *kernel* is the set of operations the must to be parallelized. In particular, kernels must be organized in a precise composition:

- a *thread* is the finest set of instructions relative to the execution of each kernel; each thread is indexed in such a way that they are able to access to the dedicated sections of memory, without the risk of reading the same part of data or overwriting the results.
- a *block* consists in a group of threads that are executed together and can access portion of shared memory. The dimension of threads in one block depends on the hardware limits of the GPU.
- Blocks are organized into a one-dimensional, two-dimensional, or three-dimensional *grid* of blocks (as shown in Figure 5.1). The number of blocks in a grid is usually dictated by the size of the data being processed or the number of processors in the system.

This composition is made according to the GPU's physical architecture: The GPU

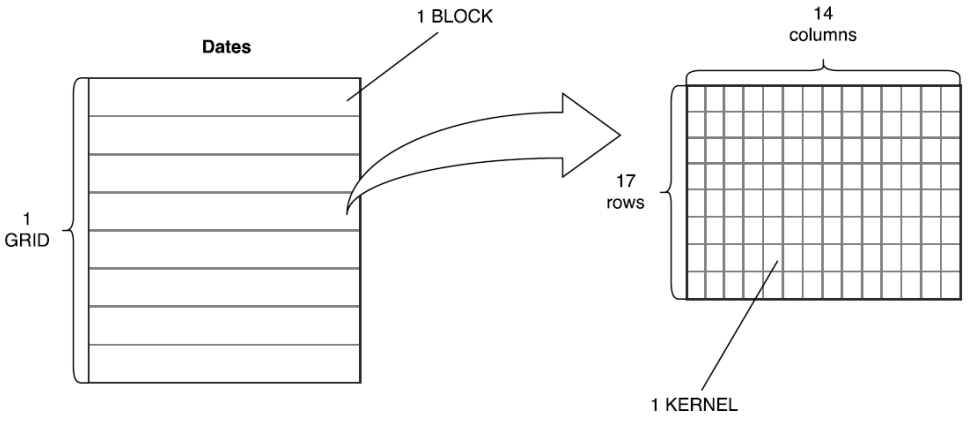


Figure 5.2: Dataset decomposition in kernels, blocks and grid

chip is organized as a collection of several Multiprocessor (MP), where each one is responsible for handling one or more blocks in a grid. Each MP is further divided into a number of Stream Processor (SP)s, with each SP handling one or more threads in a block. All threads in a block reside on the same MP core and must share the limited memory resources of that core; for this reason, the design of the structure of the code, in particular the size of the blocks, is very important because it highly affects the scheduling of the kernel executions and the efficiency of the computation [Sanders and Kandrot, 2010].

Focusing on the project of this thesis, the code and dataset related to pricing must be structured in kernels, blocks, and grids. Prices have to be evaluated for a sequence of reference dates, and for each date there is a matrix of fixed dimensions of swaptions, where the computation of each price depends only on the daily input (G2++ parameters, discount and forward curve).
Hence, the choice is very straightforward: each kernel consists in the pricing of a single swaption, i.e. putting the value in one cell of a matrix. This directly defines the dimension of a block: each reference date is related to one block. The threads maintain the same relation of the prices: the number of tenors and expiries fix the dimensions of the size of the block, and the daily input information can be passed directly to the blocks without being structured in some complicated manner. A representation of the division of the dataset into grid, blocks, and threads is illustrated in Figure 5.2

After the implementation on CUDA, the computation of the feedback achieves an incredible speedup: the pricing procedure on a single date takes 15 seconds on the CPU, while on the GPU less than 0.1 seconds are needed. Hence there is a speedup of 150 times per date. Moreover, the GPU is able to perform a massive parallelization on the several MPs (performing the feedback computation of several days at the same time): considering a training set of 450 daily samples, the feedback computation on the GPU takes 2.3 seconds, while almost 2 hours are needed with the CPU. The speedup in this case is of over 2900 times.

5.2 Gradient computation

The last major topic regarding the implementation of the calibrator is implicit in one of the formulas in the BFGS method, whose algorithm is described in Section 3.3.2. In particular, let's take a closer look at Equation 3.10:

$$\boldsymbol{p}_k = -H_k^{-1} \nabla F(\boldsymbol{\omega}_k).$$

In the overall optimization procedure, the main focus goes to the computation of the gradient of the feedback function with respect to the set of variables $\boldsymbol{\omega}$, defined as $\nabla_W F$. Actually, there is not a unique method to achieve it; different algorithms were considered during the project and will be presented in this section, along with their results.

5.2.1 Finite Differences

The first and simplest method to compute the gradient with respect to the vector of weights $\boldsymbol{\omega}$ is simply performed through finite differences: this means that there is an estimation of the impact on the feedback function for every weight w_i in the network.

In order to use a shorter notation and to underline the network structure of the model, a different notation for the generation of the target set Θ is used:

$$\Theta = \mathcal{N}_{\boldsymbol{\omega}}(X) := g(\boldsymbol{\omega}, Q)$$

where X is the result of the preprocessing procedure applied to the set of input feature Q, namely the standardization and PCA methods.

Algorithm 5 Finite Differences gradient estimation (FDM)

Input:

- Definition of feedback function $F(\cdot, \cdot)$,
- Definition of the network model $\mathcal{N}_{\boldsymbol{\omega}}$, defined by set of weights $\boldsymbol{\omega}$,
- Input set of features X, such that $\Theta = \mathcal{N}_{\boldsymbol{\omega}}(X)$,
- Increment h.

Output: Estimation of the gradient $\nabla_W F$ w.r.t. the given set of weights $\boldsymbol{\omega}$

Evaluate

$$\Theta = \mathcal{N}_{\boldsymbol{\omega}}(X)$$
$$f = F(\Theta, Q)$$

for $i = 1, \ldots, W$ **do**

$$\boldsymbol{\epsilon} = h \ \mathbf{e_i}$$
$$\hat{\boldsymbol{\omega}}_i = \boldsymbol{\omega} + \boldsymbol{\epsilon}$$

Evaluate the new target

$$\widehat{\Theta}_i = \mathcal{N}_{\widehat{\boldsymbol{\omega}}_i}(X)$$
$$\frac{\partial F(\Theta, Q)}{\partial \omega_i} = \frac{F(\widehat{\Theta}, Q_i) - f}{h}$$

end

$$\nabla_{\boldsymbol{\omega}} F = \left\{ \frac{\partial F(\Theta, Q)}{\partial w_i} \right\}_{i=1}^{W}$$

As Section 5.2.4 will show later, this method allows a big gain per iteration in terms of decreasing of the feedback; although, looking at the algorithm, its major issue emerges immediately: the feedback function is evaluated $W + 1$ times. This means that, as the complexity of the network increases, the computational time needed to perform this algorithm gets bigger and bigger. For example, considering an ANN with 20 neurons per hidden layer, the number of weights may be around 750 units (depending on the number of features selected in input). If the evaluation of the feedback takes around 4 seconds to be performed, the gradient for a single iteration is completely evaluated in 50

minutes. Consequently, since there are thousands of feedback computations, it is clear that BFGS optimization is unfeasible with this method, especially with networks with higher complexity (the number of weights increases exponentially with the number of neurons).

5.2.2 SFDM

One of the possibilities for estimating the gradient involving fewer feedback computations can be found in robotics and automation: in these fields, people have traditionally used deterministic model-based methods for obtaining the gradient of a system. However, in real systems the number of variables is huge, and we cannot expect to be able to model every detail of the robot and the environment. As a result, researchers have considered a variety of estimation methods over the last years: one of the oldest approaches involves a regression applied to the result of stochastic simulations of the systems; for this reason, this method is called Stochastic Finite Difference Method (SFDM) ([Peters and Schaal, 2006], [C. Fu, 2005]).

The procedure is quite simple: the set of reference weights is varied by small increments, generated by some random vector; from these variations the feedback function is computed, as well as their difference with the reference value (the feedback of the original weights). Finally the gradient is estimated by a Ridge regression.
There is plenty of random vectors that can be considered, each with a different result. In this project, every value in the set consists of a fixed perturbation multiplied by the result of a discrete uniform random variable in {-1,0,1}.

In the following page the general algorithm is presented: the result is an approximation of the gradient, with an accuracy increasing with the number of simulations P. It is easy to see that the feedback function is evaluated $P + 1$ times, so there is once again the trade-off between accuracy and computational speed.
The other important variable to keep into account is the Ridge parameter λ. Indeed, in the algorithm there is a matrix inversion, and λ ensures that the matrix is invertible. Moreover, it controls the size of the output, avoiding the *gradient explosion*, and reducing its variance. Different Ridge parameters have been tested on a reduced dataset: the results are shown in Figure 5.3.

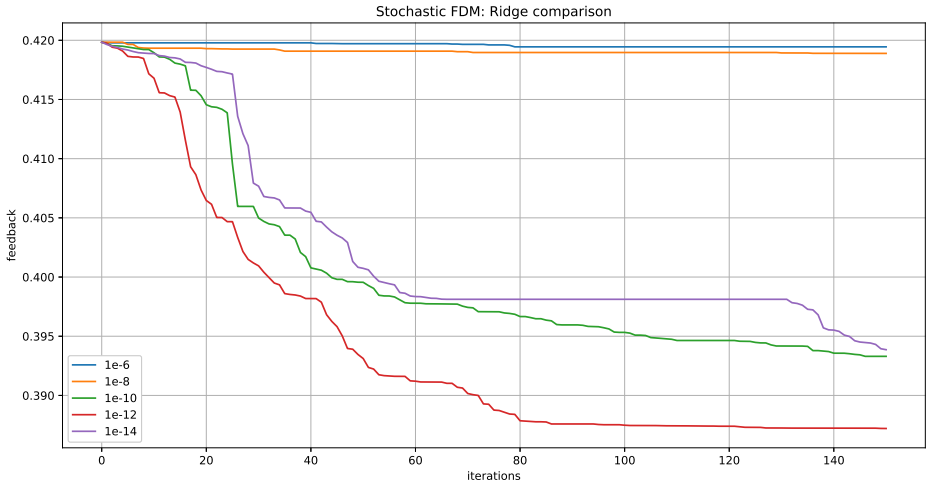


Figure 5.3: Comparison of different Ridge parameters λ. The training set contains 10 daily samples and the network is made of 25 neurons per hidden layer. The number of perturbations os SFDM is fixed to 24.

The easiest consideration that can be done is that large λ can negatively affect the optimization process. Indeed, the calibrators with $\lambda = 10^{-6}$ and $\lambda = 10^{-8}$ have a worse performance than the cases when the parameters are lower.
However, if λ assumes too small values, it may not be able to avoid the gradient explosion: for example, with $\lambda = 10^{-14}$ it is possible to see a slight worsening with respect to $\lambda = 10^{-10}$, if λ is set to even smaller values, the calibration diverges.
Consequently, the Ridge parameter chosen to perform this algorithm is set as the best performing, i.e. $\lambda = 10^{-12}$.

Algorithm 6 Stochastic Finite Differences Method (SFDM)

Input:

- Definition of feedback function $F(\cdot, \cdot)$,
- Definition of the network model $\mathcal{N}_{\boldsymbol{\omega}}$, defined by set of weights $\boldsymbol{\omega}$,
- Input set of features X, such that $\Theta = \mathcal{N}_{\boldsymbol{\omega}}(X)$,
- Increment h.
- Number of perturbations P.
- Ridge parameter λ

Output: Estimation of the gradient $\nabla_W F$ w.r.t. the given set of weights $\boldsymbol{\omega}$

Evaluate

$$\Theta = \mathcal{N}_W(X)$$
$$f = F(\Theta, Q)$$

for $i = 1, \ldots, P$ **do**
  for $j = 1, \ldots, W$ **do**
    Generate weight variation

$$x_j \sim U_{\{-1,0,1\}}$$
$$\Delta w_j = h\ x_j$$
$$\Delta \boldsymbol{\omega}_i = \{\Delta w_j\}_j$$

    Evaluate the new target

$$\widehat{\Theta}_i = \mathcal{N}_{\boldsymbol{\omega}+\Delta\boldsymbol{\omega}_i}$$
$$\Delta F_i = F(\widehat{\Theta}_i, Q) - f$$

  end
end

Perform Ridge Regression

$$\boldsymbol{\Delta\Omega} = \left[\Delta\boldsymbol{\omega}_1, \ldots, \boldsymbol{\Delta\omega}_P\right]^T \qquad \in \mathcal{R}^{PxW}$$
$$\boldsymbol{\Delta F} = \left[\Delta F_1, \ldots, \Delta F_P\right]^T \qquad \in \mathcal{R}^{P}$$
$$\nabla_{\boldsymbol{\omega}} F = \left(\boldsymbol{\Delta\Omega}^T \boldsymbol{\Delta\Omega} + \lambda \mathbf{I}_W\right)^{-1} \boldsymbol{\Delta\Omega}^T \boldsymbol{\Delta F}$$

5.2.3 Backpropagation

The last method considered is perhaps the most famous and used to train standard neural networks in supervised learning problems, where the ANN must fit known targets. In each iteration, an error signal is produced by comparing the output of the network with the desired response (the real target) by using a simple error measure (usually the mean square error). The resulting error signal is then propagated through the network, but the propagation is performed layer by layer in the backward direction [Haykin]. This is why this method is called *Backpropagation Algorithm (BACK)*

In the case of black box calibrations, as the one used in this project, things get more difficult, but this approach can still be considered for the computation of the gradient. Recalling that feedback $F(\Theta, Q)$ is the mean value of the several daily errors $E(\Theta_d, Q_d)$ as defined in Equation 3.3, it is possible to consider separately the derivatives with respect to the weights in the various days:

$$\frac{\partial F(\Theta, Q)}{\partial w_{ij}} = \frac{1}{D} \sum_{d=1}^{D} \frac{\partial E(\Theta_d, Q_d)}{\partial w_{ij}}. \tag{5.1}$$

Moreover, considering that $\Theta_d = \{a_d, b_d, \sigma_d, \eta_d, \rho_d\}$, it is possible to compute the gradient of E with respect to the weights by considering one target at a time[1]:

$$\frac{\partial E(\Theta_d)}{\partial w_{ij}} = \sum_{\theta \in \Theta_d} \frac{\partial E}{\partial \theta} \frac{\partial \theta}{\partial w_{ij}} \tag{5.2}$$

The first term in the sum refers to the derivative of the feedback error as function of one of the targets: recalling Equations 3.1 and 3.2 of the previous chapter the daily error is defined as:

$$E(\Theta_d, Q_d) = \sqrt{\sum_i \frac{1}{\nu_i^2} (\widehat{y}_i(\Theta_d, Q_d) - y_i)^2}. \tag{5.3}$$

From this formula, it is easily possible to consider the first term in the sum in Equation 5.1 as:

$$\frac{\partial E(\Theta_d, Q_d)}{\partial \theta} = \frac{\sum_i \frac{1}{\nu_i^2} \left(\widehat{y}_i - y_i\right)^2 \partial_\theta \widehat{y}_i(\Theta, Q_d)}{E(\Theta_d, Q_d)}. \tag{5.4}$$

Here the only component still not known is $\partial_\theta \widehat{y}_i(\Theta, Q_d)$: it regards exclusively the pricing model considered and not the ANN.
There are two possible approaches to compute this element:

- In the former the derivative is estimated empirically with finite differences for a suitable increment of the targets. The increment chosen is about 10^{-8}, since it

[1] the dependency of the feedback on Q_d has been omitted for simplicity

seems sufficiently low for an adequate approximation of the derivative, and sufficiently stable (lower values seem to generate some other approximation errors due to the limited floating precision of the GPU).
This means that the pricing procedure is considered 6 times: the first calling is necessary for the computation of the predicted targets; the others for the evaluation of $\partial_\theta \widehat{y}_i$, one for every parameter of the model.
This approach is referred as "Approximated BACKPRO".

- Otherwise, the derivatives can be computed analitically: these derivations required some efforts to be done and are presented in Appendix A. The numerical computation of these derivatives is performed on the GPU along with the pricing procedure. This means that every kernel has more tasks to perform, but they are called only once.
This method will be referred as "Analytic BACKPRO" or "BACK_AN".

The second term in the sum in Equation 5.2 does not depend on the financial model, but only on the structure of the ANN. To compute it, it is necessary to analyze separately its layers, using the "Backpropagation property" typical of the neural networks: for this reason, the derivatives are computed starting from the weights at the end of the network going backwards to the very first weights, with the formulas shown in Appendix B.

Algorithm 7 BackPropagation (BACK)

Input:

- Feedback function $F(\cdot,\cdot)$,
- Network model $\mathcal{N}_{\boldsymbol{\omega}}$, defined by set of weights $\boldsymbol{\omega}$,
- Input set of features X, such that $\Theta = \mathcal{N}_{\boldsymbol{\omega}}(X)$,
- Increment h.

Output: Estimation of the gradient $\nabla_W F$ w.r.t. the given set of weights $\boldsymbol{\omega}$

for $i = 1, \dots, W$ **do**
  for $d = 1, \dots, D$ **do**
    Evaluate and Initialize

$$E_d = E(\Theta_d, Q_d); \qquad \frac{\partial E_d}{\partial \omega_i} = 0$$

    Compute the derivative of price w.r.t. G2++ parameters for every date:

    - (Analytic) using formulas in Appendix A
    - (Approx) with finite differences: $\partial_\xi \widehat{y}_i = \frac{\widehat{y}_i(\Theta_d + h\mathbf{e}_\xi, Q_d) - \widehat{y}_i(\Theta_d, Q_d)}{h} \quad \forall \xi \in \Theta_d$

$$\nabla_{\Theta_d} E_d = \left\{ \frac{\sum_i \frac{1}{\nu_i^2} \left(\widehat{y}_i - y_i\right)^2 \partial_\xi \widehat{y}_i}{E(\Theta_d, Q_d)} \right\}_{\xi \in \Theta_d}$$

    Compute using formulas in Appendix B:

$$\partial_{\omega_i} \Theta_d = \left\{ \frac{\partial \xi}{\partial \omega_i} \right\}_{\xi \in \Theta_d}$$

$$\frac{\partial E_d}{\partial \omega_i} = \nabla_{\Theta_d} E_d \cdot \partial_{\omega_i} \Theta_d$$

  end
end

$$\nabla_{\boldsymbol{\omega}} F = \left\{ \frac{1}{D} \sum_{d=1}^{D} \frac{\partial E_d}{\partial w_i} \right\}_{i=1}^{W}$$

5.2.4 Comparison

Several approaches for computing the gradient have been considered and they are now compared. The first ANN considered is rather simple (15 neurons per hidden layer), but the results show what the behavior of the different algorithms in more complex scenarios would be. The same argument applies to the size chosen for the dataset: to perform calibrations in a short time, the training dataset considered contains only 10 daily samples, the same as the test size.
The setting of the weights from which the calibrations start is the same, and it is the one resulting after the global CE optimization, which led to a set of weights whose related mean feedback is equal to 0.4223. Then, the BFGS algorithm is used with the different methods for a maximum of 250 iterations.

The SFDM algorithm has been used many times, with an increasing number of perturbations. For example, the notation "SFDM8" denotes the calibration in which the gradient computation is performed through the SFDM algorithm with 8 perturbations. Table 5.1 shows the time needed to perform the 250 iterations for every method, followed by the final training feedback and on the testing datasets.
Moreover, in Figure 5.4 it is possible to see how the mean feedback on the training set decreases from iteration to iteration.

From the analysis of the results, some observations can be made:

- Backpropagation algorithms perform clearly better than other methods from both the points of view of optimization and of time. They take a really small number of iterations to converge and each iteration is fast to compute. Even if the final results are the same, the analytic backpropagation is slightly faster than the approximate version.

- SFDM is by far the method getting the worst results: even with a high number of perturbations, 250 iterations are not enough to reach convergence. As expected, while the number of perturbations increases, the optimization performs better, but, on the other hand, the computational time per iteration gets higher and higher.

- Finally, FDM needs a small number of iterations to reach convergence (still worse than backpropagation). However, the time needed with this method is huge if compared to the other ones.

When the number of neurons of the ANN and the dataset considered are bigger, all the feedback computations take more time to be evaluated. Hence, all the differences in terms of performance get bigger and bigger. For example, if the dataset has hundreds of daily samples and there are at least 30 neurons per hidden layer, there is no chance to

adopt FDM (it would need weeks to converge). Consequently, the method adopted for this project for the calibration of the complete dataset is the analytic backpropagation.

Method	*Time(s)*	Train_fb	Test_fb
Analytic BACKPRO	655	0.2971	0.2994
Approx BACKPRO	760	0.2971	0.2993
SFDM8	730	0.3478	0.3499
SFDM16	1010	0.3218	0.3289
SFDM24	1254	0.3174	0.3236
SFDM32	1740	0.3152	0.3219
SFDM64	2810	0.3133	0.3193
FDM	16447	0.3016	0.3067

Table 5.1: Comparison of the final performance after 250 BFGS iterations for every gradient algorithm.

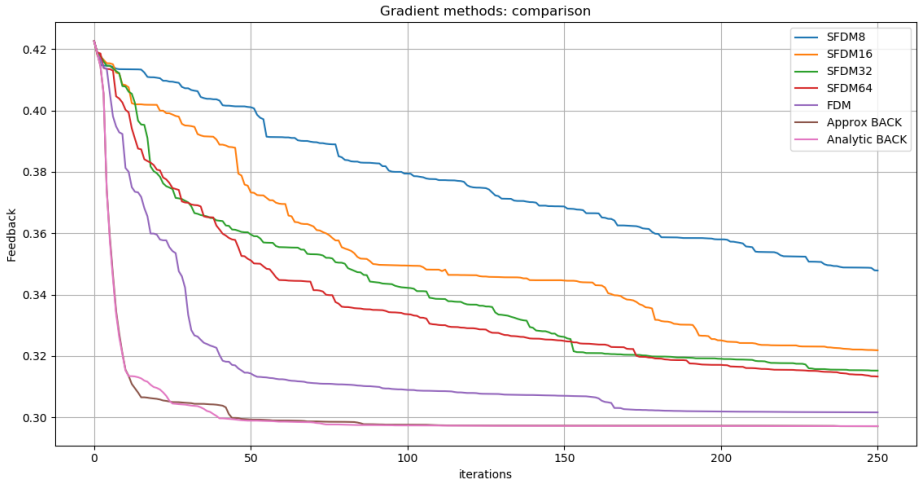


Figure 5.4: Comparison of the performance of different gradient algorithms: the gradual optimization of the parameters in the network is shown for the first 250 iterations for every method. The network considered has 15 neurons for every layer and the dataset contains only 20 samples, 10 used for training, the other for testing.

Chapter 6

Experimental Results

This chapter is dedicated to presenting the results obtained; three different types of experiments are analyzed:

- Offline calibration on the EUR dataset;
- Online calibration on the EUR dataset;
- Offline calibration on a multicurrency dataset.

In particular, every test is compared to the feedback error obtained through the calibration made by BANCA IMI using Vasicek's model. Moreover, in the single-currency offline calibration, the results are compared also with the calibrator built using Vasicek's model [Donati, 2018].

6.1 Offline calibration

The first kind of experiment is the offline or batch calibration. As explained in Section 3.1.1, the dataset is split into two subsets; the first (which accounts for 66% of the total size) is the training set, which is the framework actually used by the calibrator to tune the weights (through CE and BFGS). Once this process is ended, it is evaluated on the test set, which contains all the remaining samples.

The whole process took globally 8 hours and 43 minutes (using a Nvidia Tesla K40c GPU with 2880 Cuda cores); the first four and a half hours were spent during the CE, while, in the remaining time, BFGS optimization was in process.
The final feedback curve is shown in Figure 6.1: globally, the result is much better than the one obtained through the bank's calibrator (which uses Vasicek's model).
Moreover, also in the test phase there is a good performance; indeed, when new samples

are observed, the error is expected to worsen. This could mean that the test samples do not show completely unseen scenarios, so the behavior of the parameters can be foreseen by looking at the past.

To get an idea of how the calibrator performed with respect to its optimal possibilities, Figure 6.2 also compares the feedback obtained with the one resulting from the single-day calibration (introduced in Section 3.4).
This calibration was performed on a reduced set of daily samples, which, more or less, coincides with the training set of the global calibration.
It is clearly depicted that in more than half of the samples, the predicted parameters are very close to the optimal ones. The feedbacks in 2015, however, show a worse performance: this is the sensitive region already underlined in Section 3.4, where small differences in parameters can generate great differences in feedback.

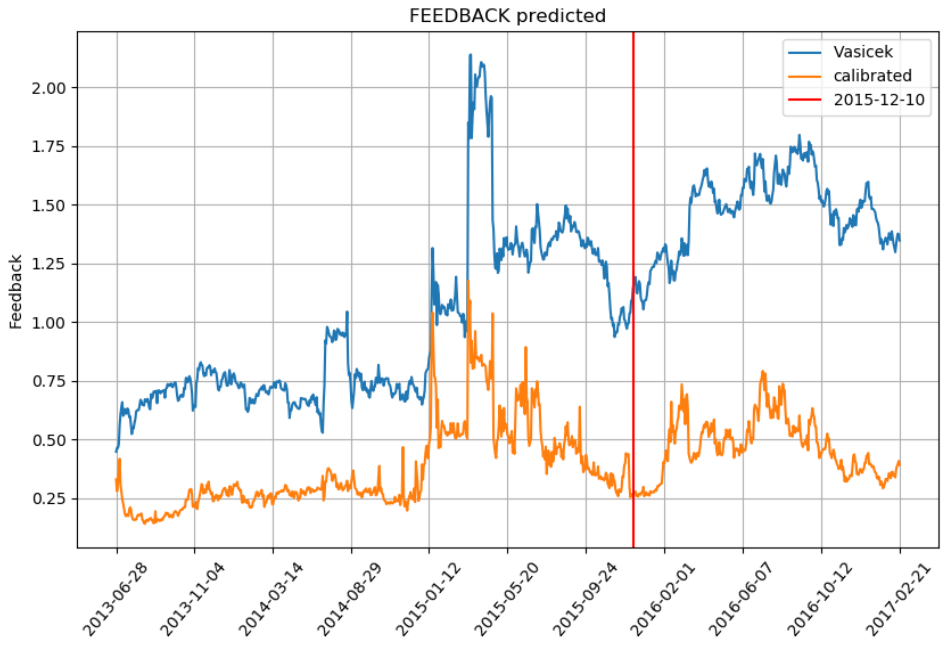


Figure 6.1: Offline calibration: feedback.
The red line denotes the end of the training phase.

By looking at Figure 6.3, it is possible to see what is the mean contribution to the

feedback of each tenor/expiry couple. The dark, central region is the subset of most liquid swaptions, which are the most important to price correctly. The biggest errors are made for the less liquid options.

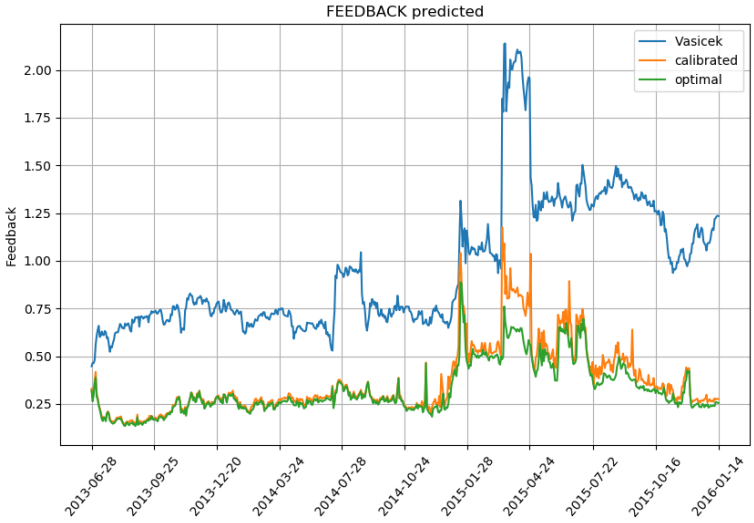


Figure 6.2: Offline calibration: comparison with single-day calibration.

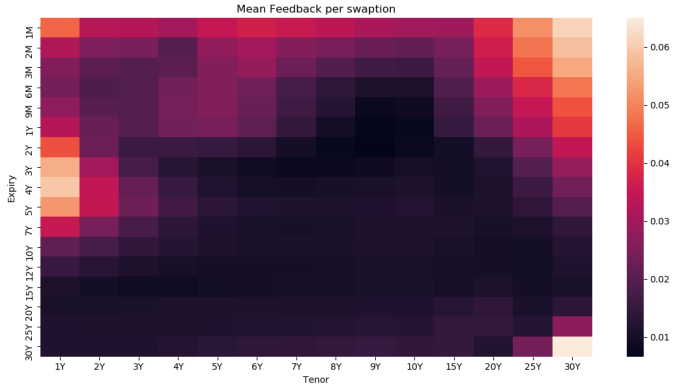


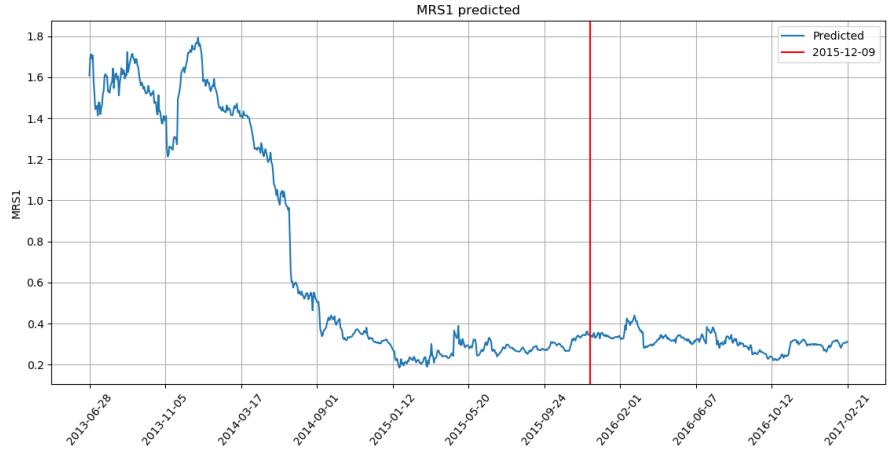
Figure 6.3: Offline calibration: maximum difference in prices

Taking a closer look at the values of the parameters resented in Figure 6.4, some observations can be made:

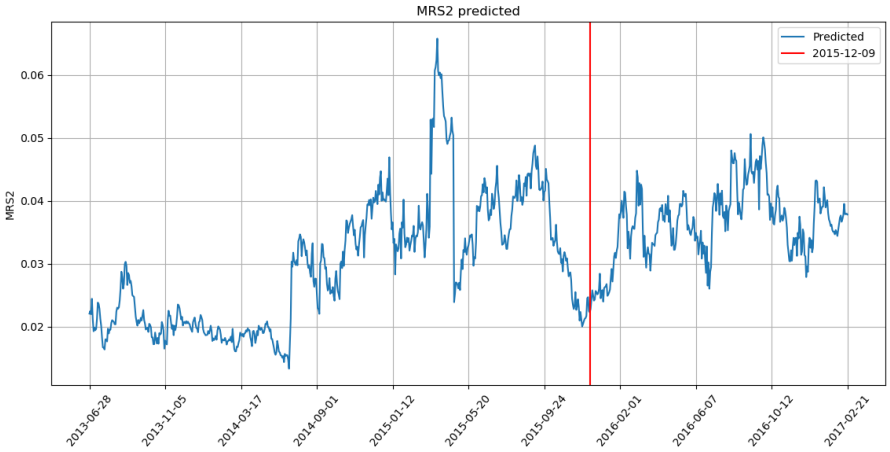
- The MRSs of the two processes defining G2++ model have a different magnitude: a has a range between (0.185, 1.79) while b assumes values in 0.013 and 0.065.
- On the other hand, the VOLs are in a similar range of values, even if σ is always slightly higher than η.
- At last, there is always a strong negative correlation between the Brownian Motions.

As a consequence, it is possible to formulate an interpretation of the behavior described by the interest model chosen: the mean reverting process described by the couple (a, σ) has the same amplitude of randomness of the other one, defined by (b, η); however, their trajectories deviate from the mean value in opposite directions. Finally, the processes have a different reaction to these deviations: the first one has a stronger force driving it back to its mean, while the other one is milder. It is like one process drives the short-term dynamics, in contrast with the long-term process.

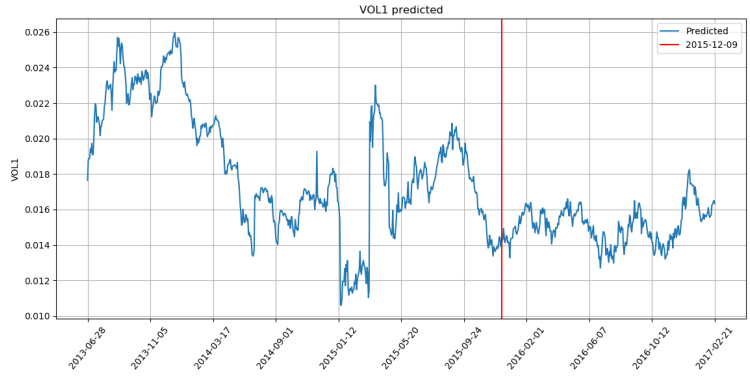
Another interesting observation can be made by looking at the trajectories described by the parameters: the volatilities have a similar shape in the second half of the dataset, while b and η show the same pattern through all the samples.
Hence, we can ask ourselves if there is some kind of correlation between the parameters; for this reason, Figure 6.5 illustrates a representation through time of all the possible couples of parameters. Among them some specific couples are more clearly shown and commented in the next images.
In particular, it is often possible to detect two group of samples containing different range of Reference Dates. In other cases, as in Figure 6.7 it is possible to detect some correlations in the behaviour of the parameters, as the pattern is close to a straight line.



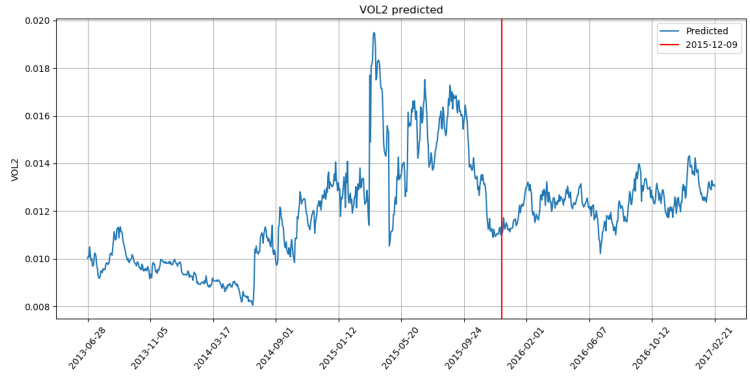
(a) Mean reversion speed of the first stochastic process (parameter a)



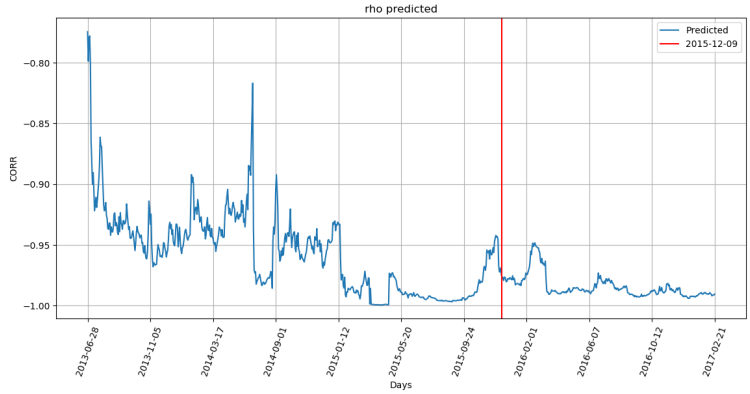
(b) Mean reversion speed of the second stochastic process (parameter b)



(c) Volatility of the first stochastic process (parameter σ)



(d) Volatility of the second stochastic process (parameter η)



(e) Correlation of the Brownian Motions (parameter ρ)

Figure 6.4: Offline calibration: predicted parameters

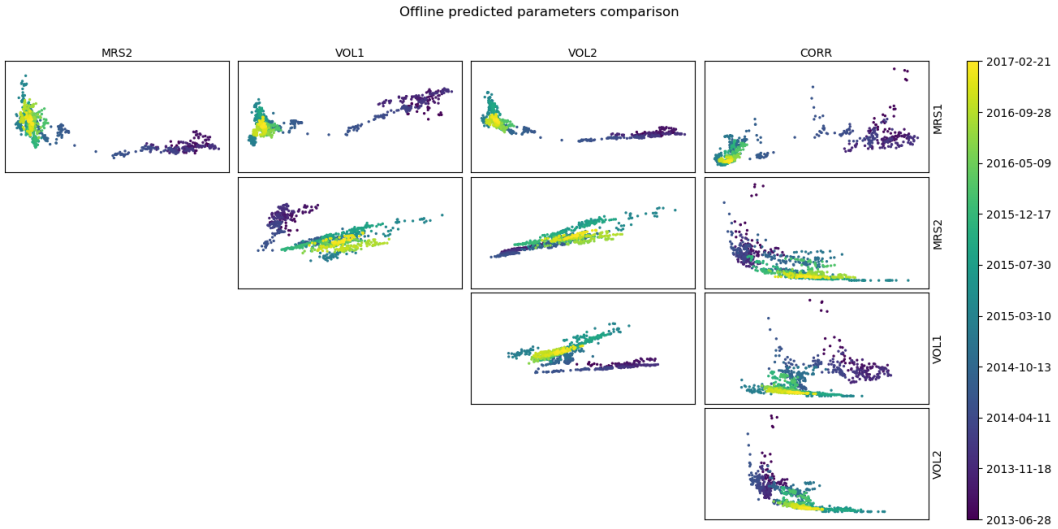


Figure 6.5: Offline calibration: representation of the couples of parameters through time

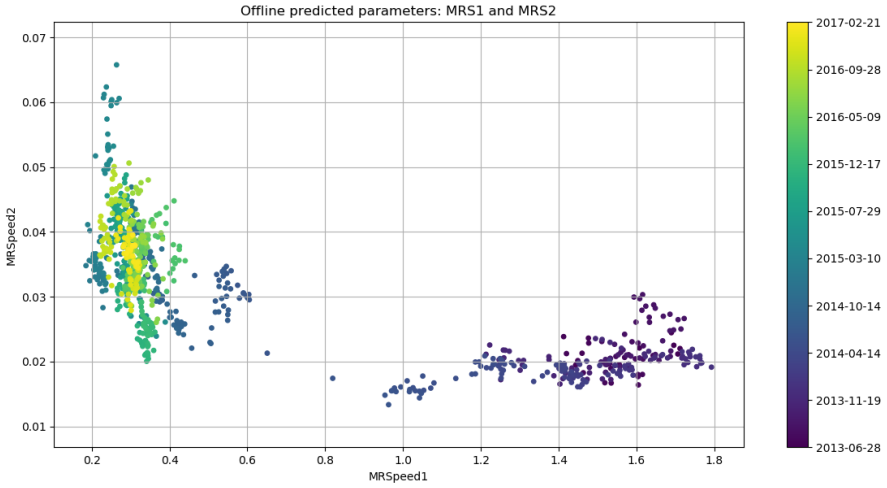


Figure 6.6: Offline calibration: representation of the predicted Mean Reversion Speeds through time.
It is possible to detect two distant groups of points; the one at the right-hand side contains the first predictions; then, as depicted also in Figure 6.4a, the first mean reversion speed decreases to lower values.

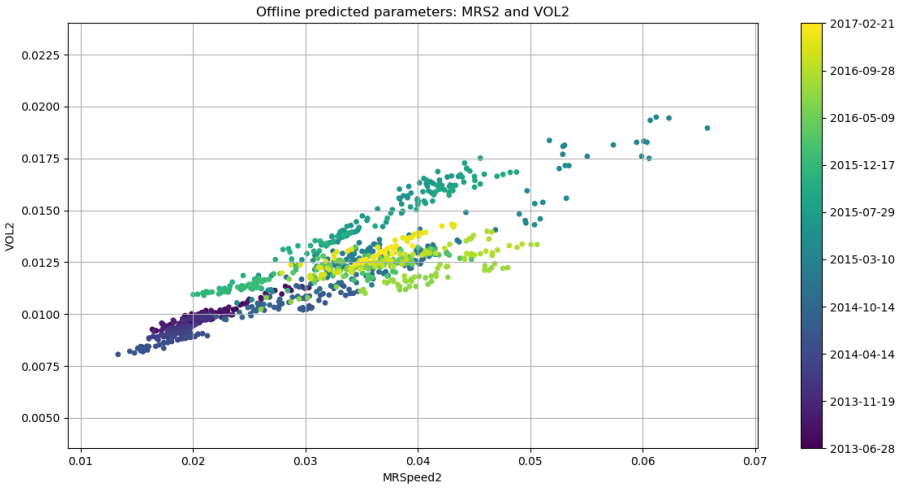


Figure 6.7: Offline calibration: representation of the couples (b, σ) through time. It is easy to see that the pattern created is close to a line.

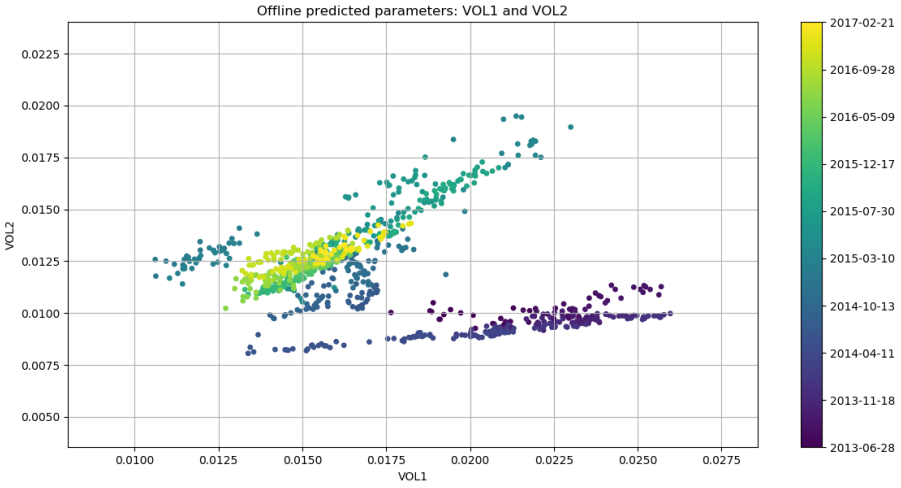


Figure 6.8: Offline calibration: representation of the predicted volatilities through time. There are two groups of daily samples, whose patterns create approximately two lines.

6.2 Online calibration

In a second instance the ANN was built in order to perform an online calibration.
As mentioned in Section 3.1.1, the dataset is split into three parts:

- The first is the usual training set, where the batch calibration is performed as in the previous experiments. Here, the optimization procedure starts from the global exploration of the space of the parameters (CE algorithm) then refined through the BFGS algorithm. This part is set to contain the first 60% of the daily samples.
- Secondly, the online phase can start: iteratively, a new sample is considered and tested using the tuned ANN; after that, it is added to the starting dataset, and a local calibration is performed through BFGS, adjusting the previous set of weights. This procedure is the most similar to the one that is daily performed by traders, so it is important to perform it quickly. This part involves 20% of the daily samples.
- The last 20% of the dataset is considered for the testing: the weights of the offline step and the final set of online weights are considered, as well as the respective set of output parameters and the feedback produced.

It is expected from the online calibrator to perform better than the offline, and the gap should get bigger as the number of new samples considered increases, since the online learner can map unseen situations.

The main drawback of this methodology lies in the computational times during the online phase: the daily calibration simply adjusts the weights with a few iterations, but this procedure is still too much time-consuming to be effectively useful for an investment bank.
Hence, for each BFGS calibration a limit on the maximum number of iterations has been put, as a trade-off between accuracy and speed. Of course this is one of the main topics to face in future researches.

As far as the predicted parameters are concerned, the results are similar to the ones shown in the offline case. The resulting feedback is more interesting, as shown in Figure 6.9: once the end of the offline training is reached, the set of neural weights is used to evaluate also the feedback on the rest of the dataset, considered as a test set. These evaluations are depicted with the green-colored curve.
First, it is possible to notice that this feedback curve relating to the offline calibration is slightly worse than the one expressed in the previous Section: this is mainly due to the fact that the maximum number of iterations in the offline procedure was lowered for these tests, and the algorithm stopped before reaching its maximum potential.

After this, the online calibration is performed, following the above-mentioned procedure. In this way, remarking that the last 20% of the dataset is dedicated to evaluation purposes, it is possible to build the online feedback curve, colored in blue.
In the comparison of the curves, emerges a slight improvement obtained by the online calibrator. As expected, the gap increases as long as new dates are introduced, even if it is reduced in the last section of the test phase.

From this consideration it becomes clear that, in order to have useful results, a great effort must be put into the reduction of the computational times: currently, convergence was reach only a few times, and more time was needed to complete the process (a complete experiment was impossible to perform in this project for practical purposes); even with this limitations, the local calibrations ended in 1 hour and 10 minutes on average: it is not possible to adopt the current solution for real applications without deep changes.

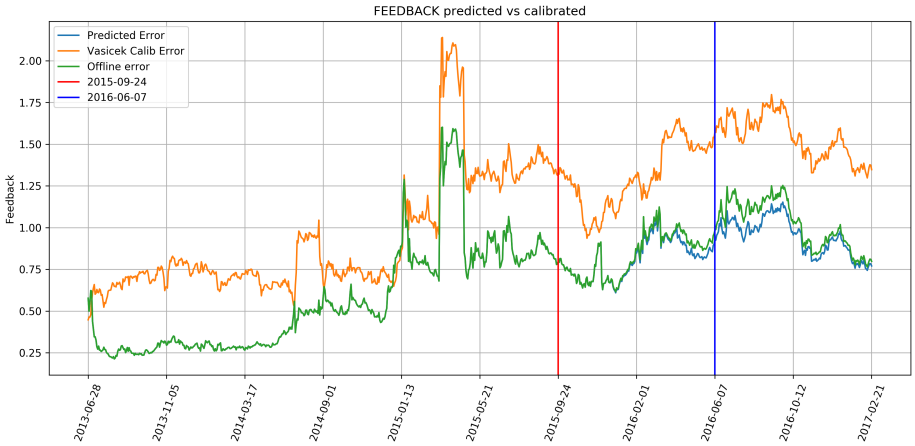


Figure 6.9: Online calibration: feedback curves comparison.
The orange curve denotes Vasicek's single-day calibration.
The green curve refers to the offline calibration, with training ending at the red vertical line. Online calibration is then performed until the blue vertical line, where the testing phase starts. The feedback of this process is shown in the blue curve.

6.3 Multicurrency offline

The last tests were made taking into account the multicurrency dataset.
In particular, four currencies were selected: EURO, USD, CHF and CAD. The procedure of selection of the Reference Dates and of the daily swaptions is explained in Section 4.1.3.
Since the range of values assumed by prices is different for each currency, the number of neurons per hidden layer has been increased to 40 per hidden layer: a more complex network might be able to express better the variability of the data.

The choice of performing a simultaneous calibration for all the datasets might deeply affect the performance: indeed, it allows the exploration of a wider range of the input manifold. The sets of prices are mapped into g2++ parameters by the ANN; the domain of this map is enlarged in a multicurrency framework.
As a consequence, once the training phase ends, during the test there should be a reduction in the risk of facing unexpected, non-mapped scenarios.
On the other hand, there is a drawback: there are small daily variations in the local datasets that might be missed. This can be seen as a "resolution limit": in order to express the global variability of the data, there is a loss of focus on a local basis, so the training feedbacks might encounter a small worsening.

The major criticality of the multi-currency calibration emerges immediately in the comparison of different simulations: the output parameters are in completely different ranges. This means that there are different local minima, and the calibrator gets stuck in one of them, depending on the points randomly selected during the CE.
There are three main scenarios from the calibrator, but the pattern shown in the offline single-currency calibration occurs in none of them.

- In the first scenario the values of the biggest Mean Reversion Speed are usually in the range [1.5, 2.6] with some exceptions in CHF dataset, while the second MRS is often negative, with values between -0.15 and 0.025. The curves of the volatilities and correlation are similar to the ones resulting from the single-currency calibration.

- The second scenario is completely different from all other results, since both the MRSs can take negative values. One is centered in zero, with values in the range [-0.18, 0.18], with the exception of the first days in CHF dataset; the second one is almost always negative, with a minimum value around -0.15. Also the correlation is remarkable and very different from the other cases, since it ranges from -0.4 to 0.6. Finally, volatilities are similar to the first scenario.

- In the last scenario, one of the Mean Reversion Speeds has large values, between 2.95 and 2.99, while its related volatility is almost always below 10^{-4}. The pattern of MRS and volatility of the second G2++ process is similar to the one shown in the first scenario, differently from the correlation, which is always positive and over 0.7.

The feedback curves are different in the various scenarios; however, it is possible to see (in Section C.5) that the first scenario has the best overall performance; for this reason, all the curves of the predicted parameters and of the feedbacks are shown in Appendix C only for this case, while only the most relevant aspects of the other are shown.

This calibration is not comparable to the one made by the bank using the Vasicek's model or the one made considering only EUR, because they are made using a different set of swaptions. The only possible comparison can be made on CHF dataset, which is made of the 100 common swaptions: Figure C.7 shows that the calibration (on a single daily samples at a time) the bank made using Vasicek's model has a better performance than the one resulting from the first scenario.

For this reason, it was decided to perform an offline calibration only over the CHF dataset, in order to be able to generate a reference feedback for G2++ model and compare it with the Vasicek's.
The result is shown in Figure 6.10: there are just a few days in which the G2++ calibration performs better than the Vasicek's; in the other cases, the best situation is the replication of the results of the simplest model, and the introduction of a more complex one does not bring any advantage (this holds only for the CHF dataset).

By comparing the curves of the single-currency and the multi-currency calibrations, it is possible to make another consideration: as expected, during the training phase the multi-currency calibration generally has a worse performance than the other one, which on the contrary has a high feedback function in the first days of the test phase.

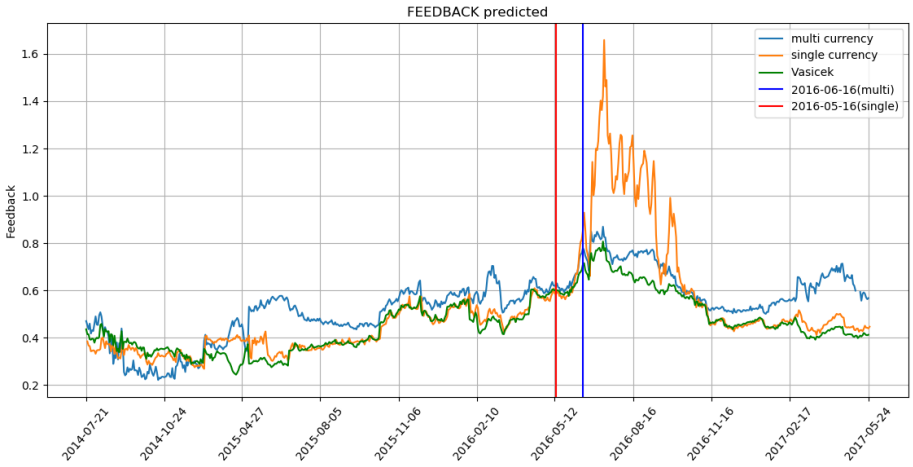


Figure 6.10: CHF feedback function: comparison of the different calibrations.
The green line denotes Vasicek's single-day calibration.
The orange curve is referring to the offline calibration on the only CHF dataset (with training ending at the red vertical line).
The blue curve is referring to the offline, multi-currency calibration (with training ending at the blue vertical line).

Chapter 7

Conclusions

This final chapter provides a synthetic summary of the work done and of the results achieved.
Then, some possible future directions for future research are suggested.

7.1 Summary of results

With regards to the analysis of the results obtained with the previous steps of this project, the necessity of using a different financial model to price swaptions was evident. This was needed, as underlined in Section 2.2.1, because the Vasicek's model was not able to represent the evolution of interest rates adequately.

For this reason a new, more expressive model is introduced in Chapter 2, the G2++.
Thus, all the financial formulas needed are obtained, starting from bond prices up to swaptions. There is no possibility of reducing the final pricing formula to an analytic one, and some approximations have to be considered.

The calibrator, introduced in Chapter 3, is built in such a way that the learning structure and the financial one can be separated, so that the model change model has been easy to implement.
The first great benefit achieved immediately is the 65% gain in the feedback function with the new model, shown in Section 3.4.

The most complex topic of the calibrator is the computational time: from one hand, once the network is tuned, the prediction of the parameters takes a few milliseconds against the order of seconds seen with the bank's calibrator; the main drawback is in the optimization process that involves the pricing procedure thousands of times. Even with

Vasicek's model, the selection of a reduced set of swaptions to calibrate was required, despite the implementation of the feedback function on GPU using CUDA allowed an incredible boost in the procedure.
As a consequence, the BFGS algorithm (presented in Section 3.3.2) has been optimized with the introduction of several methods for the computation of the gradient of the feedback: The best one is the backpropagation algorithm, explained in Section 5.2.3, which allows the overall set of daily swaptions to be considered in calibration.

As a result (Section 6.1), offline calibration had a great performance in short times on the overall EUR dataset. This is clearly a big improvement in the project, especially compared to the calibrator using Vasicek's model.
Some tests were also made on a multicurrency dataset, and the results, shown in Section 6.3, are not as good as the previous offline calibration on a single currency, but they are still promising.
The online calibrator, the results of which are presented in Section 6.2, is the closest to the one needed for the project, and it had still a good performance, but the daily local optimization was still substantially slow.

Thus, with regard to the objectives of this thesis, it is possible to say that they are achieved; at the same time, there is still a lot of work to do as well as challenges to face in order to fulfill the purpose of the whole project.

7.2 Future research

Starting from the results achieved with the work presented in this thesis and looking at the final purpose of the project, it is possible to suggest some potential future developments and improvements.

Online improvements. The final calibrator must be able to calibrate and price the contracts in a short time on a daily basis; for this reason, great importance is given to the online calibrator. As the results shown in Section 6.2, this calibrator is still not reaching its full potentialities and should be improved as a short-term goal to achieve better results in a shorter time.

Intraday exploitation. The dataset considered in this thesis provides information on a daily basis. As a matter of facts, prices and discount curves change continuously; as a consequence, fixing the interest rates model's parameters in a whole day introduces a small bias. The ideal calibrator should be able to adapt the predictions according to the intraday information given; however, the current solution is not able to manage in

short times a dramatically increased size of the input dataset.

Multicurrency extension. Currently, the multicurrency calibration considered a dataset composed of contracts in four currencies. One of the natural future developments consists in adding more currencies to the dataset, and eventually introduce the online calibration, once fixed the topic related to multiple local minima.
Moreover, as a long-term goal can also be considered the introduction of exchange rates: in a multicurrency framework, the G2++ dynamics introduced in Chapter 2 are considered as independent for every currency; actually, in an arbitrage-free setting, they are bounded to satisfy the exchange rates, and they should be considered especially if the parameters are used to price multicurrency derivatives. In this scenario, the calibration must follow a constrained optimization procedure which adds a huge complexity to the problem.

Contract Generalization. As already explained, the calibration produces a model representation of the interest rate that could be used to price different instruments and not only swaptions. Another natural generalization of this project consists into moving to an extended space of contracts. In particular, the first derivatives that can be included are swaptions which are not At The Money. This extension may increase the number of features since it may require more information about the properties of the contracts.

Model Generalization. Investment banks deal every day with a huge amount of interest rate derivatives, but they are usually priced with different rate models. As explained previously, this calibrator can be adapted with a small effort to a different model; however, an interesting improvement (and perhaps the most difficult to achieve) is its generalization on the simultaneous calibration of different models.

Appendices

Appendix A

Derivatives in Swaption Pricing Model

The purpose of this Section is to provide useful formulas to compute the derivatives of swaption prices with respect to G2++ parameters. This task is important in order to apply backpropagation in the neural network.

Here we recall the final formula used to price swaptions

$$SO(0,T_\alpha,\Gamma,K) = \frac{P_d(0,T_\alpha)}{\sqrt{\pi}} \sum_{k=1}^{n} \omega_k \left[\sum_{j\in\Gamma} \lambda_j(x_k) e^{\kappa_j(x_k)} \Phi[-h_j(x_k)] \right], \tag{A.1}$$

where

$$x_k = \sqrt{2}\sigma_x \widehat{x}_k + \mu_x \tag{A.2}$$

$$h_j(x) = \overline{h} + B(b,T,t_j)\sigma_y\sqrt{1-\rho_{xy}^2} \tag{A.3}$$

$$\overline{h} = \frac{\overline{y}(x)-\mu_y}{\sigma_y\sqrt{1-\rho_{xy}^2}} - \frac{\rho_{xy}(x-\mu_x)}{\sigma_x\sqrt{1-\rho_{xy}^2}} \tag{A.4}$$

$$\lambda_j(x) = c_j e^{-B(a,T,T_j)x} \tag{A.5}$$

$$\kappa_j(x) = -B(b,T,t_j)\left[\mu_y - \frac{1}{2}(1-\rho_{xy}^2)\sigma_y^2 B(b,T,t_j) + \rho_{xy}\sigma_y\frac{x-\mu_x}{\sigma_x}\right] \tag{A.6}$$

$$\mu_x = -M_x^T(0,T_\alpha) \tag{A.7}$$

$$\mu_y = -M_y^T(0,T_\alpha) \tag{A.8}$$

Moreover,

- $\overline{y}(x)$ is the only solution of

$$\sum_{j\in\Gamma} c_j e^{-B(a,T_\alpha,t_j)x - B(b,T_\alpha,t_j)\overline{y}(x)} = 0 \tag{A.9}$$

- c_j is the union in a single set of days of the following coefficients:

$$\begin{aligned} d_i^1 &= \frac{\Phi_d(S_{i-1}, S_i)}{\Phi_F(S_{i-1}, S_i)} \Phi_d(T_\alpha, S_{i-1}) A(T_\alpha, S_{i-1}) && i = \overline{\alpha}+1, ..., \overline{\beta} \\ d_i^2 &= -\Phi_d(T_\alpha, S_i) A(T_\alpha, S_i) && i = \overline{\alpha}+1, ..., \overline{\beta} \\ d_i^3 &= -K\tau_i \Phi_d(T_\alpha, T_i) A(T_\alpha, T_i) && i = \alpha+1, ..., \beta \end{aligned} \tag{A.10}$$

- $\widehat{x}_j$ and w_j are respectively the j-th Hermite-Gauss node and weight (thus, not depending on a variable).

- K is the ATM strike of the underlying swap, evaluated at the reference date.

We consider one element at a time, along with its derivatives.

A.1 Bond prices and shift ratio

Let's consider the first element: bond prices. When $t_0 = 0$, they do not depend on the interest rates model selected, but only on the discount curve. However, Formula 2.28 includes the term $\Phi_d(t,T)A(t,T)$ that must be evaluated at the expiry $t = T_\alpha$.

Hence, we start with the term:

$$\begin{aligned} \Phi_d(t,T)A(t,T) &= \Phi_d(t,T) exp(V(T-t)/2) \\ &= \frac{P_{OIS}(0,T)}{P_{OIS}(0,t)} \frac{\overline{P}(0,t)}{\overline{P}(0,T)} exp(V(T-t)/2) \\ &= \frac{P_{OIS}(0,T)}{P_{OIS}(0,t)} exp(\frac{V(t) - V(T) + V(T-t)}{2}) \end{aligned}$$

A.1.1 B and ω

Let's consider the usual B used in the pricing chapter

$$B(z, \Delta) = \frac{1 - e^{-z\Delta}}{z} \tag{A.11}$$

To make thing easier later, we introduce another variable similar to the previous one, called ω:

$$\omega(p, q, \Delta) = \frac{1 - e^{-(p+q)\Delta}}{p+q} \tag{A.12}$$

Their derivatives are immediate to compute:

$$\begin{aligned} \partial_z B(z,\Delta) &= -B(z,\Delta)(\Delta + \frac{1}{z}) + \frac{\Delta}{z} && := \widehat{B}(z,\Delta) \\ \partial_p \omega(p,q,\Delta) = \partial_q \omega(p,q,\Delta) &= -\omega(p,q,\Delta)(\Delta + \frac{1}{p+q}) + \frac{\Delta}{p+q} && := \widehat{\omega}(p,q,\Delta) \\ \partial_p \omega(p,p,\Delta) &= 2\ \widehat{\omega}(p,p,\Delta) \end{aligned}$$

A.1.2 V and its exponential

Let us remark that

$$\begin{aligned}
V(\Delta) = \quad & \frac{\sigma^2}{a^2}[\Delta + \frac{2}{a}e^{-a\Delta} - \frac{1}{2a}e^{-2a\Delta} - \frac{3}{2a}] \\
& + \frac{\eta^2}{b^2}[\Delta + \frac{2}{b}e^{-b\Delta} - \frac{1}{2b}e^{-2b\Delta} - \frac{3}{2b}] \\
& + 2\rho\frac{\sigma\eta}{ab}[\Delta + \frac{e^{-a\Delta}-1}{a} + \frac{e^{-b\Delta}-1}{b} - \frac{e^{-(a+b)\Delta}-1}{a+b}] \\
= \quad & \frac{\sigma^2}{a^2}[\omega(a,a,\Delta) - 2B(a,\Delta) + \Delta] \\
& + \frac{\eta^2}{b^2}[\omega(b,b,\Delta) - 2B(b,\Delta) + \Delta] \\
& + 2\rho\frac{\sigma\eta}{ab}[\omega(a,b,\Delta) - B(a,\Delta) - B(b,\Delta) + \Delta] \\
= \quad & \frac{\sigma^2}{a^2}\Psi(a,a,\Delta) + \frac{\eta^2}{b^2}\Psi(b,b,\Delta) + 2\rho\frac{\sigma\eta}{ab}\Psi(a,b,\Delta)
\end{aligned}$$

Where, in order to simplify future formulas, we have defined

$$\Psi(a,b,\Delta) := \omega(b,b,\Delta) - 2B(b,\Delta) + \Delta \tag{A.13}$$

With derivatives:

$$\begin{aligned}
\partial_a\Psi(a,b,\Delta) &= \widehat{\omega}(a,b,\Delta) - \widehat{B}(a,\Delta) \\
\partial_b\Psi(a,b,\Delta) &= \widehat{\omega}(a,b,\Delta) - \widehat{B}(b,\Delta) \\
\partial_a\Psi(a,a,\Delta) &= 2\ [\widehat{\omega}(a,a,\Delta) - \widehat{B}(a,\Delta)]
\end{aligned}$$

This leads to the computation of the derivatives of V:[1]

$$\begin{aligned}
\partial_a V(\Delta) = & -\frac{2\sigma^2}{a^3}\Psi(a,a,\Delta) + \frac{\sigma^2}{a^2}\partial_a\Psi(a,a,\Delta) \\
& - 2\rho\frac{\sigma\eta}{a^2b}\Psi(a,b,\Delta) + 2\rho\frac{\sigma\eta}{ab}\partial_a\Psi(a,b,\Delta) \\
\partial_b V(\Delta) = & -\frac{2\eta^2}{b^3}\Psi(b,b,\Delta) + \frac{\eta^2}{b^2}\partial_b\Psi(b,b,\Delta) \\
& - 2\rho\frac{\sigma\eta}{ab^2}\Psi(a,b,\Delta) + 2\rho\frac{\sigma\eta}{ab}\partial_b\Psi(a,b,\Delta) \\
\partial_\sigma V(\Delta) = & \frac{2\sigma}{a^2}\Psi(a,a,\Delta) + 2\rho\frac{\eta}{ab}\Psi(a,b,\Delta) \\
\partial_\sigma V(\Delta) = & \frac{2\eta}{b^2}\Psi(b,b,\Delta) + 2\rho\frac{\sigma}{ab}\Psi(a,b,\Delta) \\
\partial_\rho V(\Delta) = & 2\frac{\sigma\eta}{ab}\Psi(a,b,\Delta)
\end{aligned}$$

[1] the dependency on a, b, σ, η, ρ is omitted.

From this last result it is possible to derive $\Phi_d A$ for every G2++ parameter $\theta \in \{a, b, \sigma, \eta, \rho\}$:

$$\begin{aligned}\partial_\theta \Phi_d(t,T)A(t,T) &= \partial_\theta \frac{P_{OIS}(0,T)}{P_{OIS}(0,t)} exp(\frac{V(t)-V(T)+V(T-t)}{2}) \\ &= \frac{P_{OIS}(0,T)}{P_{OIS}(0,t)} exp(\frac{V(t)-V(T)+V(T-t)}{2}) \frac{\partial_\theta V(t) - \partial_\theta V(T) + \partial_\theta V(T-t)}{2} \\ &= \Phi_d(t,T)A(t,T)W_\theta(t,T)\end{aligned}$$

where

$$W_\theta(t,T) := \frac{\partial_\theta V(t) - \partial_\theta V(T) + \partial_\theta V(T-t)}{2} \tag{A.14}$$

An important remark should be made with respect to the shift ratio $\Phi_d(t,T)/\Phi_F(t,T)$: since it holds that

$$\begin{aligned}\Phi_d(t,T) &= \frac{P_{OIS}(t,T)}{\overline{P}(t,T)} \\ &= \frac{P_{OIS}(t_0,T)}{P_{OIS}(t_0,t)} \frac{\overline{P}(t_0,t)}{\overline{P}(t_0,T)}\end{aligned}$$

and analogously, for $\Phi_F(t,T)$, using the forward market curve $P_F(t_0,T)$, the shift ratio becomes

$$\frac{\Phi_d(t,T)}{\Phi_F(t,T)} = \frac{P_{OIS}(t_0,T)}{P_{OIS}(t_0,t)} \frac{P_{FWD}(t_0,t)}{P_{FWD}(t_0,T)}$$

Hence, this term depends only on market data, and does not rely on the financial model chosen.

A.2 Conditional variables

It is time to consider the variables generated by the change of probability in the forward measure $T = T_\alpha$:

$$M_x^T(0,t) = (\frac{\sigma^2}{a^2} + \rho\frac{\sigma\eta}{ab})[1-e^{-at}] - \frac{\sigma^2}{2a^2}[e^{-a(T-t)} - e^{-a(T+t)}] - \rho\frac{\sigma\eta}{b(a+b)}[e^{-b(T-t)} - e^{-bT-at}]$$

$$M_y^T(0,t) = (\frac{\eta^2}{b^2} + \rho\frac{\sigma\eta}{ab})[1-e^{-bt}] - \frac{\eta^2}{2b^2}[e^{-b(T-t)} - e^{-b(T+t)}] - \rho\frac{\sigma\eta}{a(a+b)}[e^{-a(T-t)} - e^{-aT-bt}]$$

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

$$\sigma_y = \eta\sqrt{\frac{1-e^{-2bt}}{2b}} = \eta\sqrt{\omega(b,b,T)}$$

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

In particular, we usually consider the mean terms at the expiry T itself:

$$\begin{aligned}
M_x^T(0,T) &= (\frac{\sigma^2}{a^2} + \rho\frac{\sigma\eta}{ab})[1 - e^{-aT}] - \frac{\sigma^2}{2a^2}[1 - e^{-2aT}] - \rho\frac{\sigma\eta}{b(a+b)}[1 - e^{-(a+b)T}] \\
&= (\frac{\sigma^2}{a} + \rho\frac{\sigma\eta}{b})B(a,T) - \frac{\sigma^2}{a}\omega(a,a,T) - \rho\frac{\sigma\eta}{b}\omega(a,b,T) \\
M_y^T(0,T) &= (\frac{\eta^2}{b^2} + \rho\frac{\sigma\eta}{ab})[1 - e^{-bT}] - \frac{\eta^2}{2b^2}[1 - e^{-2bT}] - \rho\frac{\sigma\eta}{a(a+b)}[1 - e^{-(a+b)T}] \\
&= (\frac{\eta^2}{b} + \rho\frac{\sigma\eta}{a})B(b,T) - \frac{\eta^2}{b}\omega(b,b,T) - \rho\frac{\sigma\eta}{a}\omega(a,b,T)
\end{aligned}$$

Simple computations allow to find their derivatives;

$$\begin{aligned}
\partial_a M_x^T(0,T) &= (\frac{\sigma^2}{a} + \rho\frac{\sigma\eta}{b})\widehat{B}(a,T) - \frac{\sigma^2}{a^2}B(a,T) - 2\frac{\sigma^2}{a}\widehat{\omega}(a,a,T) + \frac{\sigma^2}{a^2}\omega(a,a,T) - \rho\frac{\sigma\eta}{b}\widehat{\omega}(a,b,T) \\
\partial_b M_x^T(0,T) &= -\rho\frac{\sigma\eta}{b^2}B(a,T) + \rho\frac{\sigma\eta}{b^2}\omega(a,b,T) - \rho\frac{\sigma\eta}{b}\widehat{\omega}(a,b,T) \\
\partial_\sigma M_x^T(0,T) &= (2\frac{\sigma}{a} + \rho\frac{\eta}{b})B(a,T) - 2\frac{\sigma}{a}\omega(a,a,T) - \rho\frac{\eta}{b}\widehat{\omega}(a,b,T) \\
\partial_\eta M_x^T(0,T) &= \rho\frac{\sigma}{b}(B(a,T) - \omega(a,b,T)) \\
\partial_\rho M_x^T(0,T) &= \frac{\sigma\eta}{b}(B(a,T) - \omega(a,b,T))
\end{aligned}$$

The derivatives of $M_y^T(0,T)$ are computed in the same manner.
For what concerns the variance and correlation terms σ_x, σ_y and ρ_{xy}:[2]

$$\begin{aligned}
\partial_a\sigma_x = \sigma\ \partial_a\big(\sqrt{\omega(a,a,T)}\big) &= \sigma\frac{\widehat{\omega}(a,a,T)}{\sqrt{\omega(a,a,T)}} = \frac{\sigma^2}{\sigma_x}\widehat{\omega}(a,a,T) \\
\partial_b\sigma_y = \eta\ \partial_b\big(\sqrt{\omega(b,b,T)}\big) &= \eta\frac{\widehat{\omega}(b,b,T)}{\sqrt{\omega(b,b,T)}} = \frac{\eta^2}{\sigma_y}\widehat{\omega}(b,b,T) \\
\partial_\sigma\sigma_x &= \sqrt{\omega(a,a,T)} \\
\partial_\eta\sigma_y &= \sqrt{\omega(b,b,T)}
\end{aligned}$$

[2]the missing derivatives are equal to 0.

$$\partial_a \rho_{xy} = \rho \frac{\sigma^3 \eta}{\sigma_x^3 \sigma_y} \omega(a,b,T)\ \widehat{\omega}(a,a,T) + \rho \frac{\sigma\eta}{\sigma_x \sigma_y} \widehat{\omega}(a,b,T)$$

$$\partial_b \rho_{xy} = \rho \frac{\sigma \eta^3}{\sigma_x \sigma_y^3} \omega(a,b,T)\ \widehat{\omega}(b,b,T) + \rho \frac{\sigma\eta}{\sigma_x \sigma_y} \widehat{\omega}(a,b,T)$$

$$\partial_\sigma \rho_{xy} = \rho_{xy}\left(\frac{1}{\sigma} - \frac{\partial_\sigma \sigma_x}{\sigma}\right)$$

$$\partial_\eta \rho_{xy} = \rho_{xy}\left(\frac{1}{\eta} - \frac{\partial_\eta \sigma_y}{\eta}\right)$$

$$\partial_\rho \rho_{xy} = \frac{\rho_{xy}}{\rho}$$

A.3 Integration nodes

The Gauss-Hermite nodes $\widehat{x}_k$ are shifted into the actual integration nodes according to

$$x_k = \sqrt{2}\sigma_x \widehat{x}_k + \mu_x$$

where $\mu_x = -M_x^T(0,T)$. Simply, their derivatives become

$$\partial_a x_k = \sqrt{2}\partial_a \sigma_x \widehat{x}_k + \partial_a m u_x$$
$$\partial_b x_k = \partial_b \mu_x$$
$$\partial_\sigma x_k = \sqrt{2}\partial_\sigma \sigma_x \widehat{x}_k + \partial_\sigma \mu_x$$
$$\partial_\eta x_k = \partial_\eta \mu_x$$
$$\partial_\rho x_k = \partial_\rho \mu_x$$

A.4 coefficients c_j and root search

Thanks to the previous results, it is possible for us to compute the derivatives with respect to the coefficients c_j; another important thing to keep in mind is that K is the ATM strike of the underlying swap at the reference date: this means that all future cash flows are not influenced by the financial model considered, but only on the market curves. Hence, strikes do not change by choosing the Vasicek model instead of the G2++, or even a completely different one. Obviously, the year fractions τ_i are independent from the model parameters.
As a consequence, the following holds:

$$\partial_\theta d_i^1 = \frac{\Phi_d(S_{i-1}, S_i)}{\Phi_F(S_{i-1}, S_i)} \partial_\theta\big(\Phi_d(T_\alpha, S_{i-1}) A(T_\alpha, S_{i-1})\big) \qquad i = \overline{\alpha}+1, ..., \overline{\beta}$$

$$\partial_\theta d_i^2 = -\partial_\theta\big(\Phi_d(T_\alpha, S_i) A(T_\alpha, S_i)\big) \qquad i = \overline{\alpha}+1, ..., \overline{\beta}$$

$$\partial_\theta d_i^3 = -K\tau_i \partial_\theta\big(\Phi_d(T_\alpha, T_i) A(T_\alpha, T_i)\big) \qquad i = \alpha+1, ..., \beta$$

in other terms:

$$\partial_\theta c_j = c_j W_\theta(T_\alpha, T_j)$$

For the next step it is necessary to consider the root $\overline{y}(x)$.
Recalling that $\Theta = \{a, b, \sigma, \eta, \rho\}$, let us define:[3]

$$F(\Theta, x, y) = \sum_j c_j e^{-B(a,t_j-T_\alpha)x - B(b,t_j-T_\alpha)y} \tag{A.15}$$

then, $\overline{y}$ is the only value for which $F(\Theta, x, \overline{y}) = 0$.
In order to compute the derivative of $\overline{y}$ with respect to $\theta \in \Theta$, we need to consider Dini theorem in a multidimensional case.

Theorem A.4.1 (Dini theorem)**.**

Let $\underline{F}(\underline{x}, \underline{y})$ a function defined on an open set $\Omega \in \mathbb{R}^{m+n}$ with values in $\mathbb{R}^m$. By convention, $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^m$, i.e. $\underline{x} = (x_1, \ldots, x_n)$, $\underline{y} = (y_1, \ldots, y_m)$. Let $(\underline{x}_0, \underline{y}_0) \in \Omega$ such that $\underline{F}(\underline{x}_0, \underline{y}_0) = 0$.
If $\underline{F} : \Omega \to \mathbb{R}^m$ is C^1 in Ω and

$$det \frac{\partial(F_1, \ldots, F_m)}{\partial(y_1, \ldots, y_m)}(\underline{x}_0, \underline{y}_0) \neq 0 \tag{A.16}$$

Then there exist a neighborhood U of $\underline{x}_0$ in $\mathbb{R}^n$ and a neighborhood V of $\underline{y}_0$ in $\mathbb{R}^m$, with $U \times V \in \Omega$, in which $\forall \underline{x} \in U \ \exists! \ \underline{y} = f(\underline{x}) \in V$ such that $\underline{F}(\underline{x}, f(\underline{x})) = 0$ and $f(\underline{x}_0) = \underline{y}_0$. Moreover, $f \in C^1$ in U and for any $\underline{x} \in U$ its Jacobian is

$$\mathbf{J}_{\underline{f}}(\underline{x}) = -\left(\frac{\partial(F_1, \ldots, F_m)}{\partial(y_1, \ldots, y_m)}(\underline{x}, f(\underline{x}))\right)^{-1} \frac{\partial(F_1, \ldots, F_m)}{\partial(x_1, \ldots, x_n)}(\underline{x}, f(\underline{x})) \tag{A.17}$$

In our case, $m = 1$ and we can consider that $\underline{x}$ contains also Θ. The condition (A.16) becomes $\partial_y \underline{F}(\underline{x}_0, y_0) \neq 0$.
The implicit function f assumes real values and its gradient, according to (A.17), has the expression

$$\nabla f(\underline{x}) = -\frac{1}{\partial_y F(\underline{x}, f(\underline{x}))}\left(\partial_{x_1} F(\underline{x}, f(\underline{x})), \ldots, \partial_{x_n} F(\underline{x}, f(\underline{x}))\right) \tag{A.18}$$

Hence

$$\frac{\partial f}{\partial x_j}(\underline{x}) = -\frac{\frac{\partial F}{\partial x_j}(\underline{x}, f(\underline{x}))}{\frac{\partial F}{\partial y}(\underline{x}, f(\underline{x}))} \tag{A.19}$$

[3] once again, it is clear that $\overline{y}$ is not just depending on x, but also on the whole set Θ; however, since it is usually computed once the network (and so the parameters) is fixed, the only term changing its value is x. For this reason, the dependency on Θ is usually omitted.

Now, considering (A.15), it is easy to see that $F \in C^\infty$ in its domain; moreover[4]

$$\partial_y F(\Theta, x, y) = -\sum_j c_j B_j(b) e^{-B_j(a)x - B_j(b)y}$$

and $\partial_y F(\Theta, x, \overline{y}) \neq 0$. This means that the Theorem holds and that it is possible to compute $\partial_\theta \overline{y}(x)$.
For the sake of completeness the derivatives of F with respect to Θ are presented:

$$\partial_a F(\Theta, x, y) = \sum_j c_j e^{-B_j(a)x - B_j(b)y} \left[W_a(T_\alpha, t_j) - \widehat{B}_j(a)x - B_j(a)\partial_a x \right]$$

$$\partial_b F(\Theta, x, y) = \sum_j c_j e^{-B_j(a)x - B_j(b)y} \left[W_b(T_\alpha, t_j) - B_j(a)\partial_b x - \widehat{B}_j(b)y \right]$$

$$\partial_\sigma F(\Theta, x, y) = \sum_j c_j e^{-B_j(a)x - B_j(b)y} \left[W_\sigma(T_\alpha, t_j) - B_j(a)\partial_\sigma x \right]$$

$$\partial_\eta F(\Theta, x, y) = \sum_j c_j e^{-B_j(a)x - B_j(b)y} \left[W_\eta(T_\alpha, t_j) - B_j(a)\partial_\eta x \right]$$

$$\partial_\rho F(\Theta, x, y) = \sum_j c_j e^{-B_j(a)x - B_j(b)y} \left[W_\rho(T_\alpha, t_j) - B_j(a)\partial_\rho x \right]$$

A.5 $\overline{h}$ and h_j

It is time to consider $\overline{h}$ and its derivatives:
from Formula (A.4)

$$\overline{h} = \frac{\overline{y}(x) - \mu_y}{\sigma_y \sqrt{1 - \rho_{xy}^2}} - \frac{\rho_{xy}(x - \mu_x)}{\sigma_x \sqrt{1 - \rho_{xy}^2}}$$

recall that σ_y depends only on b and η; moreover, when considering Hermite-Gauss nodes, we had in (A.2):

$$x_k = \sqrt{2}\sigma_x \widehat{x}_k + \mu_k$$

As a consequence, we can simplify the previous Formula as follows

$$\overline{h} = \frac{\overline{y}(x_k) - \mu_y}{\sigma_y \sqrt{1 - \rho_{xy}^2}} - \sqrt{2}\widehat{x}_k \frac{\rho_{xy}}{\sqrt{1 - \rho_{xy}^2}} \tag{A.20}$$

[4]in order to make things easier to read, the notation for $B(z, t_j - T_\alpha)$ will be shortened to $B_j(z)$.

Its derivatives are:

$$\partial_a \overline{h} = \frac{\partial_a \overline{y}(x_k) - \partial_a \mu_y}{\sigma_y \sqrt{1-\rho_{xy}^2}} + \frac{\partial_a \rho_{xy}}{(1-\rho_{xy}^2)^{3/2}} \big(\rho_{xy} \frac{\overline{y}(x_k) - \mu_y}{\sigma_y} - \sqrt{2}\widehat{x}_k\big)$$

$$\partial_b \overline{h} = \frac{\partial_b \overline{y}(x_k) - \partial_b \mu_y}{\sigma_y \sqrt{1-\rho_{xy}^2}} + \frac{\overline{y}(x_k) - \mu_y}{\sigma_y \sqrt{1-\rho_{xy}^2}} \frac{\partial_b \sigma_y}{\sigma_y} + \frac{\partial_b \rho_{xy}}{(1-\rho_{xy}^2)^{3/2}} \big(\rho_{xy} \frac{\overline{y}(x_k) - \mu_y}{\sigma_y} - \sqrt{2}\widehat{x}_k\big)$$

$$\partial_\sigma \overline{h} = \frac{\partial_\sigma \overline{y}(x_k) - \partial_\sigma \mu_y}{\sigma_y \sqrt{1-\rho_{xy}^2}} + \frac{\partial_\sigma \rho_{xy}}{(1-\rho_{xy}^2)^{3/2}} \big(\rho_{xy} \frac{\overline{y}(x_k) - \mu_y}{\sigma_y} - \sqrt{2}\widehat{x}_k\big)$$

$$\partial_\eta \overline{h} = \frac{\partial_\eta \overline{y}(x_k) - \partial_\eta \mu_y}{\sigma_y \sqrt{1-\rho_{xy}^2}} + \frac{\overline{y}(x_k) - \mu_y}{\sigma_y \sqrt{1-\rho_{xy}^2}} \frac{\partial_\eta \sigma_y}{\sigma_y} + \frac{\partial_\eta \rho_{xy}}{(1-\rho_{xy}^2)^{3/2}} \big(\rho_{xy} \frac{\overline{y}(x_k) - \mu_y}{\sigma_y} - \sqrt{2}\widehat{x}_k\big)$$

$$\partial_\rho \overline{h} = \frac{\partial_\rho \overline{y}(x_k) - \partial_\rho \mu_y}{\sigma_y \sqrt{1-\rho_{xy}^2}} + \frac{\partial_\rho \rho_{xy}}{(1-\rho_{xy}^2)^{3/2}} \big(\rho_{xy} \frac{\overline{y}(x_k) - \mu_y}{\sigma_y} - \sqrt{2}\widehat{x}_k\big)$$

From the other hand, considering formula (A.3), the derivatives of $h_j(x_k)$ are:

$$\partial_a h_j(x_k) = \partial_a \overline{h} - B_j(b)\sigma_y \frac{\rho_{xy}}{\sqrt{1-\rho_{xy}^2}} \partial_a \rho_{xy}$$

$$\partial_b h_j(x_k) = \partial_b \overline{h} + \big(\widehat{B}_j(b)\sigma_y + B_j(b)\partial_b \sigma_y\big)\sqrt{1-\rho_{xy}^2} - B_j(b)\sigma_y \frac{\rho_{xy}}{\sqrt{1-\rho_{xy}^2}} \partial_b \rho_{xy}$$

$$\partial_\sigma h_j(x_k) = \partial_\sigma \overline{h} - B_j(b)\sigma_y \frac{\rho_{xy}}{\sqrt{1-\rho_{xy}^2}} \partial_\sigma \rho_{xy}$$

$$\partial_\eta h_j(x_k) = \partial_\eta \overline{h} + B_j(b)\partial_\eta \sigma_y \sqrt{1-\rho_{xy}^2} - B_j(b)\sigma_y \frac{\rho_{xy}}{\sqrt{1-\rho_{xy}^2}} \partial_\eta \rho_{xy}$$

$$\partial_\rho h_j(x_k) = \partial_\rho \overline{h} - B_j(b)\sigma_y \frac{\rho_{xy}}{\sqrt{1-\rho_{xy}^2}} \partial_\rho \rho_{xy}$$

A.6 λ_j

The computations of the derivatives of λ_j are simpler than the last one presented; indeed its definition is

$$\lambda_j(x) = c_j e^{-B_j(a)x}$$

Hence, it is easy to obtain

$$\begin{aligned}
\partial_a \lambda_j(x) &= \lambda_j(x)\Big[W_a(T_\alpha, t_j) - \widehat{B}_j(a)x - B_j(a)\partial_a x\Big] \\
\partial_b \lambda_j &= \lambda_j\Big[W_b(T_\alpha, t_j) - B_j(a)\partial_b x\Big] \\
\partial_\sigma \lambda_j(x) &= \lambda_j(x)\Big[W_\sigma(T_\alpha, t_j) - B_j(a)\partial_\sigma x\Big] \\
\partial_\eta \lambda_j(x) &= \lambda_j(x)\Big[W_\eta(T_\alpha, t_j) - B_j(a)\partial_\eta x\Big] \\
\partial_\rho \lambda_j(x) &= \lambda_j(x)\Big[W_\rho(T_\alpha, t_j) - B_j(a)\partial_\rho x\Big]
\end{aligned}$$

A.7 κ_j

The last element we have to consider is $\kappa_j(x_k)$. Again, considering Equation (A.2) it can be simplified in:

$$\kappa_j(x_k) = -B_j(b)\Big[\mu_y - \frac{1}{2}(1-\rho_{xy}^2)\sigma_y^2 B_j(b) + \sqrt{2}\widehat{x}_k\sigma_y\rho_{xy}\Big] \tag{A.21}$$

As a consequence, the derivatives are as follows:

$$\begin{aligned}
\partial_a \kappa_j =& - B_j(b)\Big[\partial_a\mu_y + \rho_{xy}\sigma_y^2 B_j(b)\partial_a\rho xy + \sqrt{2}\widehat{x}_k\sigma_y\partial_a\rho_{xy}\Big] \\
\partial_b \kappa_j =& - \widehat{B}_j(b)\Big[\mu_y - \frac{1}{2}(1-\rho_{xy}^2)\sigma_y^2 B_j(b) + \sqrt{2}\widehat{x}_k\sigma_y\rho_{xy}\Big] \\
& - B_j(b)\Big[\partial_b\mu_y - (1-\rho_{xy}^2)\big(\widehat{B}_j(b)\frac{\sigma_y^2}{2} + B_j(b)\sigma_y\partial_b\sigma_y\big) \\
& \qquad + \rho_{xy}\sigma_y^2 B_j(b)\partial_b\rho xy + \sqrt{2}\widehat{x}_k\sigma_y\partial_b\rho_{xy}\Big] \\
\partial_\sigma \kappa_j =& - B_j(b)\Big[\partial_\sigma\mu_y + \rho_{xy}\sigma_y^2 B_j(b)\partial_\sigma\rho xy + \sqrt{2}\widehat{x}_k\sigma_y\partial_\sigma\rho_{xy}\Big] \\
\partial_\eta \kappa_j =& - B_j(b)\Big[\partial_\eta\mu_y - (1-\rho_{xy}^2)B_j(b)\sigma_y\partial_\eta\sigma_y + \rho_{xy}\sigma_y^2 B_j(b)\partial_\eta\rho xy + \sqrt{2}\widehat{x}_k\sigma_y\partial_\eta\rho_{xy}\Big] \\
\partial_\rho \kappa_j =& - B_j(b)\Big[\partial_\rho\mu_y + \rho_{xy}\sigma_y^2 B_j(b)\partial_\rho\rho xy + \sqrt{2}\widehat{x}_k\sigma_y\partial_\rho\rho_{xy}\Big]
\end{aligned}$$

A.8 Final formulas

Finally, all the single blocks have been considered; the only thing left is putting everything together to find the final derivatives of the prices with respect to G2++ parameters. Let's recall once again the pricing formula in a shortened format:

$$SO(0, T_\alpha, \Gamma, K) = \frac{P_d(0, T_\alpha)}{\sqrt{\pi}} \sum_{k=1}^{n} \sum_{j \in \Gamma} \omega_k \Psi_{jk} \tag{A.22}$$

where

$$\Psi_{jk} = \lambda_j(x_k) e^{\kappa_j(x_k)} \Phi[-h_j(x_k)] \tag{A.23}$$

Consequently, since $P_d(0, T_\alpha)$ does not depend on the model, we obtain directly:

$$\partial_\theta SO(0, T_\alpha, \Gamma, K) = \frac{P_d(0, T_\alpha)}{\sqrt{\pi}} \sum_{k} \sum_{j} \omega_k \partial_\theta \Psi_{jk} \tag{A.24}$$

where

$$\begin{aligned} \partial_\theta \Psi_{jk} = \quad & \partial_\theta \lambda_j(x_k) e^{\kappa_j(x_k)} \Phi[-h_j(x_k)] \\ & + \lambda_j(x_k) e^{\kappa_j(x_k)} \partial_\theta \kappa_j(x_k) \Phi[-h_j(x_k)] \\ & - \lambda_j(x_k) e^{\kappa_j(x_k)} f(h_j(x_k)) \partial_\theta h_j(x_k) \end{aligned} \tag{A.25}$$

and $f(\cdot)$ denotes the Gaussian distribution function.

Appendix B

Backpropagation Derivatives

The purpose of this Section is to provide useful formulas to obtain the derivatives of the ANN's weights with respect to the final output.
This task is performing by following backwards the structure of the neural network starting on the final layers: for this reason, it is called *backpropagation*.

Without loss of generality, all the following derivatives have as reference the target a. The weights are classified in the following way:

- the L input features are labelled with $l = 1, \dots, L$;
- the M neurons in the first hidden layer are labelled with $m = 1 \dots, M$; similarly, the N neurons in the second layer are labelled with $n = 1, \dots, N$; the activation function Ψ is meant to be the same for every neuron in both hidden layers.
- the targets are simply labelled with the parameter they are representing as well as the activation function used: the target a has as Ω_a as activation function, σ has Ω_σ and so on.
- the weights have as notation w_{ij}^k: the subscript i refers to the input neuron (0 is meant for the bias), j to the output one; k is a superscript referring to the set of weights it belongs to[1]:
  - "in" refers to the weights between the feature layer and the first hidden layer. This set of weights is defined in matrix form as $\mathbf{W}^{in->1}$; its dimensions are $[L+1; M]$, where the last row is meant for the bias.

[1]For example, the weight $w_{03}^{(1)}$ is the weight connecting the bias of the first hidden layer to the third neuron in the second hidden layer.

  - (1) is used to indicate the weights between the hidden layers. This set of weights is defined as $\mathbf{W}^{1->2}$;
  - (2) denotes the set of weights connecting the neurons of the second hidden layer to the ones in the output layer. This set of weights is referred as $\mathbf{W}^{2->out}$.

- a similar notation is used to refer to the values assumed by the neurons: there is the net value and the activation value, respectively called as net_i^k and act_i^k; as before, i is for the enumeration term of the neuron in the set and k is the label of the layer: "*in*" for the features layer, (1) and (2) for the hidden layers and "*out*" for the output layer[2].

The derivative of a target with respect to a specified weight is obtained thanks to the following formulas.

$\mathbf{W}^{2->out}$

Let's consider the set of weights between the second hidden layer and the output. Recalling that:

$$a = \Omega_a(net_a^{out})$$
$$net_a^{out} = \sum_n act_n^{(2)} w_{na}^{(2)} + w_{0a}^{(2)}$$

the derivative of a with respect to the weights incoming from k-th neuron ($k = 1, \ldots, N$)

[2]remark: the net value is the linear combination of the values incoming, whose coefficients are the weights; the activation value is the result of the application of the activation function on the net value.

in the second hidden layer is computed as follows:

$$\begin{aligned}\frac{\partial a}{\partial w_{ka}^{(2)}} &= \frac{\partial a}{\partial net_a^{out}} \frac{\partial net_a^{out}}{\partial w_{ka}^{(2)}} \\ &= \dot{\Omega}_a(net_a^{out}) \frac{\partial}{\partial w_{ka}^{(2)}} \big(\sum_n act_n^{(2)} w_{na}^{(2)} + w_{0a}^{(2)} \big) \\ &= \dot{\Omega}_a(net_a^{out})\ act_k^{(2)}\end{aligned}$$

$$\begin{aligned}\frac{\partial a}{\partial w_{0a}^{(2)}} &= \frac{\partial a}{\partial net_a^{out}} \frac{\partial net_a^{out}}{\partial w_{0a}^{(2)}} \\ &= \dot{\Omega}_a(net_a^{out}) \frac{\partial}{\partial w_{0a}^{(2)}} \big(\sum_n act_n^{(2)} w_{na}^{(2)} + w_{0a}^{(2)} \big) \\ &= \dot{\Omega}_a(net_a^{out})\end{aligned}$$

Obviously, all the weights whose outcome is different from the neuron related to the target a have null derivative.

$\mathbf{W}^{1->2}$

Again, for the weights between the hidden layers, we consider that:

$$act_k^{(2)} = \Psi(net_k^{(2)})$$

$$net_k^{(2)} = \sum_m act_m^{(1)} w_{mk}^{(1)} + w_{0k}^{(1)}$$

The derivative of a with respect to the weights between the j-th neuron of the first

hidden layer and the k-th in second one ($j = 1, \ldots, M;\ k = 1, \ldots, N$) is:

$$\begin{aligned}
\frac{\partial a}{\partial w_{jk}^{(1)}} &= \frac{\partial a}{\partial net_a^{out}} \frac{\partial net_a^{out}}{\partial act_k^{(2)}} \frac{\partial act_k^{(2)}}{\partial net_k^{(2)}} \frac{\partial net_k^{(2)}}{\partial w_{jk}^{(1)}} \\
&= \dot{\Omega}_a(net_a^{out})\ w_{ka}^{(2)}\ \dot{\Psi}(net_k^{(2)})\ \frac{\partial}{\partial w_{jk}^{(1)}}\Big(\sum_m act_m^{(1)} w_{mk}^{(1)} + w_{0k}^{(1)}\Big) \\
&= \dot{\Omega}_a(net_a^{out})\ w_{ka}^{(2)}\ \dot{\Psi}(net_k^{(2)})\ act_j^{(1)}
\end{aligned}$$

$$\begin{aligned}
\frac{\partial a}{\partial w_{oj}^{(1)}} &= \frac{\partial a}{\partial net_a^{out}} \frac{\partial net_a^{out}}{\partial act_k^{(2)}} \frac{\partial act_k^{(2)}}{\partial net_k^{(2)}} \frac{\partial net_k^{(2)}}{\partial w_{0j}^{(1)}} \\
&= \dot{\Omega}_a(net_a^{out})\ w_{ka}^{(2)}\ \dot{\Psi}(net_k^{(2)})\ \frac{\partial}{\partial w_{0k}^{(1)}}\Big(\sum_m act_m^{(1)} w_{mk}^{(1)} + w_{0k}^{(1)}\Big) \\
&= \dot{\Omega}_a(net_a^{out})\ w_{ka}^{(2)}\ \dot{\Psi}(net_k^{(2)})
\end{aligned}$$

$\mathbf{W}^{in->1}$

Finally, in the first hidden layer:

$$act_j^{(1)} = \Psi(net_j^{(1)})$$

$$net_j^{(1)} = \sum_l x_l w_{lj}^{in} + w_{0j}^{in}$$

where x_l is the l-th input feature.

The derivative of a with respect to the weights from the input layer (feature l) to

the j-th neuron of the first hidden one is:

$$\begin{aligned}
\frac{\partial a}{\partial w_{ij}^{in}} &= \frac{\partial a}{\partial act_j^{(1)}} \frac{\partial act_j^{(1)}}{\partial w_{ij}^{in}} \\
&= \Big(\sum_n \frac{\partial a}{\partial act_n^{(2)}} \, \frac{\partial act_n^{(2)}}{\partial act_j^{(1)}} \Big) \, \frac{\partial act_j^{(1)}}{\partial net_j^{(1)}} \, \frac{\partial net_j^{(1)}}{\partial w_{ij}^{in}} \\
&= \dot{\Omega}_a(net_a^{out}) \, \sum_n \big[w_{na}^{(2)} \, \dot{\Psi}(net_n^{(2)}) \frac{\partial net_n^{(2)}}{\partial act_j^{(1)}} \big] \, \dot{\Psi}(net_j^{(1)}) \, \frac{\partial net_j^{(1)}}{\partial w_{ij}^{in}} \\
&= \dot{\Omega}_a(net_a^{out}) \, \sum_n \big[w_{na}^{(2)} \dot{\Psi}(net_k^{(2)}) w_{jn}^{(1)} \big] \, \dot{\Psi}(net_j^{(1)}) \, \frac{\partial}{\partial w_{ij}^{in}} \big(\sum_l x_l w_{lj}^{in} + w_{0j}^{in} \big) \\
&= \dot{\Omega}_a(net_a^{out}) \, \sum_n \big[w_{jn}^{(1)} w_{na}^{(2)} \dot{\Psi}(net_j^{(1)}) \dot{\Psi}(net_n^{(2)}) \big] \, x_i
\end{aligned}$$

$$\begin{aligned}
\frac{\partial a}{\partial w_{0j}^{in}} &= \frac{\partial a}{\partial act_j^{(1)}} \frac{\partial act_j^{(1)}}{\partial w_{0j}^{in}} \\
&= \Big(\sum_n \frac{\partial a}{\partial act_n^{(2)}} \, \frac{\partial act_n^{(2)}}{\partial act_j^{(1)}} \Big) \, \frac{\partial act_j^{(1)}}{\partial net_j^{(1)}} \, \frac{\partial net_j^{(1)}}{\partial w_{0j}^{in}} \\
&= \dot{\Omega}_a(net_a^{out}) \, \sum_n \big[w_{na}^{(2)} \dot{\Psi}(net_n^{(2)}) \frac{\partial net_n^{(2)}}{\partial act_j^{(1)}} \big] \, \dot{\Psi}(net_j^{(1)}) \, \frac{\partial net_j^{(1)}}{\partial w_{0j}^{in}} \\
&= \dot{\Omega}_a(net_a^{out}) \, \sum_n \big[w_{na}^{(2)} \dot{\Psi}(net_n^{(2)}) w_{jn}^{(1)} \big] \, \dot{\Psi}(net_j^{(1)}) \, \frac{\partial}{\partial w_{0j}^{in}} \big(\sum_l x_l w_{lj}^{in} + w_{0j}^{in} \big) \\
&= \dot{\Omega}_a(net_a^{out}) \, \sum_n \big[w_{jn}^{(1)} w_{na}^{(2)} \dot{\Psi}(net_j^{(1)}) \dot{\Psi}(net_n^{(2)}) \big]
\end{aligned}$$

In particular let's consider what are Ω_θ, Ψ and their partial derivatives in the case of this project:

- Ω_θ is an affine transformation of the hyperbolic tangent; we can generalize in this way

$$\theta = \Omega_\theta(net_\theta^{out}) = A \tanh(net_\theta^{out}) + B$$
$$\dot{\Omega}_\theta(net_\theta^{out}) = A \cdot (1 - \tanh(net_\theta^{out})^2)$$

- Ψ is the so-called *logistic* function:

$$\Psi(x) = \frac{1}{1 + e^{-x}}$$
$$\dot{\Psi}(x) = \Psi(x)\big(1 - \Psi(x)\big)$$

Appendix C

Multicurrency results

This appendix has the purpose of showing the results of the offline calibration in a multi-currency framework. In order not to be too long in the presentation, only the curves of the predicted parameters in the first scenario are presented, as well as the complete presentation of the feedback curves. For the other scenarios, just the most relevant features are shown.

In the final section the feedback of the different scenarios are compared, just to have an idea of the range of errors that can result from the multi-currency calibration.

There is the need to remark that this calibration is done on the subset of couples tenor/-expiry which are common in all the currencies, while the calibration made using Vasicek's model involved all the available swaptions. Hence, in the figures comparing the curves of the predicted feedback and Vasicek's calibration error, this last value is considered just to give a reference point in the plot; it is not possible to compare the two models by looking at the result of these tests, with the exception of the CHF case, where the set of swaptions considered coincides with the traded one.
In section 6.3 the CHF case is considered and analyzed.

C.1 First scenario: predicted curves

Figure C.1: Multi currency, test 1: Mean reversion Speed 1 (a).
With some exception in CHF, the curves are usually between 1.5 and 2.6.
The patterns shown in the CAD and USD curves are similar (to some extent they are also similar to the EUR one).

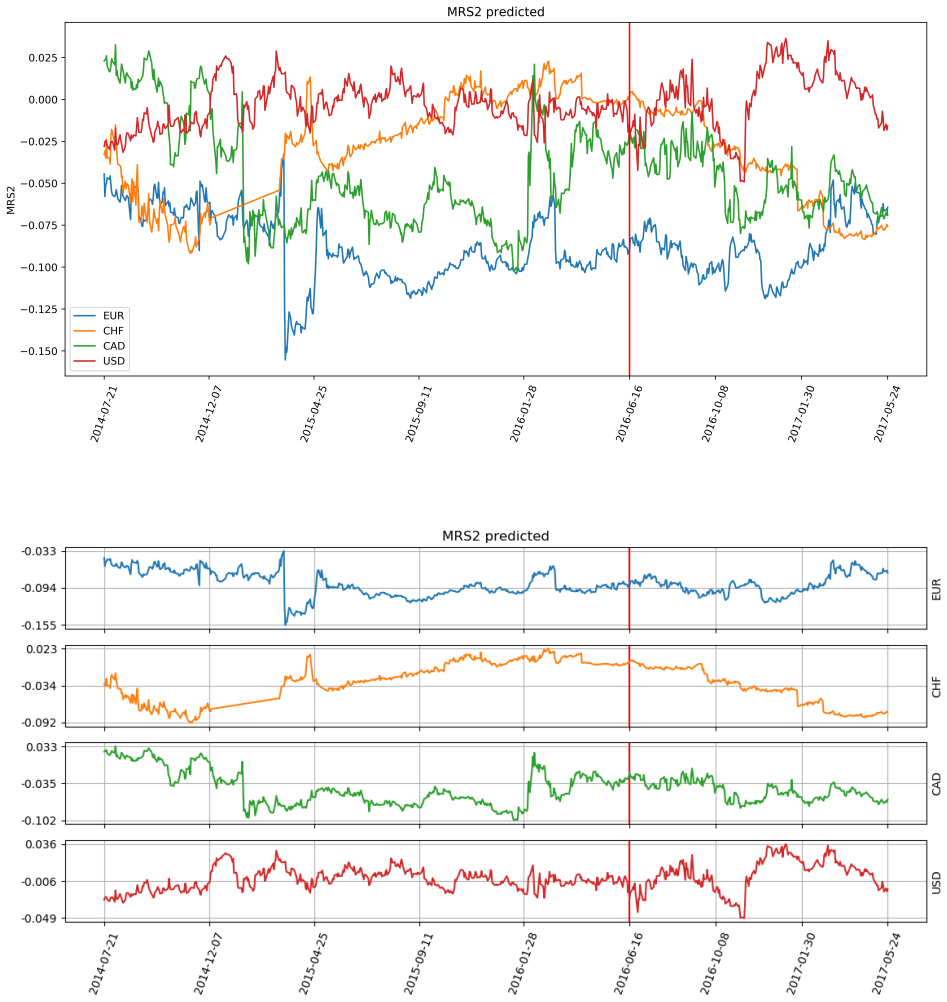


Figure C.2: Multi currency, test 1: Mean reversion Speed 2 (b)
In the majority of the cases, these Mean Reversion Speed assumes negative values, also for the EURO currency.

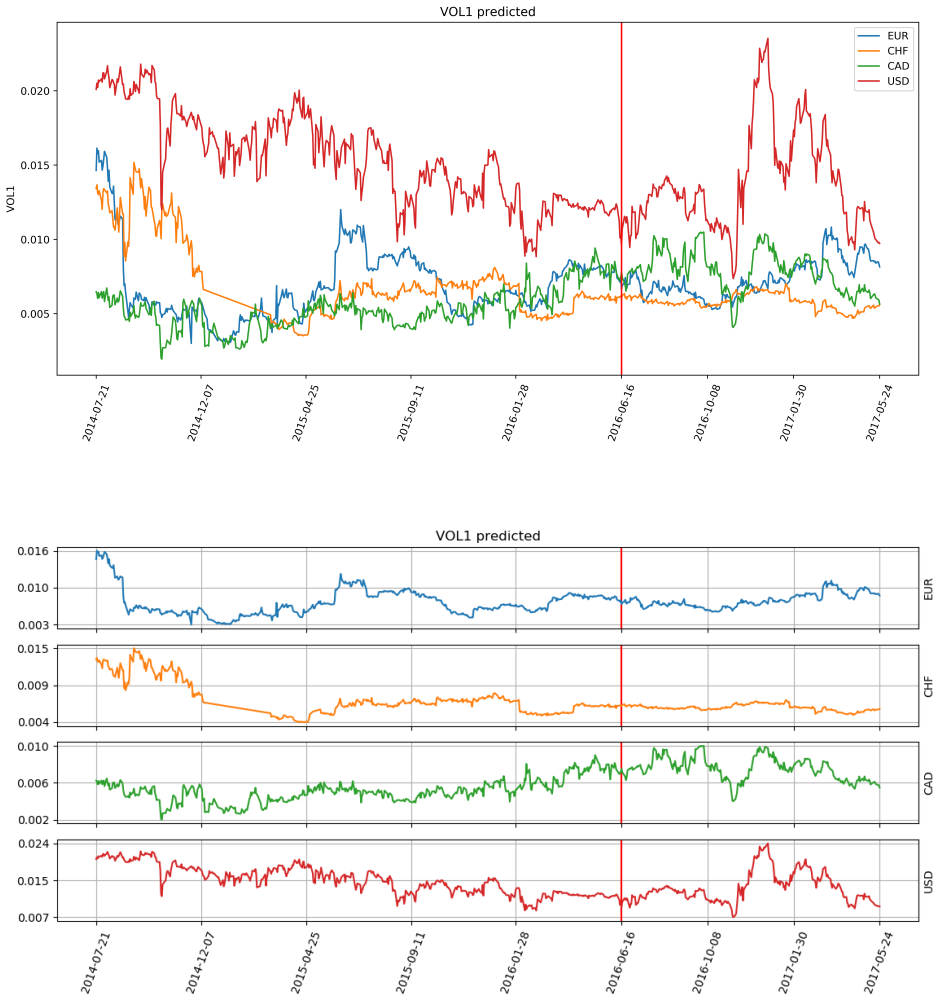


Figure C.3: Multi currency, test 1: Volatility 1 (σ)

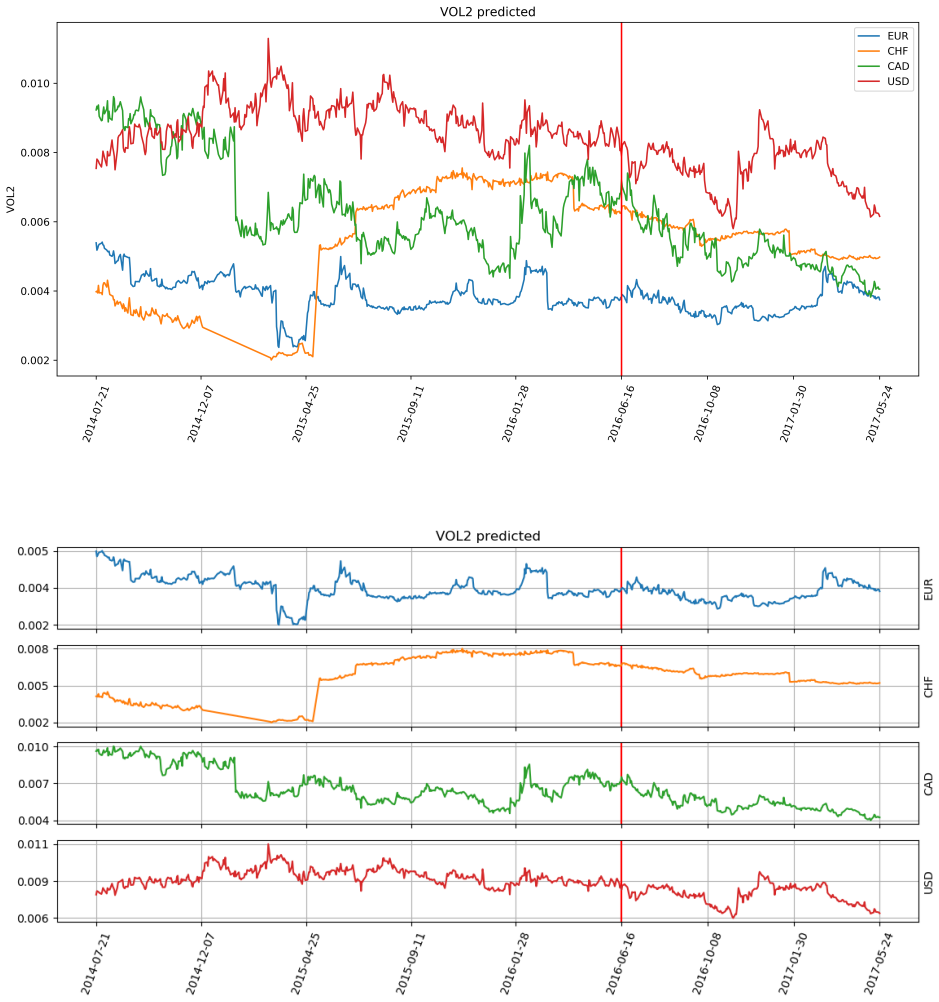


Figure C.4: Multi currency, test 1: Volatility 2 (η)
As far as the volatilities are concerned, the range of values assumed is similar to the one shown in the single currency calibrations.

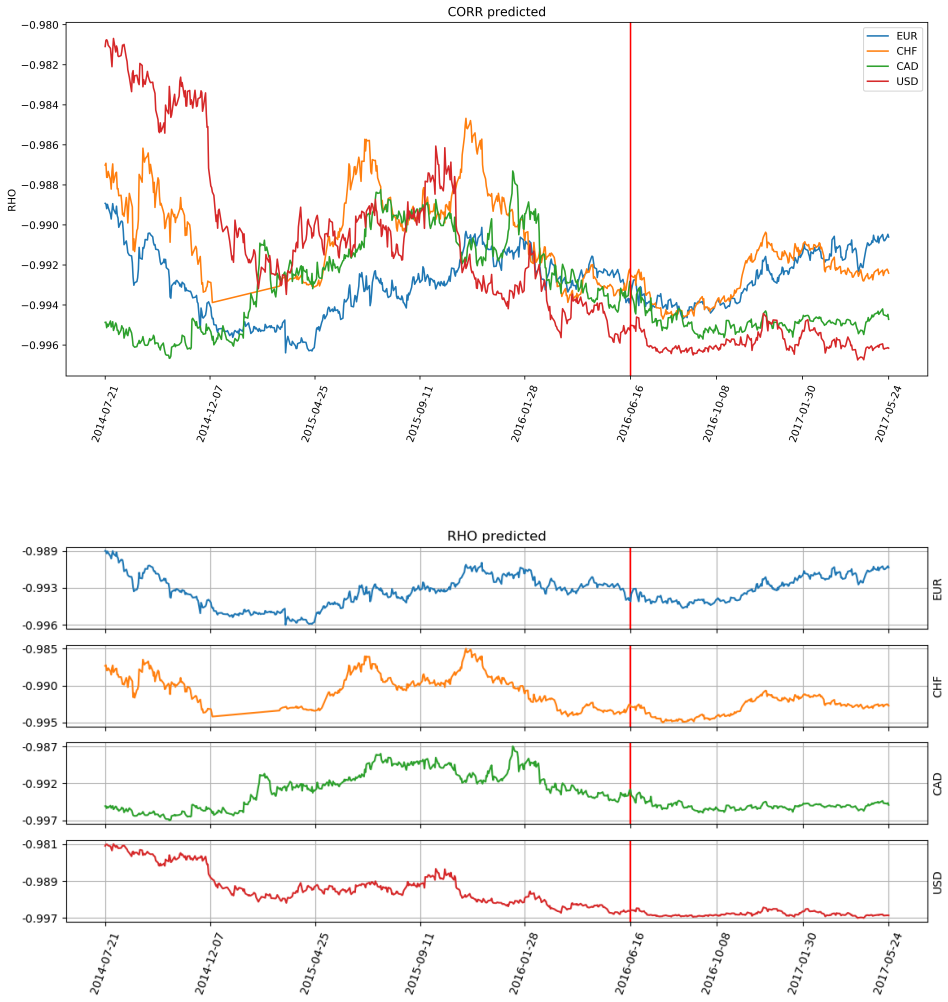


Figure C.5: Multi currency, test 1: Correlation (ρ)
As for the volatilities, the correlations are similar to the single currency case: the Brownian Motions have always a high negative correlation.

C.2 First scenario: feedback curves

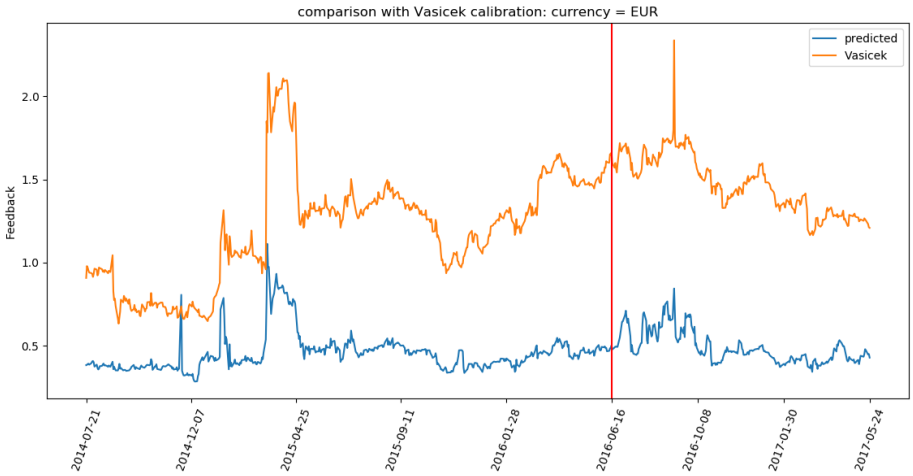


Figure C.6: Multi currency, test 1: EUR feedback function
In this dataset the calibration is very stable, since there are few peaks, and the test phase has still a good performance.

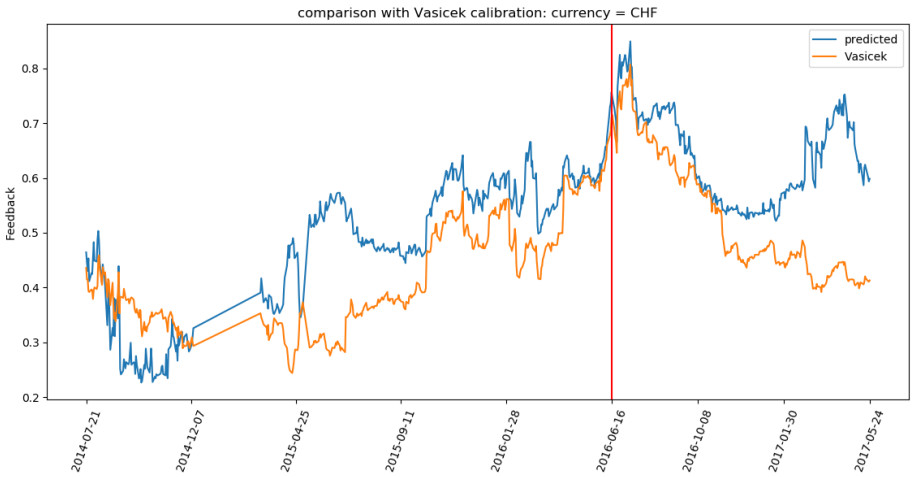


Figure C.7: Multi currency, test 1: CHF feedback function
In this dataset the calibration has a worse performance with respect to the one made using Vasicek's model (single-day calibration).

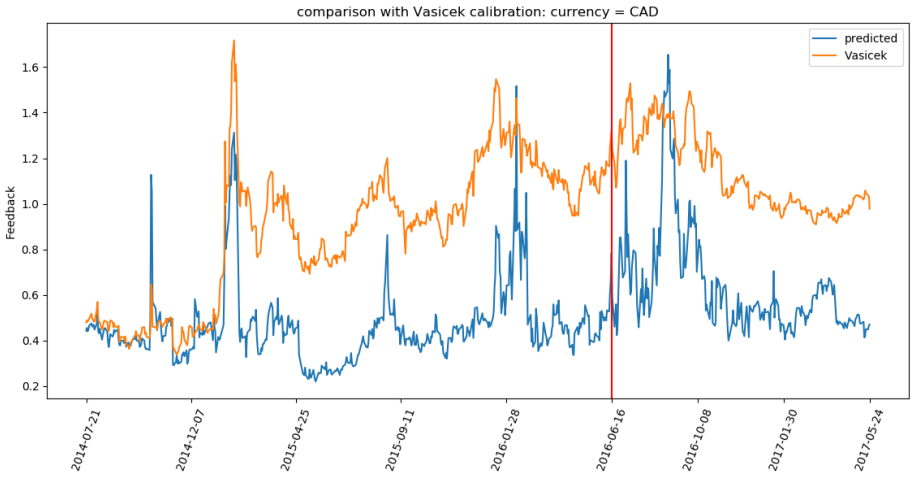


Figure C.8: Multi currency, test 1: CAD feedback function
This seems to be the most unstable currency for what regards the feedback, since there are several peaks in the feedback.

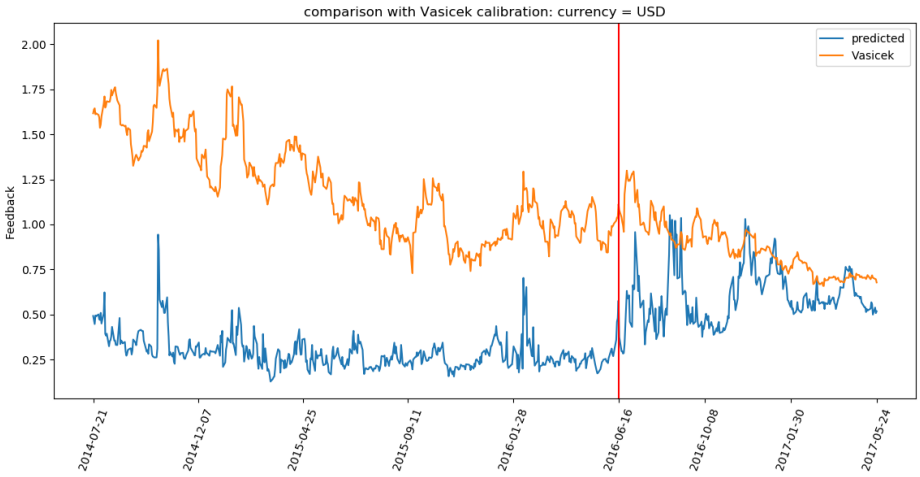


Figure C.9: Multi currency, test 1: USD feedback function
In this curve it is possible to see that the calibration is stable across the training dataset, with worse performances in the test phase.

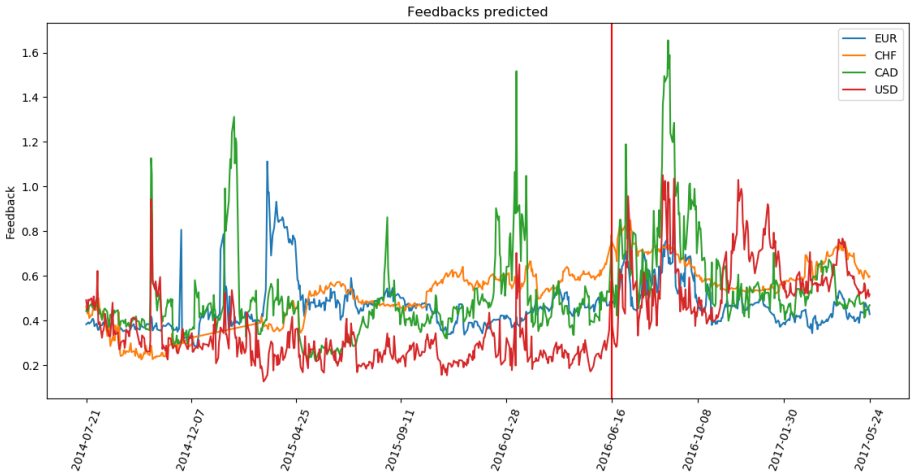


Figure C.10: Multi currency, test 1: comparison of the feedback functions.
It is clear that the feedback curve for all the currencies is usually in the same range of values.
It is also evident that the CAD curve shows the worst peaks.

C.3 Second scenario: predicted curves

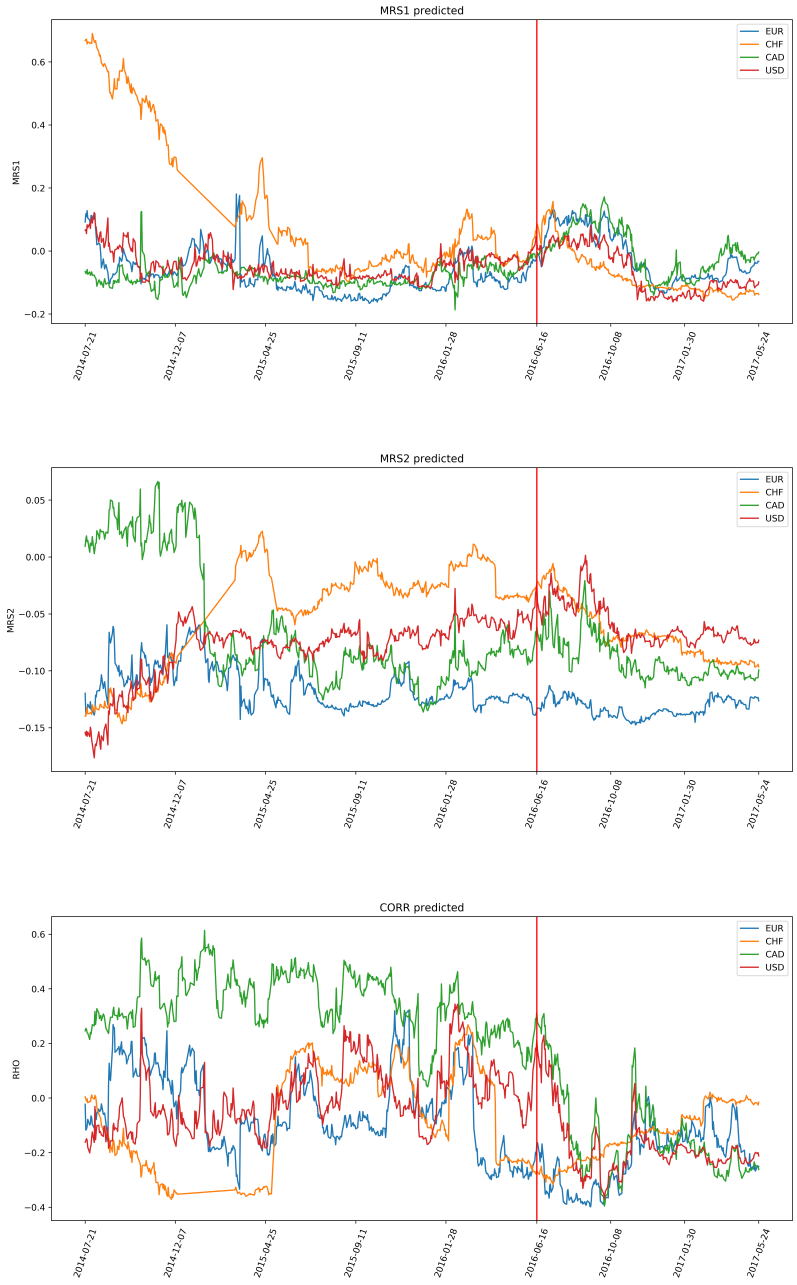


Figure C.11: Multi currency, test 2: Relevant curves of parameters (a, b, ρ)

C.4 Third scenario: predicted curves

Figure C.12: Multi currency, test 3: Relevant curves of parameters (a, σ, ρ)

C.5 Comparison of the scenarios

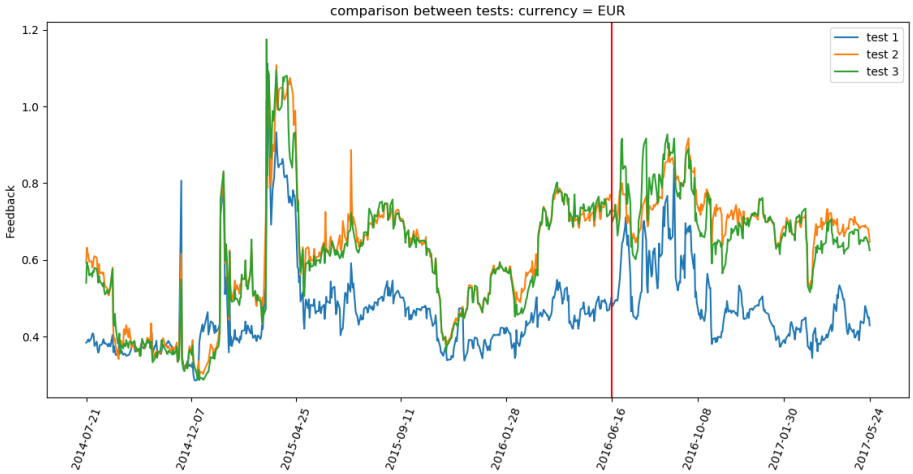


Figure C.13: Multi currency, comparison of the feedback functions for EUR dataset. It is easy to see that the first scenario has the best performance over the others, especially in the test phase.

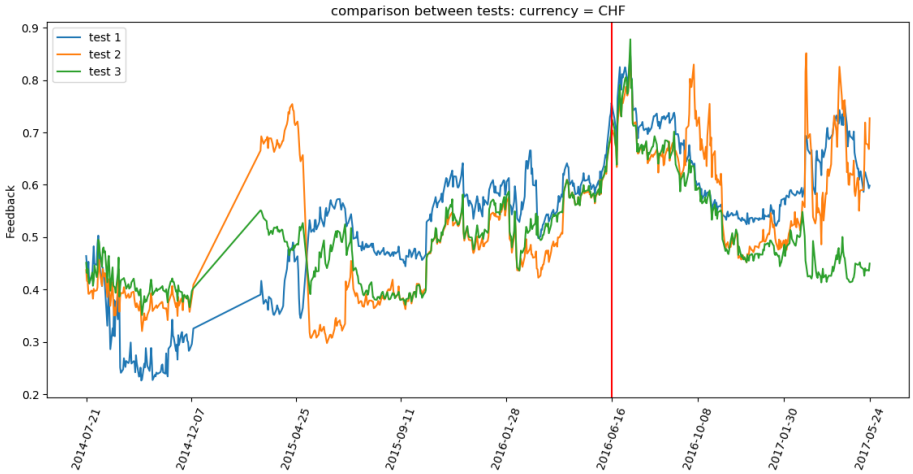


Figure C.14: Multi currency, comparison of the feedback functions for CHF dataset. In this case it is difficult to say which is the best scenario; in the test phase, the third scenario seems to have the lowest feedbacks.

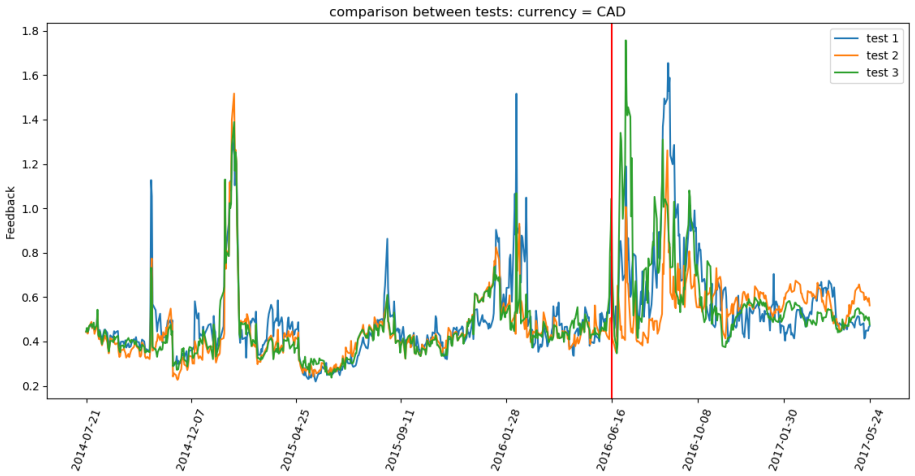


Figure C.15: Multi currency, comparison of the feedback functions for CAD dataset. In this case the curves are very similar, it is not possible to detect the best scenario.

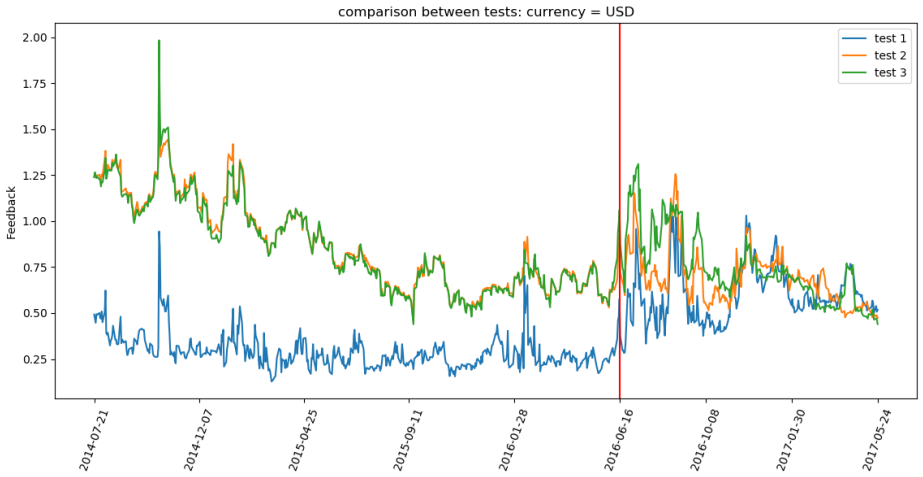


Figure C.16: Multi currency, comparison of the feedback functions for USD dataset. Here it is clear that first scenario is the most able to replicate market prices.

Bibliography

[Alon et al., 2005] Alon, G., Kroese, D. P., Raviv, T., and Rubinstein, R. Y. (2005). Application of the cross-entropy method to the buffer allocation problem in a simulation-based environment. *Annals of Operations Research*, 134(1):137–151.

[Bianchetti, 2008] Bianchetti, M. (2008). Two Curves, One Price :Pricing & Hedging Interest Rate Derivatives Decoupling Forwarding and Discounting Yield Curves. MPRA Paper 22022, University Library of Munich, Germany.

[Bishop, 2009] Bishop, C. (2009). *Pattern Recognition and Machine Learning*. Springer.

[Björk, 2009] Björk, T. (2009). *Arbitrage Theory in Continuous Time*. Oxford University Press.

[Brigo and El-Bachir, 2007] Brigo, D. and El-Bachir, N. (2007). An exact formula for default swaptions' pricing in the ssrjd stochastic intensity model. ICMA Centre Discussion Papers in Finance icma-dp2007-14, Henley Business School, Reading University.

[Brigo and Mercurio, 1998] Brigo, D. and Mercurio, F. (1998). On deterministic shift extensions of short-rate models. Internal Report, Banca IMI, Milan.

[Brigo and Mercurio, 2001] Brigo, D. and Mercurio, F. (2001). *Interest Rate Models - Theory and Practice*. Springer-Verlag.

[C. Fu, 2005] C. Fu, M. (2005). Stochastic gradient estimation. *Handbook on Simulation Optimization, chapter 5*, page 32.

[Cella, 2016] Cella, L. (2015-2016). A supervised learning approach to swaption calibration. Master's thesis, Politecnico di Milano.

[Christensen, 2007] Christensen, J. H. E. (2007). *Default and Recovery Risk Modeling and Estimation*. PhD thesis, Copenhagen Business School.

[Donati, 2018] Donati, A. (2017-2018). Black-box calibration of interest rate models for the pricing of swaptions. Master's thesis, Politecnico di Milano.

[Haykin, 2009] Haykin, S. S. (2009). *Neural networks and learning machines*. Pearson Education, Upper Saddle River, NJ, third edition.

[Hernandez, 2016] Hernandez, A. (2016). Model calibration with neural networks. *SSRN*.

[Hotelling, 1933] Hotelling, H. (1933). *Analysis of a complex of statistical variables into principal components*. Journal of Educational Psychology.

[Hull, 2009] Hull, J. (2009). *Options, Futures and Other Derivatives*. Pearson.

[J.M. Rondinelli and Marks, 2007] J.M. Rondinelli, B. D. and Marks, L. (2007). Enhancing structure relaxations for first-principles codes: an approximate hessian approach. *Comput. Mater. Sci.*, 40:345–353.

[LeCun et al., 1998] LeCun, Y., Bottou, L., Orr, G. B., and Müller, K.-R. (1998). Efficient backprop. In *Neural Networks: Tricks of the Trade, This Book is an Outgrowth of a 1996 NIPS Workshop*, pages 9–50, London, UK, UK. Springer-Verlag.

[McNelis, 2005] McNelis, P. D. (2005). *Neural Networks in Finance*. Academic Press Advanced Finance. Academic Press, Boston.

[Peters and Schaal, 2006] Peters, J. and Schaal, S. (2006). Policy gradient methods for robotics. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Beijing, China.

[Rubinstein, 2004] Rubinstein, Y. Reuven, K. P. D. (2004). *The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation, and Machine Learning*. Springer-Verlag.

[Sanders and Kandrot, 2010] Sanders, J. and Kandrot, E. (2010). *CUDA by Example: An Introduction to General-Purpose GPU Programming*. Addison-Wesley Professional, 1st edition.

[Sender, 2017] Sender, N. A. (2017). Multi-curve bootstrapping and implied discounting curves in illiquid markets. Master's thesis, University of Cape Town.

[Shloof and Salmi Noorani, 2012] Shloof, A. and Salmi Noorani, M. (2012). Halley irrational-homotopy analysis method. *International Journal of Applied Mathematical Research*, 1.

[Witten et al., 2011] Witten, I. H., Frank, E., and Hall, M. A. (2011). *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann, Amsterdam, 3 edition.