POLITECNICO DI MILANO
Corso di Laurea Magistrale in Computer Science and Engineering
Dipartimento di Elettronica e Informazione

BLACK-BOX CALIBRATION OF INTEREST RATE MODELS FOR THE PRICING OF SWAPTIONS

AI & R Lab
Laboratorio di Intelligenza Artificiale
e Robotica del Politecnico di Milano

Relatore: Prof. Marcello Restelli
Correlatore: Dott. Matteo Pirotta

Tesi di Laurea di:
Andrea Donati, matricola 860982

Anno Accademico 2016-2017

Abstract

In finance, pricing models represent the dynamics of interest rates and are used by financial institutions to price different financial instruments. A central consideration for any pricing model is the ability to calibrate the model to current market data, and a second important aspect is the speed at which that calibration can be performed.
We propose a black-box calibration of interest rate models for swaption pricing using machine learning techniques. The advantages of the black-box approach over the traditional one are many, where the most important is the independence between the machine learning model and the interest rate model, so that the latter can be changed without any significant change in the former. The second advantage is the calibration speed, reduced from several seconds to milliseconds, achieved by offloading the computational intensive tasks to an offline training process while the online evaluation can be performed in a considerably shorter time.
Then, we show how to optimize the intensive computation by moving the execution from the CPU to the GPU, achieving a considerable speedup.

Sommario

In finanza, i modelli di pricing rappresentano le dinamiche dei tassi di interesse e sono utilizzati dalle istituzioni finanziarie per valutare diversi strumenti finanziari. Una considerazione centrale per qualsiasi modello di pricing è la capacità di calibrare il modello agli attuali dati di mercato e un secondo aspetto importante è la velocità con cui è possibile eseguire tale calibrazione.
Proponiamo una calibrazione black-box di modelli di tassi di interesse per la valutazione di swaption utilizzando tecniche di machine learning. I vantaggi dell'approccio black-box rispetto a quello tradizionale sono molteplici, in cui il più importante è l'indipendenza tra il modello di apprendimento automatico e il modello di tasso di interesse, in modo che quest'ultimo possa essere modificato o sostituito senza alcun cambiamento significativo nel primo.
Il secondo vantaggio è la velocità di calibrazione, ridotta dall'ordine di diversi secondi a millisecondi, ottenuta delegando le attività computazionali più onerose ad un processo di ottimizzazione che non viene eseguito in tempo reale, mentre la valutazione online può essere eseguita in un tempo notevolmente più breve.
Quindi, mostriamo come ottimizzare i calcoli più critici spostando l'esecuzione dalla CPU alla GPU, ottenendo una notevole accelerazione.

Acknoledgements

Finalmente è arrivato il tanto atteso momento di coronamento della mia carriera universitaria.
Vorrei ringraziare i miei amici, quelli nuovi che mi hanno accompagnato in questa avventura universitaria in una città a me nuova, e gli amici di una vita che non hanno mai smesso di supportarmi.
Vorrei anche ringraziare la mia famiglia che mi ha sempre incoraggiato durante il mio percorso e anche confortato nei momenti di difficoltà.
Ringrazio in modo particolare il professor Marcello Restelli, che mi ha guidato durante tutto il lavoro della tesi fornendomi un esempio da seguire e ottimi insegnamenti di cui farò tesoro per il futuro sia professionale che non.
Ringrazio anche Matteo Pirotta per avermi assistito nella stesura della tesi.
Infine vorrei ringraziare Banca IMI e tutti coloro che mi hanno aiutato nell'affrontare il primo approccio con il mondo della finanza, fornendomi assistenza durante lo sviluppo del progetto.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Nowadays, the derivative market is one of the most pupular markets in the financial world. The reason of the popularity of derivatives is their versatility, in fact they can be used for several purposes.
The derivative is a financial instrument whose value depends on the value of other underlying assets, and its own structure gives the power to create very complex instruments on top of existing assets. This is one of the reasons of their big diffusion. There are different types of derivatives, the ones that are traded on traditional exchanges, and the ones traded *Over The Counter* (OTC), i.e. traded directly between financial institutions like banks and funds. The last one is our case, where we are dealing with swaptions, that are options on swaps and are traded exclusively over the counter.
As we mentioned before, derivatives are very versatile and can be used for different purposes. The main applications are for the discovering of prices of the underlying assets, for risk management or hedging, and of course for speculation. The hedging case is one of the most relevant, in particular for the fact that we are dealing with the swaptions, that can be used for risk management in loans.
In the last years, the derivatives have been demonized for the role they had in the financial crisis of 2008 for the way they have been used, but they can have lots of good applications.

In finance, the ability of pricing financial instruments in short time has become of vital importance, and that is one the reasons that led to the exploration of machine learning methods.
In our work, we perform a black-box optimization in order to calibrate the interest rate model that is used for the pricing of the swaptions. The goal of

the thesis and the problem that we are going to solve are explained in more details in the following section.

This project is been developed with the collaboration of Banca IMI, so we are working on a real-world scenario with data that have been collecetd by the bank from the market.

1.1 The Problem: Swaption Calibration

On every day, the bank has to tune a model that they use to price different financial instruments. This model is obtained from a process that is named calibration, that is an optimization performed starting from current market data. The model characterizes the evolution of the interest rate in time, and it is dependent on the market data relative to that particular day.
The goal of the thesis is to provide a solution for the problem of the swaption calibration.
The calibration of the swaptions is the problem to provide the optimal parameters η of a model M that describes the dynamics of the interest rate from the daily market quotes. This task is performed by the *calibrator* f.
For a specific day, the inputs of the calibrator are the market quotes of a set of swaptions, i.e. prices and volatilities of the swaptions, with the addition of the information about the discounting and forwarding curves.
The outputs of the calibrator are the *calibrated parameters* η of the model M, that in our case for the *Vasicek model* are the mean reversion speed k and the volatility σ.
The model M is the *Vasicek model*, that is defined by the couple of parameters (k, σ) and it is used to model the dynamics of the interest rate in the future. This is a key element in the pricing of the swaptions. In fact, the calibration produces a model representation of the interest rate that could be used to price different instruments and not only the swaptions. This allows a much wider use of the pricing model that is not exclusively restricted to the swaptions.
An important element of the calibration problem is the *feedback function* J, that is a function of the inputs and the outputs of the calibration, that gives a measure of the goodness of the model parameters η in representing the interest rate. This metric of goodness of the model is computed in terms of an error given by the difference between the swaption prices in input and the model prices computed from η.
Thus, the calibrator is a function f that takes in input the market quotes, the discounting and forwarding curves, and produces the calibrated param-

eters η.
The goal of the thesis is to build a model that replaces the calibrator f with a machine learning model that approximates the function f.

1.2 Our Solution

For the swaption calibration problem, we propose a neural network that approximates the calibration function f. The network is trained using a black-box approach by maximizing the goodness of the model parameters η. This solution aims to replace entirely the calibrator in its use: given the market data, it will compute the optimal parameters of the Vasicek model. The most obvious advantage of our solution is the speedup of the calibration, in fact the computation of the calibrated parameters η would require just a multiplication of few small-sized matrices, instead of a full optimization run. In fact, the optimization part, i.e. the training of the network, is moved offline so it can be performed in a separate moment and so the computation times are not a constraint.
Even if the time gain can be important, the real advantage of our solution is that our metodology could be applied to all the interest rate models different and more complex than the Vasicek that is currently used, with no changes in the learning procedure. The only part that would need to be implemented would be the feedback relative to the new model, because the new model would have a different formula to compute the prices of the swaptions, so a different way to compute the error given by the feedback. By replacing the calibrator and using the same network for different interest rate models, we can save a significant amount of time in the development of the original calibrator, that is no more needed.

1.3 Structure of the Document

The thesis is structured as follows.
Chapter 2 provides a general explanation of the swaps and the swaptions, and a more detailed analysis of the feedback function and the calibrator.
Chapter 3 gives an overview of the main machine learning techniques used in our solution.
Chapter 4 provides an analysis of the state of the art for the calibration problem, in particular two machine learning methods are explained.
In Chapter 5 we explain our solution, giving a formal definition of the problem and describing the procedure used for the training of the network.

In Chapter 6 we provide and comment the results of the experiments, analyzing some problems and their solutions.
Finally, in Chapter 7 we provide some closing comments about the whole solution and perspectives for future work.

Chapter 2

Swaption

In this chapter we will provide an overview from a financial point of view of the problem that will be addressed in the next chapters. This is not intended to be a exhaustive explanation of these topics, but it aims to give sufficient knowledge to understand the problem and the proposed solution.
First of all, we will introduce the concept of derivative. A derivative can be defined as a financial instrument whose value depends on (or derives from) the values of other, more basic, underlying variables. [Hull, 2009]
Derivatives are one of the three main categories of financial instruments, the other two being stocks and debt. In our study we will focus on two type of derivatives, the *Swap* and the *Swaption*.

2.1 Swap

2.1.1 Overview

The Interest Rate Swap (IRS) is an agreement between two parties to exchange cash flows in the future. In particular, one party agrees to pay cash flows at a fixed rate of interest based on a notional principal for a fixed period of time. In return, it receives interest at a floating rate on the same notional principal for the same period of time. The floating rate is defined by a *Forward Curve* (FWD), that usually corresponds to LIBOR or EURIBOR curves.

For instance, consider a swap between A and B, where A agrees to pay B a fixed annual interest rate of $K\%$ on a principal N, and in return B agrees to pay A the 6-months EURIBOR rate on the same principal. Now A is the *fixed-rate payer* (*fixed leg*) because she will pay always the same amount to

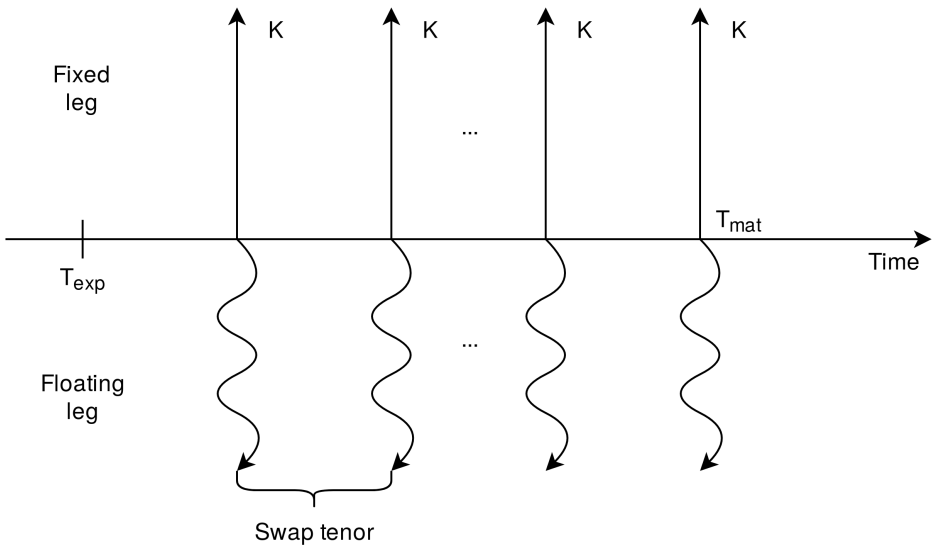


Figure 2.1: Representation of an Interest Rate Swap with fixed and floating payments on the same dates

B, while B is the *floating-rate payer* (*floating leg*) because the amount paid by B depends on the variations of the index used to compute the interest. For simplicity we can assume that all the payments will be performed every 6 months, but in this case we need to adapt the fixed interest rate with semiannual compounding. It is important to point out that this is not mandatory, and that the floating and fixed payments can be performed on different dates. The swap that we have defined is represented in figure 2.1.

A swap contract is fully defined by the following elements:

- Notional principal N: the initial amount on which the interests are paid, it is not actually exchanged;
- Strike price K: the fixed interest rate payed by the fixed leg;
- Floating rate: the interest rate payed by the floating leg, it is defined by a market curve, for instance: EURIBOR, LIBOR;
- Expiry date T_{exp}: the date when the contract begins;
- Floating tenor $\tau_{floating}$: the time interval between two floating payments;
- Fixed tenor τ_{fixed}: the time interval between two fixed payments;

- Maturity date T_{mat}: the date when the contract ends.

2.1.2 Pricing

Now that we have defined all the elements of the Interest Rate Swap, we can move to the valuation of a swap contract, i.e. define the price of the swap. The swap value, also denoted as the Net Present Value (NPV), is given by the contribution of two elements: the NPV of the fixed (NPV_{fixed}) and the NPV of the floating leg ($NPV_{floating}$), that are the sum of the actualized payments respectively of the fixed and floating legs. The complete formula to compute the NPV of the swap is provided in Equation 2.2.

2.2 Swaption

As for the swap, we will first introduce the concept of the swaption through an example, and then we will provide all the key elements that characterize it.

Swaptions are options on the interest rate swaps, and are one of the most popular types of interest rate options. The swaption gives the holder the right to enter in a interest rate swap at a specified strike price on a specified date. The owner of the swaption, of course, is not forced to enter the swap, but it can decide to not exercise his right if, for example, the swap conditions are less favorable than those available in the market.

Consider a company that knows that in 1 year will enter in a floating-rate loan for 3 years, and it would like to transform the floating-rate loan into a fixed-rate loan. In order to do that, the company can enter in a swap contract where it will receive the floating leg and pay the fixed leg. In this way, the company will effectively pay a fixed-rate loan instead of the floating one.
For this reason, the company decides to enter in a swaption at a cost, gaining the possibility to enter in a swap in 1 year at the specified strike price K. After one year, if the fixed rate of a regular 3-years swap turns out to be greater than K, then the company will enter the swap at more favorable conditions than those available in the market. In the other case, the company will enter in a regular swap with the fixed rate given by the market, because this will be more convenient than the swap relative to the swaption.

The main elements that characterize a swaption are:

- Expiry date T_{exp}: when the underlying swap will start;
- Tenor τ: the duration of the swap;
- Maturity date T_{mat}: when the swap will end;
- Strike price K: the strike price of the swap starting at T_{exp}. If at T_{exp} the fixed rate K of the underlying swap is higher than the fixed rate of a regular swap of the market starting in T_{exp}, then the underlying swap will be favourable and the holder will exercise his right to enter the swap.

2.3 Vasicek Model

In these settings, the Vasicek model is a mathematical model used by the bank to describe the evolution of the interest rates. The model is used in the valuation of the interest rate derivatives, e.g. the swaptions.
Based on the Vasicek model, the dynamics of the instantaneous interest rate are described by the following stochastic differential equation

$$dr(t) = k\left(\theta - r(t)\right)dt + \sigma dW_r(t)$$

where $W_r(t)$ is a Wiener process modelling the random market risk factor, i.e. the non-predictable part of the market. 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" k: the speed at which the rate r will converge to the long-term mean θ after a perturbation. Higher values of k mean that r will return very fast to the θ value;
- "instantaneous volatility" σ: the factor that controls the amount of randomness entering in the system. Higher values of σ will bring more randomness in the dynamics because the contribution of $W_r(t)$ will be greater.

Between these three parameters, only k and σ will be the outputs of our machine learning solution. In fact, the pricing formula of the swaption does not depend on the θ parameter, nor does the feedback function (Section 2.4.1). Also the bank calibrator does not produce θ as an output, because it is not relevant for the pricing of instruments.

2.4 Calibration

Now that we have defined what the swap and the swaption are, let's introduce the concept of the calibration, that is the task that we are going to replace with our machine learning solution.
The calibration is the process that finds the optimal model parameters given the market prices of the swaptions that are considered, by minimizing the error given by the feedback function.
We will first describe in detail how the feedback function is computed and then we are going to introduce the bank calibrator, i.e. the algorithm currently used by the bank to perform the calibration.

2.4.1 Feedback Function

For our solution it is important to define the feedback function used to compute the error of the model parameters, because it will be extensively used in our model. We will provide all the formulas needed to compute the final value, but we will omit some details of the theoretical background in order to keep the explanation lighter.
The feedback computes the error between the model and the market prices of the swaptions, and it is the objective that we want to minimize to obtain good model parameters that can describe the interest rate in an accurate way.
For each input market price x_i, let $h_i(k, \sigma)$ be the price given by the Vasicek model for the parameters (k, σ), so the error is

$$\epsilon_i(k, \sigma) = |x_i - h_i(k, \sigma)|$$

Then we have to compute the norm over the whole set of swaptions to obtain the final feedback value

$$J(k, \sigma) = \sqrt{\sum_i w_i^2 \epsilon_i \left(k, \sigma\right)^2} \tag{2.1}$$

where the weights w_i are the *Vegas* provided as inputs to the problem. The vega is defined as the rate of change in the fair value of the option per 1% change in the volatility.

Now we have to define the model price $h_i(k, \sigma)$ of each swaption, and the computation can be summarized in the following steps:

- Compute the strike price of the underlying swap;

- Compute the interest rate $\bar{r}$ such that the NPV of the swap is zero at T_{exp};
- Compute the price of the swaption.

Strike Price

The strike price K of a swaption is the fixed interest payed by the fixed leg. In order to compute it we have to set the swap price to zero, and then solve for K.
The price at time t of the swap with fixed payments in the dates $\Upsilon = \{T_{\alpha+1}, \ldots, T_\beta\}$ with fixed interest K and floating payments in the dates $\overline{\Upsilon} = \{T_{\bar{\alpha}+1}, \ldots, T_{\bar{\beta}}\}$, can be expressed as follows

$$NPV\left(r_t, t, \Upsilon, \bar{\Upsilon}, K\right) = \sum_{i=\bar{\alpha}+1}^{\bar{\beta}} \left[\frac{\Phi_d(S_{i-1}, S_i)}{\Phi_f(S_{i-1}, S_i)} P^d(t, S_{i-1}, r_t) - P^d(t, S_i, r_t)\right] - K \sum_{i=\alpha+1}^{\beta} \tau_i P^d(t, T_i, r_t) \tag{2.2}$$

Where T_α is the expiry date of the swap, and τ_i is the year fraction between the dates T_{i-1} and T_i.
Then, setting the price equals to zero, K can be expressed as

$$\sum_{i=\bar{\alpha}+1}^{\bar{\beta}} \left[\frac{\Phi_d(S_{i-1}, S_i)}{\Phi_f(S_{i-1}, S_i)} P^d(t, S_{i-1}, r_t) - P^d(t, S_i, r_t)\right] - K \sum_{i=\alpha+1}^{\beta} \tau_i P^d(t, T_i, r_t) = 0$$

$$K = -\frac{\sum_{i=\bar{\alpha}+1}^{\bar{\beta}} \left[\frac{\Phi_d(S_{i-1}, S_i)}{\Phi_f(S_{i-1}, S_i)} P^d(t, S_{i-1}, r_t) - P^d(t, S_i, r_t)\right]}{K \sum_{i=\alpha+1}^{\beta} \tau_i P^d(t, T_i, r_t)} \tag{2.3}$$

Now, in order to compute K, we still need to define the discount and forward curves and their shifts.
The short term interest rate is described by a shifted affine stochastic process $\bar{r_t} = (r_t + \phi_t)$. From this, we can define the discount curve as

$$P^d(t, T, r_t) = \Phi^d(t, T) A(T - t) e^{-B(T-t) r_t}$$

and the forward curve as

$$P^f(t, T, r_t) = \Phi^f(t, T) A(T - t) e^{-B(T-t) r_t}$$

Here, the terms $\Phi^d(t,T)$ and $\Phi^f(t,T)$ are the deterministic shifts of the curves, and can be calibrated with, respectively, the discount and forward curves provided as inputs to the problem with an initial interest rate $r_0 = 0.01$. From the previous two formulas we can compute the shifts as

$$\Phi^d(t,T) = \frac{P^d(t,T,r_0)}{A(T-t)e^{-B(T-t)r_t}}$$

$$\Phi^f(t,T) = \frac{P^f(t,T,r_0)}{A(T-t)e^{-B(T-t)r_t}}$$

by using the values of the two input curves for $P^d(t,T)$ and $P^f(t,T)$.
Finally, the coefficients $A(T-t)$ and $B(T-t)$ depend on the model used to describe the interest rate that, in our case, is the Vasicek model. Then, the coefficients are defined as follows

$$B(T-t) = \frac{1}{k}\left[1 - e^{-k(T-t)}\right]$$

$$A(T-t) = e^{\left(\theta - \frac{\sigma^2}{2k^2}\right)[B(T-t)-T+t] - \frac{\sigma^2}{4k}B(T-t)^2}$$

Where k, σ and θ are the parameters of the Vasicek model described in section 2.3.

Interest Rate $\bar{r}$

Now that we have computed the strike price K of the swap, we want to compute the interest rate $\bar{r}$, that is the value of the interest rate that makes zero the Net Present Value (NPV) of the swap at T_{exp}. So it can be defined as the value of r_t at T_{exp} such that

$$NPV(r_t = \bar{r}, t, \Upsilon, \bar{\Upsilon}, K) = 0$$

where the NPV of the swap is computed with 2.2.
In order to compute the value of $\bar{r}$ we need to perform a numerical search for the zero of the function, by using root-finding algorithms like bisection or Newton's methods. This procedure will be described more in depth in Section 5.6, dedicated to the implementation of the feedback function.
The NPV of the swap is a monotonic increasing function with a unique zero, so the solution can be easily found. The NPV at time T_{exp} represents the value of the swap at time T_{exp} by receiving the floating leg and paying the fixed leg. Since the NPV is a function of the interest rate r, when r increases we receive higher payments because we are receiving the floating leg, so the NPV increases. On the other hand, when r decreases our income decreases

and so does the NPV. When the NPV is zero, then the r value is such that the floating and the fixed payments are equal.

Model Price

At this point we have all the elements needed to compute the NPV of the swaption, that can be written like

$$\begin{aligned} SO(t, T_\alpha, \Upsilon, K) = &\sum_{i=\bar{\alpha}+1}^{\bar{\beta}} \frac{\Phi_d(S_{i-1}, S_i)}{\Phi_f(S_{i-1}, S_i)} \Phi_d(S_{i-1}, S_i) \hat{A}_{(i-1)\alpha} \Psi_t(T_\alpha, r_0, \hat{B}_{(i-1)\alpha}, \bar{r}) \\ &- \sum_{i=\bar{\alpha}+1}^{\bar{\beta}} \Phi_d(t, S_i) \hat{A}_{i\alpha} \Psi_t(T_\alpha, r_0, \hat{B}_{i\alpha}, \bar{r}) \\ &- K \sum_{i=\alpha+1}^{\beta} \tau_i \Phi_d(t, T_i) A_{i\alpha} \Psi_t(T_\alpha, r_0, B_{i\alpha}, \bar{r}) \end{aligned} \tag{2.4}$$

Finally, we only need to introduce $\Psi_t(T, r_0, \rho, b)$ and all of its coefficients, that are defined as follows

$$\Psi_t(T, r_0, \rho, b) = P^d(t, T, r_0) e^{-b\rho + \Theta(t,T)\rho + \frac{\rho^2}{2}\Omega(t,T)^2} N(-h^+(b))$$

$$h^+(b) = \frac{\Theta(t, T, b)}{\Omega(t, T)} + \Omega(t, T)\rho$$

$$\Theta(t, T, b) = (b - r_0) + (r_0 - \theta)kB(T - t) + \frac{\sigma^2}{2} B(T - t)^2$$

$$\Omega(t, T) = \sigma \sqrt{\frac{1 - e^{-2k(T-t)}}{2k}}$$

2.4.2 Calibrator

After the definition of the feedback function, we can introduce the calibrator. The goal of the calibrator is to provide the parameters of the Vasicek model, that are mean reversion speed k and volatility σ, by minimizing the error on the market prices given by the feedback function.
The minimization performed by the calibrator is made by the Levenberg-Marquardt algorithm (LMA), that is generally used to solve non-linear least-squares problems. The algorithm is a combination of the Gauss-Newton

Algorithm (GNA) and the method of Gradient Descent.
Although this method usually converges to the global minimum, it can still find a local minimum instead of the global one. For this reason, multiple runs of the algorithm with different initializations may be required in order to find the global minimum with a good confidence.
In Chapter 4 we are going to analyze the main limitations of this calibrator and describe two machine learning approaches to solve these problems.

In the next chapter, we are going to briefly explain the main machine learning techniques that will be used in Chapters 4 and 5.

Chapter 3

Technologies

In this chapter we will provide an overview of the main technologies and algorithms that has been used in the development of the project.

3.1 Parallel Computing

Parallel computing is a type of computation in which big computational tasks can be splitted in different sub-tasks that can be executed at the same time.
In the last few years parallel computing, in particular GPU computing, has seen a huge diffusion and development mainly thanks to the great explosion of deep learning, a family of machine learning methods. In fact, deep learning techniques like convolutional neural networks (CNN), deep neural networks (DNN) and others, make a heavy use of parallel computing, and bringing the computation to the GPU has given a huge performance boost in their tasks.
The most used framework for GPU computing is CUDA, a proprietary solution by Nvidia, that dominates the market over its open-source counterpart OpenCL in these applications.

3.1.1 CUDA

In this project we used CUDA to move the non-parallel feedback computation from CPU to the GPU, so we will give a brief explanation of how it works. In particular we used a Python library that is an interface to the C CUDA implementation.
In CUDA, the kernel is the set of operations the are going to be parallelized. Each kernel can be potentially executed on a CUDA core in parallel to all

the other ones, where the only limits are the hardware limits of the GPU that is being used.
This kind of parallelization is called Data Parallelism, where the same set of instructions (kernels) are performed over different data. This is in contrast with to the Task Parallelism, where different tasks are executed at the same time and on the same data.

For instance, data parallelism can be applied to matrices, where the same function can be applied simultaneously to all the cells of the matrix. This will be our case, where we have the price matrix of the swaptions and we have to compute the model price for each combination of tenor and expiry (i.e. for each cell of the matrix) in order to compute the overall feedback error relative to the date that we are analyzing (applying the procedure explained in section 2.4.1).
In particular, kernels must be organized in a precise structure, made of grids, blocks, and threads. The thread is the finest grain decomposition and represents the execution of one kernel. Then, threads are grouped in blocks of a certain dimension that depends on the hardware limits of the GPU. Finally, the blocks are put together in grids, where all the blocks in one grid must be executed on the same physical GPU. The design of this structure, in particular the size of the blocks, is very important to maximize the GPU utilization because it highly affects how the threads are scheduled on the GPU cores. The structure is represented in Figure 3.1

The common processing flow of a CUDA program can be decomposed in three steps:

1. Copy input data from CPU memory to GPU memory
2. Load the GPU program (kernel) and execute it, caching data on chip for improved performance
3. Copy back results from GPU memory to CPU memory

3.2 Machine Learning

Machine learning is a field of Computer Science where a computer program learns from experience E with respect to some class of tasks T and performance measure P, and improves with experience E. [Mitchell, 1997]
There are three big different families of machine learning methods: super-

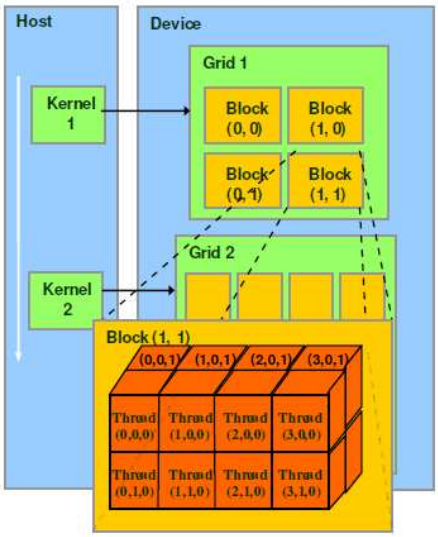


Figure 3.1: Representation of the organization of the CUDA kernels in grids, blocks and threads

vised learning, unsupervised learning and reinforcement learning.

In supervised learning the system learns a function mapping input data to output parameters by viewing at input-output pairs as examples, i.e. a set of labeled data. Such a model can learn from the known data and can generalize (i.e. provide the target parameters) even for unseen data. Usually, the known data used for the training of the model are defined as the training set, while the new data used as a performance evaluation of the model are identified with the test set.
The two common supervised tasks are classification, where we want to classify the input data by assigning them a class, and regression, where the output parameters to be predicted are scalar values.
Supervised learning models can be further divided into parametric and non-parametric methods. Parametric models represent the mapping function from the inputs to the targets with a fixed set of parameters, that will be later tuned by the training algorithm. Some examples of parametric models are neural networks, linear regression or support vector machines. On the other hand, non-parametric models are not characterized by a fixed set of parameters, since the number of parameters can grow with the size of the training set. Some examples are decision trees, K-nearest neighbour, where the predictions for new data are performed by looking at the training data

instead of a model built from them, and also kernel support vector machines.

In unsupervised learning we work only with unlabeled data, and the system tries to learn hidden structures in the data, like for instance the underlying probability distribution. Since it works only with unlabeled data, there is not a true benchmark to test it against, but it tries to minimize some performance metrics.
Some examples of unsupervised techniques are clustering, density estimation, and dimensionality reduction methods like principal component analysis. Clustering is the task of finding sets among the data such that data points inside the same set are very similar and at the same time as different as possible from data in the other sets. It can be used to find hidden communities and relationships of similar data inside a dataset. There are also particular types of neural networks that are employed for unsupervised learning, like the autoencoders, that perform a sort of dimensionality reduction providing a new, more compact representation of the data.

Finally, there is reinforcement learning, where we want to find the best agent that acts in an environment by taking actions in order to maximize a reward. Reinforcement learning is very different from the previous two families of machine learning algorithms, and so are the techniques employed to solve the problem. This is a very interesting field since it can be defined as the "true artificial intelligence", where there is an intelligent agent taking actions in the environment to achieve a goal.

3.2.1 Principal Component Analysis

Principal Component Analysis (PCA) is a dimensionality reduction technique and it is used to create a new representation of the data by using a reduced number of dimensions.
Reducing the number of dimensions of the data can be very important to overcome the problem commonly defined as "curse of dimensionality". By curse of dimensionality we mean the phenomenon for which when the dimensionality increases, the volume of the space increases so fast that the available data become sparse [Bishop, 2009]. In order to obtain a statistical significant result, the amount of needed data usually grows exponentially with respect to the number of dimensions. Another big problem of the high dimensionality is that all data points appear to be sparse and dissimilar, so it can be very difficult to apply grouping techniques like clustering.

For all these reasons we can use PCA to reduce the number of the dimensions of the data, trying to lose the minimum amount of information possible.

PCA can be defined as the orthogonal projection of the data onto a lower dimensional linear space, known as the principal subspace, such that the variance of the projected data is maximized [Hotelling, 1933]. The dimensions of the principal subspace are known as principal components. The first principal component is the component that captures the largest percentage of variance of the data, and can be seen in a intuitive way as the dimension such that, when the data are projected onto that dimension, it has the maximum variance of the data. Then, the second principal component can be seen as a new dimension, orthogonal to the first one, that has maximum projected variance, and so on for the other principal components. More rigorously, the principal components are the eigenvectors of the correlation matrix S of the data, defined as follows

$$S = \frac{1}{N}\sum_{n=1}^{N}(x_n - \bar{x})(x_n - \bar{x})^T$$

where x_n is a general data point and $\bar{x}$ is the sample mean

$$\bar{x} = \frac{1}{N}\sum_{n=1}^{N}$$

Then, the eigenvector e_k of S with the k^{th} largest eigenvalue λ_k is the k^{th} principal component and its respective percentage of explained variance is given by

$$explained\ variance_k = \frac{\lambda_k}{\sum_{i=1}^{k}\lambda_i}$$

The full set of the eigenvectors will form a new orthogonal basis of the initial space, while the projection of the initial data onto the first k principal components produces a new reduced representation of the data. From this new representation, it is possible to reconstruct the original data despite having some error, depending on the number of principal components used to project the data. An example of projection of some bidimensional data into the first principal component is provided in Figure 3.2, where the data are projected into the dimension that captures the most variance. When the data are reconstructed using only the first principal component, we will lose some information depending on the percentage of the captured variance.

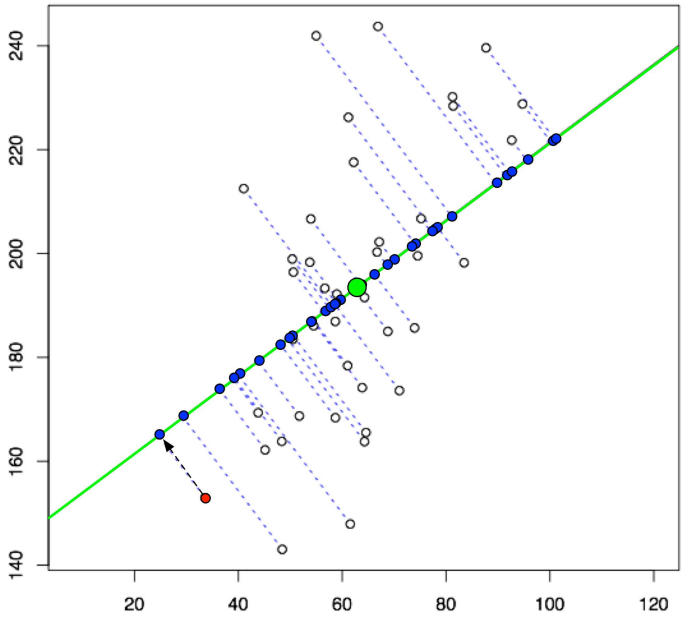


Figure 3.2: Example of projection of 2D data points into the first principal component. The white circles are the original data, the green line represents the principal component, the green dot is the mean of the data and the blue dots are the projected points, i.e. the points reconstructed using only the first principal component.

3.2.2 Feed-forward Neural Networks

Neural networks are simply non-linear functions from a set of input variables to a set of output variables controlled by a vector of parameters. They are very popular in machine learning because of their representation power. In fact, they are said to be *universal approximators* because, with the right structure, they can approximate any continuous function. [Bishop, 2009].
The big popularity of this model came mostly in the last ten years, because it needs many samples to be able to generalize in a good way and therefore the computational resources needed for training can be very high. For this reason, in the last years these models became very popular thanks to the increased computational power available in the cloud and the shift of the computation on the GPUs (3.1).

Feed-forward neural networks have a great advantage with respect to the more traditional regression and classification models, in fact they can perform feature extraction procedures automatically during the training phase instead of having to manually design the basis functions on the input features. The basis functions are non-linear functions of the inputs, used to create new features. This can be very helpful, because usually it is not trivial to find good basis functions for the problem. Neural networks can perform the step of learning the basis functions in the first hidden layer of the network, and then elaborate more the result in the other layers.

The structure of a feed-forward neural network is made of an input layer, whose dimension is equal to the number of features in input, then zero or more hidden layers and finally the output layer. Each layer is made of one or more neurons, and each neuron applies a function, the *activation function*, to a linear combination of its inputs with the following formula

$$a_j = h\left(\sum_{i=1}^{D} w_{ji}x_i + w_{j0}\right)$$

where a_j is the output of the j^{th} neuron, D is the number of inputs, x_i are the inputs, w_{ji} are the weights of the connections between the input i and the neuron j, w_{j0} is the bias and finally $h(.)$ is the activation function.
An example of structure of a feed-forward neural network with a single hidden layer is represented in Figure 3.3.

The choice of the activation function is important to achieve a good result with the network. The function of the output layer depends mostly on the task we are performing, i.e. classification and regression, where we could choose a logistic sigmoid function for the former and a linear one for the latter. In the hidden neurons the choice is wider, some of the possibilities are: identity, hyperbolic tangent (tanH), sigmoid, rectified linear unit (ReLU) and exponential linear unit (ELU). Each of these functions has different properties and can be better for different classes of tasks. For instance, in deep learning using convolutional neural networks, the ReLU activation function is very popular for image analysis.
A representation of the different activation function is provided in Figure 3.4.

The actual learning of the network to fit a particular function is performed in the training, where we find the parameters of the network that minimize the error between the predicted and the original outputs. This

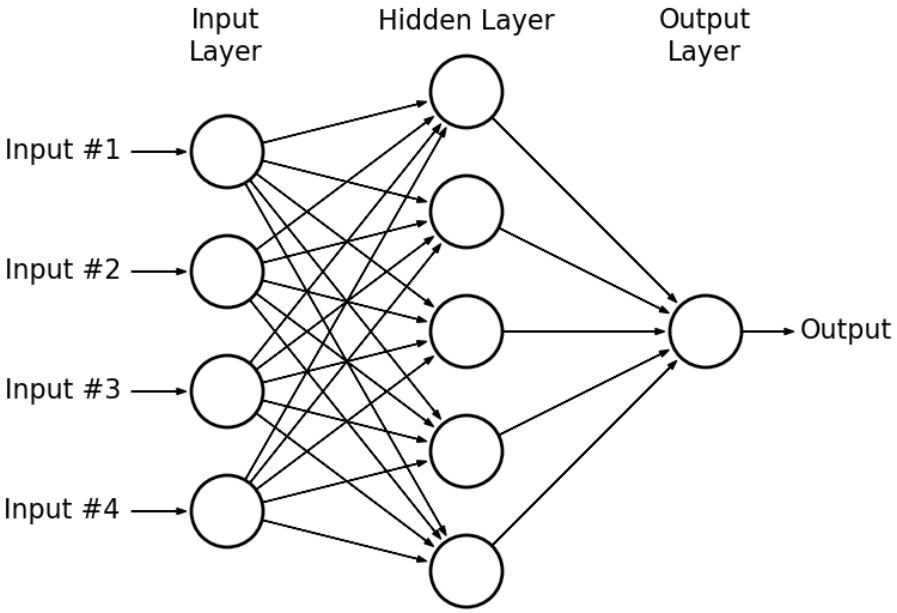
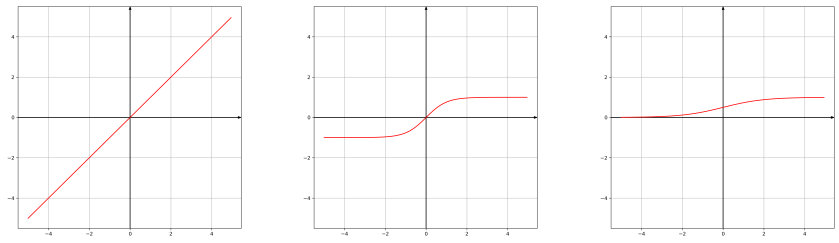
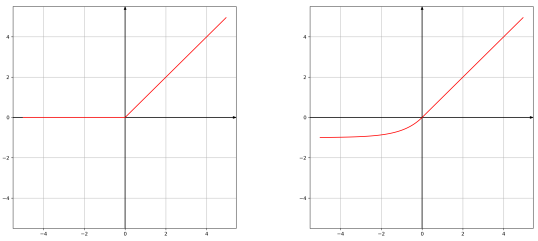


Figure 3.3: Example of a feed-forward neural network with four input neurons, one hidden layer with five neurons and one output layer with one neuron

procedure is called *error backpropagation* and lets to perform weight updates in a very efficient way by using the gradient of the error function with respect to the weights.
Although the error backpropagation is fundamental for supervised learning, in our solution we will not use it because we will perform a sort of unsupervised learning. In particular, the algorithm that will be used for the network optimization are presented in the next section.



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



(d) ReLU *(e) Elu*

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

Chapter 4

State of the art

In this section we are going to present two different machine learning methods that address the calibration problem with a supervised approach. Their goal is to replace the calibrator currently used by the bank with a model trained on the calibrator solutions.

4.1 Limitations of the Calibrator

The calibrator briefly described in Section 2.4.2 has some problems and limitations. The main drawback of calibrator's algorithm is speed, in fact it is too slow to be applied on real-time data because the calibration must be performed before each pricing and this can be extremely slow. For instance, a single calibration relative to one day took around 40 seconds, that have been reduced to roughly 4 seconds after several optimizations in the computation of the feedback function.
For this reason, the adopted solution is to use the calibrator only on a subset of the instruments in order to reduce the calibration time, but obviously at the cost of a less precise calibration. The optimal calibrator is run over the whole set of swaptions, that are 238, while the traded-off calibrator only runs on a subset of 7 swaptions in order to reduce the computational time. Obviously, this approximation has a cost in term of the accuracy of the produced parameters of the model.
This is one of the main reasons that led to the exploration of new methods in order to speed up the model estimation.

4.2 Machine Learning

The solution to the problem was to apply machine learning techniques to replace the calibration procedure. With this approach, only one evaluation of the model is needed in order to obtain the optimal parameters of the Vasicek model, instead of having to perform a complete LMA optimization, that would take way more time.
In the two following sections we will present two different methods to address the problem of the calibration, both of them using a supervised learning approach.
Supervised learning is a branch of machine learning, and it is the task of learning a function that maps the inputs to the outputs based on example input-output pairs. The function can be represented by different models, that can be parametric (i.e. entirely defined by a fixed set of parameters θ) or non-parametric.
In particular, the goal of these two methods is to replicate the behavior of the LMA algorithm, i.e. the bank calibrator, by building a parametric or non-parametric representation of the calibration algorithm, since it is trained on its solutions.

4.2.1 Online Supervised Method

The first solution using a supervised approach is described in Master's thesis [Cella, 2016], where they tried both Decision Trees and K-nearest neighbour with $K = 1$ to represent the function from the inputs to the outputs. Both of these models are non-parametric.

The input data used for this method are similar to the data that have been used in the project presented in this document. In particular, the available data about the swaptions are prices matrix, volatilities matrix, log-normal shifts matrix, discount and forward curves, and finally vegas matrix. Each matrix has a 17 rows and 14 columns, and each cell represent a combination of T_{exp} and tenor τ. Then, both the discount and forward curves contains 120 points for different days. I will describe in more detail how the dataset is organised and the exact meaning of every feature we have in the Section 5.4.
As we can see, the data have a very high dimensionality, since for each date we have 3 matrices of 238 (17*14) elements each, and also 120 points from the discount curve and again 120 points from the forward curve. For this reason it was necessary to perform a dimensionality reduction to represent

the data in a more compact way.

First of all, Principal Component Analysis (PCA, explained in section 3.2.1) is applied to the prices and volatilities matrices of the swaptions. As a result of the PCA, the new dimensions of the price matrix are 4, maintaining the 99% of the variance, while for the volatility matrix the dimensions are reduced to 6, again capturing the 99% of the variance.
For what concerns the matrix of the log-normal shifts, it has been observed that the values are always constant for the same date, then the matrix can be replaced by a scalar value without any loss of information.
The last matrix we had was the matrix of the vegas, but these are not used for the training of the model.
At this point, we still need to reduce the number of dimensions of the discount and forward curves. In order to solve this problem, the Nelson-Siegel model [Charles R. Nelson, 1987] has been used to fit the curves, reducing the representation to only four parameters per curve, instead of the initial 120 components.
In the end, the initial high-dimensional data has been reduced to the following features: 4 principal components of the price matrix, 6 principal components of the volatility matrix, 1 log-normal shift, 4 Nelson-Siegel components for the discount curve and other 4 components for the forward curve, for a total of 19 features.
After these dimensionality reduction techniques, they performed feature selection using first a supervised feature selection approach, where a model is fitted to the data and a score is assigned to each feature depending on how much it is important in predicting the target values. Then, only the features with scores greater than the average score are kept. The other method is Iterative Feature Selection (IFS) that iteratively fits a supervised model to different subset of features, adding a new feature in each iteration based on a score.
The whole process of feature selection and construction is summarized in Figure 4.1.

After the preprocessing and decomposition of the original data, it is time for the real training of the model. The data are splitted into the training and test sets, maintaining the temporal sequence between them. Since the data are temporal, i.e. the samples are sorted per reference date, they can present trends based on the evolution of the underlying market. For this reason, the classical batch approach in which the training is performed on the training set and then the results are evaluated on the test set could be

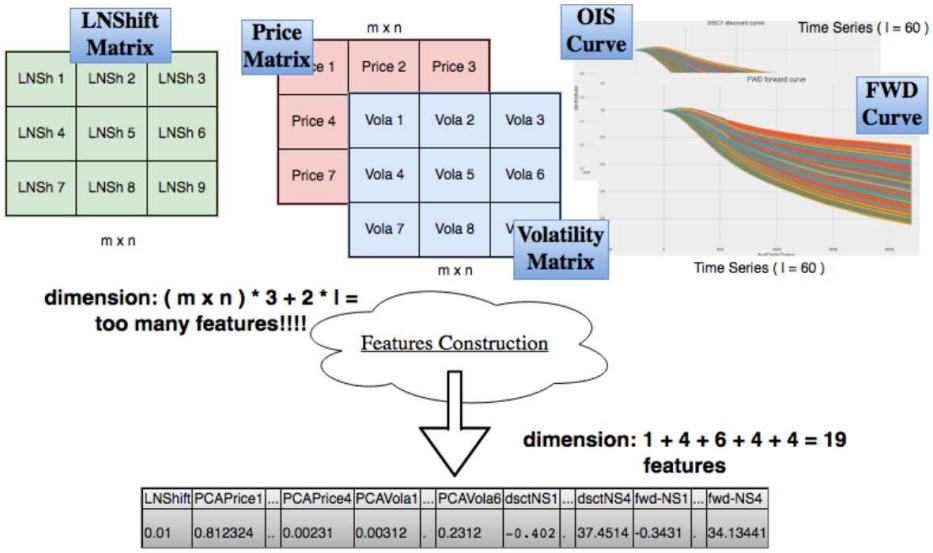


Figure 4.1: Summary of the dataset construction from the original features

not good for this application. In fact, the model could not capture the variation in the test data due to some changes in the market. For this reason, they chose to use a online approach, i.e. first performing a batch training over the training set, and then an online phase where the model is adjusted for each new sample of the test set. For sure this online method is slower than the offline one, but it brings some advantages because in this way the model will be able to catch eventual variations in the distribution of the data. In addition, the online evaluation is closer to the real application of the method, where new data are coming on a daily basis and the model could be trained again.

In the offline learning they trained a decision tree on the training set, without giving any attention to not overfit the data. This is because we can see that overfitting over the training data does not compromise the prediction accuracy on the test data.
After the batch phase, the model is evaluated online, retraining the decision tree for each new sample in the test set. The results of the prediction of the mean reversion speed k can be seen in Figures 4.2 and 4.3, while the results for the volatility σ in Figures 4.4 and 4.5.

In the end, this method gave good results with small errors and drastically reduced the computational times needed for calibration with respect to the

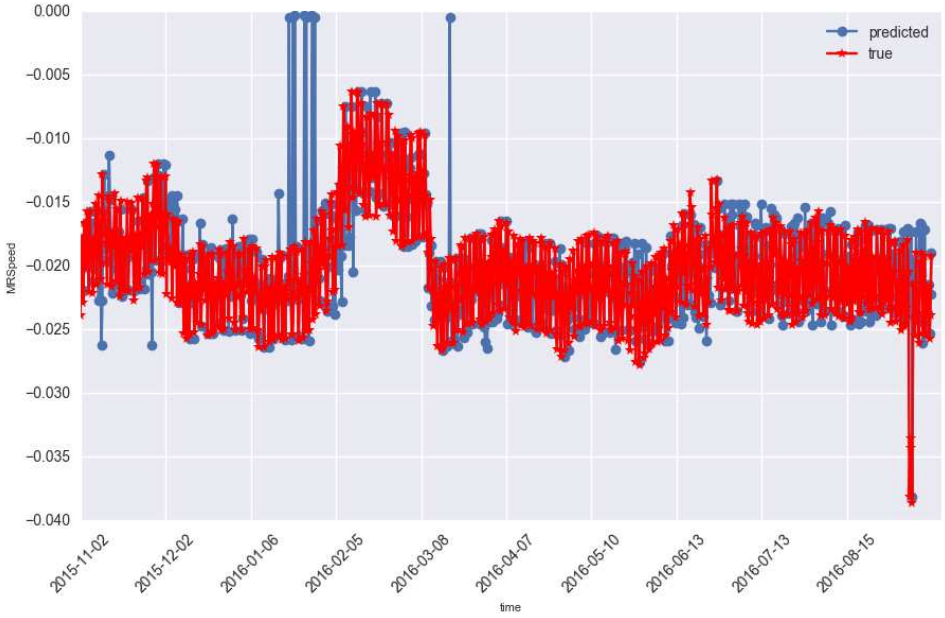


Figure 4.2: Mean reversion speed prediction during the online stage on the test set

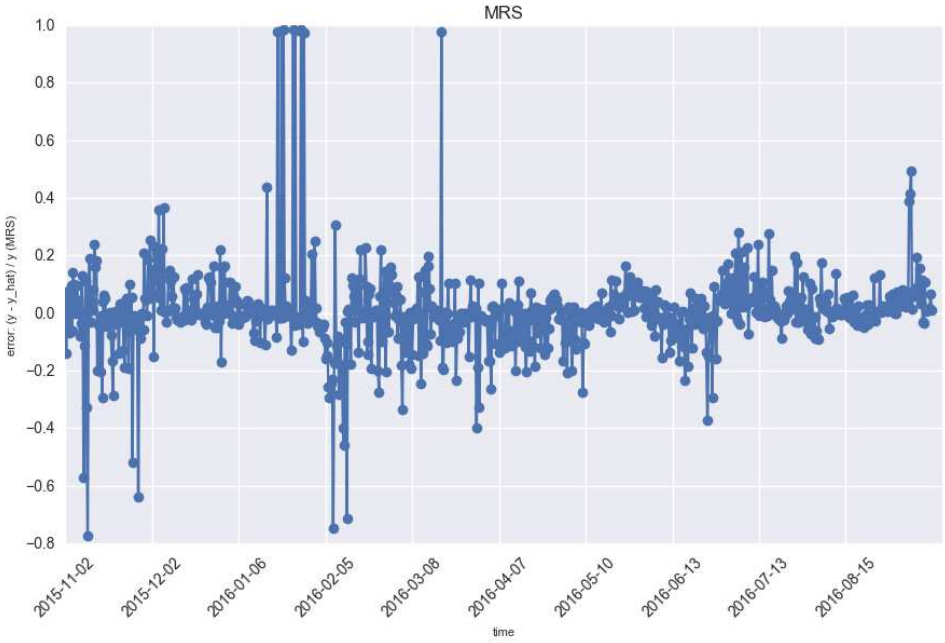


Figure 4.3: Relative error of the mean reversion speed prediction during the online stage on the test set

Figure 4.4: Volatility prediction during the online stage on the test set

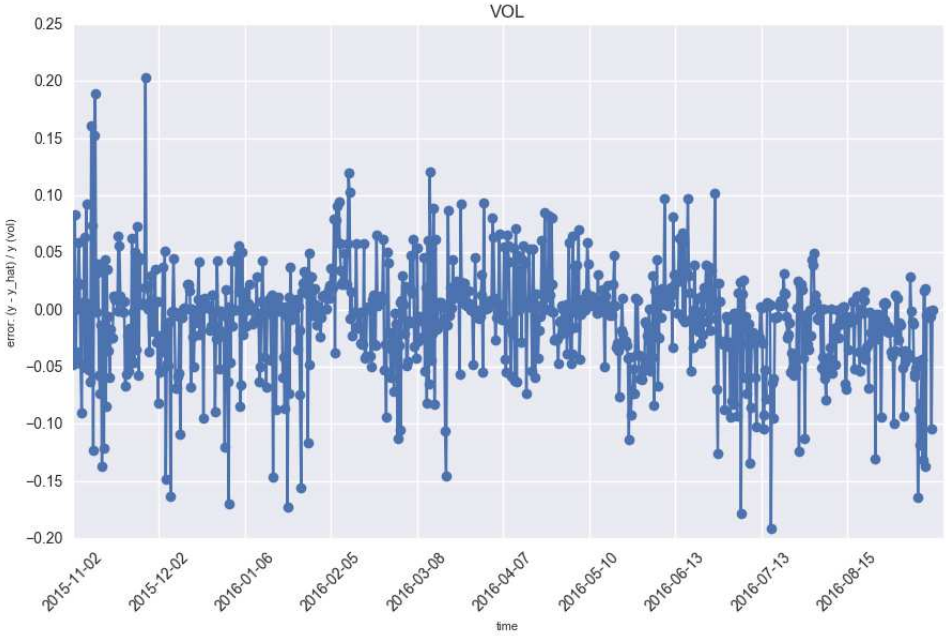


Figure 4.5: Relative error of the volatility prediction during the online stage on the test set

calibrator used by the bank.

4.2.2 Supervised Method with Augmented Data

A different approach is presented in the paper [Hernandez, 2016]. This method addresses exactly the problem of the calibration, still using a supervised approach.
Like the method described in previous section, this builds a model to predict the model parameters by training on input-output pairs. The parameters will be mean reversion speed k and volatility σ. Unlike the previous model, they don't use an online test phase, but it first perform the training and validation on the generated training set and finally evaluate the model on the historical data. In addition, the training set will be made of generated data in order to train the model on a high number of samples, significantly higher than the available historical data.

First of all, they need to obtain the model parameters from the input data, so they perform the calibration on all the historical data using a Levenberg-Marquardt local optimizer and repeat the calibration twice for each sample with two different initializations: one standard with parameters $k = 0.1$ and $\sigma = 0.01$, and the second using the calibrated parameters from the previous day.
After the calibration of the history is completed, they generate the training set using a particular technique made of the following steps:

- Compute the errors for each swaption for each day
- Take the natural logarithm of the model parameters
- Rescale yield curves, parameters and errors to have zero mean and unitary variance
- Apply PCA to the yield curve and take the principal components that account for the 99.5% of the explained variance
- Compute covariance of errors, normalized log-parameters and yield curve principal components
- Sample vectors from a normal distribution with zero mean and the given covariance
- Transform everything back to the original mean, variance, dimensionality and take the exponential of the parameters

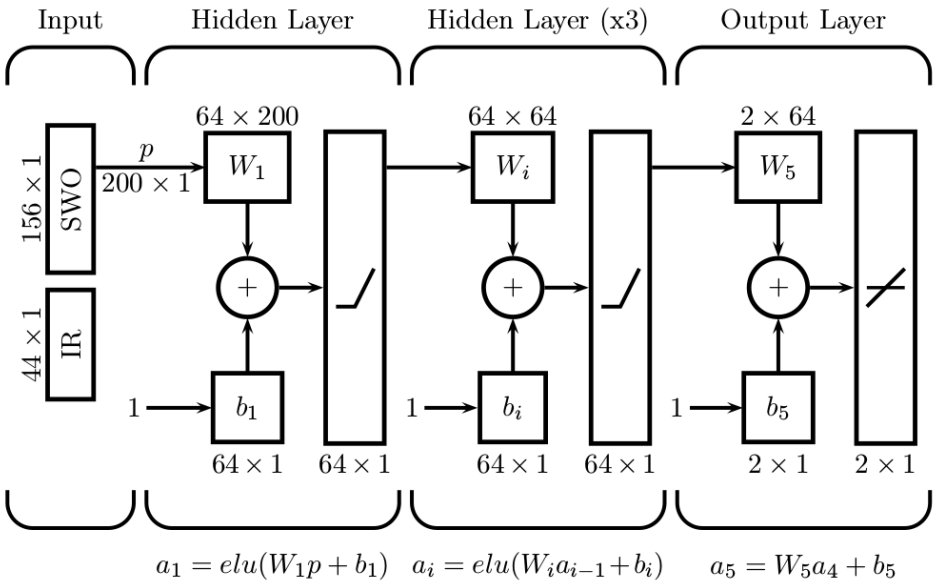


Figure 4.6: Topology of the feed-forward neural network used for the calibration of the model (source [Hernandez, 2016])

- For each new generated sample take a random reference date from the set used for covariance estimation
- For all the the swaptions compute the implied volatilities from the model parameters and apply random errors to the results

After the generated training data are available, they define the model. They use a feed-forward neural network that takes swaption volatilities and the yield curve as input and provides the two model parameters as output. For the hyperparameter tuning they apply a truncated grid search combined with a manual search, and at the end they come up with the following hyperparameters for the network:

- 4 hidden layers with 64 neurons each using a ELU activation function
- dropout rate of 0.2 for all the layers
- Nadam optimizer with a learning rate of 0.001

The structure of the network is represented in Figure 4.6.
The network is trained on the data generated from the estimated covariance, using the 80% of the data as training set and the remaining 20% as the validation set. In the end, the network is tested on the historical data, that

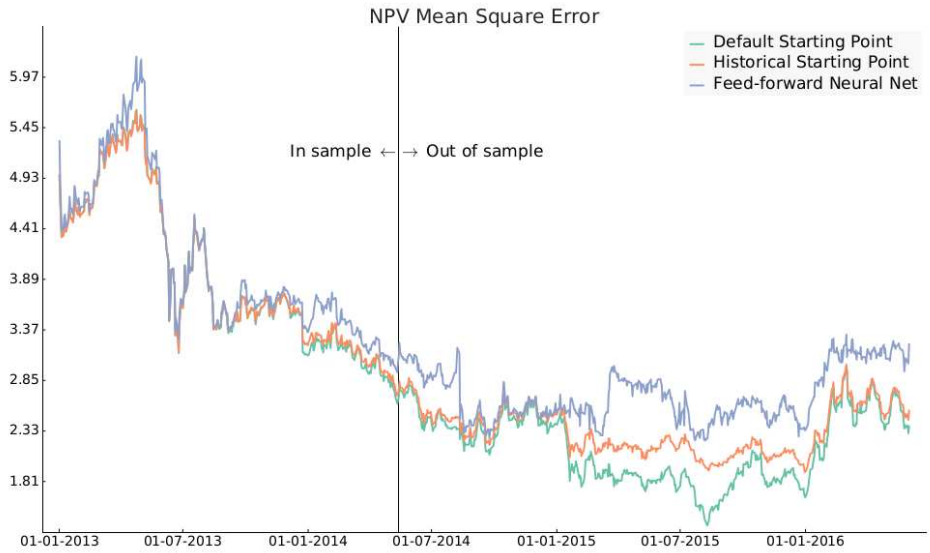


Figure 5: Correlation up to June 2014

Figure 4.7: Errors between the original and the model volatilities with the training set generated on covariance estimated on data until June 2014 (source [Hernandez, 2016])

will serve as a backtesting of the model.
The results of the training phase are shown in Figures 4.7 and 4.8, where is represented the error on the historical data.

4.3 Considerations

The previous two machine learning methods are the two solutions that have been developed to tackle the problem of the slowness of the calibration. In the next section, we are going to propose our solution, whose main goal is to completely replace the original calibrator. The main advantage of our procedure is that we propose a black-box optimization that is independent from the interest rate model, so that the model can be changed without huge efforts that were required before. The only thing that would need to be reimplemented is the feedback function, but not the calibrator itself.

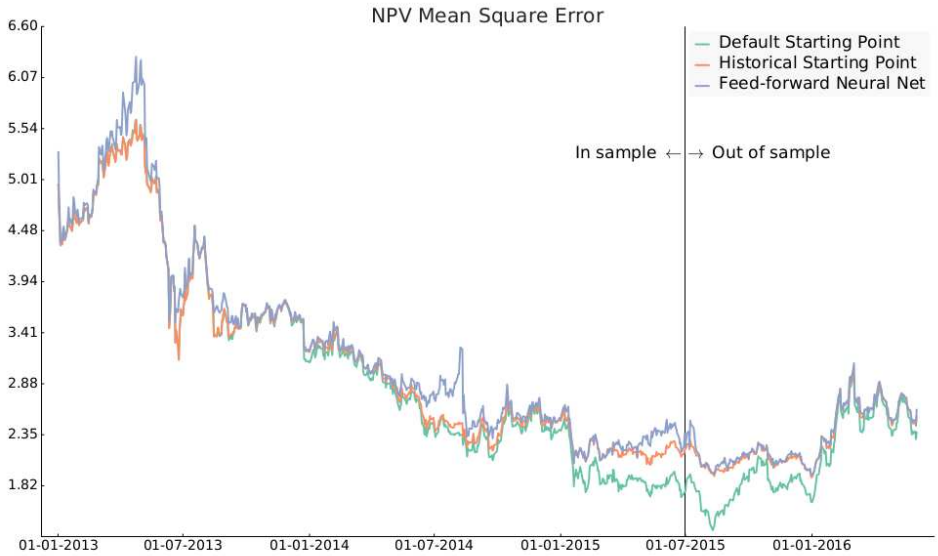


Figure 6: Correlation up to June 2015

Figure 4.8: Errors between the original and the model volatilities with the training set generated on covariance estimated on data until June 2015 (source [Hernandez, 2016])

Chapter 5

Proposed solution

In this chapter we will present the procedure we followed and all the implementation details of our solution.

5.1 Limitations of the Supervised Method

All the machine learning methods presented in Chapter 4 used a supervised approach to solve the calibration problem. These two methods achieved very good and accurate results, with a small error on the model parameters they wanted to predict. The main advantage of these methods with respect to the original calibrator used by the bank is the speed gain, in fact the calibration of new data from the market is performed by a single evaluation of the models, i.e. decision tree, K-nearest neighbours or neural network, that are quite fast.
Although this great advantage, there are still some problems that have not been addressed and that we are trying to solve with our method. For instance, the supervised methods need the historical data to be calibrated using the bank calibrator in order to have the original target parameters. This process can be very time consuming, but the time is not the real problem here because this process should be performed only once, and it could be performed offline.
The real drawback of supervised methods is that we still need the calibrator, that can be very complex to design and implement. In addition, the calibrator is model dependent, thus if the bank changes the model used to represent the interest rate we would have to implement again a new calibrator, that is a long and complex task.
For these reasons, we are proposing a black-box solution that doesn't make

use of the calibrator to calibrate the historical data but it only needs the market data as inputs. The only part that is still required is the feedback function, used to evaluate the goodness of the model parameters, and that would be necessary in any case for the development of the calibrator. The advantage of our solution is that if the bank changes the interest rate model, for instance from a simpler to a more complex one, then we would only need to implement the new feedback.

5.2 Formalization of the Problem

Before continuing with the explanation of our solution, we are going to provide a formalization of the problem we are solving.
As we have already mentioned, we want to solve the calibration problem, that is the design of a calibrator for the interest rate model. This means that the calibrator, starting from the current market data, must provide the calibrated model parameters η where the correspondent model best describe the market data. In our particular case, the vector η contains the parameters k and σ of the Vasicek model.

First, we define the inputs of our problem like the market quotes Q_i relative to the date i, containing the swaption prices, volatilities, the discount and the forward curve. Q_i refers to a set of N swaptions, so $|Q_i| = N$. Then, we have the target model parameters η, where $|\eta| = n$.
Our goal is to approximate the calibration function

$$f(Q_i) : \mathbb{R}^N \to \mathbb{R}^n$$

that provides in output the model parameters η calibrated to the input market quotes Q_i, with the neural network

$$\bar{f}(Q_i; \theta^*) : \mathbb{R}^N \to \mathbb{R}^n$$

where θ^* are the optimal parameters of the network.

In Section 2.4.1 we have defined the feedback function $J(k, \sigma)$, that is expressed by Equation 2.1, and can be be written as $J(\eta)$. The feedback function is non-differentiable, this is the main difficulty in our problem because for this reason we cannot use backpropagation in the training of the neural network and we had to set up a custom optimization process.
At this point, we can define the non-differentiable objective function g over

all the market quotes Q_i that we are going to minimize

$$g(\theta; Q) : \mathbb{R}^p \to \mathbb{R}$$

$$g(\theta; Q) = \frac{1}{|dates|} \sum_{i \in dates} J(f(Q_i))$$

where $p = |\theta|$.
Finally, we find the optimal network parameters θ^* that minimize the function $g(\theta; Q)$ over all the dates

$$\theta^* = \min_{\theta} g(\theta; Q)$$

5.3 Overview

Our goal is to design a model that predicts the interest rate model parameters by minimizing the feedback function, i.e. the error of the model prices computed against the input market data. Thus, this is a kind of unsupervised learning task because we are not going to use the target parameters in the training process.
We are going to explain the main steps of our solution, that is a black-box optimization of the known objective function $g(\theta; Q)$.

As a first step, we perform an exploration of the available data to spot interesting patterns in them and to find possible correlations to exploit. After that, we apply dimensionality reduction to the data because of their high dimensionality.
After the dataset has been preprocessed, it can be used to calibrate the interest rate model. However, since this process should be performed in real-time, the simple CPU implementation was not fast enough. For this reason, we decided to reimplement it using GPU code.
Finally, with the final dataset and a fast implementation of the feedback, we define our model, the model selection procedure, and the training and the evaluation techniques.
The whole calibration process is summarized in Figure 5.1.

5.4 Data Exploration and Preprocessing

Let's start our analysis from the exploration of the data. In our solution we mainly use the data relative to the euro currency (EUR), but in the Section

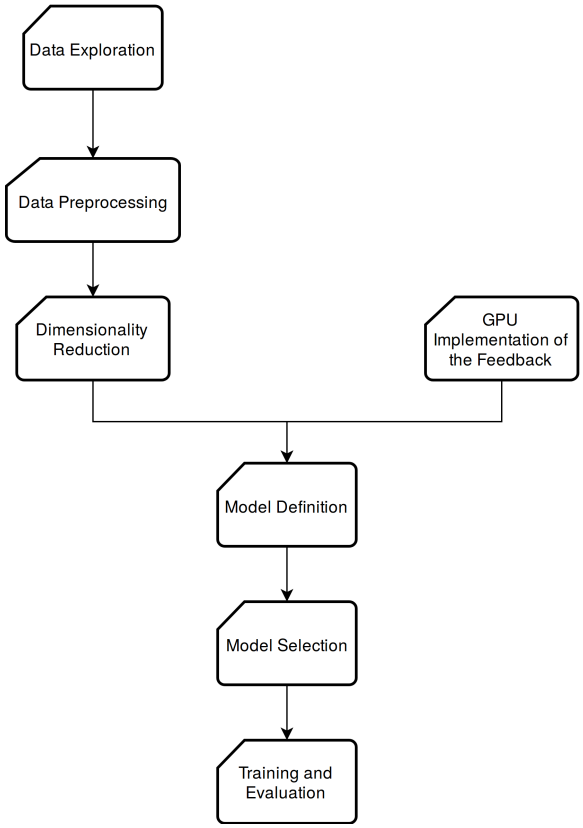


Figure 5.1: Summary of the whole calibration process, from data exploration to the evaluation of the solution

5.9 we will describe how it is possible to extend the model to use data from more than one currency.
The data at our disposal are 902 daily samples from the market with the addition of the two model parameters *mean reversion speed* (MRS) and *volatility* (σ) provided by the bank calibrator executed on the daily samples. We have these two parameters because they were already available in the dataset since the bank already developed the calibrator for the Vasicek model, but even if we have them we don't use them in the training phase, but only in our exploration to better understand their behaviours. The time span covered by the data goes from 2013-06-28 to 2017-09-12.
Let's start from the description of how the daily samples are structured. The relevant fields of each sample are described in Table 5.1.
We performed some minor processing on the data, like transforming the forward and discount dates into the equivalent delta days, i.e. the difference in

Table 5.1: 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 expiry periods of the swaptions. The expiry dates T_{exp} could be computed as T_{ref} + swaption expiry. For the *EUR* dataset we have 17 different expiries
Swaption tenors	The list of tenor intervals of the swaptions. The maturity dates T_{mat} could be computed as T_{exp} + swaption tenor. For the *EUR* dataset we have 14 different tenors
Swaption prices	Matrix containing the swaption prices for each combination of T_{exp} and tenor τ. The matrix shape is *#expiries* x *#tenors*
Swaption volatilities	Matrix containing the swaption volatilities for each combination of T_{exp} and tenor τ. The matrix shape is *#expiries* x *#tenors*
Swaption vegas	Matrix containing the swaption vegas for each combination of T_{exp} and tenor τ. The matrix shape is *#expiries* x *#tenors*
Swaption log-normal shifts	Matrix containing the swaption log-normal shifts for each combination of T_{exp} and tenor τ. The matrix shape is *#expiries* x *#tenors*
Forward dates	The dates for which the values of the forward curve are available
Forward values	The values of the forward curve correspondent to the forward dates
Discount dates	The dates for which the values of the discount curves are available
Discount values	The values of the discount curves correspondent to the discount dates
Mean reversion speed	The mean reversion speed of the Vasicek model provided by the calibrator for the reference date
Volatility	The volatility of the Vasicek model provided by the calibrator for the reference date
Calibration error	The feedback error made by the bank calibrator

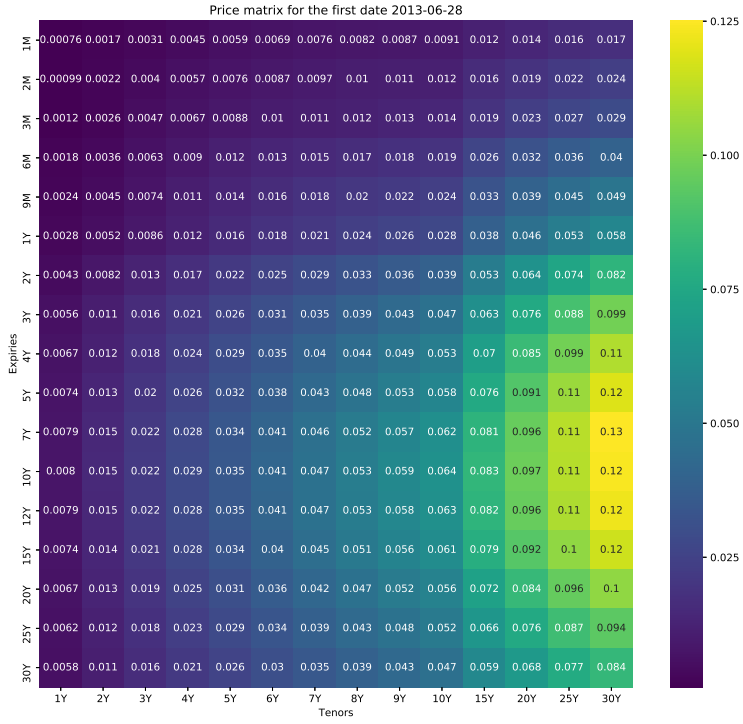
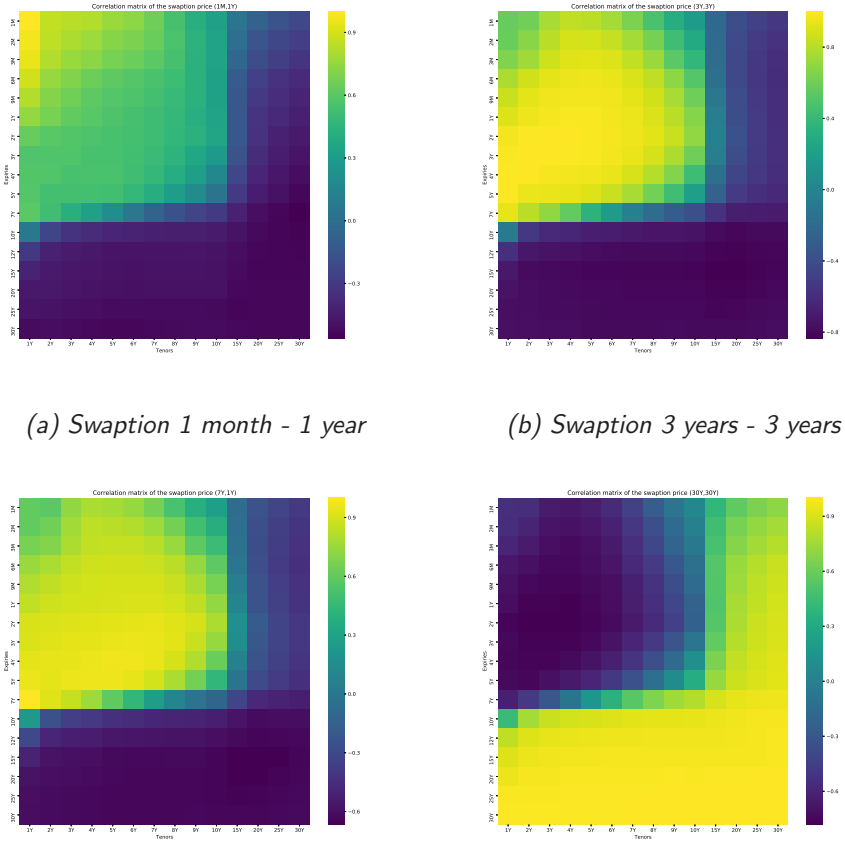


Figure 5.2: Visualization of the price matrix relative to the date 2013-06-28

days between the date of the curve point and the reference date. This new representation makes easier the manipulation of the curves, in particular for the computation of the feedback function.

At this point, we can start analyzing the data from the swaption prices. A representation of the matrix for the first date *2013-06-28* is provided in Figure 5.2. As we can see, the price of a swaption is more similar to the closest ones in the matrix, thus the correlation of close swaptions is higher than the correlation of the far ones. In addition, from the Figure 5.2 we can see that the price has a gradual and smooth variation over the whole matrix. This behaviour is expected, in fact close cells in the matrix represent swaptions with similar expiries and tenors. This effect can be seen in the correlations of single swaptions, identified by the couple expiry - tenor, with respect to the other ones, as is represented in Figure 5.3. In this figure we show the correlation of four different swaptions, one in each subfigure, with respect



(a) Swaption 1 month - 1 year

(b) Swaption 3 years - 3 years

(c) Swaption 7 years - 1 year

(d) Swaption 30 years - 30 years

Figure 5.3: Four matrices representing the correlations between the prices of a swaption, identified by the expiry-tenor couple, and all the other ones.

to the other swaptions and we can clearly see the yellow areas where the correlation is high, and the correlation decreases while moving away from the considered swaption.

In Figure 5.4 we show the correlation matrix between all the swaptions, after that the matrix has been flattened. In this figure we can see some patterns that are due to the fact that the matrix has been flattened and we use a row major representation, reflecting the behaviour we observed in Figure 5.3.

The same procedure can be applied to the matrix of the volatilities, that can be visualized in Figure 5.5. From this figure we can clearly see that the most volatile swaptions are the ones closer in time to the reference date,

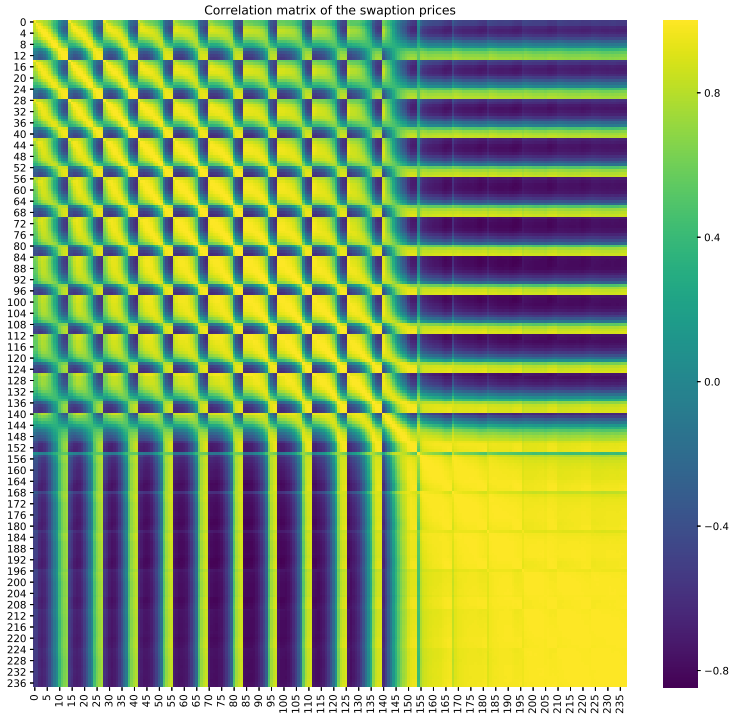


Figure 5.4: Correlation matrix of the flattened matrix of the swaption prices

and the swaption volatility decreases the more we are moving further in the future.

As we can see, the volatilities change gradually over the matrix and swaptions that are close to each other have similar volatilities like in the price matrix. This behaviour can be seen in Figure 5.6 that shows the correlations of single swaptions, identified by the couple expiry - tenor, with respect to the other ones. Here we can see a behaviour similar to the prices, where the volatilities of swaptions close to each other are highly correlated.

In the same way as the prices of the swaptions, we show the correlation matrix relative to the flattened volatility matrix in Figure 5.7. Here we can observe the effect due to the row major representation of the matrix, and that the correlations of the volatilities are stronger than those of the prices.

In addition to the prices and the volatilities of the swaptions, we have also

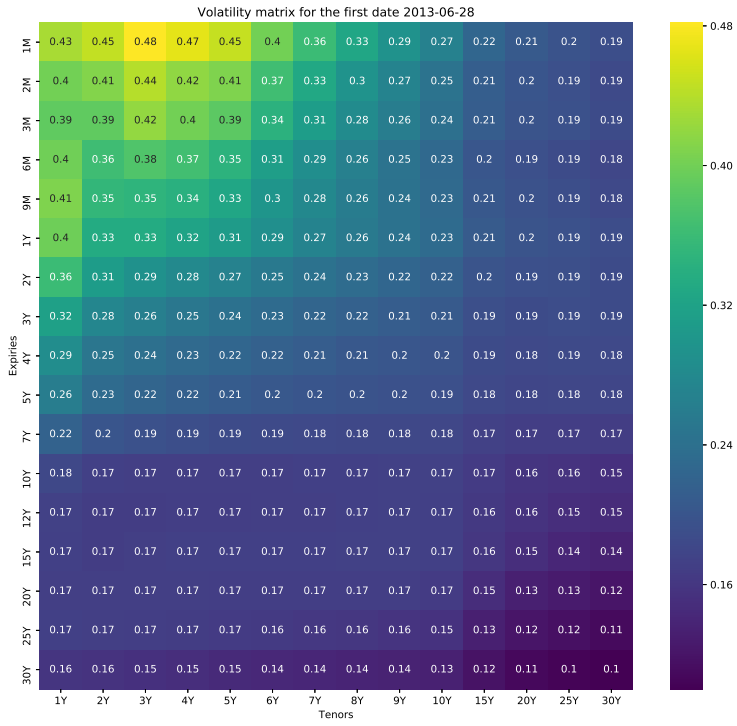
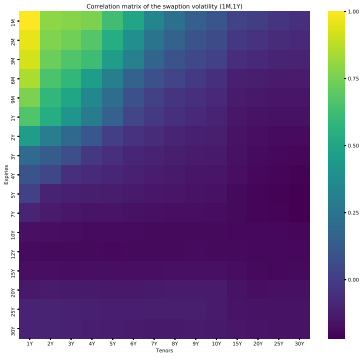
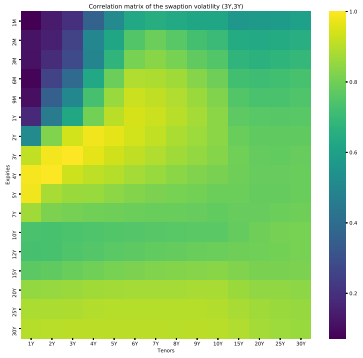


Figure 5.5: Visualization of the volatility matrix relative to the date 2013-06-28

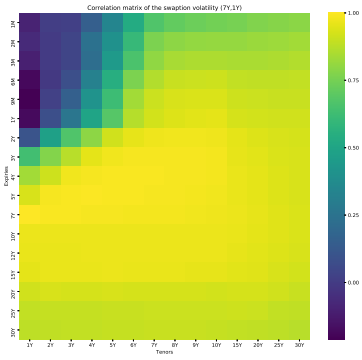
the discount and forward curves relative to the reference date. These two curves are made of around 60 points per date, and they cover a time span of roughly 20000 days starting from the reference date. The distribution of the available points (delta days) is plotted in Figure 5.8 for both the discount and forward curve relative to the first reference date 2013-06-28. The points are more frequent in the first part of the curve, the one closer to the reference date, because there an higher precision is needed. On the other hand the points become sparser while moving in the future. The curves have a mostly regular behaviour that can be seen in Figure 5.9.
Despite we have the data relative to the discount and forward curves at our disposal, we are not going to use them in our solution because we observed that they do not add useful information to the model. In addition, in this way we are able to have a lower dimensionality of the data, helping a lot the training of the model.



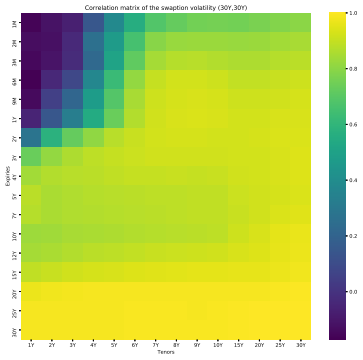
(a) Swaption 1 month - 1 year



(b) Swaption 3 years - 3 years



(c) Swaption 7 years - 1 year



(d) Swaption 30 years - 30 years

Figure 5.6: Four matrices representing the correlations between the volatilities of a swaption, identified by the expiry-tenor couple, and all the other ones.

Finally, there are the two parameters of the Vasicek model: mean reversion speed and volatility. These are the two targets that our model is going to predict. Their evolution in time is represented in Figure 5.10. From this plot, we can see that the two parameters are somehow correlated between each other, even if they are on different scales.

An interesting representation of the mean reversion speed and the volatility over time is provided in Figure 5.11. We can see that the distribution of the points changes over time moving in the 2D space, probably because of the evolution of the market, and this behaviour could be useful for our regression.

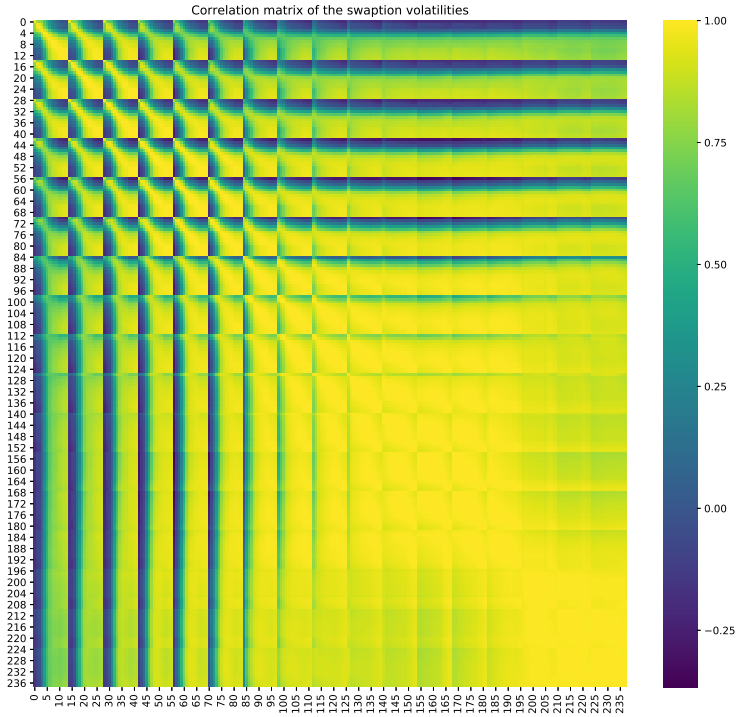


Figure 5.7: Correlation matrix of the flattened matrix of the swaption volatilities

5.5 Dimensionality Reduction

After the exploration, we want to reduce the dimensionality of the data. We already described the Principal Component Analysis (PCA) algorithm and some of its benefits in Section 3.2.1. In our particular case, the number of dimensions of the data is very high with respect to the number of samples available. In fact, after the flattening of the price and volatility matrices, we have 238 features for each matrix, that sum up to 576 total features. This number of features is obviously too high for the number of samples we have, that is around one thousand. For this reason we have to reduce the number of dimensions.

For the PCA step we have different options. First, we have to decide whether to perform PCA on the prices and volatilities in a separated way

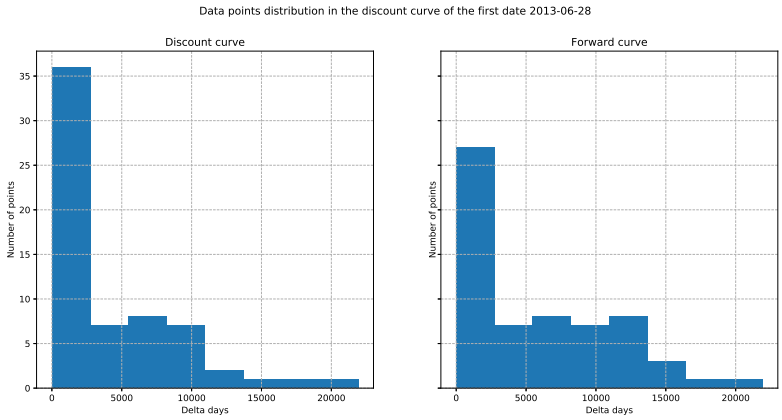
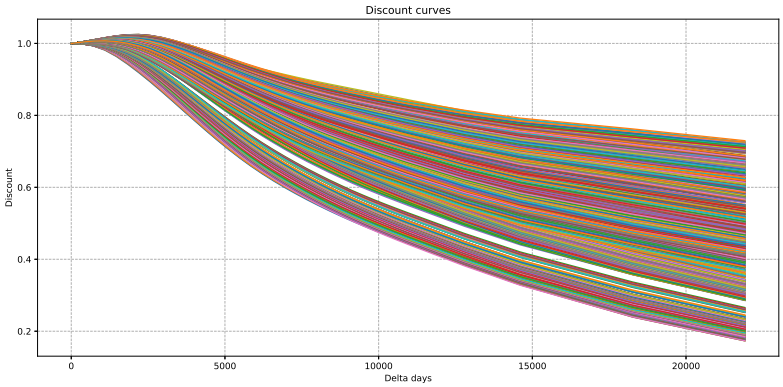


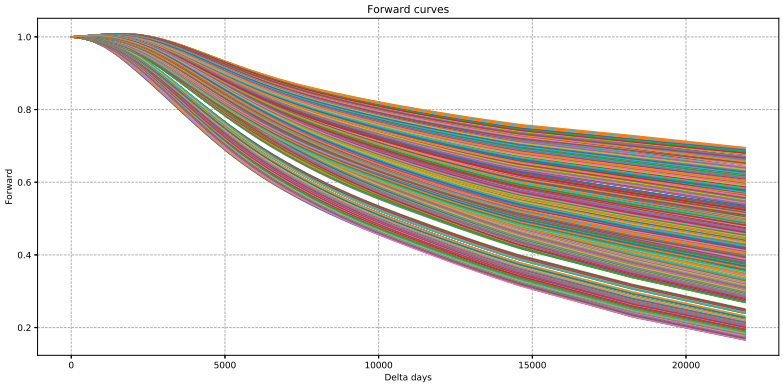
Figure 5.8: Distribution of the curve points for the date 2013-06-28.

and then merge the provided principal components, or apply PCA to the prices and volatilities together. The difference between the two is that in the former case we would keep the two data spaces divided and exploit only the variances and correlations internal to the prices and the volatilities. In the latter case, we would be able to exploit all the correlations between prices and volatilities, being able to reduce more the number of dimensions and to produce a more compact representation of the same data.
We have performed PCA in both the cases to compare and understand the results in order to choose the best representation possible. The principal components of the two cases are presented in Figure 5.12. In this figure we can see that the first principal component of the prices captures alone the 95% of the explained variance, while the other three the remaining 4%. For what concerns the volatilities the situation is different, in fact we have 8 principal components and the first one captures roughly the 78% of the explained variance. In addition, we can note that the PCA performed on the volatilities and the PCA performed on the prices plus the volatilities produce almost identical results. This is because the volatilities dominate the prices in the PCA, and this could lead to a loss of the information given by the prices. For this reason we decided to use the principal components given by the PCA performed separately on the prices and the volatilities.

The second important choice is whether to normalize the data before performing the PCA or not. This depends on the different cases, and in our situation we decided to not normalize the data before and so to perform the PCA on the raw prices and volatilities. We choose that option because we



(a) Discount curves



(b) Forward curves

Figure 5.9: Plot of the discount and forward curves for all the dates.

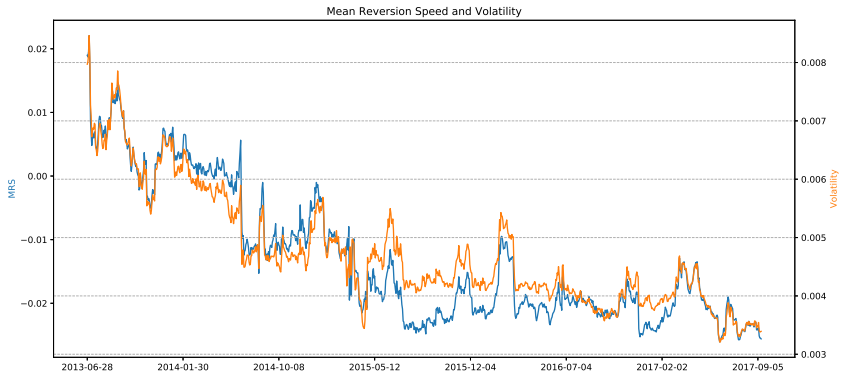


Figure 5.10: Plot of the mean reversion speed and the volatility from 2013-06-28 to 2017-09-05.

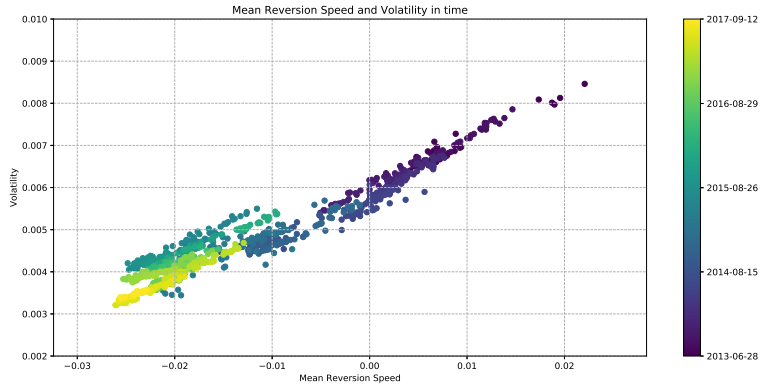
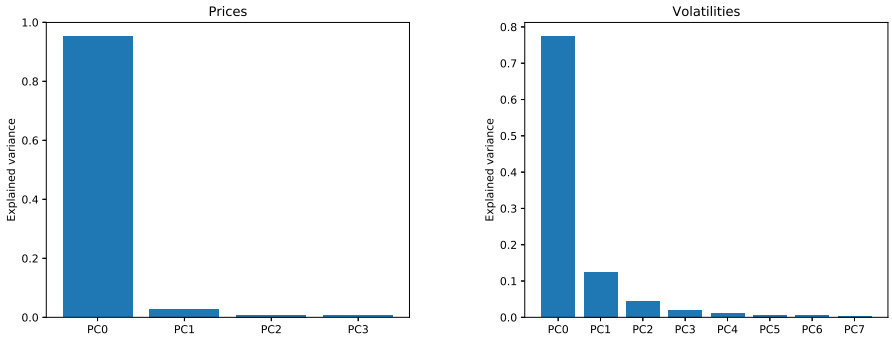
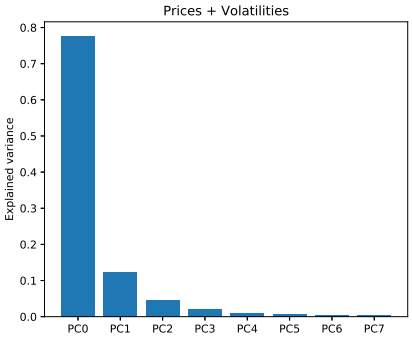


Figure 5.11: Two-dimensional representation of the model parameters evolution over time. On the x-axis there is the mean reversion speed, on the y-axis the volatility and the color represents the reference date of the sample.



(a) Prices

(b) Volatilities



(c) Prices and volatilities

Figure 5.12: Principal components of the PCA performed on: (a) the swaption prices, (b) the swaption volatilities and (c) the prices and the volatilities together

are performing the PCA separately on the prices and volatilities, and we are not performing the PCA on heterogeneous features that could have different measurement units or ranges. In fact, the normalization is required in the cases where some features are measured differently from the others (i.e. times in seconds and distances in meters) because they could have significantly different ranges that would bias the PCA. Probably we would need the normalization if we were performing the PCA on the merged prices and volatilities, in order to scale the variances and the ranges.
At the end, the principal components we choose to build our dataset are the ones provided by the PCA performed sperately on the prices and the volatilities, that are represented in Figures 5.12a and 5.12b.

At this point we can analyze the composition of the principal components given by the PCA. A principal component is a linear combination of the

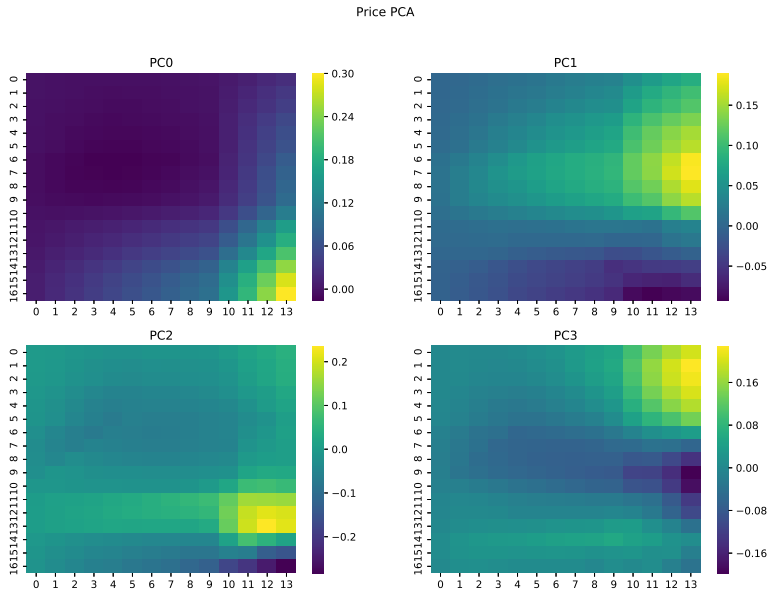


Figure 5.13: Contributions of the original prices to the four principal components provided by the PCA

original features, so we can visualize the contributions of the original dimensions to better understand what are the most important for the PCA. The visualization of the price principal components is provided in Figure 5.13.

In the figure we can see that the swaptions that contribute the most to the first component are the ones with higher expiries and tenors, this is because they have the highest variance. Then, the other three components concentrate in other areas of the matrix, especially on the right part where we have high tenors.
On the other hand, for the volatilities we have eight principal components, but we are going to represent the compositions of the first four in Figure 5.14.

PCA components provide also a nice way to visualize high dimensional data, that otherwise would be impossible to represent. In fact, by plotting the first two principal components we can have an idea of the distribution of the data and of their evolution over time. The plot of the first two principal components of the prices is provided in Figure 5.15, and the first two components

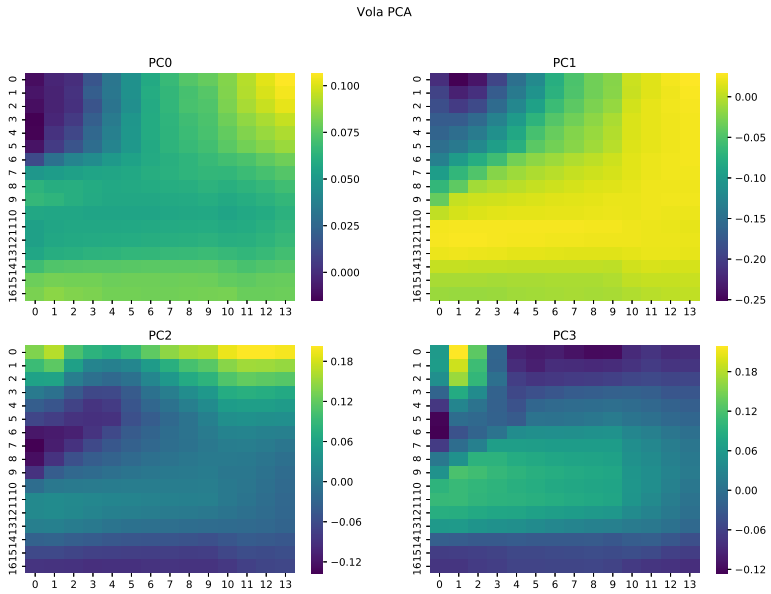


Figure 5.14: Contributions of the original volatilities to the first four principal components provided by the PCA

of the volatilities in Figure 5.16.

Figure 5.15: Representation of the prices with respect to the first two principal components

In the end, our final dataset for the training of the model is made of the four principal components from the PCA on the prices and the eight principal

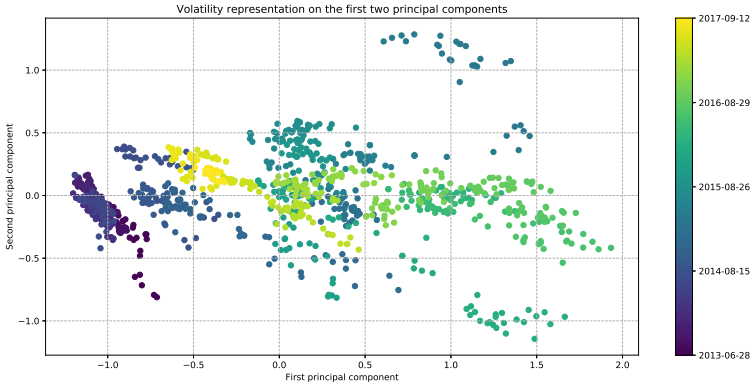


Figure 5.16: Representation of the volatilities with respect to the first two principal components

components from the volatilities, for a total of twelve features.

5.6 Feedback Implementation

After the dimensionality reduction, a huge part of the effort was focused on the implementation of the feedback function on the GPU using CUDA. The mathematical steps of the computation are described in Section 2.4.1.
This step was necessary because the computation time of the feedback was too high for our time constraints. In fact, the computation on the CPU takes around 690 seconds per one thousand dates and we have to perform several thousands of evaluations of the feedback function over roughly one thousand dates, and this is clearly unfeasible. For instance, one thousand evaluations over one thousand dates would take 690000 seconds, equivalent to roughly 8 days.

Here we are going to explain the important steps in the implementation of the feedback function.
First of all we had to choose which framework to use, and the choice fell on CUDA, in particular we chose a Python wrapper of CUDA because the entire project is developed in Python.
The first step is to organize the parallel computation in kernels, blocks and grids. The threads are the execution units relative to the execution of each kernel. We have to compute the feedback value for several dates, and for each date we have to compute the matrix of the model prices. Luckily the choice for the kernel is not very hard here, because for each date we have

a matrix of prices to compute, so we can have one kernel to compute the model price for each cell of the matrix, i.e. for each swaption. In fact, the computation of each model price of the matrix is independent, so it can be perfomed in parallel for each cell. Thus, the kernel takes the inputs and compute the price of a cell of the matrix.
Now that we have defined our kernel, we have to group the kernels in blocks. This is a important step because it directly affects the efficiency of the computation and the scheduling of the kernel executions. We have to define a block size such that the number of kernels inside is not too low or too high, in order to maximize the utilization of the GPU multiprocessors. In fact, a GPU has a limited number of Streaming Multiprocessors (SM) and, in each of them, only a limited number of blocks and a limited number of warps can be executed at the same time. The warps are the executors of the kernels, and usually they can execute a maximum of 32 kernels at the same time. For the limit of the concurrent active blocks, if the number of kernels in the block is too low, then only a limited number of kernels will be executed at the same time and not all the warps in the SM will be utilized, leading to a low utilization of the GPU. For instance, if the SM can execute 16 blocks and 64 warps and we have blocks made of 32 kernels (1 warp), then in the SM we could use only 1 warp per block, so 16 warps per SM would be active, leading to the following occupancy

$$occupancy = \frac{\#active\ warps}{\#total\ warps} = \frac{16}{64} = 0.25 = 25\%$$

In this case, increasing the block size to 128 kernels leads to 100% occupancy because there are 4 active warps per block and 16 active blocks per SM, so in total we have $16 * 4 = 64$ active warps per SM.
Another limitation for the occupancy is the number of registers per SM, that are the registers shared by all the active threads (kernels) in the SM. If the kernel uses too many registers during its execution, then the number of active threads will be limited by the maximum number of available registers. As a consequence, the number of warps would be limited, leading to a non-optimal occupancy. The solution is to reduce the number of registers used by each kernel by introducing optimizations in the code and, in the case the code optimizations are not enough, by limiting in the CUDA compiler the number of registers to be allocated to each thread. In this way the scheduling of the threads into the warps would not be limited by the number of used registers, but the execution time of the single threads would increase because they would have less registers to use than needed, so they would perform more accesses to the memory, that is slower than

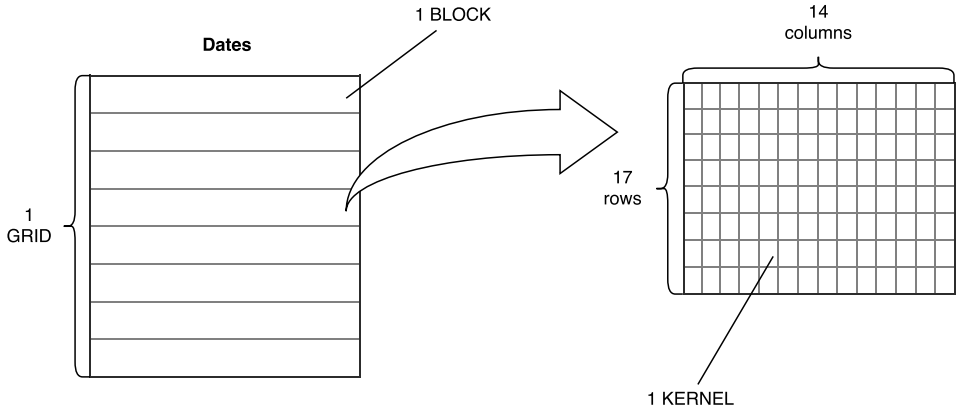


Figure 5.17: Organization of the kernels in blocks and grids

the registers. For this reason, the optimal solution is a trade-off between the number of registers used per thread and the total execution time, and the optimal number of registers can be found experimentally by tracing the computation times with kernels compiled with different numbers of registers per thread.

After these considerations, let's consider the kernels organization in our solution. In our implementation we have a set of dates, and for each date we have to compute the matrix of model prices. Considering this structure, in our solution the kernel computes the price of one swaption in one date, i.e. one cell of the matrix. The choice of the organization of the kernels in blocks is pretty straightforward, and we choose to make one block for each date. In this way one block contains one matrix of kernels, so one block contains $17 * 14 = 238$ kernels. This is a reasonable value for the block size for what concern the theoretical occupancy that could potentially be 100%. After that, we group all the blocks in one single grid, because have no need to perform the computation on multiple GPUs. In fact, all the blocks contained in one grid must be executed on the same GPU. For this reason, if it was needed to perform the computation on multiple GPUs, then we would have to split the blocks into more grids, so that each grid could be executed on a different GPU.
The final organization of the kernels for the feedback computation is represented in Figure 5.17.

5.6.1 Data Preparation and Optimizations

Now that we have defined the structure of the kernels in blocks and grids, and the task to be executed by each kernel, we have to prepare the data for the actual computation of the prices.
The first thing to do is to prepare the input data to have the right shape such that the kernels can identify their own inputs. In fact, a kernel is identified by its 3D coordinates inside the block, but in our case we use only two dimensions, and every kernel can see all the data. Thus, the kernel uses its coordinates to identify its inputs, so the inputs must be shaped in a coherent way with respect to the kernel structure. There are six main inputs that must be prepared for the feedback computation, that are:

- Floating payment dates: list of the dates when the payments of the floating leg are performed
- Fixed payment dates: list of the dates when the payments of the fixed leg are performed
- Discount curve: the OIS curve, that is needed to compute the OIS values for different dates
- Forward curve: the FWD curve, that is needed like the OIS curve to compute values in different dates
- Mean reversion speed: parameter of the Vasicek model
- Volatility: parameter of the Vasicek model

These are all the inputs needed to compute the model price of each swaption. The first two inputs, i.e. the payment dates, are different for each kernel for each block, so we need to compute them for each kernel and organize them in a list of matrices, one matrix for each date, where the cell of the matrix contains the payment dates to be used by that particular kernel.
The other inputs are common for all the kernel of one date, so can provide them as shared inputs to the blocks. This can save some memory and time in data transfer between the CPU and the GPU.

An additional analysis is needed for the discount and the forward curves. In fact, one of the main limitations of using CUDA, or every GPU framework in general, is that the majority of Python libraries cannot be used in the kernels. The problem is that we have to interpolate the curves and then sample the values for new dates during the computation in the kernels. We

noted that in all blocks the kernels were using the same dates for the curve values, so the solution we came up with is to find all the dates for which we need the discount and forward values during the computation and save them. Then, for each date we precompute the interpolated values and we pass these values as inputs to the kernels as a shared input at block level, and each kernel can find the needed value for a date by performing a binary search. In this way we have a double advantage, we save time by precomputing the interpolated values on the CPU for all the used dates and we reduce the complexity of the computations on the GPU.

One important part in the feedback implementation is the step where we perform the search of $\bar{r}$, that is the value of the interest rate that makes zero the Net Present Value (NPV) of the swap at T_{exp} (Section 2.4.1). In fact, this step requires a search for the zero of the NPV function of the swap and we apply a numerical root-finding algorithm to find the value of $\bar{r}$, in particular the bisection method because the NPV function is monotonic. The bisection algorithm involves several evaluations of the NPV, so the optimization of this function is crucial. For this reason, instead of computing the NPV like in the Equation 2.2, we compute it with a sum using a list of coefficients. The point is that we can precompute the coefficients in the kernel before calling the NPV function and the bisection method, in this way we can compute only one time the coefficients that, without this optimization, should be computed at every call of the function. In the end, this optimization yields a good speedup for the whole feedback function.

5.6.2 Performance

The performance of the GPU implementation of the feedback value is very good, both in terms of errors on the original feedback values and of computational time. In fact, the errors made by our function are acceptable, with an average of percentage error of less than 0.5%. All the errors for the feedback computed over the Euro dataset are reported in Figure 5.18.
From the figure we can see that for more than 75% of the dates the error is lower than 1%, that is acceptable. In particular, we can accept these errors if we consider the huge speedup we achieve. In fact, the GPU implementation of the feedback takes 0.48 seconds for 1000 dates, with respect to the previous CPU implementation that takes 690 seconds for 1000 dates, yielding a speedup of over 1400 times. This is a remarkable result, because without it our method would have been impossible.

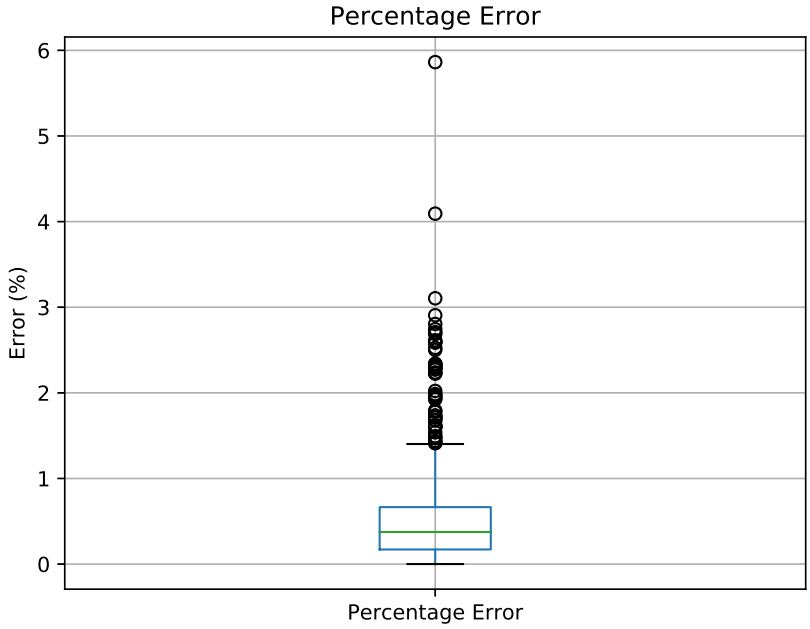


Figure 5.18: Organization of the kernels in blocks and grids

5.7 Model

With the new GPU feedback and the final dataset ready, we can move to the definition of our model. We want to approximate the function f that from the market prices and the volatilities computes the model parameters mean reversion speed k and volatility σ. This is the function that we want to approximate with our model. For this task, we choose to use a feed-forward neural network (FNN, explained in Section 3.2.2) because of its big representational power. Our choice of the model is limited by the fact that we need a parametric model because we need a model that can be entirely defined by a limited and exact number of parameters. In fact, in the learning process we are going to find the optimal set of parameters such that the feedback function computed from the model parameters is minimized, and we have to be able to create our model from a given set of parameters. An additional advantage of a parametric model is the speed in the evaluation of new samples, in fact a FNN requires just a multiplication of few small matrices.

In order to define the hyperparameters like the number of layers, the activa-

tion functions of the hidden and output layers, and the number of neurons in the hidden layers we perform a first grid search and then we refine the search with some manual educated tests because of the high training time of the model. In fact a full grid search would be infeasible due to the high computational times.
In the model selection process we faced some problems due to the high variability of the results even for small changes in the size of the training set, that is probably due to the instability of the model. One possible reason could be that we have a small dataset at our disposal, that is composed by daily samples of real market data, so it would not be easy to extend the dataset significantly without the addition of other currencies. The main problem to the extension of the dataset is that the increase in size of the training set brings a proportional increase in the computation times. For this reason we should try to solve the instability problem without increasing too much the size of the training set due to time constraints. We see that even with small changes of roughly 10% in size of the training set, the model couldn't learn anymore the output parameters, predicting a constant value. The possibile solution to this issue is to carefully tune the network structure taking into account also the number of dates on which we are training, in order to find the optimal number of layers and number of neurons per layer. In addition, we are trying to keep the network as simple as possible, in order to have the minimum number of parameters to train, because the training time grows pretty fast the more parameters we have.
Another problem to account is that with small variation of the number of neurons the results change drastically, adding difficulty to the model selection.
The first obvious choice for the structure of the network is the size of the output layer. We have two output neurons, one for each model parameter to predict, and a linear activation function because we are performing a regression on two scalar values with no particular constraints.
The input layer is made of twelve inputs, that are the principal components given by the PCA on the prices and the volatilities.
For the choice of the activation function of the one or more hidden layers, we found that the ReLU function have a good behaviour and the model can fit the data. Actually, different activation functions bring very similar results, so this choice is not very critical for the training process.
The most important and tricky choice in the network structure is the number of hidden layers and, in particular, the number of neurons per layer. Keeping in mind that we want the network to be as simple as possible, we started with one hidden layer with an increasing number of neurons, until

no significant improvements are found. The results of this test suggest that a single hidden layer with five or seven neurons is enough to capture all the patterns in the data and to ensure that the model could perfectly fit the data. As we mentioned earlier, the optimal number of neurons in the first hidden layer is very sensitive to the data, so using different data could yield a different optimal number of neurons, for example with the addition of other currencies to the dataset, as an attempt to reduce the instability of the learning process by increasing the available samples used for the training.

In the end, the result of the model selection procedure is a feed-forward neural network with twelve inputs, one hidden layer with seven neurons and ReLU activation function. The output layer is made of two neurons with linear activation functions, one for each Vasicek parameter to predict.

5.8 Training

In the previous section we have defined the model, now we are going to describe the learning procedure. We have a feed-forward neural network as model, but unfortunately we cannot use the backpropagation algorithm for the training phase because of the nature of our problem. In fact, despite backpropagation can be very good, it is designed for supervised learning problems, that is not our case.
Our goal is to find the optimal parameters θ, i.e. weights and biases, of the network that is approximating the function f. Since we cannot use backpropagation because the objective function g is non-differentiable, we have to use alternative optimization algorithms like Cross Entropy, BFGS and L-BFGS (explained in Sections 5.10.1 and 5.10.2). These are numerical optimization algorithms used to find the global minimum of a given function. For this reason, we have to reconduct our learning problem to the minimization of a function $g(\theta)$ that takes the parameters θ as inputs and produces the feedback value as output. The function g will be our objective function to be minimized.
In this function, first we take the parameters θ and put them in the network, then we predict the parameters k and σ for the training set using the updated network. After that, we compute the feedback values for all the dates in the training set using the Vasicek targets k and σ we just predicted, and finally we compute the final value of the function as the mean of all the feedback values.
At the end, given the network parameters θ with $|\theta| = n$, our goal is to

minimize the function

$$g : \mathbb{R}^n \to \mathbb{R}$$

We are dealing with temporal data, i.e. daily samples taken from the market, so we want to train and evaluate the model in a way that is closest as possible to the actual use in production. For this reason, we split the training into two phases: the offline and the online training.
We split the dataset in two parts, the first one is used for the offline training where a step of the optimization is performed and the second one is used for the online phase, where the model is trained for each new sample added to the training set. In this way we can simulate the real world scenario, in fact the bank needs the calibrated parameters k and σ on each new day, and these are provided by a single evaluation of the network. By adding an online phase, we retrain the network with the new samples potentially on each day, but the interval between the online updates can change. With this procedure the network can be updated with the latest samples and can react to the changes in the market.
A more detailed explanation of the offline and the online phases is provided in the following sections.

5.8.1 Offline Learning

In the previous section we mentioned that we split the dataset in two parts, that we can name offline and online data.
All the data preparation and dimensionality reduction are performed at this step on the data, but we fit the PCA only to the offline data, that would be the only data available at this step in the real application.
In the offline training we perform a minimization of the objective function $g(\theta)$ based on the offline data. We use a combination of two numerical optimization algorithms: Cross-Entropy (Section 5.10.1) and BFGS (Section 5.10.2). There are two reasons why we are using two different algortihms, the first one is that we have no important time constraints for the offline phase, in fact it is performed when the model is deployed and can be considered as a sort of initialization of the system. The second reason, a consequence of the first one, is that we want to perform a very good optimization without caring too much about the time, so we start it with the Cross-Entropy, that is a method that performs a very good exploration of the input space in order to find the best area where the global minimum could be, and then we perform a refinement step with the BFGS, that is a method where the solu-

tion is found by following the gradient of the objective function. The BFGS method is initialized with the final solution found by the Cross-Entropy, so with this combination we are more confident to find a good vector of parameters θ that minimizes the objective function $g(\theta)$.

In order to evaluate the result of the training process, we predict the target parameters k and σ for the first date of the online data to compare the error between the feedback computed from the predicted parameters and the original one. Of course, we can do this comparison because we have the original calibrated parameters k and σ and the relative feedback value, so we can exploit them, but in general in our method we should not rely on these values. In this case, we are using the original feedbacks as a benchmark to evaluate our solution, but in a general case with a different interest rate model from the Vasicek, our only evaluation metric would be the feedback computed from the predicted target parameters.

5.8.2 Online Learning

After the offline training is completed, we start the online part. Here, we apply iteratively the same steps of the offline training, one time for each sample of the online data. In detail, as the first step of the online training, we train the network on the offline data plus the first sample of the online data, and we evaluate the result on the second sample of the online data, comparing the feedback computed from the predicted target parameters k and σ and the original one. After that, we repeat the procedure starting from the last optimal vector of parameters θ produced by the last optimization, we add also the second sample to the dataset, we train on the new data, we evaluate on the third sample and so on. This procedure is reapeated for all the dates in the online data, simulating the daily adjustment of the network parameters θ that would be performed on a daily basis by the bank.

The online optimization step is performed by using the L-BFGS algorithm (Section 5.10.2), that is a version of the BFGS that uses some numerical approximations to make the optimization faster and lighter in terms of memory, at a cost of a less precise solution. This is a trade-off, in fact in the online phase we have time constraints to satisfy, because the bank should perform one step on each day, so the time is an important factor in the choice of the optimization algorithm to use.

We use a customized versione of the L-BFGS algorithm in order to speedup

the optimization. The custom part is in the computation of the gradient, that in the original version requires one function evaluation for each dimension of the vector θ. This evaluation is the most expensive part of the algorithm, so we want to minimize the number of the evaluation of the objective function. The solution is to use a technique called *Finite Difference Gradient Estimation* to estimate the gradient. In this method, we generate a number N of random perturbations around the point P for which we want to compute the gradient, where N is less than the number of dimensions, and we compute the gradient along the directions given by the perturbations and the point P and finally we perform a Ridge regression to find the gradient.

We have said that L-BFGS is an approximation of the original BFGS algorithm, so the goodness of the solution will not be as good as the one provided by BFGS, and even less good than a solution given by the combination of Cross-Entropy and BFGS. For this reason, after some time, e.g. weeks or monts, from the beginning of the online phase, the system could be restarted by performing again the offline training with all the past offline and online data. This could prevent the model to lose precision over the long term because of the approximation introduced by the online steps.

5.9 Multi-currency

During the analysis of the problem, we found out that the function f that we are trying to approximate with our neural network is not dependent on the currency from which we are taking the samples used for training. In fact, the function should only depend on the market prices and the market conditions, so we could exploit this property to create a larger dataset and merge more currencies together.
For this purpose, we merge the daily data relative to three currencies, that are Euro (EUR), US Dollar (USD) and Swiss Franc (CHF). For the same reason of before, we can take all the dates we have, without paying attention to select only the samples relative to common dates.
The only problem with the samples taken from different currencies is that they refer to different set of swaptions, so they have different lists of expiries and tenors. The solution we adopt in order to use all the data together is to keep only the intersection of the swaptions, that are identified by the expiry-tenor couple, and discard all the other ones.
In our solution, we just merged the data so that they are ready to be used in the training process, but this application will be further devolped in future

work, as it is explained in Section 7.1.

5.10 Optimization Algorithms

In Section 5.8 we have explained the training procedure used to fit the model to the data. In this procedure we use some numerical optimization algorithm to minimize the objective function, that is our $g(\theta)$ function.
In this section we are going to present the three main optimization algorithms we use: Cross-Entropy, BFGS and L-BFGS.

5.10.1 Cross Entropy

The cross-entropy method (CE) is an optimization algorithm used for finding the global minimum of noisy functions. [Rubinstein, 2004]
Cross-entropy performs a very good exploration of the search space, with respect to other gradient-based algorithms. For this reason, it can help in finding the global minimum instead of being stuck in a local one.

In our solution we used a slightly customized version of the cross-entropy method, that we have implemented in Python. The algorithm consists of two main phases

1. Sample data points from a normal distribution
2. Update the distribution parameters (i.e. mean and covariance) based on the sampled data in order to produce better samples in the next iteration
3. Repeat until the termination condition is met

The custom part of the algorithm is that we don't only use the data sampled in the current iteration to update the normal distribution parameters, but we also use the best K points we found during the full run of the method.

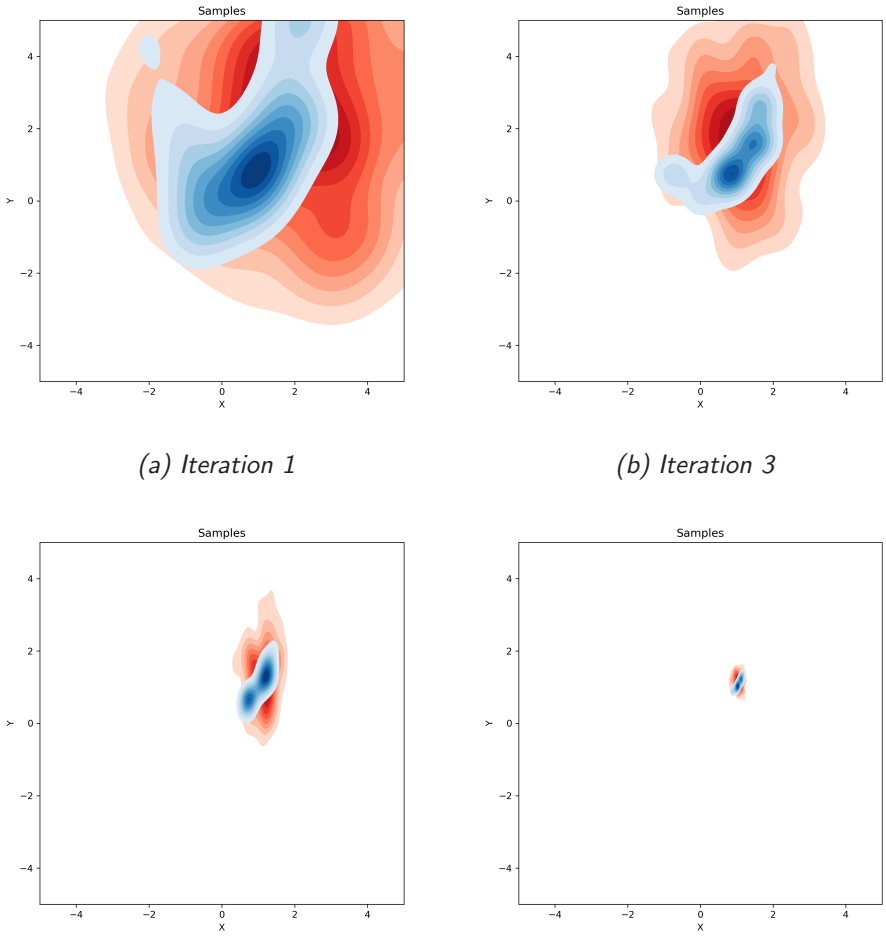
At the beginning, the normal distribution is initialized with some initial values for the mean and the covariance. After that, we obtain N samples from the current normal distribution and sort them based on the value of the objective function to be minimized. Then, the best N_e points of the N sampled, together with the K best ever found points, are used to update the mean and the covariance of the normal distribution. At this point, the termination condition is checked, so the algorithm will end if the variance of the normal distribution is less than a given value ϵ. On the other case,

the algorithm will proceed with the generation of the next N points from the updated normal distribution.
A visualization of the distributions of the data points during different iterations of the algorithm is provided in Figure 5.19.

5.10.2 BFGS and L-BFGS

The Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm belongs to the class of quasi-Newton methods, and it is an iterative method for solving non-linear, unconstrained optimization problems. [Fletcher, 1987]
The algorithm searches for a stationary point of the function, where the necessary condition for the optimality is the gradient to be zero. In general, for quasi-Newton methods, the termination is guaranteed only if the function has a quadratic Taylor expansion near the minimum, but it has been observed that BFGS has good performance even for non-smooth optimization problems.
Quasi-Newton methods like BFGS use a variant of the secant method to find the root of the first-order derivative. BFGS uses only the first-order derivatives and an approximation of the inverse Hessian matrix.

Limited-memory BFGS (L-BFGS) is a variant of the BFGS algorithm that approximates the behaviour of the original algorithm by reducing the memory used during the optimization [Byrd, 1995]. In fact, like BFGS, L-BFGS uses the Hessian matrix to guide the search of the minimum but it stores a very compact representation of this matrix, while the original BFGS algorithm stores a NxN representation of the inverse Hessian matrix, where N is the number of dimensions of the problem. For this reason, the L-BFGS algorithm is better suited for problems with a large number of dimensions and, in addition, it requires less time to converge, but at the cost of a less accurate solution.



(a) Iteration 1

(b) Iteration 3

(c) Iteration 7

(d) Iteration 10

Figure 5.19: Four iterations of the cross-entropy method. The red area shows the distribution of the N data sampled from the normal distribution, while the blue area shows the distribution of the N_e best data points and the K best ever found points.

Chapter 6

Experimental results

In this chapter we are going to present and comment the results of our method, with the main problems we have faced and the solutions we have found. The results are produced by the online evaluation of the model, by retraining the network using the online approach described in Section 5.8.2.

At the beginning, we tried with a network with two hidden layers with eleven neurons each, and the results are provided in Figures 6.1 and 6.2.
In Figure 6.1 we can see that feedback computed from the predicted parameters η is pretty close to the feedback computed from the original calibrated parameters, and in the same way the absolute errors represented in Figure 6.2 are not too high.
This could make us think that this is close to a good solution, but if we look at the predicted parameters we see that the network actually is not learning anything and that the parameters (k, σ) are constant. The parameters are represented in Figures 6.3 and 6.4.

As we can see, the parameters k and σ are constant, but the feedback computed from these parameters is actually not far from the original one.
The Root Mean Squared Errors for this test are reported in Table 6.1. The RMSE between two vectors y and $\hat{y}$ is defined as

$$RMSE(y, \hat{y}) = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}}$$

At this point we realize that there is something wrong in the procedure. The reason of this problem is that the interest rate model we are using is not very sensitive to the model parameters, in the sense that its expressive power is low and so the feedback function is not sensitive to the changes in

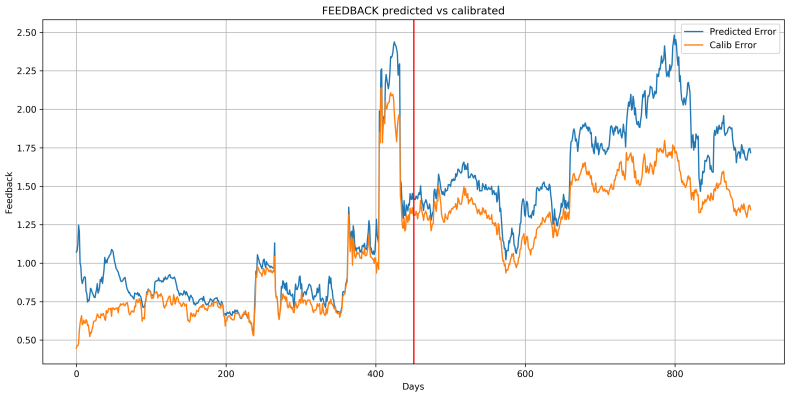


Figure 6.1: Comparison between the two feedbacks computed from the predicted parameters η and from the original parameters

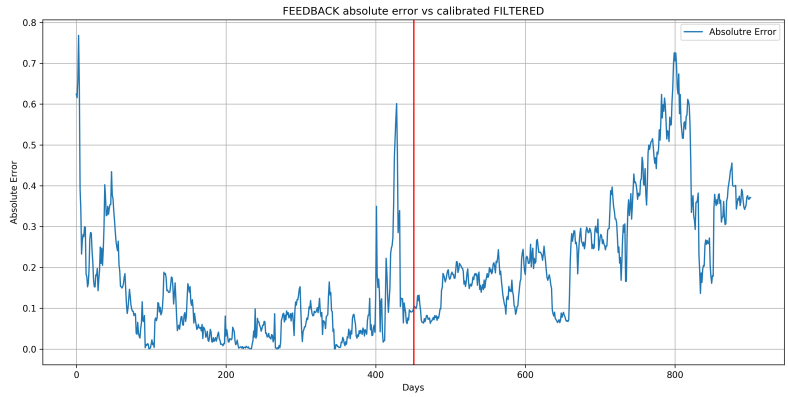


Figure 6.2: Absolute errors between the two feedbacks

the parameters. In fact, as we can see in the Figures 6.1, 6.3 and 6.4, the feedback looks not bad although the parameters are constant. This happens because even for the optimal model parameters, the error provided by the feedback function is relatively high because of the low expressivity of the model that is not complex enough.
The right solution to this problem would be to use a more complex interest rate model that would be able to better capture the dynamics of the interest rate, and so to provide a more accurate pricing of the swaptions, reducing the feedback error. The feedback of a more complex model would be more sensitive to the changes in the model parameters, making the optimization easier even in high dimensional spaces.

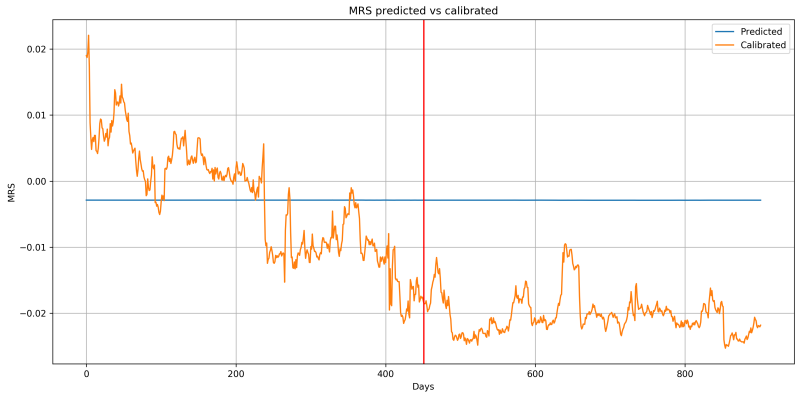


Figure 6.3: Predicted mean reversion speed vs the original one

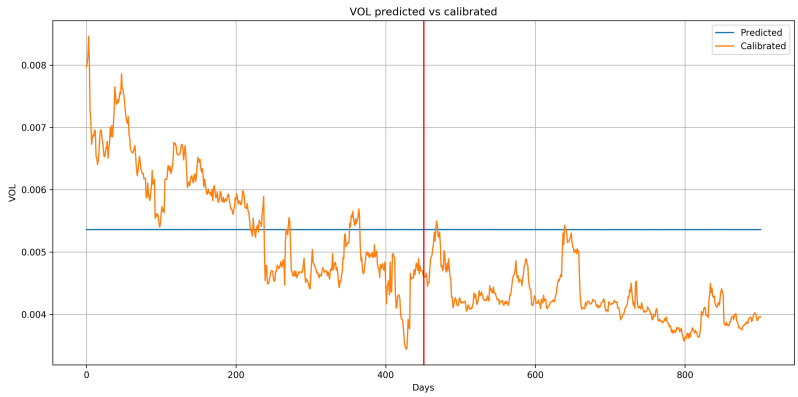


Figure 6.4: Predicted volatility vs the original one

Another reason of the poor performances is probably the high dimensionality of the input space of the function g that we are minimizing. In fact, the dimensions of the inputs are equal to the number of the parameters θ of the network, and we started with two hidden layers, that are equivalent to roughly 200 parameters. That is a very high dimensional space for an optimization, also considering the relatively small number of samples we have. For this reasons, we started to find a way to reduce the number of parameters of the network, i.e. reducing the number of hidden layers and/or the number of neurons per layer. In order to do that, we iteratively trained in a supervised way using the original calibrated model paramters networks with different number of neurons in the first hidden layer, and we found out that

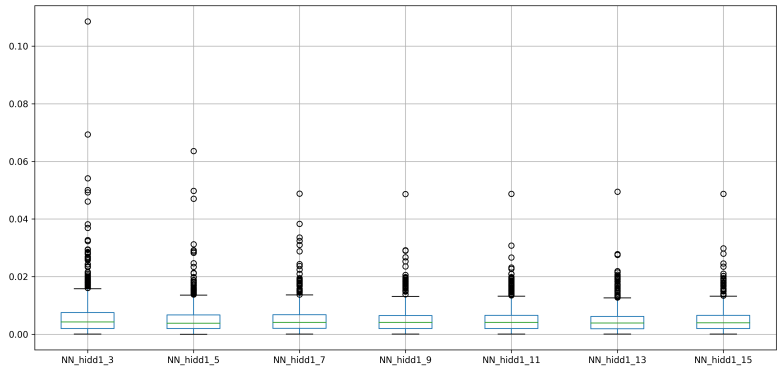


Figure 6.5: Boxplots of the absolute errors between the feedbacks for several configuration of the network. In particular, from left to right, there are networks with 3, 5, 7, 9, 11, 13 and 15 hidden neurons

around 7 neurons in a single hidden layer were sufficient to approximate our function f with a small error. Higher number of neurons in the layer do not bring great improvements to the solution.
In Figure 6.5, we can see the training errors of the networks for the different number of neurons in the hidden layer. Then, the resulting feedback and model parameters for the supervised experiments with one hidden layer made of seven neurons are provided in Figures 6.6, 6.7 and 6.8, where the red line represents the limit between the offline and online data, even if in this test no online training has been performed.

In Figure 6.6 we can note that the fitting of the feedback function is almost perfect, and in Figures 6.7 and 6.8 we can see that the predicter model parameters fit very well the orinal ones for the training data, and they start to diverge in the unseen data.
After these considerations, we can say that the network with seven hidden neurons is totally capable of representing our function f.
The RMSE resulting from the supervised test relative to the network with seven hidden neurons is reported in Table 6.1.

At this point, the problem we faced about the constant parameters was mitigated but still present in some particular situations, so we performed additional changes to the training procedure. In particular, we customized the Cross-Entropy optimization algorithm (Section 5.10.1) in order to keep the best solutions through all the iterations, and this helped in reducing the areas where the parameters were constant.

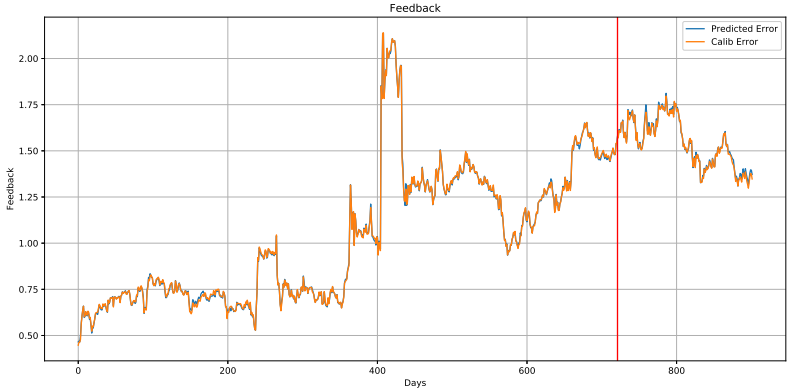


Figure 6.6: Comparison of the feedback computed from the parameters predicted by a network with 7 hidden neurons and the feedback relative to the original parameters

The results of the offline training over the whole dataset of a network with seven hidden neurons are provided in Figures 6.9, 6.10 and 6.11.

In Figure 6.9 we can see the feedback computed with the predicted parameters k and σ compared with the feedback of the original parameters. The fit is good and the error between the two feedback functions, that is reported in Table 6.1, is acceptable. In Figures 6.10 and 6.11 we can see that the network is able to capture the dynamics of the parameters and that the values are no more constant. The fit is not perfect, but the parameters that we are using as a benchmark are not guaranteed to be optimal and, in addition, there is still the problem of the low expressivity of the Vasicek model, that penalizes our learning process.
Overall, the results of this offline test are good and the potential performances with a richer interest rate model are even greater.
The resulting RMSE for the offline test is reported in Table 6.1.

Finally, we present some results of the online training for a network with seven hidden neurons, where we have performed the online updates with intervales of 50 days using the L-BFGS algorithm. In Figure 6.12 we can see the feedback of the predicted parameters compared with the original one, where the red line represents the limit between the offline and online phases. The feedback given by the predictions follows the original one with a small error, that is reported in Table 6.1. If we look at the model parameters k and σ in Figures 6.13 and 6.14, we can find again the problem we faced at the beginning of our tests, in fact the parameters are constant in some sets

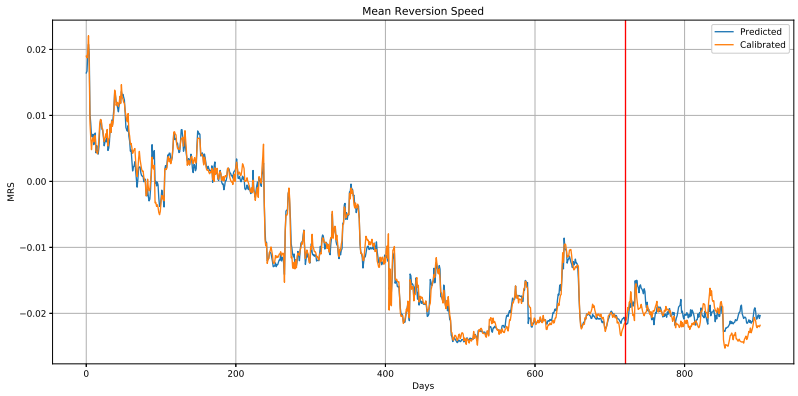


Figure 6.7: Comparison of the mean reversion speed k predicted by a network with 7 hidden neurons and the original mean reversion speed

Table 6.1: Offline and online RMSE (Root Mean Squared Errors) for the different tests that we have performed

		RMSE	
Test	**Offline Size (%)**	**Offline**	**Online**
Online with two hidden layers	50%	0.15587	0.30877
Supervised	80%	0.00732	0.01126
Offline only	100%	0.03165	—
Online	80%	0.03021	0.07248

of dates. As we have mentioned, this is a recurrent problem and the outcome of the training is very variable between different runs of the learning process, where for similar runs we can obtain very different results.

In the end, the errors of the four main tests we have performed are summarized in Table 6.1.

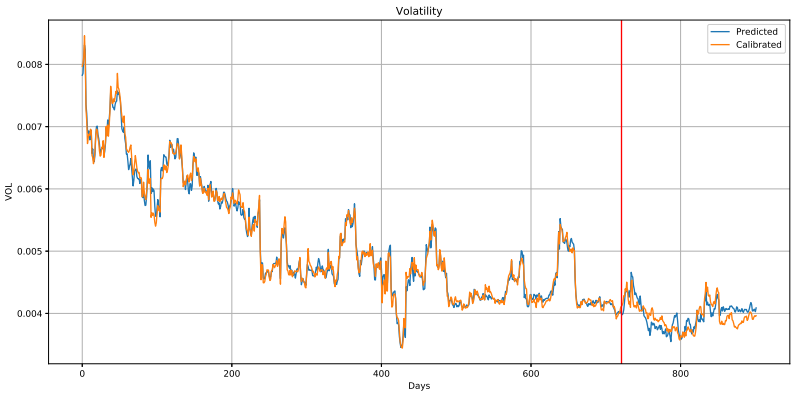


Figure 6.8: Comparison of the volatility σ predicted by a network with 7 hidden neurons and the original volatility

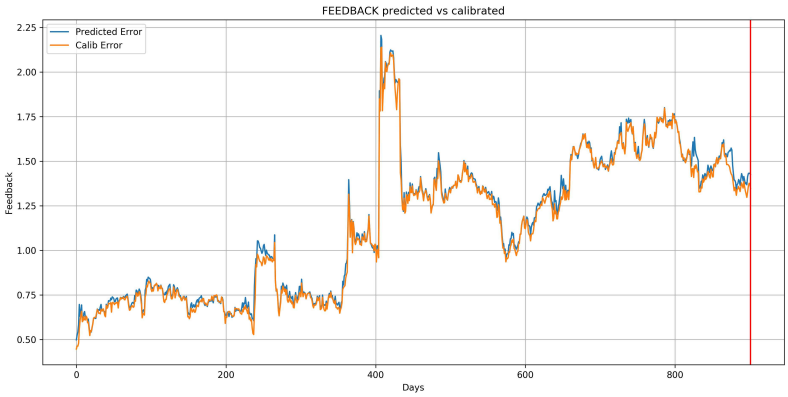


Figure 6.9: Comparison of the resulting feedback for the offline test with the original one

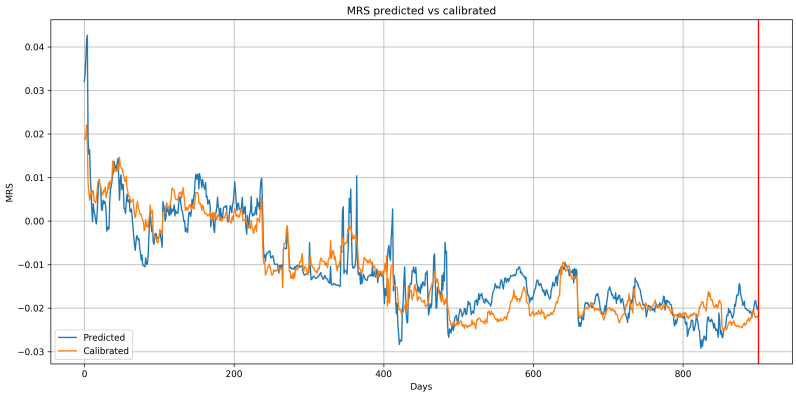


Figure 6.10: Comparison of the predicted mean reversion speed with the original one in the offline test

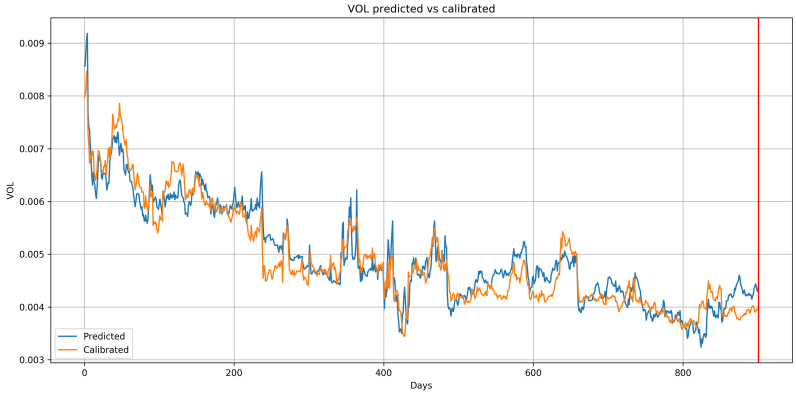


Figure 6.11: Comparison of the predicted volatility with the original one in the offline test

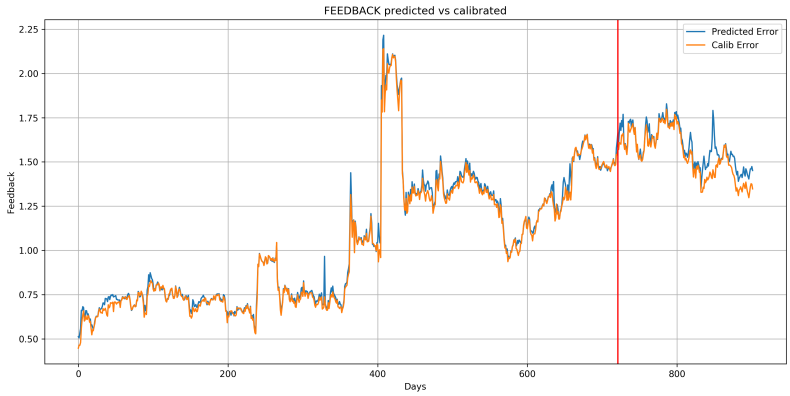


Figure 6.12: Comparison of the resulting feedback for the online test with the original one

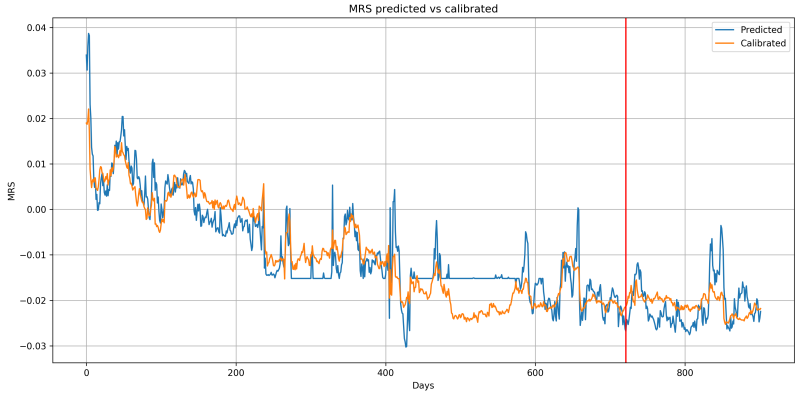


Figure 6.13: Comparison of the predicted mean reversion speed with the original one in the online test

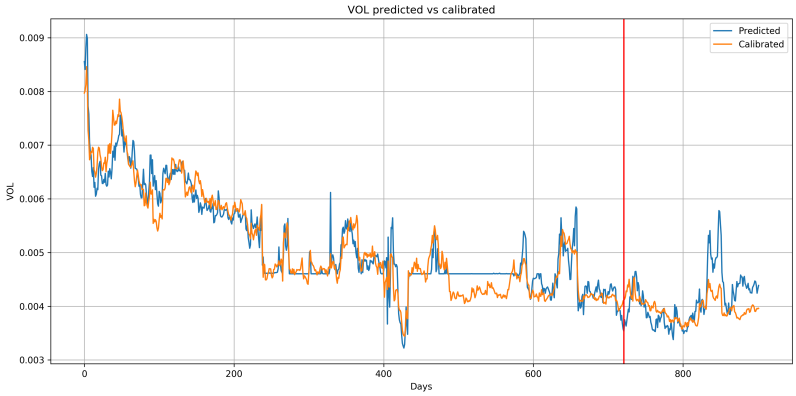


Figure 6.14: Comparison of the predicted volatility with the original one in the online test

Chapter 7

Conclusions

In the previous section we have seen the results of our black-box optimization method. They are not perfect yet but they look very promising thanks to the advantage that our solution brings over the old calibrator. The main benefit of this new approach is the fact that the development of a calibrator for a particular model would not be needed anymore, that means that it will be easier to change the interest rate model to more complex ones, or simply different ones, by only reimplementing the feedback function. This is a great advantage, in fact the development of the calibrator can be difficult because the optimization that it performs is complex. The second advantage is the time required by the calibration, that with our solution is in the order of milliseconds instead of several seconds.
We have seen that the offline training is usually capable of approximating the calibration function that we want to replace, even though sometimes our prediction model gets stuck in some areas where the model parameters are constant. This problem cannot be really solved without changing the interest rate model used for the pricing of the swaption. In fact, a more expressive, i.e. more complex, model would help the optimization process and problably it would produce better results.

7.1 Future Reaserch

Starting from the results we have achieved with our solution, the first and most important further development should be the use of our black-box optimization with a richer interest rate model. The choice would be to replace the simple Vasicek we use with a multi-factor Vasicek model that can capture more complex dynamics of the interest rate, so it could provide a much

more accurate pricing of the swaptions. Such a model would help the optimization thanks to a more sensitive feedback with respect to the variations of the model parameters, providing an actual metric to measure improveents between different solutions.

In addition, the multicurrency approach (described in Section 5.9) should bring some benefits to the whole procedure thanks to the enriched dataset, so the trained model should be able to generalize more on the unseen cases. In fact, the calibration function that we are minimizing does not depend on the particular currency, but only on the current market data, that will be probably different for different currencies, bringing an increase variety to the input data.
The use of a multicurrency dataset should bring advantages but we would still need a more complex interest rate model like the multi-factor Vasicek model to effectively see good results.

Bibliography

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

[Byrd, 1995] Byrd, R. H.; Lu, P. N. J. Z. C. (1995). A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*.

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

[Charles R. Nelson, 1987] Charles R. Nelson, A. F. S. (1987). Parsimonious modeling of yield curves. *The Journal of Business*.

[Fletcher, 1987] Fletcher, R. (1987). *Practical methods of optimization (2nd ed.)*. John Wiley and Sons.

[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.

[Mitchell, 1997] Mitchell, T. M. (1997). *Machine Learning*. McGraw Hill.

[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.