Computational Science and Engineering
(Int. Master's Program)

Technische Universität München

Master's Thesis

Calibrating Short-Rate Models Using Differential Evolution on a GPU

Afshin Loni

Computational Science and Engineering
(Int. Master's Program)

Technische Universität München

Master's Thesis

Calibrating Short-Rate Models Using Differential Evolution on a GPU

Author:	Afshin Loni
1st examiner:	Prof. Dr. Michael Gerndt
2nd examiner:	Univ.-Prof. Dr. Michael Bader
Assistant advisor:	Grant Bartel (risklab - Allianz Global Investors)
Thesis handed in on:	March 29, 2017

I hereby declare that this thesis is entirely the result of my own work except where otherwise indicated. I have only used the resources given in the list of references.

March 29, 2017 Afshin Loni

Acknowledgments

First and foremost, I would like to thank Grant Bartel, who helped me throughout the whole process of this thesis. He had taught me a lot, about the financial part of this work. Also, I would like to express my gratitude to Professor Gerndt, for giving me the opportunity of working within his chair.

There were a lot of other people, who helped me from time to time during this work. I would like to thank, Mohammadreza Basirati and Mohsen Ahmadvand for reading the thesis and giving me their suggestions, Amirhesam Shahvarani who gave me a lot of programming hints, Arash, Mohammad Moein, Nazanin, Mehdi Saleh, Ghazal, Laszlo, for being beside me all along, my brother Babak, who was such an inspiration for me, and my sister Mina as well as Amir and Bardia who encouraged me to move forward.

Finally, I would like to dedicate this thesis to my parents, Mosi and Sodi, who gave me their wonderful support all these years and helped me be who I am now. Love you both.

Abstract

Modeling interest rates, plays an important role in the financial services industry because it helps provide insight into future market movements and financial instrument prices. This thesis, focused on the calibration of two popular short-term interest rate models (short-rate models) to make the most accurate and timely predictions possible of future interest rates, more specifically the spot rate. The two models to be explored include the one-factor Vasicek model and one-factor Cox-Ingersoll-Ross (CIR) model, which are both well known in finance industry. To calibrate these models, a global optimization method called differential evolution – a well-known evolutionary algorithm – was implemented. To implement this global optimizer, a program that runs on a GPU using C++ and CUDA was written. Following this process, analysis were formulated regarding the performance between serial and parallel implementations, performance when perturbing the differential evolution parameters, and accuracy when comparing model and market values. We found out, that using the differential evolution algorithm would give us a properly accurate result and by using GPU, we can now solve the problem in about 7-10 times faster than before.

Contents

Outline of the Thesis

1. Introduction

When an individual, as an investor, wants to invest her/his money into finance market and earn some benefits from that, she/he has to consider a lot of aspects of the market and be able to predict it's movements. This is not an easy task, and to achieve the best result, the investor must have a rather good knowledge of finance. Therefore, investors usually ask advisers to help about their investment and tell them where should they put their money. Investment advisers, conduct a set of financial and mathematical analysis and give the investor a proper recommendation in return for a fee. These analysis, are very complicated, because one have to consider a lot of features of the market. The advisers, need to model all scenarios that influence the market behavior, and predict the market movements by combining all these models. Interest rate modeling is among one them.

Interest rate, as a well known term in finance, is the amount charged by a lender to a borrower for the use of assets. To put it simply, let us consider a bank offers a €1000 loan for buying a commodity and asks the borrower to return €1050 after two years. Here, the bank is getting 5% interest over it's money. This is because of the fact that the bank could have invested its money on something else and earn some profit after two years; but instead, it is lending it. Interest rates are not just taken into account when lending and borrowing money (debt), but they are also used in other types of assets like bonds, options, commodities, etc. In fact, interest rate has a rather great important role in finance, that it affects everything in finance directly or indirectly. Take inflation as an example. The rise or fall of prices for everything is based on inflation each year. If inflation is too low, it means there is no demand to buy products and therefore the economy is in danger. To fix this issue, banks reduce interest rates of debts, which means they encourage people to borrow money for a good deal and consequently they can buy the products. Eventually, when the demand to buy a product goes up the inflation rises as well. Therefore, to control the inflation interest rate is used here. Now the question is how can we define the proper interest rate that is fair for the economical situation and profits both the borrowers and lenders. This is where interest rate modeling comes into play.

An interest rate model is a probabilistic description of how interest rates can change over time and optimally it will describe the evolution of interest rate in future. Like all other financial terms, interest rate is dependent on so many scenarios, from a well known factor, like a country's financial situation, to a totally random incident like a sudden earthquake. This is the reason why, it is always uncertain to predict interest rate movement. Therefore, the interest rate model comes in handy by characterizing these uncertainties [14]. These models, must be able to present the properties of an interest rate movement. For example Vasicek interest rate model[35] describes the movement of the interest rate using the properties such as time, mean diversion equilibrium value and market risk.

In finance science, there are numerous interest rate models which have their advantages and disadvantages. In this thesis, we will study two different interest rate models; the one-factor Vasicek model[35] and the one-factor CIR[9] model. These models are of the most famous interest rate models. They are also called as short-rate models. A short-rate model is used to predict the interest rate at the current time with infinitesimally small amount of expiry time. We will talk about these in details in the following chapters. For now, it is necessary to know that short-rate model is a term for specific interest rate models and for the Vasicek and CIR model we can use both terms.
Vasicek model can sometimes produce negative interest rates which was not realistic in the past, but since 2009 we do have negative interest rates as well. Adding this, to the fact that Vasicek model is simple, will prove why nowadays still most of the financial institutes and banks use this model to describe the evolution of short-term interest rates[4]. If we want to describe shortly how Vasicek and CIR models work, it is that they have some parameters in the model and try to find the best values for these parameters in a way that the model can estimate the interest rate values from the past market data. The process of estimating these parameters is called calibration. It is very crucial for us to make these parameters consistent so that the model values are close to the market values. This helps the model to be realistic and hence usable for future predictions.

Calibration is basically a minimization problem, because we want to find a set of model parameters that minimizes the difference between the market data and the model data. There are some difficulties in calibrating the model which depends on the complexity of the model, number of parameters in the model, and other factors.
It is necessary to know, that calibration is an important step of interest rate modeling, because no matter how complex or how good are the models, if we are not able to calibrate them properly even if they are the best models they would fail.

To calibrate these models we have to come up with a cost function. One can model the cost function, using Maximum Likelihood Estimation (MLE) or the Least Square Method (LSM). There have been different studies about the MLE and LSM and according to these studies it is important to be very careful with these methods, because it can lead to a very biased parameter estimation[30]. One study showed that using MLE and LSM could result in a bias of up to 400%[21]. So, optimizing the cost function is an important fact in solving our problem.

To clarify the problem more, let's take a short look at the mathematics behind one of the short-rate models we are going to study, i.e. the Vasicek model. This way we can get a picture of the whole project in a glimpse.

The Vasicek model is a stochastic differential equation which is described with the following equation.

$$dr_t = \alpha(\beta - r_t) + \sigma dW_t \tag{1.1}$$

This model has three parameters, namely α, β, and σ. We will talk more about these parameters later.

We have a historical data of the interest rate, and with this data we want to calibrate the model, which means we want to find the parameters α, β, and σ in a way that it matches the best with our historical data. For this purpose, we use e.g. the Least Square Method to create a cost function. A general form of the cost function is written in equation 1.2.

$$f_{cost} = \sum_{k=1}^{N} (d_k - m_k(P))^2 \tag{1.2}$$

where d_k is our k^{th} real data point, $m_k(P)$ is our model prediction function, and P is a vector of our parameters. Also the dimension of our function depends on size of P.

Next step, would be to use an optimization algorithm to minimize the cost function. There are several algorithms to optimize the minimization problems among them one can suggest Gradient decent[2] and Gauss-Newton[5] which are of the most well known algorithms.

Gradient decent uses the first derivatives of the cost function and simply goes downhill to find the minimum, as Gauss-Newton uses the second derivative of the cost function which leads to the faster convergence. There are also more complicated algorithms, like Levenberg-Marquardt Method[5], which is a mixture of gradient decent and Gauss-Newton. The problem with these algorithms is that for they depend a lot on a starting point and if we cannot offer a good starting point, there is a high possibility of falling into a local minimum. Therefore, one have to come up with an appropriate starting point which is close to the real global minimum and then one can be confident that the output is the global minimum. Another problem is that the cost function should be differentiable and also continuous, which is considered as a big constraint in so many least square problems.

Another method that is worth to mention is Downhill Simplex Algorithm[25]. This algorithm is based on a simple direct search which means it does not need any computation of the derivatives of the cost function. This method is not as fast as Levenberg-Marquardt and the other problem is that according to a study, there is a possibility that it will miss the global minimum even for simple parameter spaces[36]. Therefore, we decided to use a better optimizer to increase the chances of finding the best values.

One of the most interesting global optimizing methods, is called Differential Evolution (DE), which is basically from the family of Evolutionary Algorithms. The way DE works, is to take a population of candidate solutions and tries to improve them iteratively by searching through the problem domain. It would then choose the best solution among all the improved candidate solutions. One of the advantages of this method is that unlike classic optimization methods like gradient decent and Gauss-Newton, this method needs no assumptions such as the cost function being differentiable or even being continuous. But on the other hand, in definition it is not guaranteed that Differential Evolution would lead to the optimal solution. This problem, depends on the number of candidate solutions we use and the criteria that we have to choose for the method and it will be all explained in the results chapter. Vollrath and Wendland[36] have compared Differential Evolution with

Levenberg-Marquardt and Downhill Simplex Algorithm and the results were showing that Differential Evolution is more reliable with 5 different cost functions.

For the aforementioned reasons, we are using the DE algorithm to calibrate the interest rate models. The DE algorithm has three parameters which by changing them the accuracy and performance of the algorithm would differ. These parameters are *F* (mutation rate), *CR* (Crossover probability) and *NP* (population size). These parameters must be tuned to achieve the maximum performance, so an analysis of them would be very important. We are going to analyze the perturbation of these parameters and their effects on the performance and accuracy of our problem.

The number of candidate solutions for DE would be usually a few dozens, but for some functions even more than a hundred can be insufficient[36]. This means that the algorithm would be computationally expensive. On the other hand, the candidates are totally separate from each other which means the algorithm is highly suitable for being parallelized.

With these introductory statements, we can now describe the problem, the questions ahead of us and our contribution to solve the problem.

Financial advisers, need to model interest rate to get a better understanding of market behavior. short-term interest rate models, or as it simply called short-rate models, are of the most useful types of interest rate models that need to be simulated. But, to model the short-rates we have to answer the following questions.

- What is the importance of calibrating short-rate models?
- Which optimization model is practical to calibrate these models?
- How can we calibrate these models accurately, yet fast enough to be acceptable for the usage of financial consulting companies like risklab??

Our suggestion to answer these questions is, calibrating Vasicek and CIR model using the differential equation algorithm. Also, to make the algorithm fast we are going to write the code on GPU and compare the performance and accuracy with the serial version.

2. Related Works

We are dividing the works related to our work in two parts. One is the calibration of interest rates, and the other is using GPU for the DE algorithm. We will show how much work has been done throughout the history.

There has been several studies of calibration of interest rates as mentioned throughout this work. It was in the 1990's, when banks started introducing new financial instruments including different derivatives and options[28]. Since then, studies concerning pricing options and consequently calibrating interest rates to forecast the market increased rapidly. Calibrating short-rate models, such as Vasicek and CIR model, was also of interest. In 1992, Chan, Karolyi, Longstaff and Sanders[7], have estimated the best parameters for a number of short rate models using the Generalized Method of Moments. They were among the first ones that realized the importance of volatility and randomness in the short-rate models. Clewlow and Strickland[8] used the Monte Carlo Method to calibrate interest rates in a study in 1996. Later, Li[18] used again the Generalized Method of Movements to estimate one factor interest rate models and showed that they fit best with the historical data.

Duan, Gauthier, Simonato and Zaanoun[13] used a different estimation method, Maximum Likelihood Estimator, to generate the best parameters for a short rate. Also in another study in 2004, De Rossi[11] has used the Maximum Likelihood method to estimate the parameters of the CIR model. There has been several other studies on the short-rate models of our concern namely the Vasicek and the CIR model. Zeytun and Gupta[37], for example, has studied both Vasicek and CIR model and their calibration process. They also compared both models by their parameter sensitivity and concluded the fact that these models are very similar in results with minor advantages in favor of the Vasicek model. Amin[4] on the other hand, focused on the calibration methods and have studied three different methods, Maximum Likelihood Estimator, Least Square Estimator and Kalman Filter with three different interest rate models, including the Vasicek and the CIR model. He found out that the Kalman Filter acts more accurately comparing to the other two methods although, one have to do a lot of cumbersome mathematical calculations and the method is not as simple as the former two. Van Elen[34] has studied the term structure models and calibrated again the Vasicek and the CIR model. His calibration was with respect to the data representation, where he used once the time series and once the cross sectional data to calibrate the models.

Differential Evolution has been introduced by Storn[31] in 1996 and since then it is considered as one of the pioneer evolutionary algorithms to optimize different problems. Using the DE algorithm to calibrate the interest rate models has become an interest of study in late 2000s. For instance, O'Sullivian[26], has studied the DE algorithm and the Kalman Filter to estimate the parameters of the CIR model. Vollrath and Wendlan[36] also used Differential Evolution to calibrate interest rate models. They have compared three different global optimization algorithms: Levenberg-Marquardt, Downhill Simplex, and Differential Evolution, and concluded that the DE algorithm performs better as a matter of accuracy than the other two. Although they have not used the GPU or any parallel technology to do the calibration.

Using the GPU to implement the Differential Evolution algorithm has become very common in the past few years with the rise of powerful Graphic Processing Units. Nashed[24], Sanchez and et al.[17], Plato and et al.[16], Veronese and Krohling[12] as well as Qin and et al. [29], are among the large group of researchers who implemented and studied the DE algorithm over the GPU. All these studies has led to different optimized implementations for the DE algorithm. The implementation by Nashed[24] is in fact an open source library which will get any cost function and with the help of DE gives the best parameter estimation. These implementations however are not meant for a sophisticated situation like ours. The cost function created for the Vasicek Model or the CIR model as mentioned have randomness and cannot be simply injected into this library. That is the reason that we had to implement our own version of DE on GPU.

However, the calibration of interest rates using DE algorithm over GPU is a rather new topic and there are not so many studies focusing on this area. Among the few studies one can mention, Aichinger and etc.[3] where they calibrated the Heston Model over the GPU. Also, Castillo et al.[6] have studied over parallelization of the Heath-Jarrow-Morton (HJM) model over CPU and GPU.

This specific work that we have done, which is calibrating two short-rate models using Differential Evolution over GPU, is one of a kind and has not been studied before.

3. Interest Rate Models

In this chapter, we are going to describe our interest rate models and their calibration process. But first, we have to look at some basic definitions in finance that are useful for us. The basic questions such as, why do we need interest rate modeling, why do we want to calibrate these models, and how is the calibration happening, are going to be answered in this chapter.

3.1. Things to know about Finance

We are going to calibrate interest rate models, but to understand what exactly this means we need to first understand where are we in the finance science. We are going to give some simple financial definitions, which make us familiar with the financial situation we are looking at and help to understand the mathematical part easier. The definitions are meant to be from very basic to advanced and are going to be related to each other. We are going to explain step by step, how from these definitions we get to our very own topic of interest rate models.

First let's take a look at the terms related to financial instruments.

- **Financial Instrument** is any virtual or real document that is considered as an asset and can be traded. Bonds, loans, stocks, options, futures and other derivatives are all considered as financial instruments. The term **Security** is also commonly referred to any form of financial instrument, although its legal definition is slightly different.
- **Fixed Income Instrument** is the type of instrument that generates a specified amount of income in a certain period.
- **Corporation** is a legal entity that is separate from its owners. The most important aspect of a corporation is that the shareholders are participated in the profits, but are not held personally liable for company's debt.
- **Debt** is the amount of money borrowed by one party from another party. When individuals or corporations, are willing to make large purchases that they cannot afford, they borrow funds from investors such as banks, etc.
- **Bond** is a debt investment in which the investor borrows money to an entity such as corporate or governmental entity. For example, if a company wants to raise €1 million to fund the purchasing of new equipment, it can issue 1000 bonds with a face

value of €1000 each. Bondholders are promised repayment of the face value of the bond at a certain time in future called **maturity date** in addition to a regular interest payment throughout the investing years. Bonds are just like loan except the fact, that the companies are the borrowers and the investors are the lenders or bondholders.

- **Zero Coupon Bond** is the bond that has no coupon payment. It means that, the interest would be paid only at the end of maturity date and nothing will be paid in the time before that. Zero coupon bond is one of the key values to calculate the interest rate.
- **Stock** is a security that represents the ownership in a publicly-traded corporation. This means the owner of the stock has a claim on part of the corporation's asset and earnings.
- **Future** is a financial contract that obligates the buyer or the seller to purchase or to sell and asset (such as a physical commodity or a financial instrument) at a predetermined future date and price. Futures can be used to hedge (reduce risk) or speculate on the price movement of the underlying asset. Futures and options are the among the most famous derivatives. **Derivative** is a contract that derives its value from an underlying asset. The underlying asset can be commodity, stock, etc.
- **Option** is also a derivative contract that gives the buyer the right but not the obligation to buy or sell an underlying asset with a specific price on a specific date. An option can also be exercised before the expiration date. Exercising means putting into effect the right specified in the contract.

It is now important to mention that the pricing of these instruments is a very cumbersome and important task in the finance industry. A corporation or financial company should consider different aspects of an specific industry to be able to price its instruments. Let's take a look at option pricing as an example.

The price to sell or buy an option is called premium. Generally speaking, there are six primary factors that influence the price of an option. These factors are:

- Underlying price which is, the current price of the underlying asset.
- Strike Price which is, the price at which an option can be exercised(bought or sold).
- Time until expiration of the option
- Volatility which is, a measure of the movement of the option price
- Dividends the distribution of underlying asset profits
- Interest rates

Now we can say that to get the price of any financial instrument knowing the current and future interest rate is an important factor.

3.2. Interest Rate Concepts

As defined before, **interest rate** is the amount of money charged from a lender to a borrower for the use of the lender's asset. Interest rates are applied on different periods such as a month or a week, but they are usually annualized. Interest rates are being set either by the national governments or the central banks. There are a lot of different factors, influencing the amount of interest rates such as inflation, the amount of supply and demand in the market and the government's situation. Figure 3.1 shows the benchmark interest rate of the European area from the European Central Bank since the year 1998.

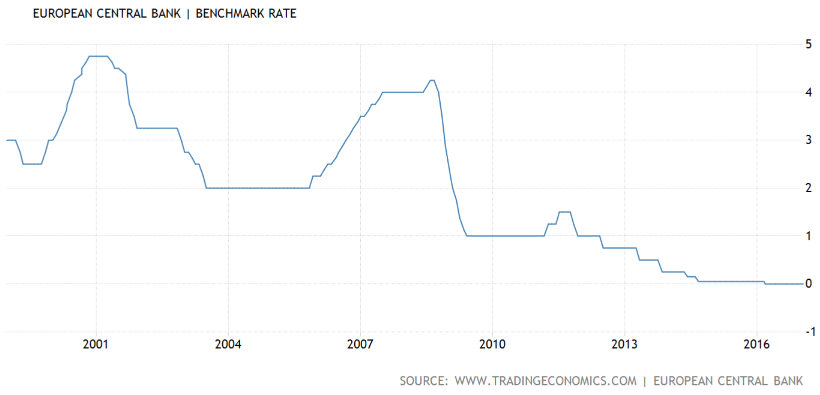


Figure 3.1.: Interest Rate in the European Area

As it is obvious in the figure, the interest rates have been reduced dramatically and at the time of writing this paper it is 0%. But as mentioned before, this figure shows the benchmark interest rate or base interest rate. Benchmark interest rate is the amount of money paid for one day. But this is not the interest rate we are talking about. There are different varieties of interest rates that we are going to talk about soon. However, to understand these definitions better, we have to take a look at some more concepts related to the interest rates which are going to be used in this thesis.

- **Time Value of Money** is an important concept in finance. Basically it means that, a dollar that you have today worth more in future, because it can be invested and earned some interest. Mathematically, this concept is also called *Market Price* or *Bond Price* which is showed as $P(t, T)$ and is the value at time t of a dollar received at time T, where T is the maturity time.

- **Yield**, usually shown in the form of percentage, is the amount of cash that returns on an investment. The relation between Yield $R(t, T)$ and Market Price $P(t, T)$ is the following:

$$R(t, T) = -\frac{\log P(t, T)}{T - t} \tag{3.1}$$

- **Instantaneous Forward Rate**, shown with $f(t,T)$ is the future interest rate describing all loans starting at time t with different maturities.
$$f(t,T) = -\frac{\partial \log P(t,T)}{\partial T} \tag{3.2}$$
- **Instantaneous Short Term Interest Rate** or simply **short rate** shown with $r(t)$ is the rate earned on shortest term loan starting at time t. it is calculated as:
$$r(t) = f(t,t) \tag{3.3}$$
- **Short-Rate Models**, use the instantaneous short term interest rate, $r(t)$, as the state variable. The Vasicek Model and it's descendants are among the short-rate models. These models usually have the following form:
$$dr(t) = A(r(t),t)dt + B(r(t),t)dW(t) \tag{3.4}$$
where A and B are called the drift and diffusion coefficients.

By now we should all be familiar with the term *maturity*. The amount of interest rate differs depending on the time of returning the loan from the borrower to the lender, which is known as the maturity date. For example, interest rate of European central bank for maturity of 10 years is currently 0.473 % while for 20 years it would be 1.127 %. To describe the relationship between interest rate and the different terms of maturities, a curve is defined called the "Yield Curve". Figure 3.2 shows the yield curve of interest rates of European central bank. Yield curve is also called "Term Structure of Interest Rate".

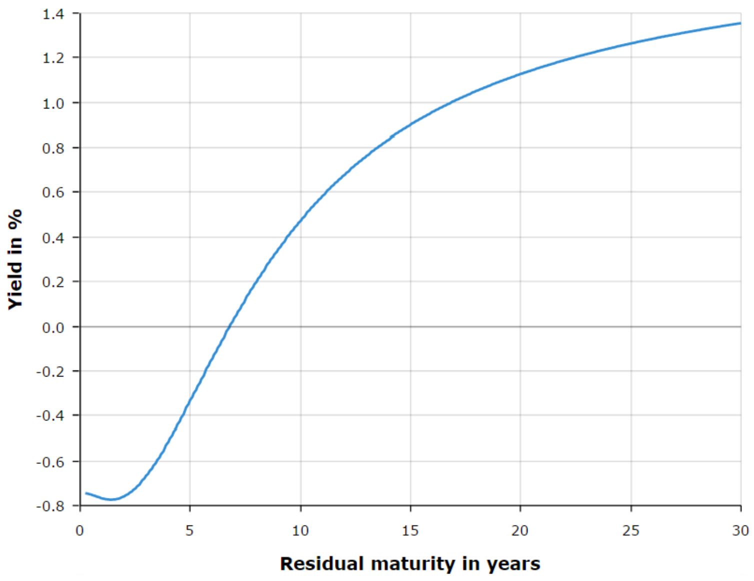


Figure 3.2.: Term Structure of European Central Bank. 03.02.2017

A noticeable point from figure 3.2 is, the interest rate for maturities less than 7 years is negative. This is a relatively new phenomenon in finance that interest rates have become negative and that is due to the fact, that since the great recession, a lot of economies had faced low growth and inflation. As a result, central banks have done a lot of different approaches to help the economies, and one of them was decreasing the interest rates to negative values. Surprisingly, this led to a helpful scenario for economies and so many countries are having negative interest rates right now.

The yield curve, is a very important curve in finance since it has a lot of usage in different finance topics, among them, is forecasting the interest rate. Knowing the future interest rate, would help the investors to make more reliable decisions. Thus, being able to extract a model from the yield curve would be a very important task. As shown in equation 3.1, the yield curve can be calculated using the bond price. For some situations, to reduce complexity, we use zero-coupon bond price to calculate the yield. And to calculate the bond price, we have to choose an interest rate model.

The interest rate models are divided by the number of uncertain factors that they have. These uncertain factors are the sources of risk in market. Most famous models are one-factor or two-factor but there are also three-factor and more multi-factor models as well. Having a multi-factor model, usually makes the model more accurate. But it also means that the model is more complicated and harder to calibrate. Also the data that we have is usable only for one-factor models. Therefore we are working on one-factor models and try to use our computation abilities to make it as accurate as possible. Equation 3.4 shows the general form of a one-factor model.

Another way to classify interest rates, is to divide them by the interest rate type they are representing an the historical data types that are available. For our data set, we have to use a class of interest rate models that are called *affine term structure models*. Affine term structure models, are the models that relate the zero-coupon bond to a short-rate model. This type of model, is specially useful to derive the yield curve. Vasicek and CIR are of the most famous affine term structure models that can be calibrated properly with risklab data.

As mentioned before, calibration of interest rates means, using an interest rate model and fit it to the existing data. If we are able to calibrate an interest rate model to the existing data, then we are able to use the model to predict the future interest rate. The whole purpose of this thesis is to calibrate two different interest rate models. The data that we have are taken from the risklab database.

So by now, it should all make sense about what we want to do and we are going to sum it up in the following definition.

Definition 3.1 (Calibration of Short-Rate Models) *We want to take an affine term structure model, like Vasicek, which is using the short rate as its variable. Using this model, we are going to generate the Zero-Coupon Bond Price. By having the Zero-Coupon Bond Price, we are able to* *generate the yield curve for different maturities. Next step would be, to make this model as close as possible to the real market yield. And this process is called Calibration of Short-Rate Models.*

We mentioned before that the two models we are calibrating are one-factor Vasicek and one factor CIR model. We also mentioned that these models are considered as *short-term interest rate models*. Short-term interest rates are on an obligation with the maturity of less than one year. It is also referred as *instantaneous interest rate*. However, instantaneous interest rate is also considered as the minimum possible maturity time, but since this term does not exist in reality in mathematics we use the same term for both definitions.

In the following section, each model is being described and the way to calibrate it is discussed.

3.3. Vasicek Model

Vasicek model is one of the most famous affine term structure models. As mentioned before, the model has been introduced by Vasicek[35] in 1977 and has been a useful model for years. Recently, there has been different variations of Vasicek model that are multi-factor. But in this thesis we are going to work on the one-factor model that has been introduced first. Let's take a look at the stochastic differential equation defining the Vasicek model.

$$dr_t = \alpha(\beta - r_t)dt + \sigma dW_t, \qquad r(0) = r_0 \tag{3.5}$$

where r_t is the instantaneous short-term interest rate or short rate.

The first part of the equation $\alpha(\beta - r_t)$ is called the drift factor. This factor, represents the instantaneous change of interest rate at time t. In this factor, β is the long term mean level, which shows that in the long run the, interest rate r will tend to β. α is the speed of reversion, which means the speed at which the interest rate would reverse to β. So, if we do not have any shock in the system meaning $dW_t = 0$ then, for example at time τ, if r is more than β, α would make the r smaller for the next time step to become closer to β. On the other hand, if r is less than β, then α would make r higher for next time step and finally we reach a point where $r = \beta$ and then it remains constant for the whole time. The existence of long term reversion, is because we should know that interest rate cannot rise indefinitely and also it cannot fall under a certain value. This is because of the fact that economies cannot accept a big value for interest rate. Imagine, if interest rates were too high, then people would just leave their money in the banks and get the interest. There would be no economy, that could give people the same amount of money. Therefore, too much interest rate doesn't make sense.

The second part σdW_t, is called diffusion factor which adds randomness to the equation and represents the shock in the market. W_t is a Wiener process which shows the random market risk factor. The Wiener process (or as it sometimes called Brownian Motion), is a continuous-time stochastic process. The Wiener process, has an important role in financial mathematics due to it's unique characteristics however, this fact is out of scope of this thesis. It is enough to know that the Wiener process gives a continuous randomness to the model. The variable σ shows the volatility of interest rate. **Volatility**, is the the degree of variation of returns for a given market security over time. Volatility is measured by calculating the standard deviation of the returns of the security in the past data. The higher the volatility, the riskier the security becomes. So, the Vasicek model considers these three factors and one shock factor to model interest rates.

The Vasicek model, can be solved analytically. The solution of the differential equation would give us a process for r_t with a Gaussian Distribution, that has a specific mean and variance. This analytic solution, is a long process which will be given in Appendix A.

For now, all we need to know is that under the Vasicek model, one can calculate the bond price. The following equation is the bond price for a specific maturity time T, at time t:

$$P(t,T) = A(t,T).e^{-B(t,T)r(t)} \tag{3.6}$$

where A and B are calculated as following.

$$B(t,T) = \frac{1 - e^{-\alpha(T-t)}}{\alpha} \tag{3.7}$$

$$A(t,T) = exp\left(\frac{(B(t,T) - T + t)(\alpha^2\beta - \sigma^2/2)}{\alpha^2} - \frac{\sigma^2 B(t,T)^2}{4\alpha}\right) \tag{3.8}$$

When we set $t = 0$, we will get the zero-coupon bond price.

Using this bond price and equation 3.1, we can calculate the yield for different maturities. The yield can be calculated with the following formula as well.

$$R(t,T) = -\frac{1}{T-t}\ln A(t,T) + \frac{1}{T-t}B(t,T)r(t) \tag{3.9}$$

Knowing this, one can create the yield curve for different maturities.

3.4. Cox-Ingersoll-Ross Model

The Cox-Ingersoll-Ross (CIR) model is another affine term structure model which is similar to the Vasicek model except the fact that it has one more term in the diffusion factor.

$$dr_t = \alpha(\beta - r_t)dt + \sigma\sqrt{r_t}dW_t, \qquad r(0) = r_0 \tag{3.10}$$

This extra term, helps the model prevent creating negative values. By solving this model, one can again calculate the bond price and get the yield to create the yield curve. The bond price, would be the same as the Vasicek model as follows:

$$P(t,T) = A(t,T).e^{-B(t,T)r(t)} \tag{3.11}$$

But the values A and B are different and are defined as follows:

$$B(t,T) = \frac{2(e^{\gamma(T-t)}-1)}{(\gamma+\alpha)(e^{\gamma(T-t)}-1)+2\gamma} \tag{3.12}$$

$$A(t,t) = \left(\frac{2\gamma e^{(\alpha+\gamma)(T-t)/2}}{(\gamma+\alpha)(e^{\gamma(T-t)}-1)+2\gamma}\right)^{2\alpha\beta/\sigma^2} \tag{3.13}$$

where $\gamma = \sqrt{\alpha^2+2\sigma^2}$. Using these two values one can again find the yield:

$$R(t,T) = -\frac{1}{T-t}\ln A(t,T) + \frac{1}{T-t}B(t,T)r(t) \tag{3.14}$$

And create a yield curve with different maturities. The procedure of calibrating the yield curve, is the same as Vasicek Model.

3.5. Calibration Process

The calibration process, is the same for both models. The basic idea of calibration, as said before, is to find the best parameters for the model (in our case α, β and σ) in a way that the model can fit the market yield curve. As we know, the yield curve uses the zero coupon bond price. This means, there is not going to be any payment before the maturity date and all the interest would be paid at once, and only at the maturity date. Thus, we have $t = 0$. This means, the yield would be calculated as follows:

$$R(0,T) = -\frac{1}{T}\ln A(0,T) + \frac{1}{T}B(0,T)r(0) \tag{3.15}$$

As mentioned before, A and B are the values that can be calculated depending on the model. The challenge here, would be to use a proper value for $r(0) = r_0$ in a way, that it will help the model yield curve, to fit best to the market yield curve. In other words, we have to fit r_0 to the model as well. In some studies, r_0 is also put in the same basket as other three parameters and is going to be fit to the model. But in this work, we separate it from them and use a rather simpler scenario to choose the best value for r_0, that is,using the *Monte Carlo Method*.

Therefore, we are dividing the calibration process into two parts. First part would be the calculation of r_0 and the second part is the parameter estimation.

3.5.1. Part I

Monte Carlo Method is a computational algorithm that is based on repeated random sampling, numerous times, to achieve a numerical result that is hard to solve analytically. This method, has been introduced first by Metropolis and Ulam[22], where they used the power of computers to solve an integration problem. There have been some studies, where Monte

Carlo method is used for estimating the interest rate models as well. Özgürel [27], for instance, has used the Monte Carlo method to generate a proper range for the parameters of the Vasicek model. In another study, Clewlow and Stricklan [8] has used different Monte Carlo techniques to evaluate different interest rate derivatives. We are going to use this method as well, just to estimate the value of r_0 and for the rest of the parameters we are going to use the Differential Evolution algorithm.

To clarify the situation, we have to first explain one more concept and that is the way of calibrating the data. There are two ways to look at the data; *Time Series* and *Cross-Sectional*. By looking at a piece of our data this can be more understandable.

Table 3.1.: Interest Rates for 4 month from risklab gmbh

Date	3 month	1 year	3 years	5 years	7 years	10 years	15 years	20 years
30.06.2016	0,30%	0,53%	0,72%	1,06%	1,32%	1,55%	1,79%	2,06%
31.05.2016	0,37%	0,73%	1,05%	1,41%	1,68%	1,91%	2,16%	2,43%
30.04.2016	0,32%	0,59%	0,95%	1,33%	1,62%	1,90%	2,20%	2,49%
31.03.2016	0,37%	0,61%	0,89%	1,28%	1,58%	1,87%	2,15%	2,43%

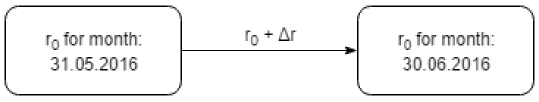
Cross-Sectional Calibration means we look at the data horizontally. We calculate the zero coupon bond for each maturity, generate a yield curve for a specific data and fit our model curve to the market curve. This is exactly what we are going to do for our calibration.

Time-Series Calibration means we look at the data vertically. This way of calibration is working only for short rate. so we cannot use it for a maturity of more than 1 year because it will not make sense. We are not going to use this way of calibration. But, to generate the best r_0 for the bond price calculation we are going to use this way. The reason is, as mentioned before, the short rate r_0 is actually the instantaneous short rate which means the interest rate at the shortest time span possible. This value, can be extracted by using the Vasicek or CIR model.

Let's say we want to get r_0 for the date $30.06.2016$, and use it for bond price calculation. To reduce confusion, we give this r_0 a new name like $nextRate$. With the time series calculation, we get this $nextRate$ from the month before, which is $31.05.2016$. That means, we first take a guess for the r_0 of month $31.05.2016$ and then, use this r_0 and the Monte Carlo Method to generate a good value for $nextRate$.

$$nextRate = r_0 + \Delta r \tag{3.16}$$

So basically we use the short rate of each month to calculate the short rate of next month, like the figure below:



Now to calculate Δr using the Monte Carlo method, we have to discretize the short-rate model. A discrete version of the Vasicek model would be as follows:

$$\Delta r_{t+1} = \alpha(\beta - r_t)\Delta t + \sigma\sqrt{\Delta t}\varepsilon_t \tag{3.17}$$

Where ε_t is a Gaussian distributed random variable. We are going to calculate Δr for the first time step and generate the next rate from r_0 which makes the equation look like:

$$\Delta r = \alpha(\beta - r_0)\Delta t + \sigma\sqrt{\Delta t}\varepsilon_0 \tag{3.18}$$

Using Monte Carlo method means, we are going to generate a big range of random values e.g. 10000 and use them as the value of ε_0. Therefore, we have 10000 different values for Δr. Then, we average these values and use the average as our best shot for Δr. As mentioned before, solving the differential equation of Vasicek model, shows that r is a Gaussian process. Therefore, by generating 10000 different values for r_0 one can claim that there is a good chance that the average value for this 10000 is a good estimation of real r_0.

Using this Δr, we are going to have the $nextRate$ as described in equation 3.16. This $nextRate$, will then be used as r_0 to calculate the zero-coupon bond price and yield in equation 3.15. The discrete version of the CIR model is also shown in equation 3.20.

$$\Delta r_{t+1} = \alpha(\beta - r_t)\Delta t + \sigma\sqrt{\Delta t}\sqrt{r_t}\varepsilon_t \tag{3.19}$$

Getting the final r_0, we are now able to move forward to the second part of calibration.

3.5.2. Part II

By getting the $nextRate$, the second part of calibration begins, which is finding the best values for model parameters. This is done, by defining a cost function from the yield formula and try to minimize that cost function using an optimization algorithm.

We have formerly discussed that, the cost function should look like the following:

$$f_{cost} = \sum_{k=1}^{N}(d_k - m_k(P))^2 \tag{3.20}$$

where d_k is our k^{th} real data point, $m_k(P)$ is our model prediction function, and P is a vector of our parameters.

For our model, we show the cost function with the following formula:

$$f_{cost} = \sum_{T=\tau_1}^{\tau_N}(R_T^M - R_T(\alpha, \beta, \sigma))^2 \tag{3.21}$$

where, R_T^M is the market yield for maturity time T and $R_T(\alpha, \beta, \sigma)$ is the model yield that is calculated as shown in equation 3.15. The maturity dates are from $\tau_1, ..., \tau_N$. For the data that we have in hand, there are 9 maturity dates that are :

$$\tau = 0.25, 1, 3, 5, 7, 10, 15, 20, 30 \quad years$$

As mentioned in the introduction, there are several algorithms to optimize a problem. These algorithms, can be used depending on the problem. In our problem, we have a function that has a lot of local minimums and one global minimum. Therefore, we should look for a global minimization algorithm. There are a huge number of global optimization algorithms, that can solve our problem. One can divide the algorithms into three categories:

- *Deterministic methods*, which if applicable, will bring theoretical guarantee that the found solution is indeed the global minimum.
- *Stochastic methods*, like Monte Carlo, which as mentioned before, uses random samples to find the best approximation.
- *Heuristic methods*, which bring an intelligent strategy to search into to the problem space and find the global minimum. Evolutionary algorithms are among the Heuristic methods and among them Differential Evolution is one of the most useful.

Therefore, we are going to use the Differential Evolution algorithm and by using this algorithm we can find the global minimum and complete the calibration process.

In the next, section we are going to take a deeper look at the DE algorithm and the way we used it to solve our problem.

4. Differential Evolution

4.1. Introduction to DE

Differential Evolution (DE), is a stochastic and population based algorithm to optimize a problem and it is from the family of Evolutionary algorithms. These algorithms, create a population of solutions and evolve these solutions to the closest possible result. The DE algorithm, was first introduced by R.Storn and K.Price [32, 33, 31]. one of the most important advantages of the DE algorithm is, that it does not need the gradient of the problem. Therefore, this algorithm is meant to work where other optimization algorithms are incapable of handling the problem, and that is when problems are non-linear and/or non-differentiable. Besides, the simplicity of the DE algorithm makes it one of the most useful algorithms in different sciences.

There is no optimization algorithm, that works perfect for all the problems and DE is not an exception and it has it's own shortcomings as well. The most important one is, that it might fall into the local minimum regions and find a local minimum which is not of our interest. This happens, because of the stochastic behavior of the DE algorithm. Another problem of the DE algorithm is that it may fall into the global region, but it cannot find the real minimum values. It will just find the global region and to go further down in that region one might need to use a local optimization algorithm after.

Despite all it's shortcomings, the DE algorithm has proved itself as a major help in the finance science and in calibrating the interest rate models. This is because of the fact, that we need global optimization algorithms for most of the finance minimization problems. From the previous works, one can mention Vollrath and Wendland [36], who used the DE algorithm to calibrate the one-factor Hull-white model, which is also one of the well known interest rate models [15]. They also, tuned the algorithm to make it suitable for a wide range of other different cost functions. Maciel, Gomide and Ballin [19], also used the DE algorithm to model the term structure of government bonds. The DE algorithm, is already being used in different financial computer applications of which FINCAD Analytics suite is one of the most famous.

As mentioned in the previous chapters, the main purpose of an optimization algorithm is to find the parameters of a cost function that fits best to the values given. Depending on the characteristics of the cost function, different optimization algorithm can be used. In our scenario, the cost function would be created using the Vasicek or the CIR model. As mentioned before our data consists of different vectors of market yield. So the cost function would be a vector of model yields which is different depending on the short rate model we are using.

DE is classified as a Global Optimization algorithm as mentioned before. This means, it will look for the minimum over all the input values and is not restricted to a specific area. This is helpful when we have a cost function that has a lot of local minimums and possibly one global minimum. A well known example to point out this fact, is the Ackley function [1], which is shown in figure 4.1.

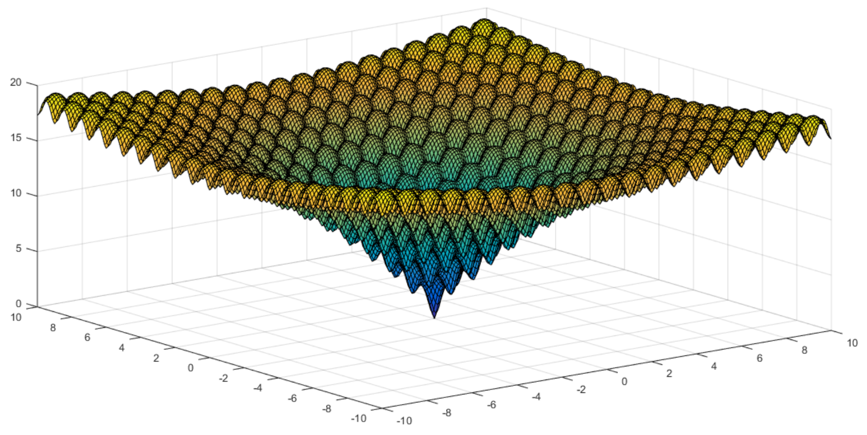


Figure 4.1.: The Ackley Function

This function, has a lot of local minimums but only one global minimum. The DE algorithm, is one of the global optimization algorithms that is perfectly capable of finding this global minimum. As mentioned before, in our scenario we are looking for the best values for α, β, and σ in Vasicek and CIR model. Since we have three parameters here, it is not possible to show them in a 3D surface but it is obvious that in our scenario there are also a number of local minimums and we are looking for the possible global minimum.

In the next section, the implementation of DE algorithm as it is used in our code will be explained.

4.2. Implementation of the DE Algorithm

In this section, we are gonna describe the DE algorithm and show how it is implemented in our code to calibrate the interest rate.

As mentioned in the previous chapter, we want to calibrate the interest rate model with a cross-functional approach. This means, that we use Vasicek or CIR model to calculate the yield and compare them with the model yield. We are going to use the zero-coupon bond price to calculate the yield. This means, we are going to use the following formula for yield calculation:

$$R(0,T) = -\frac{1}{T}\ln A(0,T) + \frac{1}{T}\ln B(0,T)r(0) \tag{4.1}$$

With the fact that $r(0) = r_0$ is a function of α, β and σ we can say that $R(0,T)$ is a function of these 3 parameters and we can show it like $R_T(\alpha,\beta,\sigma)$ and call it the model function. Considering this fact, the cost function would be:

$$f_{cost} = \sum_{T=\tau_1}^{\tau_N} (R_T^M - R_T(\alpha,\beta,\sigma))^2 \tag{4.2}$$

Where, R_T^M is the market yield for maturity time T and $R_T(\alpha,\beta,\sigma)$ is the model yield.

Table 4.1 shows a sample of our market data or R_T^M with which, one can draw a yield curve.

Table 4.1.: Yield Curve for June 2016

Maturity	3 month	1 year	3 years	5 years	7 years	10 years	15 years	20 years
Yield	0,30%	0,53%	0,72%	1,06%	1,32%	1,55%	1,79%	2,06%

Now that we have the cost function, the DE algorithm can be implemented. As described before DE algorithm uses a population of potential solutions, which are chosen randomly and tries to find the best set among the population.

Let's call a specific configuration of model parameters as a *parameter set* or *individual*. In our case, (α,β,σ) form a parameter set. A collection of parameter sets, will be then called a *population*. If we use the index i to show each individual, then in our case each individual would contain 3 model parameters and it would be like:

$$P_i = (\alpha,\beta,\sigma)$$

The DE algorithm generally has 4 steps which are, *Population Initialization*, *Mutation*, *Crossover*, and *Best Selection*. These steps, are described in the following.

(I) Population Initialization: In this step, we choose a number of population sets, $P_i = P_1, P_2, ..., P_{NP}$. NP is the number of populations. Note again, that each population contains the model parameters which in our case are α, β and σ. The model parameters that are selected for each population are selected randomly. The algorithm is defined in a way, that *NP* needs to be at least 4. The reason will be clarified in the next step.

(II) Mutation: In this step, we are going to mutate the initial population and create new parameters. To do so, we pick three random populations and using a scale factor F, we generate a new mutated population as following:

$$P_i' = P_a + F.(P_b - P_c)$$

P_i' is the mutated population. all three populations P_a, P_b and P_c have to be different from each other and also different from P_i. And this is the reason that NP should be at least 4. This procedure has to be done NP times, so we have a complete set of mutated parameters. It is important to mention, that in this step the limits of the model parameters might exceed and thus one have to be careful about that and check the limits for each model parameter every time.

The factor F, plays an important role in the mutation process and it actually controls the size of the mutations. if $F = 0$, then there will be no mutation, but the larger F gets the larger becomes the mutation. Maximum value for F is usually 1.

(III) Crossover: By doing step II, we will have a population of size NP of mutated values. Now we want a process which determines which mutated populations should be entered into the original population. This process, is called crossover. We show an original population with P_i, a mutated population with P_i' and a crossover population with P_i'' which is a mixture of both original and mutated population. P_i'' is calculated as follows. for each model parameter α_j in each population we check the following condition:

$$\begin{aligned} \alpha_j'' &= \alpha_j' \quad if \quad U(0,1) \leqslant CR \quad or \quad j = r \\ \alpha_j'' &= \alpha_j \quad otherwise \end{aligned}$$

Where r is a random index from $1, 2, ..., j$, and CR is called the *crossover ratio* which is always between $(0, 1)$. By $U(0, 1)$, we mean a random value from a uniform distribution between $(0, 1)$. if this random value is less than or equal to CR, we choose the mutated parameter and otherwise we choose the original parameter. the use of r is that at least one mutated parameter would be chosen. This process, is done for each model parameter α_j, which in our case are α, β or σ.

With this selection process, we now have a new population set P'' of size NP with a mixture of both original and mutated population.

(IV) Best Selection: By now, we have an original population set P_i and a newly generated population set P_i''. The next step would be to calculate the cost function for each of these population sets. We then compare the results from each corresponding individual and choose the one with the lower error. This, would count as one generation and at the end of this generation we have a new population of size NP which is a mixture of P_i and P_i''.

We will repeat step 2 to 4 until we reach convergence or the maximum number of generations is reached. The maximum number of generations is dependent on the complexity of the problem. Sometimes it can be thousands and sometimes it is just around 50. We have checked the best value for number of generations in the results chapter. To give a better perspective of the algorithm and the way we have done it for our problem, a flowchart is shown in figure 4.2 followed by the pseudocode in Appendix B.

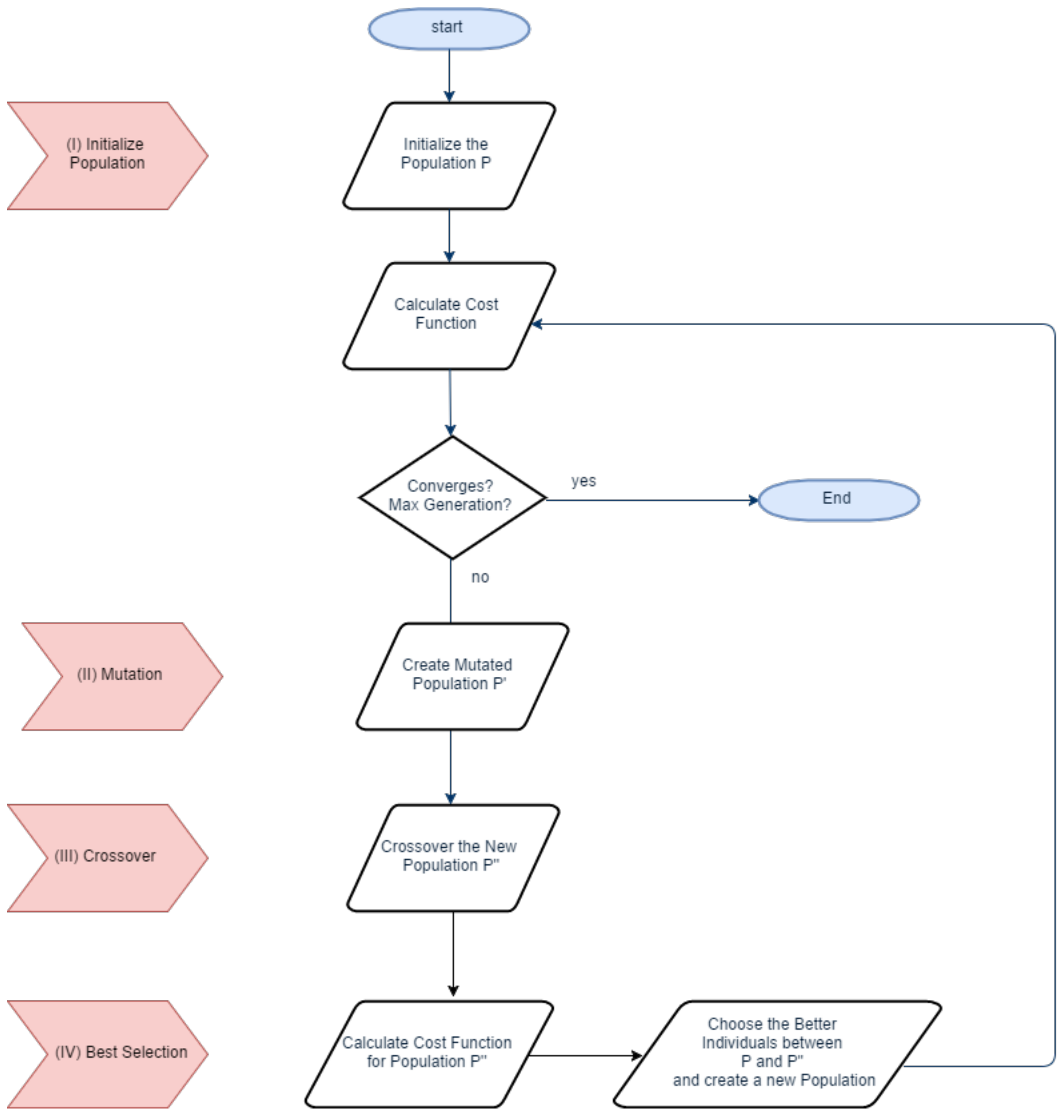
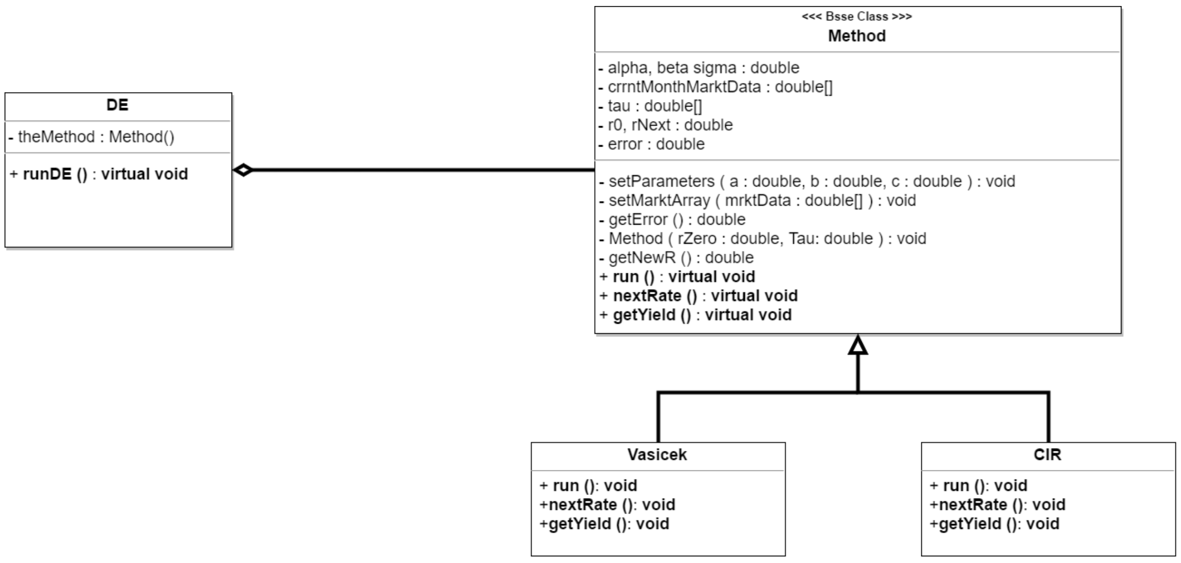

start
(I) Initialize Population
Initialize the Population P
Calculate Cost Function
Converges? Max Generation?
yes
End
no
(II) Mutation
Create Mutated Population P'
(III) Crossover
Crossover the New Population P''
(IV) Best Selection
Calculate Cost Function for Population P''
Choose the Better Individuals between P and P'' and create a new Population

Figure 4.2.: Flowchart of the DE algorithm

4.3. Code Structure of the DE algorithm

The code, has been written with C++ and we used the advantages of C++11 to make it more elegant. We have used the *strategy pattern* in our code, so the user will be able to choose between different interest rate models, of which we have implemented the Vasicek and the CIR model. The UML of the strategy pattern is shown in figure 4.3.



<<< Bsse Class >>>
Method
- alpha, beta sigma : double
- crrntMonthMarktData : double[]
- tau : double[]
- r0, rNext : double
- error : double
- setParameters (a : double, b : double, c : double) : void
- setMarktArray (mrktData : double[]) : void
- getError () : double
- Method (rZero : double, Tau: double) : void
- getNewR () : double
+ run () : virtual void
+ nextRate () : virtual void
+ getYield () : virtual void
DE
- theMethod : Method()
+ runDE () : virtual void
Vasicek
+ run (): void
+nextRate (): void
+getYield (): void
CIR
+ run (): void
+nextRate (): void
+getYield (): void

Figure 4.3.: Strategy Pattern used for the Interest rate model selection

As it can be seen in the UML graph, `Method` is a super class that implements the common functions for both Vasicek and CIR method. But it uses virtual definition for three functions, `run`, `nextRate` and `getYield`. These functions are being implemented separately for each model. It is useful to mention, that the function `nextRate` is the one that uses a Monte Carlo procedure to produce the next rate as described in the previous section.

The whole code consists of the following files:

- `main.cpp`: includes the main function in which we read the data , call the DE class, get the output and write the results in a new file.
- `DE.cpp`: the DE Class is written in this file. it has a function called *runDE()* in which we call one of the interest rate models as our cost function and then implement the DE algorithm on that function.
- `Method.h`: is a base (parent) class for our interest rate models. As mentioned before we have implemented the strategy pattern. So in the `DE` class we create an object of type `Method` where depending on the selection of the interest rate model by user, it's

gonna be a Vasicek or CIR model. The child classes will then implement the proper cost functions.

- `Vasicek.cpp`: Implements the cost function for the Vasicek Model and passes it to `DE` class for further usage.
- `CIR.cpp`: Implements the cost function for the CIR Model and passes it to `DE` class for further usage.
- `helper.h`: This file, implements the functions for reading and writing the data which is being used in the main file.

The pseudocode for the algorithms are also shown in appendix B.
In the Next Chapter, we are going to look at the parallel version of our code.

5. From Serial to Parallel

So far, we have learned the DE algorithm and the way to use it for the Vasicek and CIR model and we are able to calibrate interest rates. As it will be discussed in the results chapter, the algorithm works with a good level of estimation. The goal now, is to find a way to make the algorithm faster. As mentioned before, we have almost 10 years of monthly data for the yield curve which is a very small amount of data used in risklab. If we want to calibrate the data for 10 years of data with the current algorithm, it will take more than one hour. This number is not acceptable since, time is a very important factor in predictions.

To fix this problem, risklab has brought some powerful GPUs to it's infrastructure and therefore they are willing to do the calibration over the GPU and compare the results with the serial version. In this thesis, we are going to implement the DE algorithm for our specific models CUDA/C++ so that it can be run over GPU and then, we are going to compare the speed and accuracy with the serial version.

For the sake of research purposes, we are going to implement a parallel version of the DE algorithm over CPU as well, to compare the performance between CPU and GPU in a rather fare scenario. The following chapters are about the way we have implemented these two parallel scenarios.

5.1. Parallel CPU

There are different interfaces to make a code parallel over CPU, from which MPI and OpenMP are the most famous. In this thesis, we have used OpenMP for the parallelization of the code over CPU. OpenMP (Open Multi-Processing), is an API which has been released since 1998 for `C/C++` programming language[10]. The main advantages of OpenMP is, that it is easy to program and debug and there is no need to remake the serial code as well.

The reason to choose OpenMP is, when looking at the serial code of the DE algorithm, one can see that there are a lot of `for loops` in the code. These loops are depending on each other, which means each of them should run in a sequential mode so their results would be used for the next `for loops`. But, inside each loop there exist a good independence, which makes them easy to get paralleled. the OpenMP API, is a simple yet effective way to make the for loops parallel. we just have to add one line of code for each loop to make the processor run it in parallel mode and this would fairly reduce the speed of calculation for the whole code.

By looking at the pseudocode in appendix B, we can see that there are 5 main `for loops`. These loops are for initialization of population, calculation of cost function, mutation, crossover and selection of the best results. We have made all these loops to run in parallel.

For this parallelization, We have used a sample PC with an `Intel core i5` 5th generation CPU, with 2.20 GHz frequency and 4 cores. Also, we ran this code over the LRZ Cluster with a much more powerful CPU `Intel Xeon E5` with 28 cores. The results of the speed increase is shown the results chapter.

5.2. Parallel GPU

Writing the code on GPU, is a bit more complicated than the parallel CPU. In this section, we are going to describe how the code is written on GPU.

There has been a lot of studies before, about using the DE algorithm on GPU. Nashed, etc [24], has introduced an open source library that runs the DE algorithm on the GPU using CUDA. Due to their study, the GPU version outperforms the CPU version in most situations and the higher the number of generations goes, the more practical the GPU version becomes. In another study by Laguna Sanchez, etc[17], where the parallel implementation of the DE algorithm over GPU led to a speed up of around 5 and also a better convergence in some scenarios. In another study by Kromer and Plato [16] the GPU version of DE is 1.4-2.4 times faster than the CPU version. There exist, a few more number of studies and different implementations of the DE algorithm on GPU which shows the fact that running DE on GPU makes sense and makes the problem solving faster and more reliable. In our case, since the models are stochastic differential equations, the cost function is a bit more challenging than a normal cost function. Therefore, we had to implement our own CUDA version of the DE Algorithm.

As it can be seen in the flowchart of the DE algorithm in figure 4.2, there are a few steps that are independent of each other and can be done in parallel. we have separated all the possible steps that can be run in parallel and we are going to describe them as follows:

Random Number Generation: There are three main parts in the code, that we need to generate random numbers.
The first part is, when we want to initialize the population for the first time. For example if we have a population of size 100 and a model with 3 parameters, we need to create a matrix of 100×3.
The second part is when we need to calculate the next short rate using Monte Carlo method. For this one, if we want to have a proper Monte Carlo calculation we need at least 10000 samples for each population. In our example, this means a matrix of random numbers with the size of 10000×100.
The third part is, when we want to generate random indexes for the mutation part of the DE algorithm. Here, we also need a matrix of size 100×3 according to the example.

All these scenarios, are independent of the whole DE algorithm and can be done at once. A function, that is running on the GPU is called *kernel*. therefore, we have implemented at first three kernels for the random number generation. the `CuRand` Library, which is a well optimized library for random number generations is used for this purpose.

Evaluation and Mutation: The evaluation of the original population, is also independent of mutating the original population. With the first one we need to get the error of the cost function while with the second one we just need to create a new population for the crossover part. Therefore in this part we are going to do the evaluation in parallel with the mutation and crossover. For this Purpose we have written 2 kernels called `Evaluation` and `MutationAndCrossOver`.

With these two parts in mind, the parallelization of the code can be proceeded properly. We actually wrote the code in a way, that it will be evaluating each population P_i using one thread in the GPU. Therefore, a good speedup of the code can be expected. The following flowchart in figure 5.1, shows how we implemented different parts of the code in parallel.

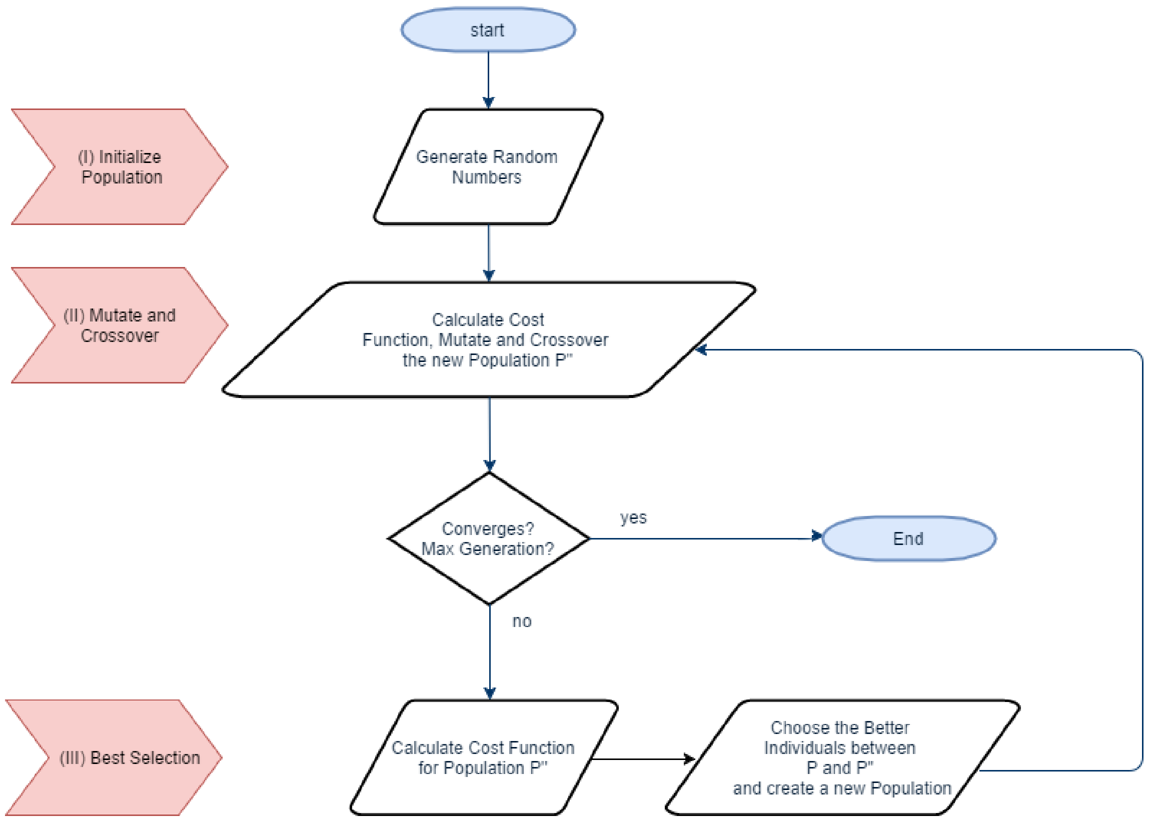

start
(I) Initialize Population
Generate Random Numbers
(II) Mutate and Crossover
Calculate Cost Function, Mutate and Crossover the new Population P''
Converges? Max Generation?
yes
End
no
(III) Best Selection
Calculate Cost Function for Population P''
Choose the Better Individuals between P and P'' and create a new Population

Figure 5.1.: The Parallel Flowchart for the DE algorithm

There are two challenges, when writing the GPU code in CUDA. One is, the CPU-GPU communication and the other one is the memory management.
Copying of data from CPU to GPU and the other way around would be usually the most time consuming part of any CUDA program. This is because of the fact, that one cannot define data directly on the GPU. Data is always in the CPU and it should be copied to GPU. And therefore, CPU and GPU should communicate, for this copy to happen. To solve this issue, we do this process once. So the data that is needed is copied only once and therefore no more extra communication is needed.
The other problem would be, since we are having a big number of values the memory might fill up and we might get memory leaks somewhere. That is why, we always need to allocate memory and clean it after the usage. To make this process less cumbersome, we have used a very useful library from CUDA called, `Thrust`. This library, is kind of the same as `STL` library in `C++`. With the help of `Thrust`, one can easily define a vector and use all the functionality a vector can offer. A vector, will delete itself from the memory automatically when it is out of scope.

The `Thrust` library, has two ways of defining vectors. `host_vector`, is used to define a vector in CPU memory and `device_vector`, is used to define a vector in GPU memory. By using this library, one should not be worried about memory leaks anymore because the `Thrust` library would define and delete vectors itself at the end of the usage. Also using this library makes the communication way easier. One can easily copy a `device_vector` to a normal vector by using the equal sign as follows.

```
thrust::device_vector < double > a(100);
vector < double > b(100);
a = b;
```

Knowing these facts, we have implemented the GPU code and in the next chapter we are going to check the results and analyze the performance and the accuracy of the written code.

6. Results

In this chapter, we are going to put everything together and try the written code. The process of testing the algorithm, divides into four stages.
First, we are going to check the stability of the model parameters. Then, we try to find the best parameters for the DE algorithm. Next, we will test the Vasicek and CIR models separately, and see how accurate they are comparing to the market data that we have. And finally, we will check the time performance of our three different implementations. The following chapters belong to the results of each test.

6.1. Stability of Model Parameters

As described before, in both Vasicek and CIR model, all model parameters should be constant. Of course, this is not the case in the real tests. But we can come up with a proper range for each model parameter, in which the model can behave realistic. Therefore, we are going to study this fact with our model parameters. It is important, to conduct this test early in the project, because with the proper range in hand, we are able to use a maximum and a minimum value as the boundaries of model parameters. Knowing the best boundary for each model parameter, will help us perform the algorithm better as a matter of time, with a guarantee of faster convergence.

We have generated the parameters for 180 different days and checked the stability of the model parameters, α, β and σ. we fixed a big range of boundary for each model parameter so we are sure that there is not going to be any missing value. In Figure 6.1, we show the results for each model parameter for the Vasicek model.

As it can be seen from the figure, Both variables α and β are rather stable and the variable σ is the one which changes the most. This makes sense, because σ is the volatility, and it shows the randomness in the equation. However, even $sigma$ values, show a reliable range that can be acceptable. Therefore, averaging the values, gives us the best range for each model parameter and that would be $\alpha \in (0.0, 0.4)$, $\beta \in (0.0, 0.05)$ and $\sigma \in (0.0, 0.005)$. These values, would be chosen for the CIR model as well, because as we can see from figure 6.2, we have the same range in this model.

Now that we have this ranges, we can claim that the model is behaving properly in this specific range and one can use this range of values to predict the future behaviors of model parameters. With this data, one can also make the algorithm perform faster which we will discuss in "Time Performance" chapter.

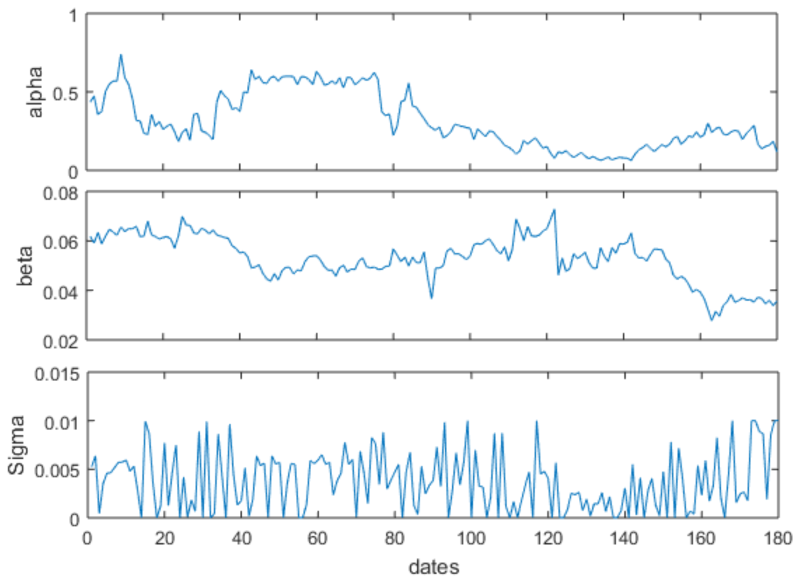


Figure 6.1.: Parameter Range for the Vasicek Model

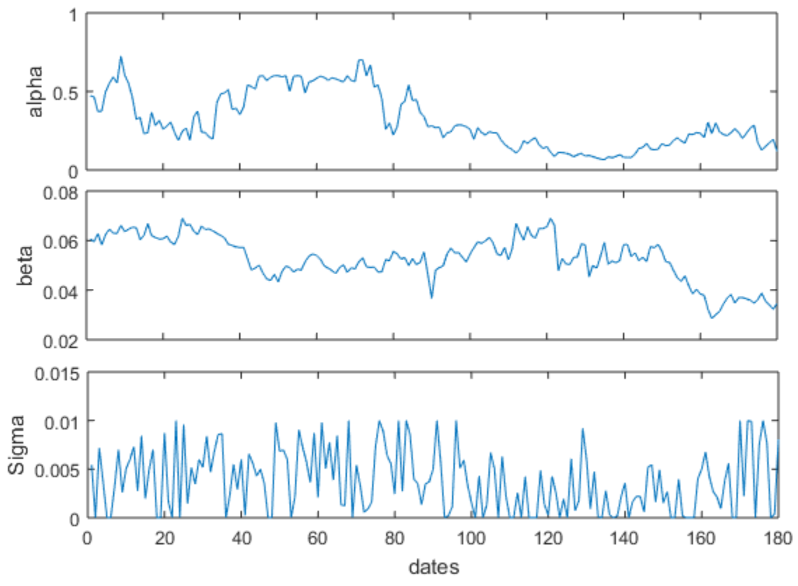


Figure 6.2.: Parameter Range for the CIR Model

6.2. DE Parameter Manipulation

The DE Algorithm, as seen in chapter 4 has three main parameters, that are Number of Population NP, Mutation Factor F, and Crossover Ratio CR. Selecting the values for these parameters, is very important for the best performance of the DE algorithm. Vollrath[36] has studied these parameters on 5 different cost functions and concluded that the best values for them are $NP = 15D$ where D is the problem dimension or the number of model parameters (3 in our case), $CR = 0.5$ and $F = 0.8$. There is another study by Magnus and Hvass[20] in which they extracted the best values for each DE parameter, depending on the dimension of the problem. For example for a problem with a dimension of 10, the best values are $NP = 28$, $CR = 0.9426$ and $F = 0.6607$. These studies show the importance of best DE parameter selection. Thus, we are going to also study this fact by choosing different values and find the best values for our own problem. To do such a test for each parameter, we make all other parameters constant and change the testing parameter and get the error for that parameter. Besides the three main parameters in DE algorithm, the maximum number of generations or $maxIters$ is an important factor. This parameter, is the first one that is being tested in our case. the Generation count has been tested in some other studies as well, because we want to see how many iterations is enough to get the better result. This test, would help us in increasing the speed of the whole code. All other variables are considered constant with the values $CR = 0.6$, $F = 0.9$, and $NP = 100$. We have averaged the error for 12 dates and normalized them. The results of the Max generation is shown in figure 6.3. Note that, the error value has been normalized due to

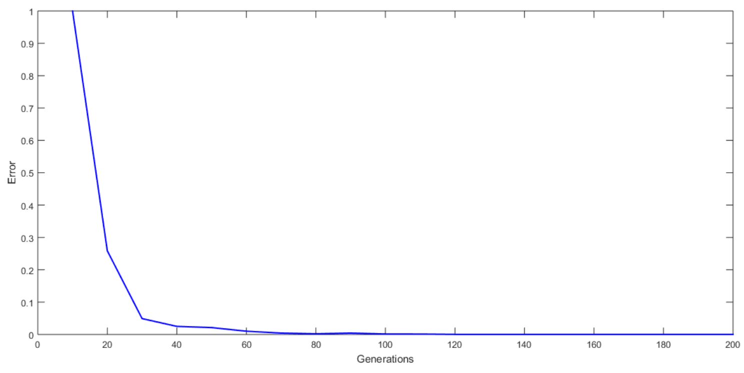
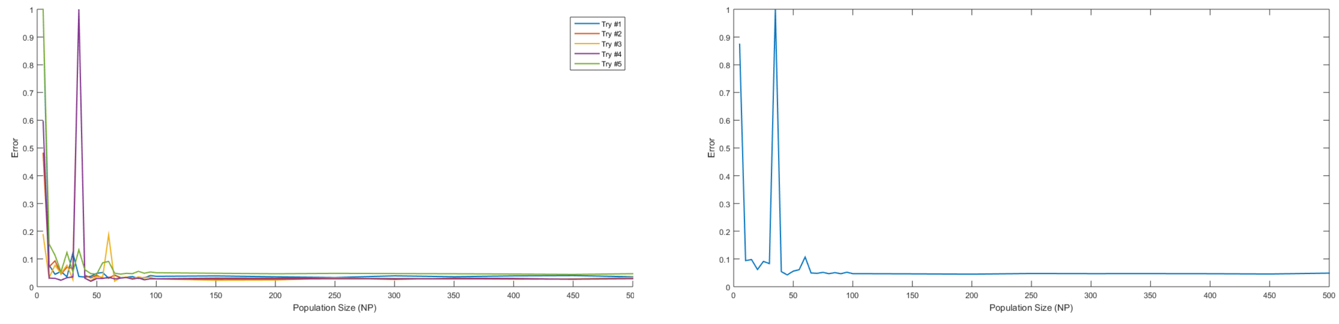


Figure 6.3.: Error vs. maximum generations for Vasicek Model

the fact that it can vary from 10^{-5} to 10^{-8} depending on other factors like the interval chosen for the model parameters. We can see in figure 6.3, that the Error will be constant after the $70th$ iteration. This means, there is not much sense in increasing the number of iterations, because at some point the error will be constant. Therefore, the value 70 would be the best for the generations of the DE algorithm.

The next parameter would be the Population size or NP. To test this parameter, we again used the values as $CR = 0.6$, $F = 0.9$, and of course $maxIters = 70$. Also to make it more accurate, we run the code for 12 dates but 5 times. Then, we averaged the errors of those 5 different tries. This way, the results would be more reliable. Figures 6.4 shows the results.

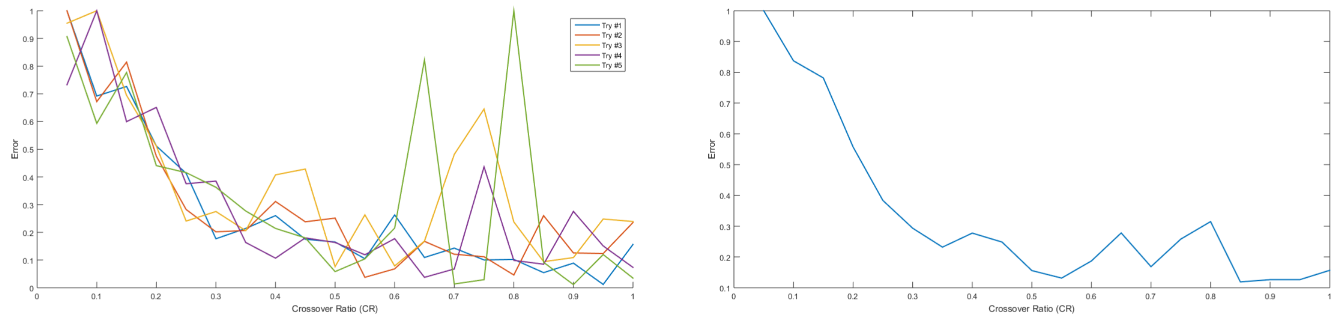


(a) 5 tries separately (b) Averaged

Figure 6.4.: Error vs. Population Size for Vasicek Model

As it is obvious in the results, the error becomes smaller by increasing the population size. However, from the value 70 it almost becomes constant and does not make much changes. Therefore, since the bigger NP gets the slower the program becomes, we choose the value 70 for population size as the best value. The jumps at the beginning of the plot shows the fact that, if the population size is not enough, the DE algorithm might fall into a local minimum and therefore the error will not be a proper one.

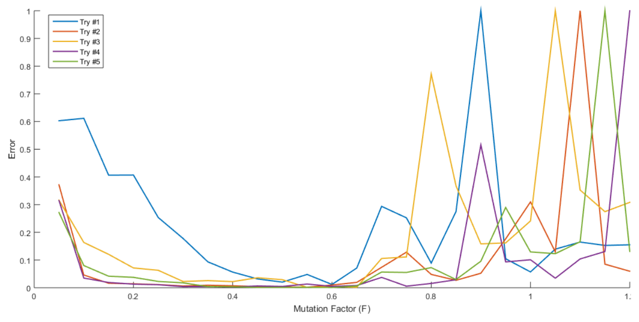
The same experiments has been conducted with CR and F and the results are shown in figures 6.5 and 6.6.



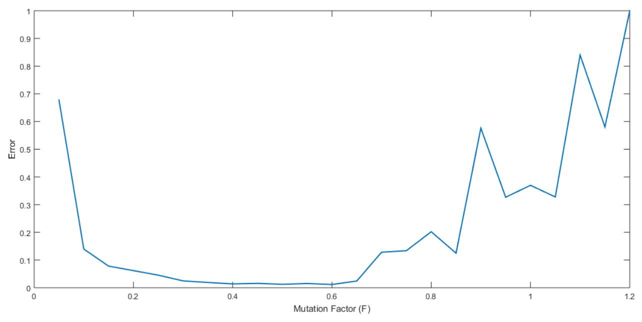
(a) 5 tries separately (b) Averaged

Figure 6.5.: Error vs. Crossover Ratio for Vasicek Model

There are two reasons adjusting the jumps in the plots. Firstly, as we know, there is a lot of randomness in the calculation process. We try to reduce the randomness as much



(a) 5 tries separately



(b) Averaged

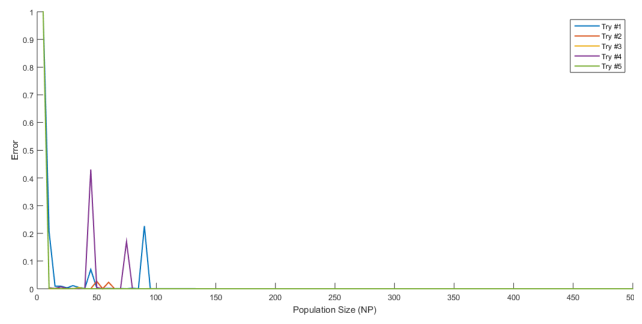
Figure 6.6.: Error vs. Scale Factor for Vasicek Model

as possible, but sometimes it still hits the algorithm and therefore we see sometimes big errors. Secondly and most importantly, the scale of the plot has been in a way, that the differences can be seen clearly. But in reality, the errors are all in a range of 10^{-6} to 10^{-8} and therefore the differences are not that large.
The same experience, is done for the CIR model and the best values chosen for each model are shown in table 6.1.

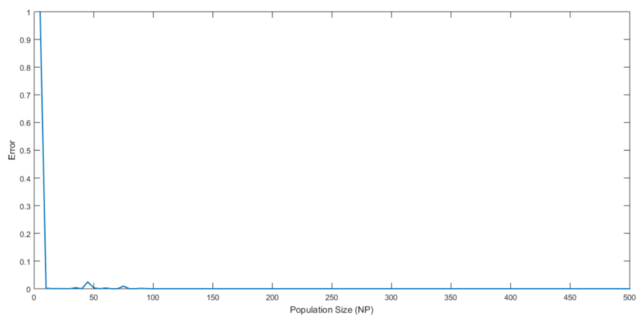
Parameter	Vasicek	CIR
NP	70	55
CR	0.85	0.6
F	0.6	0.5

Table 6.1.: Best values for the parameters of the DE algorithm

The Plots for the CIR model are shown in figure 6.7, 6.8 and 6.9. A look at figure 6.7 shows, that the CIR model converges with less NP and this is an advantage for the model, because it makes the model perform faster.

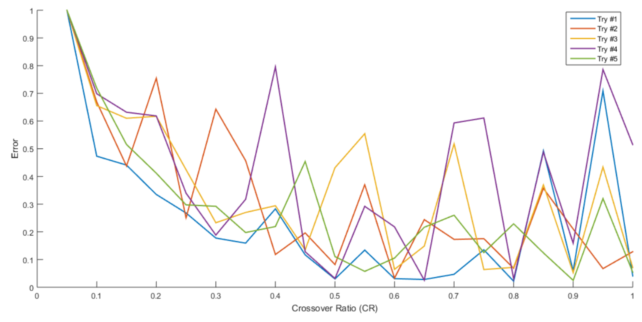


(a) 5 tries separately

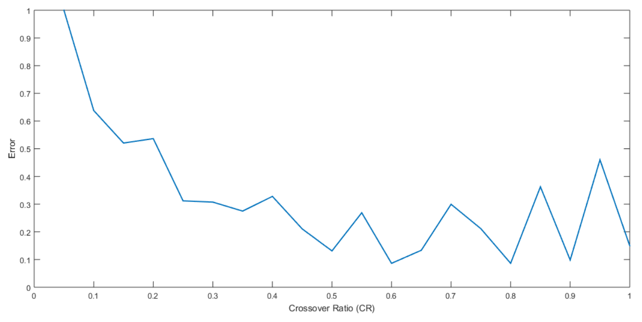


(b) Averaged

Figure 6.7.: Error vs. Population Size for CIR Model

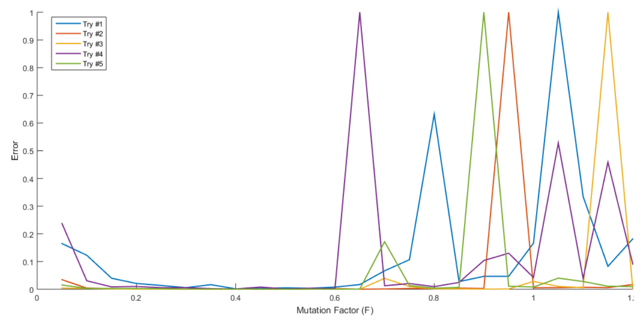


(a) 5 tries separately

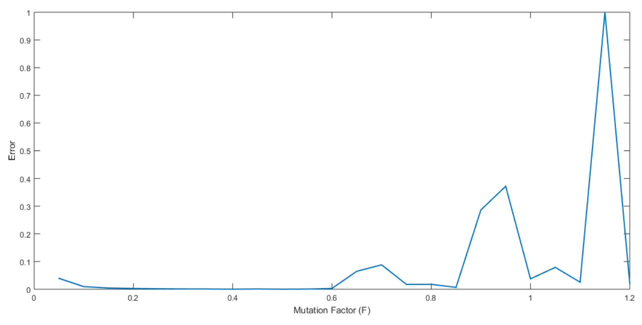


(b) Averaged

Figure 6.8.: Error vs. Crossover Ratio for CIR Model



(a) 5 tries separately



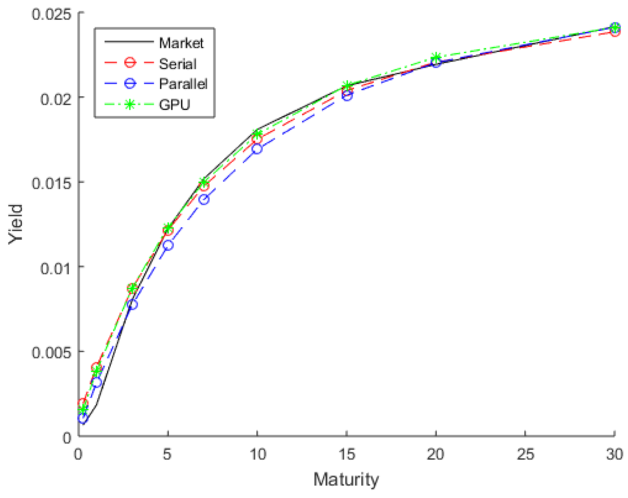
(b) Averaged

Figure 6.9.: Error vs. Scale Factor for CIR Model

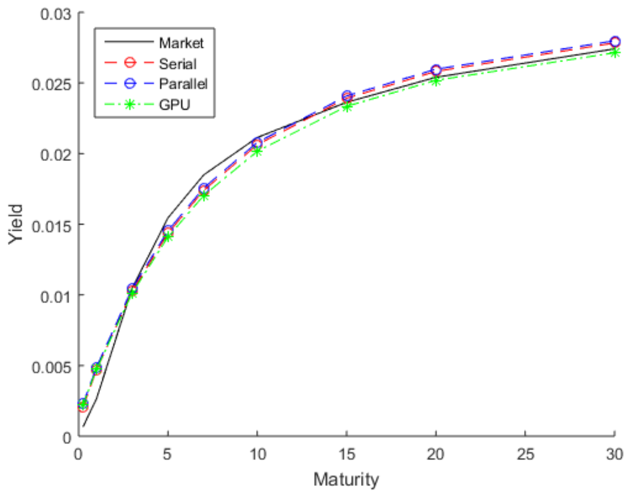
6.3. Accuracy of Models

In this section, we are checking the accuracy of the models. For each model, we run the code with best parameters. Also, for each model we run different versions of the written codes that are the *CPU Version*, *Parallel CPU Version* and the *GPU version*.

As mentioned before, we have the data for 180 different dates. Here, we present the accuracy for 12 dates of 1 year and the rest are not included here. The accuracy of the Vasicek model is compared to the market values and are shown in figures from 6.10 to 6.15.



(a) January 2015



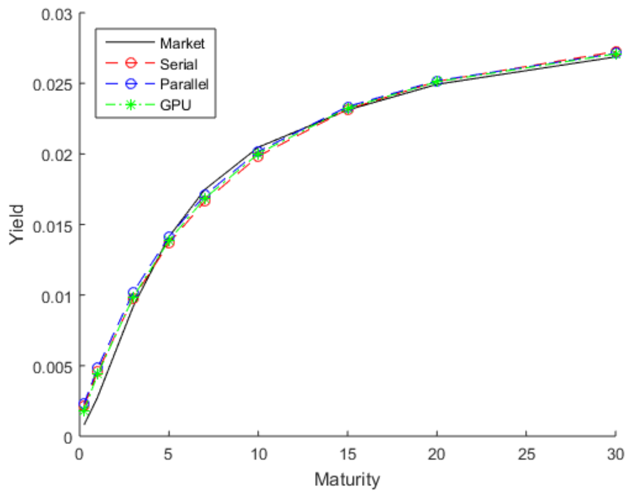
(b) February 2015

Figure 6.10.: Vasicek accuracy for different implementations

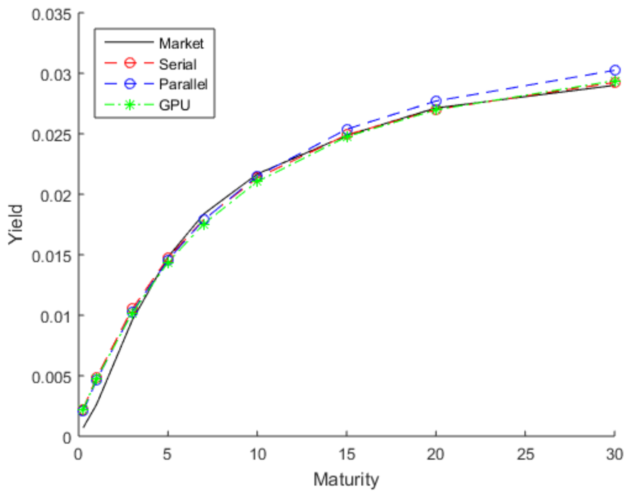
As it can be seen from the plots, All three versions have a very good accuracy. In some scenarios, one is slightly better than the other. But this can be neglected, because of the randomness we have in the model. The same results are taken for the CIR model and the results are the same.

Another Comparison is done between the Vasicek and CIR model again for 12 months. The plots can be seen in figures from 6.16 to 6.21.

From These Figures one can notice the fact, that how similar Both models are to each other. The reason of course, is because they are both one factor models and from one family of interest rate models. In this part, there is really no way to tell one model is better than another. Both models, have their advantages and disadvantages. The CIR model is more complicated to calculate the bond price, but unless the Vasicek model, it can prevent the short rate to be a negative value. But again as said before, having a negative value is not a big deal in today's finance world. Also, in some plots where both models are not that accurate (like July and December 2015), we can see that the Vasicek model performs a

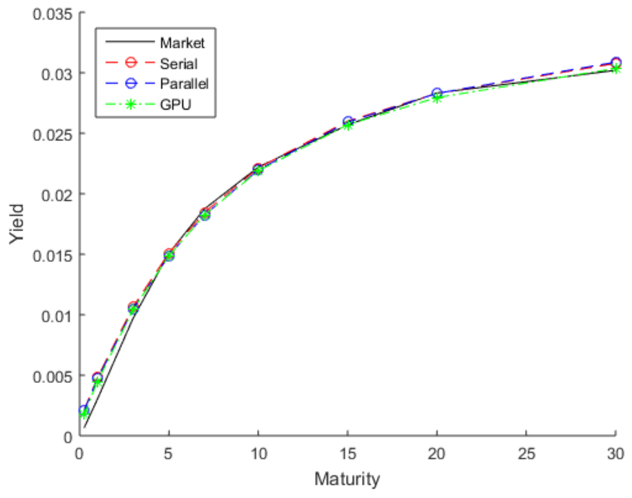


(a) March 2015

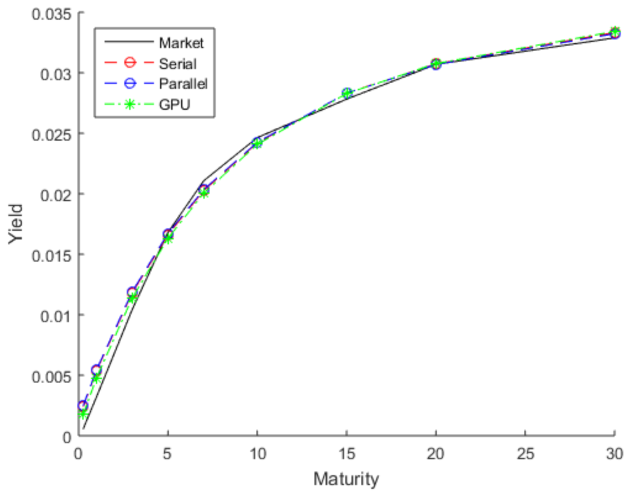


(b) April 2015

Figure 6.11.: Vasicek accuracy for different implementations

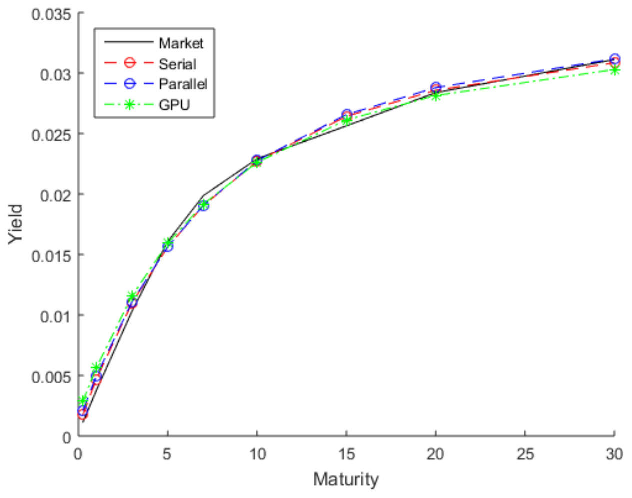


(a) May 2015

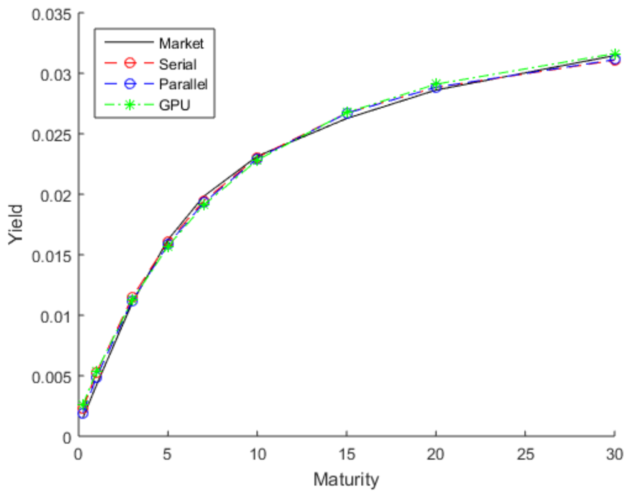


(b) June 2015

Figure 6.12.: Vasicek accuracy for different implementations

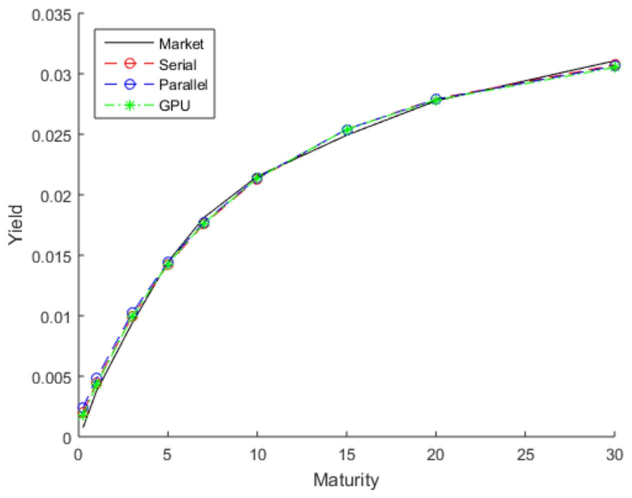


(a) July 2015

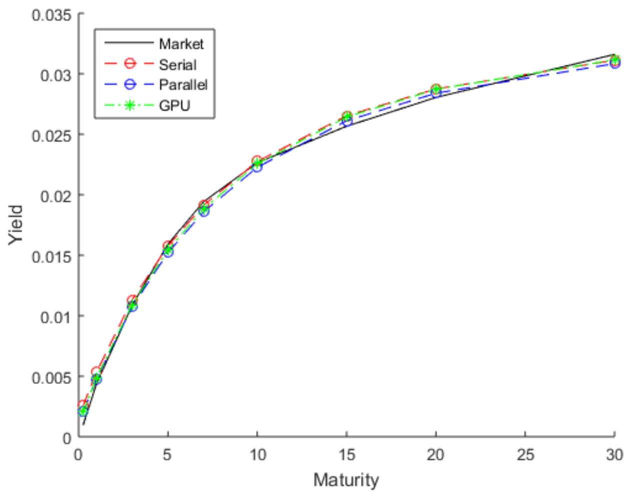


(b) August 2015

Figure 6.13.: Vasicek accuracy for different implementations

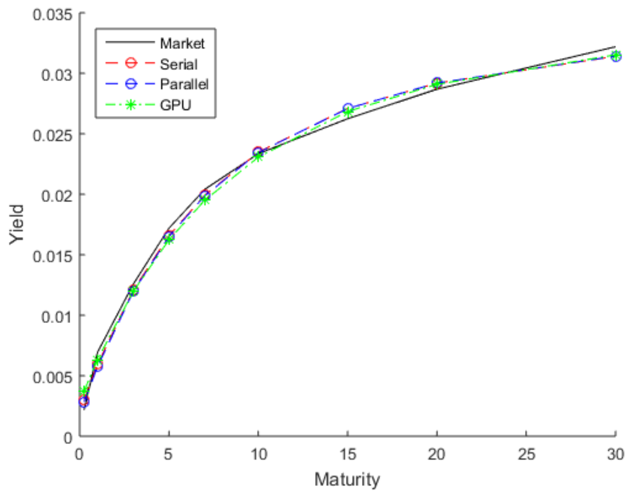


(a) September 2015

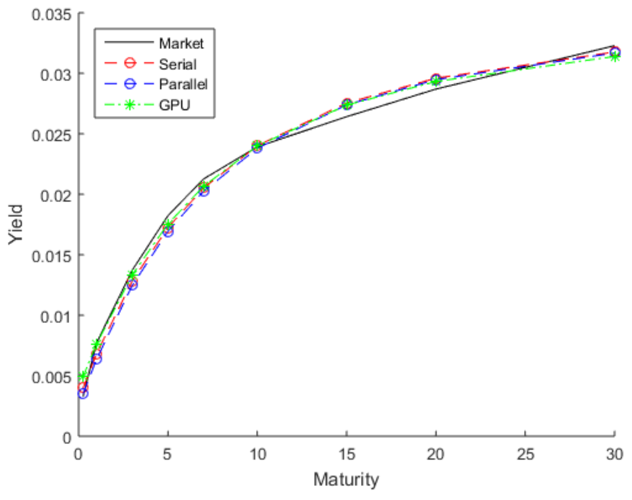


(b) October 2015

Figure 6.14.: Vasicek accuracy for different implementations

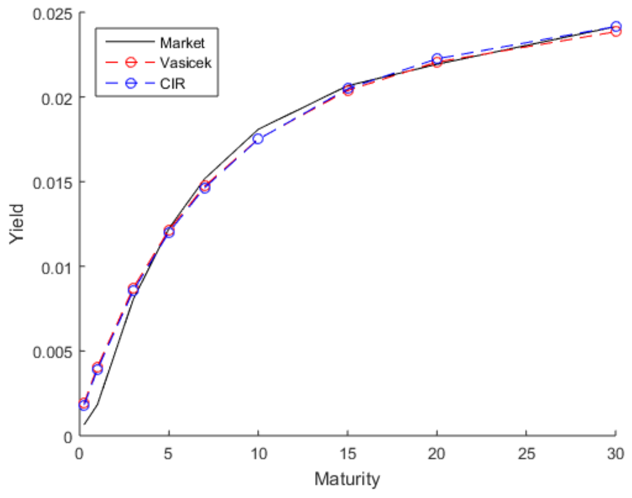


(a) November 2015

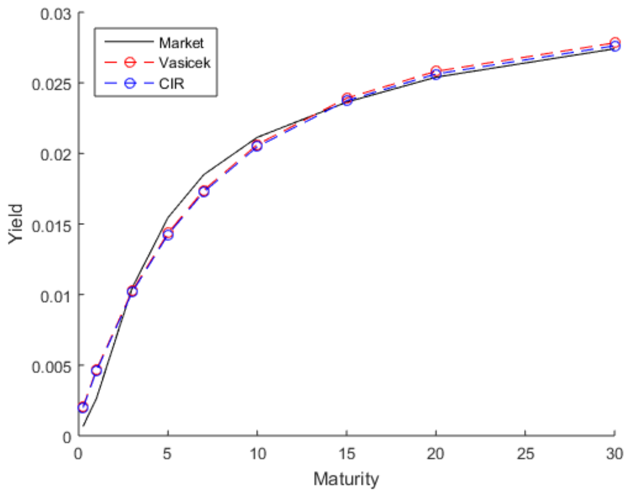


(b) December 2015

Figure 6.15.: Vasicek accuracy for different implementations

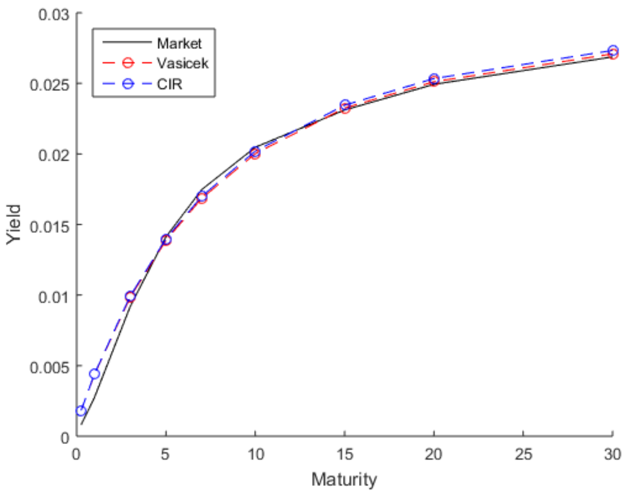


(a) January 2015

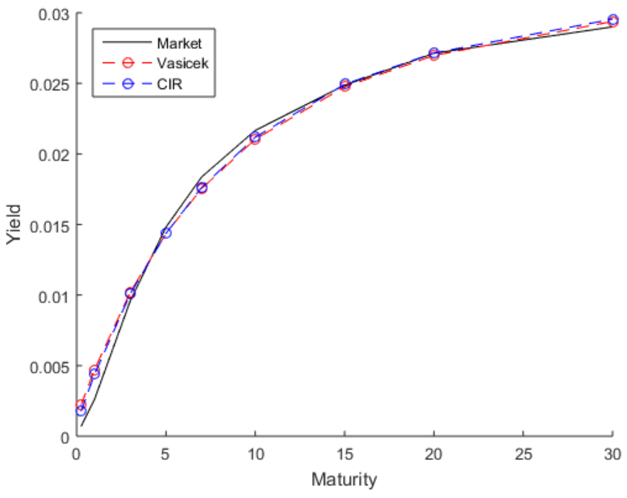


(b) February 2015

Figure 6.16.: Accuracy Comparison between the models

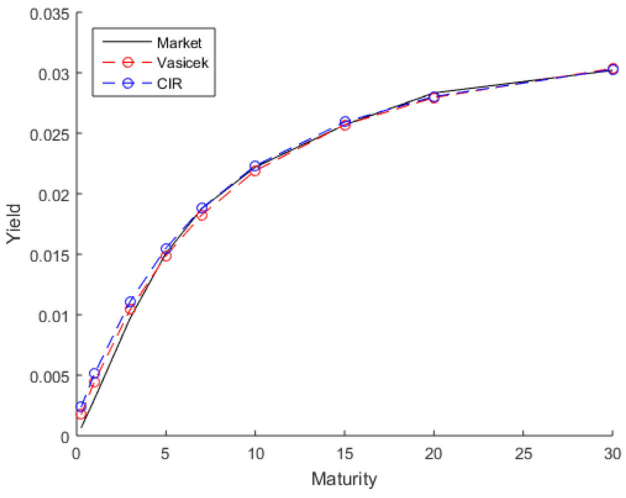


(a) March 2015

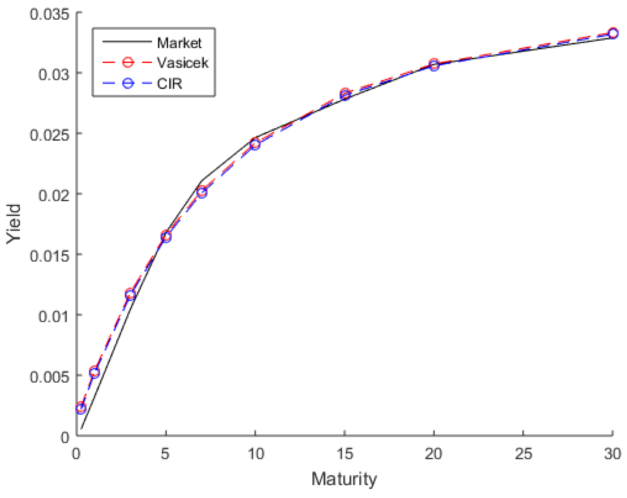


(b) April 2015

Figure 6.17.: Accuracy Comparison between the models

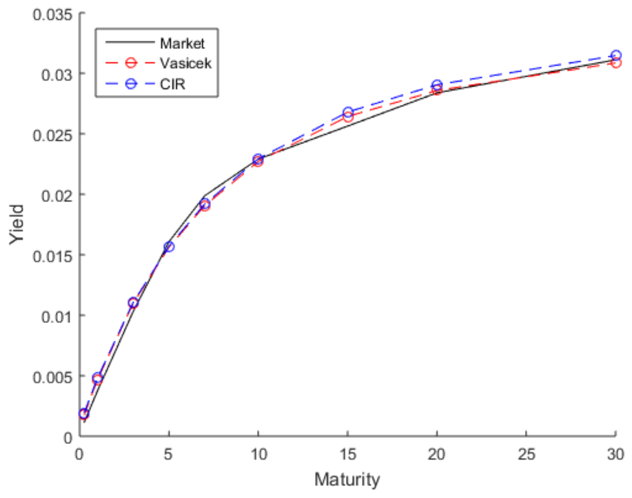


(a) May 2015

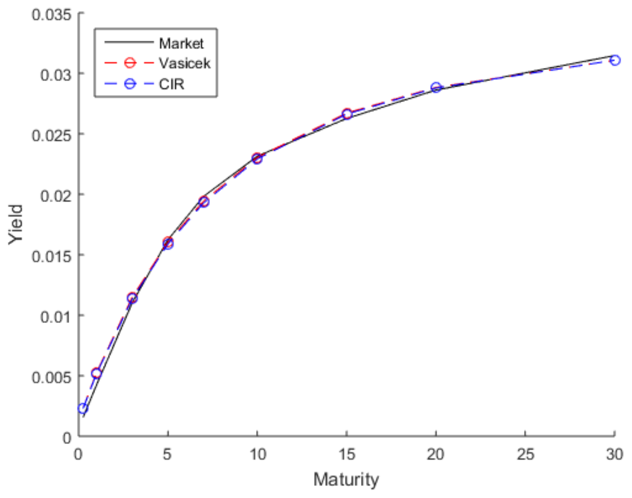


(b) June 2015

Figure 6.18.: Accuracy Comparison between the models

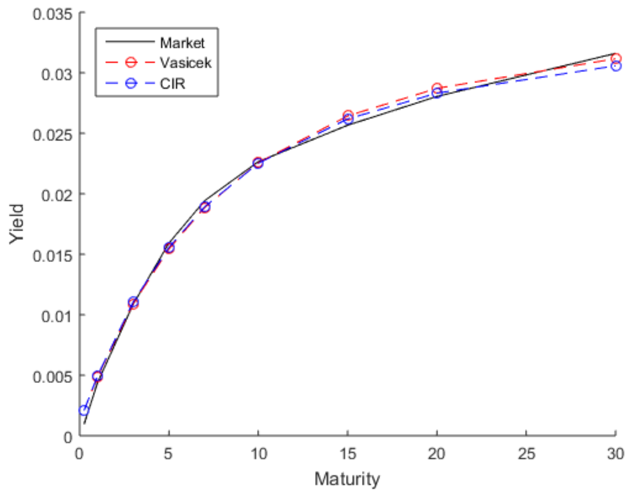


(a) July 2015

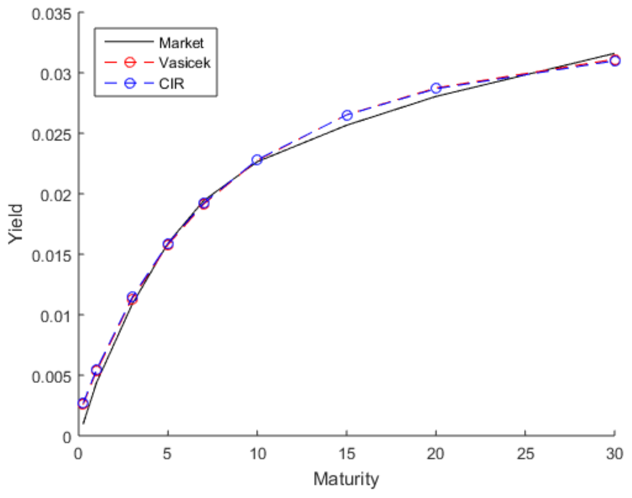


(b) August 2015

Figure 6.19.: Accuracy Comparison between the models

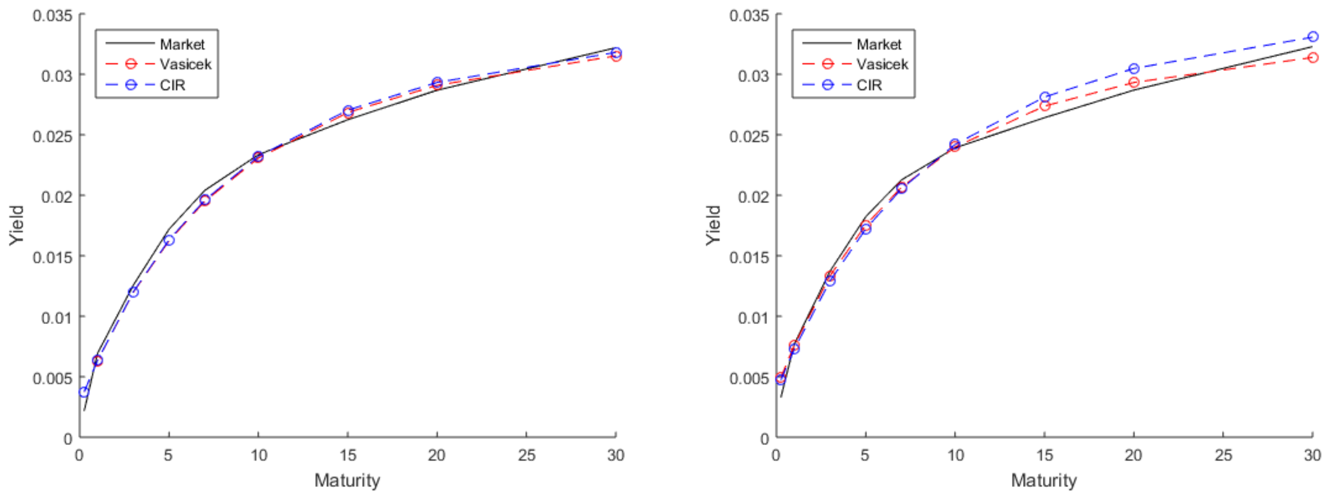


(a) September 2015



(b) October 2015

Figure 6.20.: Accuracy Comparison between the models



(a) November 2015 (b) December 2015

Figure 6.21.: Accuracy Comparison between the models

little better. But, generally we can say both models are behaving the same and as a matter of accuracy they are quiet similar.

6.4. Time Performance

Finally, We are going to check the time performance of the three code versions. We run the code for each version with different data size, with the diversity from 1 year to 15 years. We have used Three different systems for our purpose and extracted four results for time performance. We call the systems as follows:

- **LRZ Cluster**, with the CPU model `Intel Xeon E5-2690 v3` with 28 cores, used for the parallel CPU version.
- **Amazon GPU**, from the Amazon EC2 Instance, with the GPU model `NVIDIA GRID K520` with 1536 cores used, for the GPU version.
- **Sample PC**, with the CPU model `Intel Core i5` with 4 cores, used for the serial and parallel CPU version.

It is important to mention here, that there was no access to the risklab systems, therefore we used a sample pc which has the closest specs to a risklab system. A risklab system, has a CPU model of `Intel Core i7` and a GPU of `Nvidia Tesla K20c`.

The main Comparison would be between the serial CPU version and the parallel GPU version, but the fair comparison would be between the LRZ CPU and the Amazon GPU. Figures 6.22 and 6.23 show the time performance for all the versions for each model.

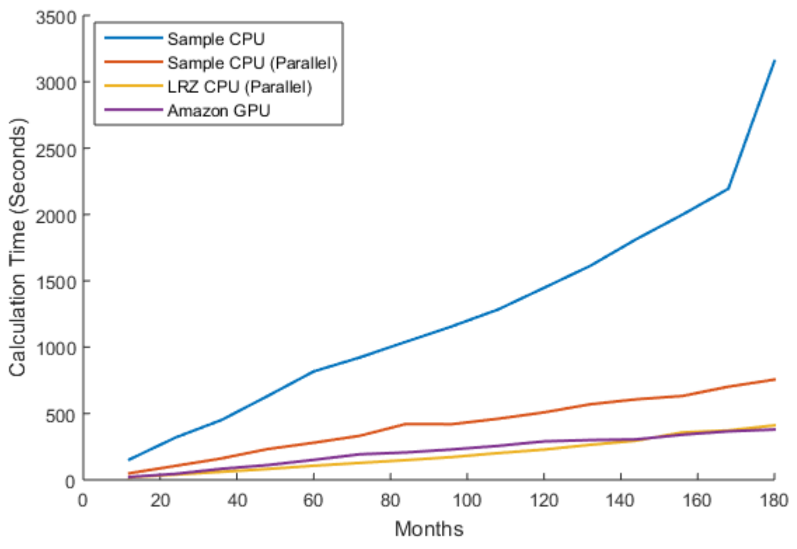


Figure 6.22.: Calculation Time for the Vasicek Model

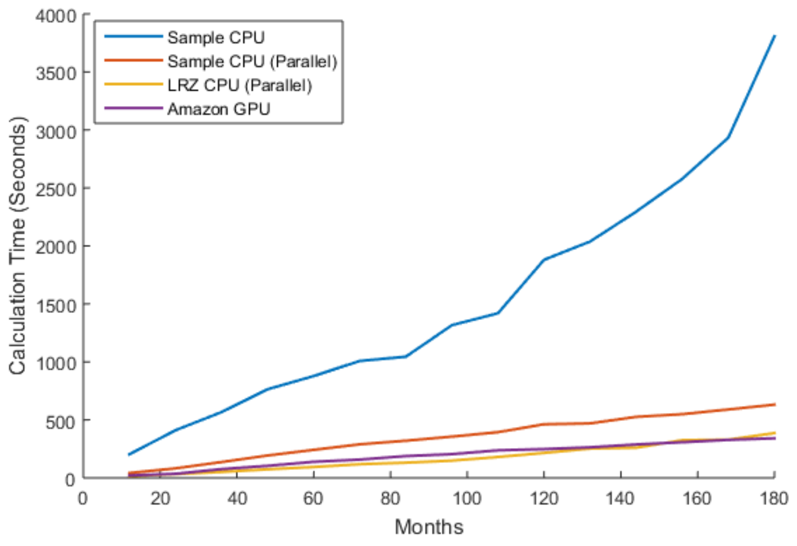


Figure 6.23.: Calculation Time for the CIR Model

As in can be seen from the values, The Speedup of the GPU version is rather good comparing to the serial version. we have an average speed up factor of 7.87 from the serial CPU to the GPU version. This amount of speedup between the CPU and GPU version, is considered perfect for risklab work purposes.
When comparing the GPU with the parallel version of the LRZ system, we can see the LRZ results are slightly better at first, but increasing the size of data makes the difference become less and at the end, the GPU version performs better. It is important to notice that the real data in risklab is way more than the 180 dates that we have, and this makes using the GPU version more necessary. In figure 6.24, we took a closer look to the results of the Amazon system and the LRZ system. We added one more data for 360 months, to shows the improvement of the Amazon GPU comparing to LRZ CPU.

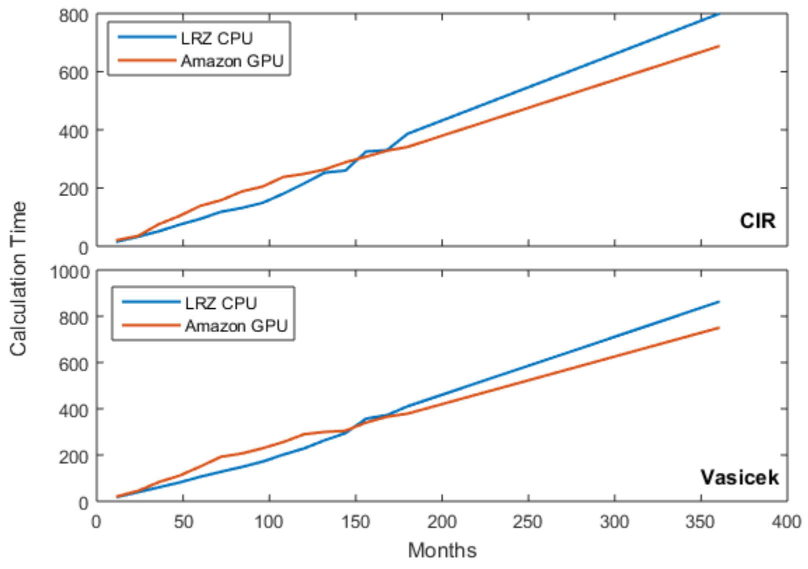


Figure 6.24.: Calculation Time for each Model

One more fact to remember is, that in risklab there is no access to a system like LRZ but instead, they have a system more like the sample pc, and there, the GPU version again performs better than the parallel CPU and has an average speedup of 1.8.
A comparison between the Vasicek and the CIR model shows, that the CIR model performs about 18% slower than the Vasicek model in the serial version. But in the parallel CPU version, CIR is around 10% faster and in GPU version it is around 16% faster than Vasicek. The reason is, although the CIR model is more complex mathematically, but it converges faster, in fewer iterations. These fewer iterations, does not help the serial version, but in parallel version it makes the CIR model perform faster. Therefore, with respect to the comparison we had in the previous chapter, that showed both models are rather similar, we can conclude that using the CIR model is more practical with the risklab data.

There is one concern about the method we use. Let's say we have a data set for 100 different dates. The GPU code we have, is going to do a parallel calculation of the DE algorithm 100 times for each of the dates. But the calculation for each date is done sequentially. In other words, we have an outer loop that is loading the data set for each date, and is running the DE algorithm on that data in an inner loop. The inner loop with the DE algorithm is parallel but the outer loop is not.

One may suggest, that we can do a parallelism for the outer loop and not the inner loop, so the algorithm would be faster. This is not applicable on the GPU, because the DE algorithm in the inner loop is computationally expensive and if we run it sequentially on the GPU it will consume way more time. The optimal way, would be to parallel both the inner and the outer loop. For that purpose, we have to write the GPU code in a way that each kernel has other kernel calls inside it. For instance, we make a block of 8 GPU cores for each date, and then use these 8 cores for parallelization of the DE algorithm. Although, calling kernel inside kernel is applicable in CUDA, but it is a very challenging work. It increases the code complexity and it was not possible to be finished in a short time span. However, this can be a research topic for future works, but for now, the current version that we have is enough for the purposes of risklab usage.

In conclusion, we can say that using the GPU version is practical for risklab and it is possible to reach the acceptable results with a faster performance to calibrate the aforementioned interest rate models.

7. Conclusion

7.1. Overview of the Work

Interest rate modeling is a vast topic in finance and it is a useful one in many scenarios. Calibrating interest rate models means that we have to find a set of parameters for a given model and use those parameters to find the best values for interest rates of a specific date. We figured out that although these parameters are meant to be constant, but in reality they are changing in a range, depending on each set of historical data. Therefore, calibration is a necessary step in modeling interest rates.

In this work, we have studied two different short rate models, the Vasicek and the CIR model. These models are considered as one-factor models, because they have only one factor of randomness in their equation.

To calibrate these models we had to follow a set of different paths. We had to first find a proper cost function for each model and try to minimize the cost function using an optimization algorithm. The cost function of the model is in fact the yield or forward interest rate of an instrument, depending on the maturity time. We had taken a little help from the Monte Carlo method and having the solutions for the differential equations of each model, we were able to extract the cost function.

The next step is to fit the extracted cost function to our given data. We have used the Differential Evolution algorithm to fit the cost function. DE is a global optimization method from the family of evolutionary algorithms. It has some advantages helpful for our problem, of which , it does not need a derivative of the cost function.

We used the DE algorithm and implemented a code with `C++` on the CPU. The CPU implementation was accurate enough for both models but the performance was not of interest. Therefore, we made the algorithm parallel, once on the CPU with the help of `OpenMP` library, and once on the GPU using `CUDA`. The results were rather good in the GPU version and it gave us a speedup of 7.87 times faster than the CPU version. Therefore, we showed that using a GPU would increase the time performance of a calibration process with an acceptable rate, while keeping the accuracy firm.

We have also studied different parameter changes in the DE algorithm. This algorithm has three main parameters: CR, NP, and F. Depending on the problem, the parameters are behaving differently. For this reason, we have checked all the three parameters for our own problem and came up with the best values for these parameters and also for both Vasicek and CIR models.

The main goal of this project was to calibrate two different short-rate models with a fine accuracy and a fast performance. We suggested using the DE algorithm to calibrate these models. Our algorithm facilitates optimal calculation of the selected short-rate models. On the other hand, we suggested using the GPU for this calibration to increase the performance. The algorithm was showing a good performance increase as well, and the goal to increase the speed while keeping the accuracy has been achieved.

7.2. Future Studies

By a quick look at the studies around the DE algorithm, one can easily see that there is more interest in using the GPU rather than the CPU for this algorithm. The reason of course is the structure of the algorithm. The algorithm has a data-parallel algorithmic structure, which means it consists of small calculation parts and that is perfect for the GPU.

With the infrastructure available at risklab, it is obvious that using the GPU implementation is the best option to calibrate the short-rate models. However, if there is access to better CPUs, like the Clusters at LRZ, using a parallel implementation over the CPU with OpenMP is much simpler and more acceptable. With a powerful CPU, one can try to implement the calibration with MPI (Message Passing Interface) and mix it with OpenMP and come up with an even faster solution. There are several packages and implemented libraries out there that already mixed MPI and OpenMP. One can go one step further and use the hybrid packages that use the power of both CPU and GPU to run an algorithm. Of course all of these comes with a cost of time and resource and one should consider these facts as well.

We have compared the Vasicek model and the CIR model in two major parts, accuracy and speed. One can also check the sensitivity of the model parameters to compare both models in a different way. Sometimes a small change in one parameter will lead to a big change in the short-term interest rate. This big changes can lead to a less accurate yield calculation and one can check which of the models perform better in this situation.

In this work, we just calibrated the one-factor models, but another future study would be using the DE algorithm and the GPU to calibrate the more complicated multi-factor models like Hull-White and Nelson-Siegel-Svensson which are among the most useful and important interest rate models. There exists various studies about calibration of these models, but few have tried to use a parallel implementation. A parallel implementation would of course help the financial engineers to calibrate more data and predict the market behavior faster.

Basically, as we have seen in the previous chapter this topic is rather new and it has a lot of other open doors to discover.

Appendix

A. Bond Price Under The Vasicek Model

In this section, first we solve the Vasicek model and then use this solution to calculate the bond price. The solution to CIR model follows the same procedure as well. The Vasicek differential equation is:

$$dr_t = \alpha(\beta - r_t)dt + \sigma dW_t, \qquad r(0) = r_0 \tag{A.1}$$

To solve this equation, we are using the integrating factor to separate the r terms.

An integrating factor is a function that can be multiplied to an ordinary differential equation and make it integrable [23]. For example, when we have a linear first-order ordinary differential equation of type

$$\frac{dy}{dx} + f(x).y = g(x) \tag{A.2}$$

then we can define an integrating factor such as $e^{\int f(x)dx}$. This factor will allow us to change the ODE into something like the following.

$$X'Y + XY' = d(x,y)$$

Which can be solved using the chain rule. Thus, we are able to separate the terms.

from equation A.1 we have:

$$dr_t + \alpha r_t dt = \alpha\beta dt + \sigma dW_t \tag{A.3}$$

Multiply it by $e^{\alpha t}$ we get:

$$e^{\alpha t}(dr_t + \alpha r_t dt) = e^{\alpha t}(\alpha\beta dt + \sigma dW_t) \tag{A.4}$$

Now we are able to separate the variables and do the integral.

$$\int_{t=s}^{t=T} d(e^{\alpha t}.r_t) = \int_{t=s}^{t=T} e^{\alpha t}(\alpha\beta dt + \sigma dW_t) \tag{A.5}$$

where s is the spot time or t_0. Solving this integral will lead to

$$r(T) = e^{-\alpha(T-s)}r(s) + \beta(1 - e^{-\alpha(T-s)}) + \sigma\int_{t=s}^{t=T} e^{-\alpha(T-t)}dW_t \tag{A.6}$$

Since we know r is a random process with a normal distribution, we have to find it's mean (expectation) and variance.

$$r(T) = N(E(r), V(r)) \tag{A.7}$$

With knowing the fact that variance of a number is zero and expectation of a Brownian motion is zero, we can separate the terms in equation A.6.

$$r(T) = \underbrace{e^{-\alpha(T-s)}r(s) + \beta(1 - e^{-\alpha(T-s)})}_{\text{Variance = 0}} + \underbrace{\sigma \int_{t=s}^{t=T} e^{-\alpha(T-t)} dW_t}_{\text{Expectation=0}} \tag{A.8}$$

Therefore the variance would be:

$$V(r(T)) = V\left(\sigma \int_{t=s}^{t=T} e^{-\alpha(T-t)} dW_t\right) \tag{A.9}$$

And we know that the expectation of a number is the number itself:

$$E(r(T)) = e^{-\alpha(T-s)}r(s) + \beta(1 - e^{-\alpha(T-s)}) \tag{A.10}$$

The remaining part is to calculate the variance in equation A.9.
When we have a deterministic function of time on a Brownian motion, we can use Itô isometry. Itô isometry, helps to compute the variance of random variables by using the following formula:

$$E\left(\int_{t=s}^{t=T} f(u) dw_u\right) = E\left(\int_{t=s}^{t=T} \underbrace{X_t^2}_{\text{square the deterministic part}} \overbrace{dt}^{\text{dw.dw=dt}}\right) \tag{A.11}$$

we also know, that variance can be decomposed into expectation terms:

$$V(X) = E(X^2) - \underbrace{[E(X)]^2}_{=0} \tag{A.12}$$

The second term is zero because as we said expectation of a Brownian motion is zero.

Therefore, from equation A.11 and the fact from A.12 we can write the variance as follows:

$$\begin{aligned} V(r_t) &= E\left(\sigma^2 \int_s^T e^{-2\alpha(T-t)} dt\right) \\ &= \sigma^2 e^{-2\alpha t} E\left(\int_s^T e^{-2\alpha(-t)} dt\right) \\ &= \sigma^2 \frac{1}{2\alpha}\left(1 - e^{-2k(T-s)}\right) \end{aligned}$$

So finally r_t would be a normal distribution with the following mean and variance.

$$r_t \sim N\left(e^{-\alpha(T-s)}r(s) + \beta(1 - e^{-\alpha(T-s)}), \sigma^2 \frac{1}{2\alpha}\left(1 - e^{-2\alpha(T-s)}\right)\right) \tag{A.13}$$

Now that we have the solution for r_t, we can get the zero-coupon bond price $P(s,t)$. For any model, bond price is calculated as follows:

$$P(s,t) = E\left(e^{\int_s^T r_t dt}/F_s\right) \tag{A.14}$$

where F_s is a constant and for the Vasicek Model $F_s = r_s$. We know that r_t is a normal distribution. Therefore:

$$E(e^X) = e^{E(X)+\frac{1}{2}V(X)} \tag{A.15}$$

And this means the bond price would be:

$$P(s,t) = exp\left(E\left[\int_s^T r_t dt\right] + \frac{1}{2}V\left[\int_s^T r_t dt\right]\right) \tag{A.16}$$

So, we need to calculate:

$$E\left[\int_s^T r_t dt\right] \quad , \quad V\left[\int_s^T r_t dt\right]$$

This calculation of these two values, is a cumbersome mathematical procedure. By solving them, we will have,

$$E\left[\int_s^T r_t dt\right] = \frac{r_s - \beta}{\alpha}(1 - e^{-\alpha(T-s)}) - \beta(T-s) \tag{A.17}$$

And,

$$V\left[\int_s^T r_t dt\right] = \frac{\sigma^2}{2\alpha^3}(2\alpha(T-s) - 3 + 4e^{-\alpha(T-s)} - e^{-2\alpha(T-s)}) \tag{A.18}$$

Putting A.17 and A.18 into equation A.16 will give us the bond price as follows,

$$\begin{aligned}
P(s,t) &= exp\bigg(\frac{r_s-\beta}{\alpha}(1-e^{-\alpha(T-s)})-\beta(T-s) \\
&\quad + \frac{\sigma^2}{4\alpha^3}(2\alpha(T-s)-3+4e^{-\alpha(T-s)}-e^{-2\alpha(T-s)})\bigg) \\
&= exp\bigg[-\bigg(\frac{1-e^{-\alpha(T-s)}}{\alpha}\bigg)r_s+\beta\bigg(\frac{1-e^{-\alpha(T-s)}}{\alpha}-(T-s)\bigg) \\
&\quad - \frac{\sigma^2}{2\alpha^2}\bigg(\frac{1-e^{-\alpha(T-s)}}{\alpha}\bigg)+\frac{\sigma^2}{2\alpha^2}(T-s)-\frac{\sigma^2}{4\alpha^2}\bigg(\frac{1-2e^{-\alpha(T-s)}+e^{-2\alpha(T-s)}}{\alpha^2}\bigg)\bigg] \\
&= exp\bigg[-B(s,t)r_s+\beta B(s,t)-\beta(T-s)-\frac{\sigma^2}{2\alpha^2}B(s,t) \\
&\quad - \frac{\sigma^2}{2\alpha^2}(T-s)-\frac{\sigma^2}{4\alpha}B(s,t)^2\bigg] \\
&= A(s,t).e^{-r_sB(s,t)}
\end{aligned} \tag{A.19}$$

By using T as the Maturity date, we will reach the Bond Price equation,

$$P(s,T)=A(s,T).e^{-r(s).B(s,T)} \tag{A.20}$$

where,

$$B(s,T)=\frac{1}{\alpha}(1-e^{-\alpha(T-s)}) \tag{A.21}$$

and,

$$A(s,T)=exp\bigg(\frac{(B(s,T)-T+s)(\alpha^2\beta-\sigma^2/2)}{\alpha^2}-\frac{\sigma^2B(s,T)^2}{4\alpha}\bigg) \tag{A.22}$$

As mentioned before, one can do the same procedure to extract the values for A and B for the CIR model.

B. PseudoCode for DE algorithm

In this Appendix, we are going to show a pseudocode for the three most important functions we have implemented. The functions are `main`, `runDE` and `runMethod`. The `main` function calls the `runDE` function and the `runDE` function calls the `runMethod` inside itself.

Function 1 main function for Calibration of interest rate models using Differential Evolution

```
function main ($fileName, methodName$)
1: $marketData \leftarrow readData(fileName)$
2: ▷ MarketData is a 2D vector with size monthsCount * MaturityCount
3: for $i = 1$ to $monthsCount$ do
4:     $currentMonthMarketData \leftarrow marketData_i$
5:     $runDE(currentMonthMarketData)$
6:     $ModelParameters_i \leftarrow getCurrentMonthModelParameters()$
7:     $ModelData_i \leftarrow getCurrentMonthModelData()$
8: end for
9: $writeNewData(ModelParameters, ModelData)$
```

Function 2 runMethod, calculates the cost funtion for Vasicek or CIR model

```
function runMethod ($Population$)
10: $maturityCount \leftarrow 9$
11: $r_0 \leftarrow getR_0()$
12: $newR_0 \leftarrow getNewR_0(methodName)$ ▷ Here, the Vasicek and CIR method is chosen
13: for $i = 1$ to $maturityCount$ do
14:     $currentMonthModelData_i \leftarrow calculateYield(newR_0)$
15: end for
16: for $i = 1$ to $maturityCount$ do
17:     $errorSum+ = (currentMonthModelData_i - currentMonthMarketData_i)^2$
18: end for
19: $currentMonthError \leftarrow \frac{errorSum}{maturityCount}$
20: return $currentMonthError$
```

Function 3 runDE, runs the Differential Evolution Algorithm

```
function runDE ($currentMonthMarketData$)
define population size, $NP$, Crossover Ratio, $CR$ and Scaling Factor, $F$
$Population \leftarrow randomGenerator(NP)$
Set Method as Vasicek or CIR
while Max Generations reached do
    for $i = 1$ to $NP$ do
        $Error_i \leftarrow runMethod(Population_i)$
    end for
    $averageError \leftarrow \sum Error_i / NP$
    if $averageError \leqslant tolerance$ then
        break
    end if
    for $i = 1$ to $NP$ do
        $mutatedPopulation_i \leftarrow generateMutatedPopulation(F)$
    end for
    for $i = 1$ to $NP$ do
        $crossOverPopulation_i \leftarrow selectFromMutatedOrOriginal(CR)$
    end for
    for $i = 1$ to $NP$ do
        $crossOverError_i \leftarrow runMethod(crossOverPopulation_i)$
    end for
    for $i = 1$ to $NP$ do
        if $crossOverError_i < Error_i$ then
            $newPopulation_i \leftarrow crossOverPopulation_i$
        else
            $newPopulation_i \leftarrow Population_i$
        end if
    end for
    $Population \leftarrow newPopulation$
end while
return final $modelParameters$ and $Population$
```

Bibliography

[1] D. H. Ackley. A connectionist machine for genetic hillclimbing. *Kluwer Academic Publishers*, 1987.

[2] G. Affken. *Mathematical Methods for Phyisicist*. Academic Press, Orlando, 1985.

[3] Michael Aichinger, Andreas Binder, Johannes Fürst, and Christian Kletzmayr. A fast and stable heston model calibration on the gpu. *Euro-Par 2010 Parallel Processing Workshops*, 2010.

[4] H.H.N. Amin. Calibration of different interest rate models for a good fit of yield curves. *Master Thesis Paper, TU Delft*, 2012.

[5] A. Bjorck. *Numerical Methods for Least Square Problems*. SIAM, 1996.

[6] E. Castillo, C. Camarero, and A. et al. Borrego. Financial applications on multi-cpu and multi-gpu architectures. *The Journal of Supercomputing*, 71.

[7] K.C. Chan, Andrew Karoyli, Franciss Longstaff, and Antony Sanders. An empirical comparison of alternative models of the short-term interest rate. *The Journal of Finace*, XLVII(3), July 1992.

[8] Les clewlow and Chris Strickland. Monte carlo valuation of interest rate derivatives under stochastic volatility. *Financial Options research center*, 1996.

[9] J.C. Cox, J.E. Ingersoll, and S.A. Ross. "a theory of the term structure of interest rates. *Econometrica*, 53.

[10] Leonardo Dagum and Ramesh Menon. Openmp: An industry-standard api for shared-memory programming. *IEEE Computational Science and Engineering*, 1:46–55, 1998.

[11] G. De Rossi. Maximum likelihood estimation of the cox-ingersoll-ross model using particle filters. *Computing in Economics and Finance*, 302, 2004.

[12] L. de Veronese and R. Krohling. Differential evolution algorithm on the gpu with c-cuda. *Evolutionary Computation (CEC) IEEE Congress*, pages 1–7, 2010.

[13] J.-CH. Duan, G. Gauthier, J.-G. Simonato, and S. Zaanoun. Maximum likelihood estimation of structural credit spread models - deterministic and stochastic interest rates. 2002.

[14] Frank J. Fabozzi. *Perspectives on Interest Rate Risk Management for Money Managers and Traders*. Frank J. Fabozzi associates, 1998.

[15] John C. Hull and Alan White. The general hull-white model and super calibration. *NYU Working Paper*, (FIN-00-024), 2000.

[16] Pavel Kromer and Jan Plato. Many-threaded differential evolution on the gpu. 2013.

[17] G. A. Laguna-Sanchez, M. Olguin-Carbajal, N. Cruz-Cortes, R. Barron-Fernandez, and R. Cadena Martinez. A differential evolution algorithm parallel implementation in a gpu. *Journal of Theoretical and Applied Information Technology*, 86:184–195, 2016.

[18] S. Li. An empirical study of australian short-term interest rate: A comparison of single factor models. *School of Finance and Business Economics*, 2000.

[19] L. Maciel, F. Gomide, and R. Ballin. A differential evolution algorithm for yield curve estimation. *Mathematics and Computers in Simulation*, 2016.

[20] Eric Magnus and Pedesen Hvass. Good parameters for differential evolution. *Hvass Laboratories*, 2010.

[21] R.C. Merton. On estimating the expected return on the market: An exploratory investigation. *Journal of Financial Economics*, 8:323–361, 1980.

[22] Nicholas Metropolis and Ulam S. The monte carlo method. *Journal of the American Statistical Association*, 44(247):335–341, 1949.

[23] Joakim Munkhammar. Integrating factor. from mathworld–a wolfram web resource, created by eric w. weisstein.

[24] Yousef Nashed, Roberto Ugolotti, Pablo Mesejo, and Stefano Cagnoni. libcudaoptimize: an open source library of gpu-based metaheuristicsg. *Proceedings of the 14th annual conference companion on Genetic and evolutionary computation*, pages 117–123, 2012.

[25] J. Nelder and Mead. R. A simplex method for function minimization. *Comp. J.*, 7:341–359, 1965.

[26] Conall O'Sullivan. Parameter uncertainty in kalman filter estimation of the cir term structure model. *Centre for Financial Markets working paper series*, 2007.

[27] Banu özgürel and Gönül Ayranci. Monte carlo simulation for vasicek interest rate model parameters. *Digital Proceeding Of The ISDS*, 2014.

[28] Kin Pang. Calibration of interest rate term structure and derivative pricing models. *Doctoral Dessrtation*, 1997.

[29] A.K. Qin, Federico Raimondo, Florence Forbes, and Yew Soon Ong. An improved cuda-based implementation of differential evolution on gpu. *GECCO '12 Proceedings of the 14th annual conference on Genetic and evolutionary computation*, pages 991–998, 2012.

[30] M.C.A. Ruijter. An interest rate model for counterparty credit risk. *Master Thesis Paper, TU Delft*, 2010.

[31] Rainer Storn. On the usage of differential evolution for function optimization. *NAFIPS*, pages 519–523, 1996.

[32] Rainer Storn and Kenneth Price. Differential evolution - a simple and efficient heuristic for flobal optimization over continuous spaces. *Journal of Global Optimization*, 11:341–359, 1997.

[33] Rainer Storn and Kenneth Price. Differential evolution a simple evolution strategy for fast optimization. *Dr. Dobb's Journal*, 1997.

[34] Emile A.L.J. van Elen. Term structure forecasting. *Thesis at Tilburg Univerity*, July 2010.

[35] O. Vasicek. An equilibrium characterization of the term structure. *Journal of Financial Econimics*, 5:177–186, 1977.

[36] Ian E. Vollrath and Jürgen Wendland. Calibration of interest rate and option models using differential evolution. *FINCAD Corporation*, 2009.

[37] Serkan Zeytun and Zeytun Gupta. A comparative study of the vasicek and the cir model of the short rate. *Frauenhofer Institut Techno- und Wirthschaft Informatik*, July 2007.

[38] W. Zhu and Y. Li. Gpu-accelerated differential evolutionary markov chain monte carlo method for multi-objective optimization over continuous space. *Proceeding of the 2nd workshop on Bio-inspired algorithms for distributed systems*, pages 1–8, 2010.