Stochastic volatility models with applications in finance

Ze Zhao
University of Iowa

Follow this and additional works at: https://ir.uiowa.edu/etd
Part of the Applied Mathematics Commons

STOCHASTIC VOLATILITY MODELS WITH APPLICATIONS IN FINANCE

by

Ze Zhao

A thesis submitted in partial fulfillment of the
requirements for the Doctor of Philosophy
degree in Applied Mathematical and Computational Sciences
in the Graduate College of
The University of Iowa

December 2016

Thesis Supervisor: Professor Palle Jorgensen

Graduate College
The University of Iowa
Iowa City, Iowa

CERTIFICATE OF APPROVAL

PH.D. THESIS

This is to certify that the Ph.D. thesis of

Ze Zhao

has been approved by the Examining Committee for the thesis requirement for the Doctor of Philosophy degree in Applied Mathematical and Computational Sciences at the December 2016 graduation.

Thesis committee: __________________________
Palle Jorgensen, Thesis Supervisor

____
Gerhard Strohmer

____
Keith Stroyan

____
Tong Li

____
Bruce Ayati

ACKNOWLEDGEMENTS

First of all, I would like to sincerely thank my advisor, Professor Palle Jorgensen, for his patience, support, and guidance. As a remarkable role model to me, his professionalism and enthusiasm for math will benefit me for the rest of my life. I am very grateful to the rest of my committee: Prof. Keith Stroyan, Prof. Bruce Ayati, Prof. Tong Li, and Prof. Gerhard Strohmer, for their invaluable assistance. I would also like thank Prof. Laurent Jay and Prof. Weimin Han, who have always made me feel supported during my study in the University of Iowa.

I would like to thank Catie and Nathaniel, for all the precious and warm memories you brought to me. Many thanks to Boshi, Tianyi, Kai, and Zhenhao, for the joyful days we spent together. Special thank to Chenhong, for her generous help for my job finding. Thank you to the rest of my friends and classmates at Iowa: yangyang, Mario, Colin, Kevin, Rachael, Christine, Rich, Gordon, Julia, and many others.

Thank you to my dad and my uncle Zhongren Zhao, for endless love and support I got from you. Last and the most important, I would like to thank my mom. There is no word to describe how much I love you. I am proud of many things in my life, but nothing beats being your son, and making your proud.

ABSTRACT

Derivative pricing, model calibration, and sensitivity analysis are the three main problems in financial modeling. The purpose of this study is to present an algorithm to improve the pricing process, the calibration process, and the sensitivity analysis of the double Heston model, in the sense of accuracy and efficiency. Using the optimized caching technique, our study reduces the pricing computation time by about 15%. Another contribution of this thesis is: a novel application of the Automatic Differentiation (AD) algorithms in order to achieve a more stable, more accurate, and faster sensitivity analysis for the double Heston model (compared to the classical finite difference methods). This thesis also presents a novel hybrid model by combing the heuristic method Differentiation Evolution, and the gradient method Levenberg–Marquardt algorithm. Our new hybrid model significantly accelerates the calibration process.

PUBLIC ABSTRACT

Financial institutions, like banks and insurance companies, are responsible for carefully managing the huge amount of assets collected from different sources (e.g. investors and the government). Although different financial institutions have various investing strategies, they are all facing the risk coming from many unpredictable future events, like the financial crisis. The financial derivatives, serve as insurance for financial institutions, and provide the practical tools to hedge many main kinds of risks. Therefore, the financial derivatives should not be free (just like you should pay the health insurance before having it). To price the financial derivative is not easy, and thus the stochastic calculus as the ideal tool comes in. The double Heston model is one of those well–known practical pricing models which computes the price of many derivatives including European options. This thesis implements the double Heston model in an optimized way which reduces the pricing time by about 15%. The sensitivity analysis of the pricing model is also very important since it provides us the window to mathematically observe the risk. Another contribution of this thesis is: a novel application of the automatic algorithmic differentiation algorithm in order to achieve a more stable, more accurate, and faster sensitivity analysis for the double Heston model (compared to the classical finite difference methods). Lastly, our study presents a novel hybrid calibration method, which mixes the Differentiation Evolution algorithm and the Levenberg–Marquardt algorithm. Our new hybrid model significantly accelerates the calibration process.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION AND PRELIMINARIES

1.1 Introduction and the Structure of the Thesis

Finance allows counties to efficiently makes full use of assets to improve the average living quality of the whole society. Finance helps the businesses and individuals reach their goals by providing them the funding. Finance makes it possible for every family to plan ahead for a rainy day by providing them the insurance. Almost everyone in our society has the need to buy one or more types of services offered by financial institutions. All financial institutions, like banks and asset management companies, hire thousands of professional experts to manage trillions of dollars collecting from many different sources. However, the financial world is extremely complicated and full of risks. One challenging problem for financial institutions is how to safely manage the huge amount of money. The solution is the financial derivatives, which can be considered as the insurance for investing strategies.

Derivatives, stocks, and debt are the three main categories of financial instruments. The use of the word "derivative" in finance refers to a product whose value derives from the value of an underlying S. There are many possibilities for S. The underlying may be made up of a stock, or of multiple stocks, of other kinds of securities, of commodities, of the foreign exchange, or of anyone of a number of indices, the Dow, S&P 500, etc. Financial institutions usually create a portfolio by investing stock and debt instruments, and expect to gain profits from doing this. At the same time, in order to hedge many kinds of risks, they buy or short derivatives which can significantly reduce the risk, although the potential gain will also be cut down meanwhile.

In our discussion of pricing, we refer to derivative securities. Fast and accurate pricing models are extremely desired in the financial industry. On one hand, the market has the most basic demand for being provided the fair price of the derivative every second, otherwise, the hedging activities cannot be conducted. On the other hand, which is more important in practice, we can derive The Greeks from the pricing model. The Greeks are the sensitivity of the price of a derivative to changes in parameter value(s) while other parameters are fixed. In other words, they are partial derivatives of the price of a derivative with respect to the parameter values. The Greeks play a significant role in risk management, since the Greeks provide traders the window to observe the risk their investing strategies are taking. For example, the Greek delta represents the partial derivative of the value of the portfolio respect to the price of the underlying. A high value of the delta means the portfolio is too sensitive to the movement of the price of the underlying. Therefore, the portfolio is experiencing too much risks, and corresponding hedging activities should be played to weaken the sensitivity of the portfolio to the underlying. Financial institutions always typically set an upper boundary for each of the Greeks that their traders must not exceed.

The Black-Scholes pricing model is the cornerstone, and it applies to all of the financial derivatives. It is based on the idea going back to P. Samuelson, F. Black, M. Scholes, and R. Merton, published in 1973 [6]. It in turn makes essential use of Ito calculus (1944 and after [21]), a key part in the study of stochastic differential equations (SDEs).The most common derivatives are options, for example, European calls or puts, or the American counter parts. European means that the holder of the European option may exercise his particular option at the future but fixed time T, but not before. An American option, by contrast, may be exercised at any time before the strike T.

The Black-Scholes pricing model makes use of continuous hedging: If C is the value of a derivative product, a new hedged portfolio Π is created: Long in one unit of C and short in some units Δ of the underlying S. Now Δ, in turn, is determined in such a way that the value of Π agrees with a risk-free investment. Via Ito calculus, this reasoning leads to a partial differential equation (PDE) for C, the Black-Scholes equation. It is a parabolic PDE, a diffusion equation, and it can be solved as a PDE; or alternatively with the use of stochastic calculus, Ito-calculus. A powerful feature of the Black-Scholes pricing model (what is called risk-neutral valuation), is the fact that holder of options is able to eliminate the risk factors which are inherent in the volatility, and the unpredictable trends of the prices of the underlying. The Black-Scholes formula thus gives the value C of the derivative which is considered, and traded.

The application of Ito-calculus is simpler if the volatility term is a constant σ times the Brownian motion, which is the case for the Black-Scholes pricing model. However, this contradicts to the observed fact. In our thesis, we stress the case when σ is instead replaced with a random variable v. The corresponding system of stochastic differential equations (SDEs) is considerably more subtle, and it is referred to as the Heston model [17]. This model, in turn, is the starting point for our considerations below. Chapter 2 will introduce the Heston model in details. As the Heston model does not fit the implied volatility surface well sometimes, Chapter 3 will discuss the double Heston model, which is a simple extension of the original Heston model, and has a better fit to the implied volatility surface. The calibration for the double Heston model is complicated, and it will be discussed in Chapter 4. Chapter 5 will introduce the powerful Automatic Differentiation (AD), which allows us quickly estimate the Greeks, and the Jacobian matrix of the price with respect to parameters. The Jacobian matrix can be used to accelerate the calibration for the double Heston model. Chapter 1- 5 prepared all needed materials for developing our new model and algorithms. The numerical results will be presented in Chapter 6.

As we concentrate in the present thesis on European options, section 1.2 of Chapter one will introduce the concepts of European call and put options. As mentioned above, a key tool in our analysis is systems of stochastic differential equations and Ito calculus; as well as ideas from hedging, and from, risk management analysis.

So, sections 1.3 - 1.5 will briefly discuss the stochastic calculus and the risk neutral measure [4], which can be found everywhere in option pricing models. Section 1.6 will present the Black-Scholes model [6], the most influential stochastic derivative pricing model in the history of mathematical finance.

1.2 European Call and Put Options

Instead of directly giving the tedious definitions of the European call and put options, I would like to share you one of my favorite stories, a story that explains the core idea of call and put options.

Example 1.1. Tom is a corn farmer who plants and sells a large amount of corn. Therefore, Tom does care about the price of the corn. Now, the market price of the corn is 3.82\$/bsh. Tom observes that an increasing number of farmers plan to plant corn, which implies that there may be more supplies for corn than the demand in the near future. So, Tom worries about that corn will be much cheaper next year. Therefore, Tom finds a corn buyer, Mike, who might think that the price of corn will increase instead since he believes that the demand for the corn will expand faster than the supplies. Tom signs a contract with Mike, a contract states that Tom has the right but no obligation to sell 10,000 tons of corn to Mike at the price of 3.82\$/bsh next year. Then, Tom can sleep well now since he will be able to sell his corn to Mike at the current price of 3.82\$/bsh next year even if the price of the corn might be as low as 3.2\$/bsh.

At the same time, Tom wants to open a new land for planting wheat. Tom believes that more and more farmers will come back to the wheat market from the corn market one year later. Therefore, he thinks that the demand for the wheat seeds will explode one year later. He worries about that the price of the wheat seeds will be much more expensive at that time. Then, Tom finds a wheat seeds seller, Jason, who thinks that the price of the wheat seeds will decrease for some reason. Tom signs a contract with Jason, a contract states that Tom has the right but no obligation to buy 1000lb wheat seeds from Jason at the current price of 25$/lb next year. By doing this, Tom will not worry too much about the uncertain big jump in the price of the wheat seeds any more because he could still get the wheat seeds at the price of 25$/lb even if the price might go to 40$/lb.

In the story above, Tom signed two contracts. Both contracts can be considered as the insurances for Tom since two contracts eliminate the risks from the potential jump in the wheat seeds price and the possible slide in the price of the corn. Here is the new question: how much should both two contracts cost? Both contracts should not cost nothing since Tom does benefit from both contracts and there should not be any free lunch. The contract signed between Tom and Jason is a call option and the contract signed between Tom and Mike is a put option. Those two contracts are two examples of the financial derivatives, which are some assets derived from prices of other assets (e.g. the wheat seeds and the corn). Here are definitions for the call and put options [27].

Definition 1.1 (Call and Put Options)**.** A call (put) option is a financial contract between two parties, the buyer and the seller of this type of option. The buyer of the call (put) option has the right to buy (sell) an agreed quantity of a particular financial instrument from the seller (to the seller) of the option at a certain time, the expiration date, for a certain price, the strike price. The seller is obligated to sell (buy) the financial instrument to (from) the buyer if the buyer so decides. The buyer pays a fee for this right.

The option has been known for hundreds of years. Trading activities of options had been remarkably increased since the Chicago Board Option Exchange was established in 1973. Nowadays, most options are in standard form and are traded on regulated options exchanges nowadays, while other over-the-counter options are still used between a single buyer and seller. There are two main styles of options: American and European options. They are different in many aspects. One main difference is that owners/buyers of American options may exercise their rights at any time before the expiration date while owners/buyers of European options can only exercise their rights at the expiration.

In this present thesis, we assume that the underlying assets are stocks and all stocks do not pay any dividend. Let S_t represent the price of the stock at time t. We also assume that the risk free interest rate is a constant as r. Without loss of generality, we will only discuss $C(S_t, t, K)$, the price of the corresponding call option at the current time of t, with the strike price of K and the expiration of T. Because given the price of the call option, the price of the corresponding put option can always be derived by using the Put-call party, which is

$$call - put = S_t - Ke^{-rt}. \tag{1}$$

From the definition of the call option, the payoff at the expiration can be expressed as $C(S_T, T, K) = (S_T - K, 0)^+$. The only indeterminate term in this formula is the value of S_T, the price of the stock at the expiration of T. In fact, it is hard to forecast the value of S_T, since the stock market is so complicated as it is influenced by a lot of factors. To reasonably predict the value of S_t at the future time of T, our first task is to find the mathematical tool to characterize S_t. The goal of following sections of this chapter is to provide the powerful mathematical tool, the stochastic calculus.

1.3 Brownian Motion

The Brownian motion is a continuous stochastic process that first introduced to model the random motion of particles, suspended in a fluid and erratically moved. Brownian motion has many interesting and attractive properties. For example, Brownian motion is nowhere differentiable despite the fact that it is continuous everywhere. Another example is that once The Brownian motion hits any particular value, it will visit it again infinitely often. Nowadays, Brownian motion is widely used in finance to portray the fluctuations in an asset's price. It can be considered as a building block of all stochastic processes. In this present thesis, stock price and volatility of stocks will be modeled by Brownian motion based processes. In order to define a stochastic process like Brownian motion, we begin with defining what a random variable [29] is first.

Definition 1.2 (Random Variable)**.** Let $(\Omega, \mathcal{F}, \mu)$ be a probability space. A random variable is a real-valued function X defined on Ω with the property that for every Borel subset B of $\mathbb{R}$, the subset of Ω given by

$$\{X \in B\} = \{\omega \in \Omega : X(\omega) \in B\}$$

, is in the $\sigma - algebra\ \mathcal{F}$.

Then, the stochastic process [29] X is defined as:

Definition 1.3 (Stochastic Process)**.** A stochastic process X is a collection of random variables

$$(X_t, t \geq 0) = (X_t(\omega), t \geq 0, \omega \in \Omega),$$

defined on the space Ω.

By definition, a stochastic process X can be viewed as a function of two variables, ω and t [29].

For a fixed time t, the stochastic process X becomes a random variable:

$$X_t = X_t(\omega), \omega \in \Omega.$$

For a fixed random outcome or event ω, the stochastic process is the function of time:

$$X_t = X_t(\omega), t \geq 0.$$

As a special case of the Gaussian process, the Brownian motion [37] is defined as following.

Definition 1.4 (Brownian Motion)**.** Let $(\Omega, \mathcal{F}, \mu)$ be a probability space. For each $\omega \in \Omega$, suppose there is a continuous function $W(t)_{t\geq 0}$ which satisfies $W(0) = 0$. Then $W(t)_{t\geq 0}$, is a Brownian motion if for all $0 = t_0 < t_1 < \cdots < t_m$, the increments

$$W(t_1) = W(t_1) - W(t_0), W(t_2) - W(t_1), \cdots, W(t_m) - W(t_m - 1)$$

are independent and each of these increments is normally distributed with

$$E[W(t_{i+1}) - W(t_i)] = 0,$$

$$Var[W(t_{i+1}) - W(t_i)] = t_{i+1} - t_i.$$

Proposition 1.1. *The path of the Brownian motion has the following properties:*

1. for almost every $\omega \in \Omega$, the path $W(t)(\omega)$ is continuous.

2. for almost every $\omega \in \Omega$, the path $W(t)(\omega)$ is not differentiable.

Another proposition that makes the Brownian motion so special is that it has nonzero quadratic variation [37], which can be defined as following:

Definition 1.5 (Quadratic Variation)**.** Let $P = \{0 = t_0 < t_1 < \cdots < t_n = T\}$ be a patition of interval $[0, T]$. $|P| = max(t_i - t_{i-1})$, where $i = 1, 2, \cdots, n$. Let $f(t)$ be a function defined on $0 \leq t \leq T$. The quadratic variation of $f(t)$ up to T is

$$[f, f](T) = \lim_{|P|\to 0} \sum_{j=1}^{n-1} [f(t_{j+1}) - f(t_j)]^2.$$

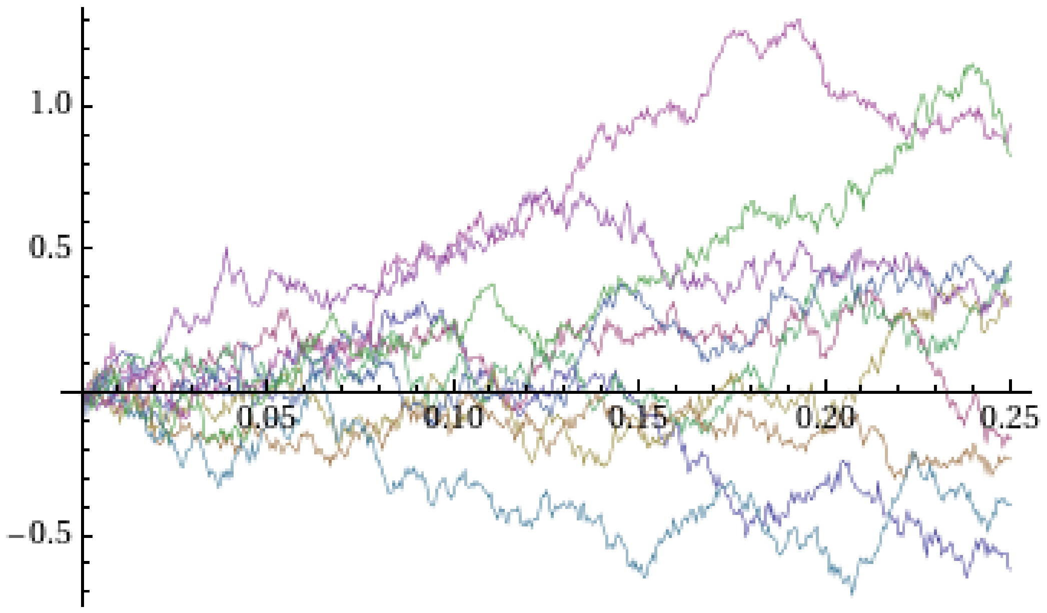


Figure 1.1: 10 Sample Paths of Brownian Motion. Every path corresponds to a different $\omega \in \Omega$

Theorem 1.1. *Let $W(t)$ be a Brownian motion. Then $[W(\omega), W(\omega)](T) = T$ for almost every $\omega \in \Omega$ and for all $T \geq 0$.*

Remark. If a function $f(t)$ has continuous derivative, then $[f, f](T) = 0$

Brownian motion is a useful tool used to describe the movements of the stock price S_t. We can represent S_t as a function of $W(t)$ and t. Therefore, we can characterize the differential dS_t as a function of $dW(t)$ and dt. Our next task is to run mathematical and statistical analysis on S_t in order to get a reasonable estimation for S_T, the value of S_t at time T. Such kind of analysis is based on the Ito's calculus, which is different from the classical calculus. In the next section, we will introduce Ito's way to define the integral and the famous Ito's formula.

1.4 Ito's Calculus with Respect to Brownian Motion

1.4.1 Ito Process

Ito's calculus, named by the famous mathematician, Kiyoshi Ito, who extended the methods of classical calculus to stochastic processes. Ito calculus, and Ito integrals, are tools from stochastic analysis which has proved extremely successful in solving stochastic differential equations based on Gaussian random variables, and Gaussian fields. It is especially powerful in quantifying volatility as it arises in the stochastic variables that serve as underlyings, say S, of derivative options. Using Brownian motion, observe that the square of the increments, $(dS)^2$ on the average are linear in the time-increment dt. Hence the differentials of Ito calculus for functions f(S) will contribute a term from the double-derivative of f. While this may seem heuristic, and perhaps counter-intuitive, K. Ito has turned it into a rigorous and very powerful tool. Since its inception in the 1950s, Ito-calculus has found many applications, of which pricing of derivative securities is the focus of the present thesis.

Ito's calculus can only be applied to Ito processes, which includes Brownian motion. It would be good to explain what Ito processes are first [29]. It is not easy to define the Ito processes in one sentence. Several key related definitions shall be presented first.

Definition 1.6 (Filtration)**.** Let Ω, the event space, be a nonempty set. Let T be a fixed positive number and assume that for each $t \in [0, T]$, there is a σ-algebra $\mathcal{F}(t)$. Assume further that if $s \leq t$, then every set in $\mathcal{F}(s)$ is also in $F(t)$. We call the collection of σ-algebra $\mathcal{F}(t)$, $t \in [0,T]$, a filtration.

Definition 1.7 (σ-Algebra Generated by X)**.** Let X be a random variable defined on a nonempty sample space Ω. The σ-algebra generated by X, denoted by $\sigma(X)$, is the collection of all subsets of Ω of the form $X \in B$, where B ranges over the Borel subsets of $\mathbb{R}$.

Definition 1.8 ($\mathcal{G}$-Measurable)**.** Let X be a random variable defined on a nonempty event space Ω. Let $\mathcal{G}$ be a σ-algebra of subsets of Ω. If every set in $\sigma(X)$ is also in $\mathcal{G}$, we say that X is $\mathcal{G}$-measurable.

Definition 1.9 (Adapted Stochastic Process)**.** Let Ω be a nonempty sample space equipped with a filtration $\mathcal{F}(t)$, $0 \leq t \leq T$. Let X(t) (a stochastic process) be a collection of random variables indexed by $t \in [0,T]$. We say this collection of random variables is an adapted stochastic process if, for each t, the random variable $X(t)$ is $\mathcal{F}(t)$-measurable.

Remark. If $X(t)$ is an adapted stochastic process under $\mathcal{F}(t)$, $t \in [0,T]$, then $\mathcal{F}(t)$ contains all the information about $X(s)$, $0 \leq s \leq t$. This means that for any Borel set B, you know whether $X(s) \in B$ happens or not if you have $\mathcal{F}(t)$, $s \leq t \leq T$.

Based on all four definitions above, we give the definition of Ito process [27].

Definition 1.10 (Ito Process)**.** Let $W(t)$ be a Brownian motion and $\mathcal{F}(t)$ be an associated filtration, given $t \geq 0$. An Ito process is a stochastic process of the form

$$X(t) = X(0) + \int_0^t H(u)dW(u) + \int_0^t M(u)du \tag{2}$$

where $X(0)$ is nonrandom and $H(u)$ and $M(u)$ are adapted stochastic processes.

Remark. Integral like $\int_0^t H(u)dW(u)$ belongs to Ito's integral, which will be defined in the following subsection.

Remark. If H and M are functions of $W(t)$ and t, then $H(t)$ and $M(t)$ are adapted under $\mathcal{F}(t)$.

Remark. To make the right-hand side of equation (2) well-defined, we should add additional assumptions, like $\int_0^t \mathbb{E}H^2(u)du$ and $\mathbb{E}\int_0^t |M(u)|du$ are finite for every $t > 0$. We will discuss such integrability assumptions in the next subsection.

1.4.2 Ito's Integral

To define the Ito's integral $\int_0^T H(u,\omega)dW(u)$, we make futher assumptions on the integrand $H(t,\omega)$.

Assumption 1.1. *Assumptions on the integrand process $H(t,\omega)$ are:*

1. *H is adapted to Brownian motion on $[0,T]$.*
2. *The integral $\int_0^T \mathbb{E}H^2(u,\omega)du$ is finite.*
3. *$\mathbb{E}(H^2(t,\omega)) < \infty$ for each $0 \leq t \leq T$.*
4. *let $\mathcal{B}$ denote the smallest σ-field that contains all of the open subsets of $[0,T]$. $H(t,\omega)$, as a map from $[0,T] \times \Omega$ to R, is jointly $\mathcal{B} \times \mathcal{F}$-measurable.*

Remark. All such functions defines the space $\mathcal{L}_T^2$.

Remark. In addition, we consider two functions in $\mathcal{L}_T^2$ as identical if they are equal for all (t,ω) except on a subset of $\mu_{\mathcal{B}} \times \mu$-measure zero. With the norm $||H||_{2,T} = \sqrt{\int_0^T \mathbb{E}(H^2)dt}$, $\mathcal{L}_T^2$ is a complete normed linear space [27].

Because the Ito's integral is defined as the limit of a sequence of random variables, we shall first clarify what the mode of convergence is. There are many modes of convergence for a sequence of random variables $Y_1, Y_2, \cdots$. The main modes are convergence in distribution, convergence in probability, almost sure convergence, and L^p-convergence. L^2-convergence is the one we will use to define the Ito's integral [32].

Definition 1.11 (L^2-Convergence)**.** The sequence of random variables $\{Y_n\}$, converges in L^2 (mean square) to Y ($Y_n \xrightarrow{L^2} Y$), if $\mathbb{E}[|Y_n|^2 + |Y|^2] < \infty$ for all n, and

$$\mathbb{E}|Y_n - Y|^2 \to 0, n \to \infty$$

The Ito's integral will be defined in two cases: the simple case and the general case.

Case 1: $H(t, \omega)$ is a simple process

Definition 1.12 (Simple Stochastic Process)**.** The stochastic process $H(t, \omega)$, $t \in [0, T]$ and $\omega \in \Omega$, is said to be simple if there exists a partition $P = \{0 = t_0 < t_1 < \cdots < t_n = T\}$ and a sequence $(Z_i, i = 1, \cdots, n)$ of random variables such that

$$H(t,\omega) = \begin{cases} Z_n, & \text{t = T} \\ \\ Z_i, & \text{if} \quad t_{i-1} \le t \le t_i, \ i = 1, \cdots, n. \end{cases}$$

Here, the sequence $\{Z_i\}$ is adapted to $\mathcal{F}_{i-1}, \ i = 1, \cdots, n$.

Definition 1.13 (Ito's Integral of a Simple Process)**.** The Ito's integral for a simple process $H(t, \omega)$ (as mentioned above) on $[0, T]$ is given by:

$$\int_0^T H(s, \omega) dW(s) = \sum_{i=1}^{n} H(t_{i-1}, \omega)(W(t_i) - W(t_{i-1})) = \sum_{i=1}^{n} Z_i(W(t_i) - W(t_{i-1}))$$

Remark. $\int_0^T H(s, \omega) dW(s)$ has mean of 0.

Proposition 1.2 (Isometry Property of a Simple Process)**.**

$$\mathbb{E}(\int_0^T H(s, \omega) dW(s))^2 = \int_0^T \mathbb{E} H^2(s, \omega) ds$$

$H(t, \omega)$ is the simple process as defined above.

Proof. The proof is very straight forward and it uses the properties of Brownian motion.

$$\mathbb{E}(\int_0^T H(s, \omega) dW(s))^2 = \mathbb{E}[\sum_{i=1}^{n} Z_i(W(t_i) - W(t_{i-1}))]^2$$
$$= \mathbb{E} \sum_{i=j, i=1}^{n} Z_i^2(W(t_i) - W(t_{i-1}))^2 + 2\mathbb{E} \sum_{i<j, i=1}^{n} Z_i Z_j (W(t_i) - W(t_{i-1}))(W(t_j) - W(t_{j-1}))$$
$$= \mathbb{E}\mathbb{E}[\sum_{i=j, i=1}^{n} Z_i^2(W(t_i) - W(t_{i-1}))^2 | \mathcal{F}_{i-1}]$$
$$+ 2\mathbb{E}\mathbb{E}[\sum_{i<j, i=1}^{n} Z_i Z_j (W(t_i) - W(t_{i-1}))(W(t_j) - W(t_{j-1})) | \mathcal{F}_{j-1}]$$
$$= \mathbb{E} \sum_{i=j, i=1}^{n} Z_i^2(t_i - t_{i-1}) + 0$$
$$= \int_0^T \mathbb{E} H^2(s, \omega) ds$$

□

Remark. By the assumptions for $H(t, \omega)$, $\int_0^T H(s, \omega) dW(s)$ is well defined.

Remark. The Isometry Property still holds for the general case.

Case 2: $H(t,\omega)$ is not a simple process

To define Ito's integral for the general case, we will follow 3 steps:

Step 1: $H(t,\omega)$ can be approximated by simple processes

$$H^n(t,\omega) = H(0) + \sum_{i=0}^{n-1} H(t_i^n,\omega)\mathcal{I}_(t_i^n, t_{i+1}^n] \tag{3}$$

where $P^n = \{t_i^n\}$ is a partition of $[0,T]$ with $|P^n| = max_i(t_{i+1}^n - t_i^n) \to 0$ as $n \to \infty$, such that

$$\lim_{n \longrightarrow \infty} \mathbb{E} \int_0^T (H^n(s,\omega) - H^n(s,\omega))^2 ds = 0$$

Step 2: $J_n = \int_0^T H^n(s,\omega)dW(s)$ forms a Cauchy sequence in the space L^2 of random variables with zero mean and finite second moments.

Step 3: Because L^2 space is complete, the limit of J_n in the sense of L^2-convergence exists. It is easy to show that the limit is unique.

Definition 1.14 (Ito's integral of Adapted Processes)**.** Assume that $H(t,\omega)$ satisfies assumption 1.1, then the Ito's integral is defined as

$$\int_0^T H(s,\omega)dW(s) = \lim_{n \longrightarrow \infty} \int_0^T H^n(s,\omega)dW(s)$$

where $H^n(t,\omega)$ is defined as equation (3).

To prove that the definition(1.14) makes sense, we just need to prove that all statements in steps 1 - 3 are true. The second and the third one are trivial as long as we prove the first one [33].

Lemma 1.1. *For each function $g(t,\omega)$ that satisfies the assumption 1.1, we can find a sequence of simple functions $g^n(t,\omega)$, $n = 1, 2, \cdots$, such that*

$$\lim_{n\longrightarrow\infty} \mathbb{E}\int_0^T (g(s,\omega) - g^n(s,\omega))^2 ds = 0$$

Proof. Let P^n be the partition of [0, T] of the form $0 = t_1^n < t_2^n < \cdots < t_{n+1}^n = T$ with $|P^n| = max(t_{j+1}^n - t_j^n) \to 0$ as $n \to \infty$, $j = 1, 2, \cdots, n$.

(a). If $\mathbb{E}(g^2(t,\omega))$ is continous in t, we define a sequence of simple processes

$$g^n(t,\omega) = g(t_j^n, \omega),\ t_j^n \le t \le t_{j+1}^n,$$

for $j = 1, 2, \cdots, n$ and $n = 1, 2, 3, \cdots$. Then $g^n(t,\omega) \in \mathcal{L}_T^2$ and

$$\mathbb{E}(|g^n(t,\omega) - g(t,\omega)|^2) \to 0, \text{when } n \to \infty,$$

for each $t \in [0, T]$. Therefore, by the Lebesgue Dominated Convergence Theorem we have

$$\lim_{n\longrightarrow\infty} \mathbb{E}\int_0^T (g(s,\omega) - g^n(s,\omega))^2 ds = 0$$

(b). If $\mathbb{E}(g(t,\omega)^2)$ is not continuous in t, for each N, we define the bounded function

$$G_N(t,\omega) = \begin{cases} g(t,\omega), & |g(t,\omega)| \le \text{N} \\ 0, & \text{otherwise} \end{cases}$$

Moreover

$$\int_0^T \mathbb{E}(|G_N(t,\omega) - g(t,\omega)|^2) dt \le 4\int_0^T \mathbb{E}(|g(t,\omega)|^2) dt,$$

so by Dominated Convergence Theorem, we have

$$\lim_{n\longrightarrow\infty} \mathbb{E}\int_0^T (g(s,\omega) - G_N(s,\omega))^2 ds = 0.$$

For such $G_N(t,\omega)$, we define a function f_k for $k > 0$ as

$$f_k(t,\omega) = ke^{-kt}\int_0^t e^{ks}G_N(s,\omega)ds.$$

It is not hard to see that $|f_k(t,\omega)| \leq N \cdot (1 - e^{-kt})$. Thus, $\mathbb{E}((f_k(t,\omega))^2)$ is finite and integrable over $[0,T]$. So, f_k belong to $\mathcal{L}_T^2$. Moreover, f_k satisfy

$$|f_k(t,\omega) - f_k(s,\omega)| \leq 2kN|t-s|,$$

which implies that $f_k(t,\omega)$ is continuous and $\mathbb{E}((f_k(t,\omega))^2)$ is continuous. From part (a), we can approximate f_k by simple functions. □

Lemma 1.2. *The J_n defined in step 2 forms a Cauchy sequence in space L^2.*

Proof. By the Isometry Property of the simple processes, we have that all J_n are in L^2 space and

$$\begin{aligned}\mathbb{E}(J_n - J_m)^2 &= \mathbb{E}(\int_0^T (H^n(s,\omega) - H^m(s,\omega))dW(s))^2 \\ &= \mathbb{E}\int_0^T (H^n(s,\omega) - H^m(s,\omega))^2 ds \\ &\leq 2\mathbb{E}\int_0^T (H^n(s,\omega) - H(s,\omega))^2 ds + 2\mathbb{E}\int_0^T (H^m(s,\omega) - H(s,\omega))^2 ds \to 0,\end{aligned}$$

as $n, m \to \infty$.

□

In fact, the limit $\int_0^T H(s,\omega)dW(s)$ does not depend on the particular choice of the approximating sequence of simple process.

Example 1.2. Find $I = \int_0^T W(t)dW(t)$ by following the definition of Ito's integral.

Let $W_j = W(\frac{jT}{n})$ and by definition

$$I = \lim_{n \to \infty} \sum_{j=0}^{n-1} W_j \cdot (W_{j+1} - W_j).$$

Because

$$\begin{aligned} \frac{1}{2}\sum_{j=0}^{n-1}(W_{j+1} - W_j)^2 &= \frac{1}{2}\sum_{j=0}^{n-1} W_{j+1}^2 - \sum_{j=0}^{n-1} W_j W_{j+1} + \frac{1}{2}\sum_{j=0}^{n-1} W_j^2 \\ &= \sum_{j=0}^{n-1} W_j \cdot (W_{j+1} - W_j) + \frac{1}{2}W_n^2, \end{aligned}$$

therefore by theorem 1.1,

$$I = \lim_{n \to \infty} [\frac{1}{2}W_n^2 - \frac{1}{2}\sum_{j=0}^{n-1}(W_{j+1} - W_j)^2] = \frac{1}{2}W^2(T) - \frac{1}{2}T.$$

1.4.3 Ito's Formula

In example 1.2, we computed $\int_0^T W(t)dW(t)$ by strictly following the way we defined the Ito's integral. In fact, the amount of work can be reduced significantly by applying Ito's formula. Ito's formula, as the most fundamental and powerful tool in Ito's calculus, offers us a really convenient method to compute the integral.

Theorem 1.2 (Ito-Doeblin formula for Brownian motion)**.** *Let $W(t)$ be a Brownian motion and $f(t, x)$ be a function such that $f_t(t, x)$, $f_x(t, x)$ and $f_{xx}(t, x)$ are defined and continuous. Then for every $T \geq 0$,*

$$f(T, W(T)) = f(0, W(0)) + \int_0^T f_t(t, W(t))dt + \int_0^T f_x(t, W(t))dW(t) + \frac{1}{2}\int_0^T f_{xx}(t, W(t))dt$$

or

$$df(t, W(t)) = [f_t(t, W(t)) + \frac{1}{2}f_{xx}(t, W(t))]dt + f_x(t, W(t))dW(t)$$

Lemma 1.3. *The Quadratic Variation of the Ito process X as defined in the definition (1.10) is*

$$[X, X](t) = \int_0^t H^2(u)du$$

Theorem 1.3 (Ito-Doeblin formula for an Ito process)**.** *Let $f(t, x)$ be a function such that $f_t(t, x)$, $f_x(t, x)$ and $f_{xx}(t, x)$ are defined and continuous. Let $X(t)$ be the Ito process as described in the definition 1.10. Then for every $T \geq 0$,*

$$\begin{aligned} f(T, X(T)) &= f(0, X(0)) + \int_0^T f_t(t, X(t))dt + \int_0^T f_x(t, X(t))dX(t) + \frac{1}{2}\int_0^T f_{xx}(t, W(t))d[X, X](t) \\ &= f(0, X(0)) + \int_0^T f_t(t, X(t))dt + \int_0^T f_x(t, X(t))dW(t) \\ &+ \int_0^T f_x(t, X(t))M(t)dt + \frac{1}{2}\int_0^T f_{xx}(t, X(t))H^2(t)dt \end{aligned}$$

Example 1.3. Find $I = \int_0^T W(t)dW(t)$ by using the Ito's formular.

We just simply apply the Ito's formula to the function $f(t, W(t)) = \frac{1}{2}W^2(t) - \frac{1}{2}t$

1.5 Risk Neutral Measure

A risk neutral measure (also called equivalent martingale measure), is a probability measure under which the price of assets is equal to the discounted expectation of the price. The market is arbitrage-free if and only if there exists at least one risk neutral measure. The risk neutral measure pricing technique has been widely used in financial engineering to compute the values of derivatives product . In this present thesis, we also derive the price of the European call options by calculating the expection of the call price at the expiration under the risk neutral measure.

I would like to repeat that our goal is to predict $C(S_T, T, K) = (S_T - K, 0)^+$, where S_T is the price of the stock at expiration of T. In the statistical sense, to predict $(S_T - K, 0)^+$, usually means to find the expectation of $(S_T - K, 0)^+$, since S_T is a random variable.

However, investors have various opinions about where the price of the stock would end at the expiration of T. Some of them might think that the stock will go up by 10% while others might think that the stock will drop by 5%. In other words, everyone believes in his or her own way of measuring the risk. Because of that, people have different expectations for $(S_T - K, 0)^+$, which can be described as

$$\mathbb{E}^*(S_T - K, 0)^+ \tag{4}$$

where $\mathbb{E}^*$ represents the expectation under one particular measure μ^* on the σ-algebra $\mathcal{F}$ of the event space Ω.

There could be many different probability measure μ^*s under which we will get different expectations. Which measure should we choose? Which measure does fit the common sense best? What is the ideal measure supposed to be? To answer these questions, we need to define the continuous-time martingale first [37].

Definition 1.15 (Martingale)**.** The stochastic process $Y(t, \omega)$ ($t \geq 0$ and $\omega \in \Omega$), which is adapted to the filtration $\mathcal{F}_t$, is called a continuous-time martingale if $\mathbb{E}|Y_t| <$

∞ for all $t > 0$ and

$$\mathbb{E}(Y_t|\mathcal{F}_s) = Y_s, \text{for all } 0 \leq s < t$$

Remark. If $H(t, \omega)$ satisfies the assumption (1.1), $\int_0^T H(s, \omega)dW(s)$ is a martingale.

Remark. What a martingale truely means is that the future change is expected to be 0, having all information at the current time.

Assuming that the risk free interest rate is r, every riskless asset is supposed to have the same increasing rate as r, otherwise arbitrage opportunities will appear. A 10 dollars worth acesst now ($t = 0$) should be worthy of $10 \cdot e^{rT}$ dollars at time of T. If an asset is supposed to be worthy of 10 dollars at some futrue time of T, it should worth $10 \cdot e^{-rT}$ dollars now ($t = 0$). In fact, we are not able to really tell the value of the risk free rate r, although the interest rate on a three-month U.S. Treasury bill can be used as an estimation for the risk free rate. Theoretically, we always assume that there exists such a risk free rate.

Definition 1.16 (Risk-Neutral Measure)**.** The risk-neutral measure is the measure μ^{RN}, under which the stochastic process $Y_t(t, \omega) = e^{-r(T-t)} \cdot S_T$ is a martingale, for all $0 \leq t \leq T$, where S_t represents the value of the asset at time t.

Remark. The future change of the discounted value of an asset is supposed to be 0.

Remark. The related risk-neutral measure μ^{RN} may not exist for some process $\{S_t\}_{0 \leq t \leq T}$.

You may also wonder what is the relation between risk-neutral measure μ^{RN} and any another measure μ^*. When it comes to changing of measures, the following theorems are very useful. I will list them here for future use [27].

Theorem 1.4. *Let* $(\Omega, \mathcal{F}, \mu)$ *be a probability space and let* X *be an almost surely nonnegative random variable with* $\mathbb{E}^{\mu}X = 1$. *For* $A \in \mathcal{F}$, *define*

$$\mu^*(A) = \int_A X(\omega) d\mu(\omega). \tag{5}$$

Then μ^* *is a probability measure, If* Y *is a nonnegative random variable, then*

$$\mathbb{E}^{\mu^*}Y = \mathbb{E}^{\mu}[YX]$$

Theorem 1.5 (Girsanov). *Let* $\{W(t)\}_{0\leq t\leq T}$ *be a Brownian motion on a probability space* $(\Omega, \mathcal{F}, \mu)$ *and let* $\{\mathcal{F}_t\}_{0\leq t\leq T}$ *be a filtration for this Brownian motion.* $\{\theta(t)\}_{0\leq t\leq T}$ *is an adapted process. Define*

$$X(t) = e^{\{-\int_0^t \theta(u)dW(u) - \frac{1}{2}\int_0^t \theta^2(u)du\}}, \tag{6}$$

$$W^*(t) = W(t) + \int_0^t \theta(u)du, \tag{7}$$

and assume that

$$\mathbb{E}^{\mu} \int_0^T \theta^2(u)X^2(u)du < \infty. \tag{8}$$

let $X = X(T)$. *Then* $\mathbb{E}^{\mu}X = 1$ *and under the probability measure* μ^* *given by equation (5), the process* $\{W^*(t)\}_{0\leq t\leq T}$ *is a Brownian motion.*

Remark. The equation (8) is called the Novikov's condition. It is a sufficient but unnecessary condtion.

1.6 Black Scholes Model and Its Drawbacks

We have almost introduced all the statistical tools we need to develop a model to price European call options. This section will introduce the most well-known stochastic option pricing model, the Black-Scholes model[6]. Although the Heston model is a model derived to overcome the shortcoming of the Black-Scholes model, the pricing strategy in the Heston model is very similar to the that in the Black-Scholes. Reviewing the Black-Scholes model would be very helpful for readers to understand following chapters.

Assumption 1.2. *Assumptions for the Black-Scholes model are:*

- *The risk free interest rate r is constant.*
- *There is no arbitrage opportunity.*
- *No cost for any transactions.*
- *It is possible to buy, sell and short any amount of the stock.*
- *It is possible to borrow and lend any amount of cash.*
- *The stock does not pay dividend.*
- *The process of the stock price $\{S_t\}_{0\leq t\leq T}$ is defined as*

$$dS_t = \mu S_t dt + \sigma S_t dW(t), \tag{9}$$

where μ (the drift rate of S_t) and σ (the standard deviation of S_t) are all constant. $\{W(t)\}_{0\leq t\leq T}$ is a Brownian motion. S_0 is constant.

Theorem 1.6. *If assumption (1.2) is satisfied, the price the call option will be:*

$$C(t, S_t) = S_t \cdot N(d_1(\tau, S_t)) - e^{-r\tau} K N(d_2(\tau, S_t)) \tag{10}$$

where

$$d_1(\tau, S_t) = \frac{1}{\sigma\sqrt{\tau}}[ln\frac{S_t}{K} + (r + \frac{1}{2}\sigma^2)\tau] \tag{11}$$

and

$$d_2(\tau, S_t) = \frac{1}{\sigma\sqrt{\tau}}[ln\frac{S_t}{K} + (r - \frac{1}{2}\sigma^2)\tau] \tag{12}$$

Proof. We will prove the Black-Scholes formula (10) in three steps:

Step 1: $S_t = S_0 e^{\{(\mu-\frac{1}{2}\sigma^2)t+\sigma W(t)\}}$. By Ito's formula,

$$d(ln(S_t)) = \frac{1}{S_t}dS_t - \frac{1}{2}\frac{1}{S_t^2}(dS_t)^2 = (\mu - \frac{1}{2}\sigma^2)dt + \sigma dW(t),$$

then

$$ln(S_t) = ln(S_0) + (\mu - \frac{1}{2}\sigma^2)t + \sigma W(t).$$

Step 2: The risk-neutral measure exists under the assumption (1.2) and

$$dW(t) = dW^{RN}(t) - \frac{\mu - r}{\sigma}dt.$$

Under the risk-neutral measure, the process of discounted price of stock $\{e^{-rt}S_t\}_{0\leq t\leq T}$ is a martingale. By Ito's formula

$$d(e^{-rt}S_t) = e^{-rt}S_t[(\mu - r)dt + \sigma dW(t)]. \tag{13}$$

To be a martingale, the coefficient of dt must be zero and thus we define

$$X(t) = e^{\{-\frac{\mu-r}{\sigma}W(t)\}}$$

and

$$W^{RN}(t) = W(t) + \frac{\mu - r}{\sigma}t.$$

By theorem (1.5), we find the unique risk-neutral measure and the Brownian motion $\{W^{RN}(t)\}_{0\leq t\leq T}$ under the risk-neutral measure.

Step 3: Based on the results above, we will prove the Black-Scholes formula (10).

We define $Y = -\frac{W^{RN}(T)-W^{RN}(t)}{\sqrt{\tau}} \sim N(0,1)$, where $\tau = T-t$. Then

$$\begin{aligned} S(T) &= S(t)e^{\sigma(W(T)-W(t))+(\mu-\frac{1}{2}\sigma^2)\tau} \\ &= S(t)e^{\sigma[W^{RN}(T)-W^{RN}(t)]+(r-\frac{1}{2}\sigma^2)\tau} \\ &= S(t)e^{-\sigma\sqrt{\tau}Y+(r-\frac{1}{2}\sigma^2)\tau}. \end{aligned}$$

Therefore

$$\begin{aligned} C(t,S_t) &= e^{-r\tau}\mathbb{E}^{RN}(S_T-K,0)^+ \\ &= \frac{1}{\sqrt{2\pi}}e^{-r\tau}\int_{-\infty}^{\infty}[S_te^{-\sigma\sqrt{\tau}y+(r-\frac{1}{2}\sigma^2)\tau}-K]^+\cdot e^{-\frac{1}{2}y^2}dy \\ &= \frac{1}{\sqrt{2\pi}}e^{-r\tau}\int_{-\infty}^{d_2(\tau,S_t)}[S_te^{-\sigma\sqrt{\tau}y+(r-\frac{1}{2}\sigma^2)\tau}-K]\cdot e^{-\frac{1}{2}y^2}dy \\ &= \frac{1}{\sqrt{2\pi}}e^{-r\tau}\int_{-\infty}^{d_2(\tau,S_t)}S_te^{-\sigma\sqrt{\tau}y+(r-\frac{1}{2}\sigma^2)\tau}\cdot e^{-\frac{1}{2}y^2}dy-\frac{1}{\sqrt{2\pi}}e^{-r\tau}\int_{-\infty}^{d_2(\tau,S_t)}K\cdot e^{-\frac{1}{2}y^2}dy \\ &= \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{d_2(\tau,S_t)}S_te^{-\frac{1}{2}y^2-\sigma\sqrt{\tau}y-\frac{1}{2}\sigma^2\tau}dy-\frac{1}{\sqrt{2\pi}}e^{-r\tau}\int_{-\infty}^{d_2(\tau,S_t)}K\cdot e^{-\frac{1}{2}y^2}dy \\ &= \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{d_2(\tau,S_t)}S_te^{-\frac{1}{2}(y+\sigma\sqrt{\tau})^2}dy-\frac{1}{\sqrt{2\pi}}e^{-r\tau}\int_{-\infty}^{d_2(\tau,S_t)}K\cdot e^{-\frac{1}{2}y^2}dy \\ &= \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{d_1(\tau,S_t)}S_te^{-\frac{1}{2}z^2}dz-\frac{1}{\sqrt{2\pi}}e^{-r\tau}\int_{-\infty}^{d_2(\tau,S_t)}K\cdot e^{-\frac{1}{2}y^2}dy \\ &= S_tN(d_1(\tau,S_t))-e^{-r\tau}KN(d_2(\tau,S_t)) \end{aligned}$$

□

From the proof of the theorem (1.6), you can see the beauty of the application of the stochastic calculus in finance. Because of their creative work on the Black-Scholes model, Merton and Scholes received the Nobel Memorial Prize in Economic Sciences in 1977. Since then, their work was widely used for pricing options in the real financial market. However, the Black-Sholes model is not perfect. Traders in wall street did not realize the drawbacks of Black-Sholes formula while they were making billions of dollars by using it. However, they did get a serious lesson when the Dow Jones Index dropped by 508 points to 1738.74 on a disastrous Monday in 1987. One of the main reasons for the Black Monday is that the assumption of Black-Sholes model that the volatility of the stock is constant is not proper. In fact, the change in the volatility of the stock plays a more significant role on determining the price of options sometimes. After the Black Monday, many great researches got started immediately to improve the Black-Scholes model. The Heston model is one of those models which perfect the Black-Scholes model remarkably. In the next chapter, we will discuss it.

CHAPTER 2 THE HESTON MODEL

After the crash of the stock market in October 1987, traders in the Wall Street wished to fix the Black-scholes model as quick as possible. However, they did not know where to get started with. It was hard to find the reason why the Black-scholes model destroyed the market, since it had been successfully applied for a decade, without showing any problem. Finally, they found the problem when they saw the following picture.

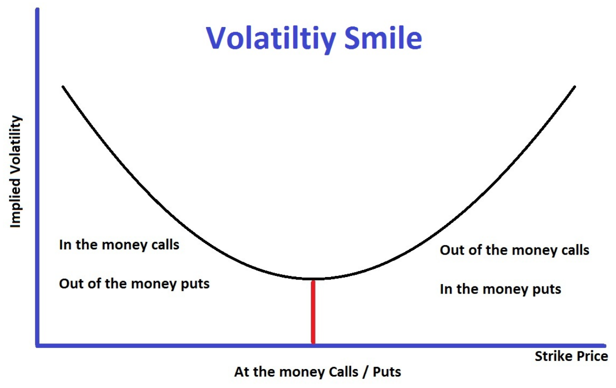


Figure 2.1: Volatility Smile

Let me explain the graph above in detail. Let $C^M(S_t, t, K)$ be the price of the call option determined by the buying and selling activities of traders in the real world. Its value is not the theoretical price derived from the statistical models (e.g. Black-scholes model).

Under the framework of Black-scholes, the theoretical price of European call option is an increasing function of the volatility of the underlying asset. Also, we assume that all parameters other than σ, the volatility of the stock, are fixed. Then, given the value of $C^M(S_t, t, K)$, we can always find an estimate $\hat{\sigma}$ for σ, a value that makes the price produced by the Black-scholes model to be equal to $C^M(S_t, t, K)$. Such value of σ is called the implied volatility. To find the implied volatility, numerical methods like the bisection method can be easily implemented.

We get the figure 2.1 when the implied volatility is plotted against the strike price. The curve was supposed to be flat since volatility was assumed to be constant in Black-sholes model. However, as all you can see, when traders lose (out of the money) or earn (in the money) a lot, the value of the implied volatility becomes larger.

As more and more researchers realized that the assumption of constant volatility should be modified, many stochastic volatility models got developed. These models include Hull and White (1987), Wiggins (1987), Scott(1987 and 1989), and Stein and Stein (1991)[20][41][36][38]. In these stochastic volatility models, the volatility is described by a stochastic process, instead of being a constant. The Heston model [17] was one of the most important models, if not the most important one.

In the next Section 2.1, we will present the derivation of the price of the European call option under the Heston model. Then section **??** will talk about the problems in the Heston model and the work has been done to improve the Heston model.

2.1 The Original Heston Model

Under some probability measure μ^p, which is often called the physical measure, the Heston model can be described as following:

$$dS_t = \mu S_t dt + \sqrt{v_t} S_t dW_{t1} \tag{1}$$

$$dv_t = k(\theta - v_t)dt + \sigma\sqrt{v_t} dW_{t2} \tag{2}$$

$$\mathbb{E}^p[dW_{t1} dW_{t2}] = \rho dt \tag{3}$$

μ the drift of the process for the stock;

$\theta > 0$ the mean reversion level for the variance;

$k > 0$ the mean reversion rate for variance;

$\sigma > 0$ the volatility of the varicance;

$\rho \in [-1, 1]$ the correlation between the two Brownian motions W_{t1} and W_{t2}.

λ the price of the risk.

As you can see, in the Heston model, the variance of the stock is not a constant any more. What is more, the stochastic variance v_t of the stock follows a CIR process[12], which is presented by equation (2).

You might wonder where the parameter λ comes from. The following subsection will talk about the risk-neutral measure for the Heston model and then you will get the answer.

2.2 Risk-Neutral Measure for the Heston Model

Now, let us discuss what the processes in the Heston model will look like under the risk neutral probability measure μ^{RN}. First of all, as mentioned, the discounted price of the stock has to be a martingale under μ^{RN}. Once again, because

$$d(S_t e^{-rt})] = e^{-rt} dS_t - re^{-rt} S_t dt = e^{-rt}[(\mu - r)S_t dt + \sqrt{v_t} S_t dW_{t1}]$$

therefore

$$dW_{t1}^{RN} = dW_{t1} + \frac{\mu - r}{\sqrt{v_t}} dt. \tag{4}$$

Then, under the risk netural measure μ^{RN}, the equation (1) becomes

$$dS_t = rS_t dt + \sqrt{v_t} S_t dW_{t1}^{RN} \tag{5}$$

What does the equation (2) look like under the risk neutral probability measure? It gets more complicated when the variance of the stock follows a stochastic process instead of being a constant. Uncertainty about the variance of the stock is a new source of risk. Thus, a risk premium offsets the effect from the risk caused by the uncertainty about v_t. Let $C(S_t, v_t, t)$ (or C for short) be the price of the option, by Ito's formula, we have

$$\begin{aligned}
&\mathbb{E}^p[\frac{dC}{C}] - E^{RN}[\frac{dC}{C}] \\
&= \frac{1}{C}[C_t + C_s\mathbb{E}^p(dS_t) + C_v\mathbb{E}^p(dv_t)) + \text{second order terms}] \\
&- \frac{1}{C}[C_t + C_s\mathbb{E}^{RN}(dS_t) + C_v\mathbb{E}^{RN}(dv_t)) + \text{second order terms}] \\
&= \frac{1}{C}[C_s(\mathbb{E}^p(dS_t) - \mathbb{E}^{RN}(dS_t)) + C_v(\mathbb{E}^p(dv_t) - \mathbb{E}^{RN}(dv_t))] \\
&= \frac{1}{C}[C_s(\mu - r)S_t + C_v(\lambda(S_t, v_t, t))]dt,
\end{aligned}$$

where $\lambda(S_t, v_t, t)$ is called the risk premium and

$$\lambda(S_t, v_t, t) = \mathbb{E}^p(dv_t) - \mathbb{E}^{RN}(dv_t).$$

Therefore

$$dW_{t2}^{RN} = dW_{t2} + \frac{\lambda(S_t, v_t, t)}{\sigma\sqrt{v_t}}dt. \tag{6}$$

By the way, as explained in Heston(1993)[17], Breeden's (1979)[7] consumption model yields a risk premium proportional to the variance such that

$$\lambda(S_t, v_t, t) = \lambda v_t,$$

where λ is a constant. Then, under the risk neutral measure, the equation (2) turns out to be

$$dv_t = k(\theta - v_t - \lambda v_t)dt + \sigma\sqrt{v_t}dW_{t2}^{RN} = k^*(\theta^* - v_t)dt + \sigma\sqrt{v_t}dW_{t2}^{RN}, \tag{7}$$

where $k^* = k + \lambda$ and $\theta^* = \frac{k\theta}{k+\lambda}$.

Moreover, it is not hard to prove that

$$\mathbb{E}^{RN}[dW_{t1}^{RN} dW_{t2}^{RN}] = \rho dt \tag{8}$$

To simplify the notation, we will drop the subscript letter t, then we get

$$dS = rSdt + \sqrt{v} S dW_1^{RN}, \tag{9}$$

$$dv = k^*(\theta^* - v)dt + \sigma\sqrt{v} dW_2^{RN}, \tag{10}$$

$$dW_1^{RN} = dW_1 + \frac{\mu - r}{\sqrt{v}} dt, \tag{11}$$

$$dW_2^{RN} = dW_2 + \frac{\lambda\sqrt{v}}{\sigma} dt, \tag{12}$$

$$\mathbb{E}^{RN}[dW_1^{RN} dW_2^{RN}] = \rho dt, \tag{13}$$

where $k^* = k + \lambda$ and $\theta^* = \frac{k\theta}{k+\lambda}$.

2.3 Pricing under the Heston Model

Let $C(S, v, t)$ represent the price of the call option. Then, the value of $C(S, v, t)$ should be equal to the discounted value of the expectation of the final payoff under the risk neutral measure.

$$\begin{aligned} C(S, v, t) &= e^{-r(T-t)} E^{RN}[(S_T - K)^+] \\ &= e^{-r(T-t)} E^{RN}[(S_T - K)\mathbf{1}_{S_T > K}] \\ &= e^{-r(T-t)} E^{RN}[S_T \mathbf{1}_{S_T > K}] - K e^{-r(T-t)} E^{RN}[\mathbf{1}_{S_T > K}] \end{aligned} \tag{14}$$

$$E^{RN}[\mathbf{1}_{S_T > K}] = \mathbf{P}^{RN}(S_T > K) = \mathbf{P}^{RN}(lnS_T > lnK) = P_2 \tag{15}$$

We introduce another measure μ^Q, such that $\frac{d\mu^{RN}}{d\mu^Q} = \frac{e^{r(T-t)}}{S_T/S_t}$, then

$$\begin{aligned} e^{-r(T-t)}E^{RN}[S_T\mathbf{1}_{S_T>K}] &= S_tE^{RN}[\frac{S_T/S_t}{e^{r(T-t)}}\mathbf{1}_{S_T>K}] \\ &= S_tE^Q[\frac{S_T/S_t}{e^{r(T-t)}}\mathbf{1}_{S_T>K}\frac{d\mu^{RN}}{d\mu^Q}] \\ &= S_tE^Q[\mathbf{1}_{S_T>K}] = S_t\mathbf{P}^Q(S_T > K) \\ &= S_t\mathbf{P}^Q(lnS_T > lnK) = S_tP_1 \end{aligned} \tag{16}$$

$$C(S, v, t) = S_tP_1 - Ke^{-r(T-t)}P_2 \tag{17}$$

$P_1 = \mathbf{P}^Q(lnS_T > lnK)$ is the probability that the call expires in-the-money under the measure Q. In fact, it is not hard to prove than Q is a measure.

$P_2 = \mathbf{P}^{RN}(lnS_T > lnK)$ is the probability that the call expires in-the-money under the risk neutral measure μ^{RN}.

Our next goal is to solve P_1 and P_2. In order to do this, we will first derive the PDE's for P_1 and P_2. To construct the PDEs, we need to construct a hedging portfolio.

We form a portfolio $\mathbf{M}$, which includes one option $V = V(S, v, t)$, δ_1 units of the stock, and δ_2 units of another option $U(S, v, t)$. Then, we have

$$d\mathbf{M} = dV + \delta_1dS + \delta_2dU. \tag{18}$$

By Ito's formula, we get

$$dV = [\frac{\partial V}{\partial t} + \frac{1}{2}vS^2\frac{\partial^2 V}{\partial S^2} + \rho\sigma vS\frac{\partial^2 V}{\partial S\partial v} + \frac{1}{2}\sigma^2 v\frac{\partial^2 V}{\partial v^2}]dt + \frac{\partial V}{\partial S}dS + \frac{\partial V}{\partial v}dv \tag{19}$$

$$\begin{aligned} d\mathbf{M} = & [\frac{\partial V}{\partial t} + \frac{1}{2}vS^2\frac{\partial^2 V}{\partial S^2} + \rho\sigma vS\frac{\partial^2 V}{\partial S\partial v} + \frac{1}{2}\sigma^2 v\frac{\partial^2 V}{\partial v^2}]dt \\ & + \delta_2[\frac{\partial U}{\partial t} + \frac{1}{2}vS^2\frac{\partial^2 U}{\partial S^2} + \rho\sigma vS\frac{\partial^2 U}{\partial S\partial v} + \frac{1}{2}\sigma^2 v\frac{\partial^2 U}{\partial v^2}]dt \\ & + [\frac{\partial V}{\partial S} + \delta_1 + \delta_2\frac{\partial U}{\partial S}]dS + [\frac{\partial V}{\partial v} + \delta_2\frac{\partial U}{\partial v}]dv \end{aligned} \tag{20}$$

Choose

$$\delta_2 = -\frac{\partial V}{\partial v}/\frac{\partial U}{\partial v},$$

and

$$\delta_1 = -\delta_2\frac{\partial U}{\partial S} - \frac{\partial V}{\partial S}.$$

Then we have

$$\begin{aligned} d\mathbf{M} = & [\frac{\partial V}{\partial t} + \frac{1}{2}vS^2\frac{\partial^2 V}{\partial S^2} + \rho\sigma vS\frac{\partial^2 V}{\partial S\partial v} + \frac{1}{2}\sigma^2 v\frac{\partial^2 V}{\partial v^2}]dt \\ & + \delta_2[\frac{\partial U}{\partial t} + \frac{1}{2}vS^2\frac{\partial^2 U}{\partial S^2} + \rho\sigma vS\frac{\partial^2 U}{\partial S\partial v} + \frac{1}{2}\sigma^2 v\frac{\partial^2 U}{\partial v^2}]dt \end{aligned} \tag{21}$$

As a hedged portfolio, $d\mathbf{M} = r\mathbf{M}dt$, where r is the risk-free rate. Thus we have

$$d\mathbf{M} = r(V + \delta_1 S + \delta_2 U)dt. \tag{22}$$

Combine the two equations above, we have

$$\begin{aligned} & \frac{\frac{\partial V}{\partial t} + \frac{1}{2}vS^2\frac{\partial^2 V}{\partial S^2} + \rho\sigma vS\frac{\partial^2 V}{\partial S\partial v} + \frac{1}{2}\sigma^2 v\frac{\partial^2 V}{\partial v^2} + rS\frac{\partial V}{\partial S} - rV}{\frac{\partial V}{\partial v}} \\ = & \frac{\frac{\partial U}{\partial t} + \frac{1}{2}vS^2\frac{\partial^2 U}{\partial S^2} + \rho\sigma vS\frac{\partial^2 U}{\partial S\partial v} + \frac{1}{2}\sigma^2 v\frac{\partial^2 U}{\partial v^2} + rS\frac{\partial U}{\partial S} - rU}{\frac{\partial U}{\partial v}} \end{aligned}$$

The only difference between the left-hand side and right-hand side of equation (2.3) is the function V and U. This implies

$$\frac{\frac{\partial V}{\partial t} + \frac{1}{2}vS^2\frac{\partial^2 V}{\partial S^2} + \rho\sigma vS\frac{\partial^2 V}{\partial S\partial v} + \frac{1}{2}\sigma^2 v\frac{\partial^2 V}{\partial v^2} + rS\frac{\partial V}{\partial S} - rV}{\frac{\partial V}{\partial v}} = f(S, v, t), \tag{23}$$

where $f(S, v, t)$ is a function of S, v and t. Heson [17] set this function as

$$f(S, v, t) = -k(\theta - v) + \lambda(S, v, t). \tag{24}$$

To explain this setup, we need to construct another portfolio. We consder a portfolio H, which longs one share of call option V(S, v, t) and shorts Δ shares of S. Then

$$H = V - \Delta S,$$

and

$$\begin{aligned} dH - rHdt &= dV - \Delta dS - rVdt + r\Delta Sdt \\ &= [\frac{\partial V}{\partial t} + \frac{1}{2}vS^2\frac{\partial^2 V}{\partial S^2} + \rho\sigma vS\frac{\partial^2 V}{\partial S\partial v} + \frac{1}{2}\sigma^2 v\frac{\partial^2 V}{\partial v^2} + rS\frac{\partial V}{\partial S}]dt \\ &+ \frac{\partial V}{\partial S}dS + \frac{\partial V}{\partial v}dv - \Delta dS - rVdt + r\Delta Sdt. \end{aligned} \tag{25}$$

To make this H to be a portfolio of delta-neutral , set $\Delta = \frac{\partial V}{\partial S}$, then

$$\begin{aligned} dH - rHdt &= dV - \Delta dS - rVdt + r\Delta Sdt \\ &= [\frac{\partial V}{\partial t} + \frac{1}{2}vS^2\frac{\partial^2 V}{\partial S^2} + \rho\sigma vS\frac{\partial^2 V}{\partial S\partial v} + \frac{1}{2}\sigma^2 v\frac{\partial^2 V}{\partial v^2} - rV]dt + \frac{\partial V}{\partial v}dv \\ &= \frac{\partial V}{\partial v}[f(S, v, t)dt + dv]. \end{aligned} \tag{26}$$

Therefore, under the risk neutral measure, we have

$$\begin{aligned} dH - rHdt &= \frac{\partial V}{\partial v}[f(S, v, t)dt + dv] \\ &= \frac{\partial V}{\partial v}[f(S, v, t)dt + (k(\theta - v) - \lambda(S, v, t))dt + \sigma\sqrt{v_t}dW_2^{RN}] \end{aligned} \tag{27}$$

$E^{RN}(dH - rHdt) = 0$ implies the equation (24).

Using the same setup, Black and Scholes(1973) and Merton(1973) demonstrate that the value of any asset $C(S, v, t)$, including the European Call option, must satisfy the following PDE

$$\begin{aligned} &\frac{1}{2}vS^2\frac{\partial^2 C}{\partial S^2} + \rho\sigma vS\frac{\partial^2 C}{\partial S\partial v} + \frac{1}{2}\sigma^2 v\frac{\partial^2 C}{\partial v^2} + rS\frac{\partial C}{\partial S} \\ &+ [k(\theta - v) - \lambda(S, v, t)]\frac{\partial C}{\partial v} - rC + \frac{\partial C}{\partial t} = 0. \end{aligned} \tag{28}$$

An European option with the strike price K and the expiration time T, satisfies the PDE above, subject to the following boundary conditions (Heston 1993):

$$C(S, v, T) = (S_T - K, 0)^+ \tag{29}$$

$$C(0, v, t) = 0 \tag{30}$$

$$\frac{\partial C}{\partial S}(\infty, v, t) = 1 \tag{31}$$

$$C(S, \infty, t) = S \tag{32}$$

$$rs\frac{\partial C}{\partial S}(S, 0, t) - k\theta\frac{\partial C}{\partial v}(S, 0, t) - rC(S, 0, t) + C_t(S, 0, t) = 0 \tag{33}$$

Set $x = ln(S)$, we get

$$\begin{aligned} &\frac{\partial C}{\partial t} + \frac{1}{2}v\frac{\partial^2 C}{\partial x^2} + (r - \frac{1}{2}v)\frac{\partial C}{\partial x} + \rho\sigma v\frac{\partial^2 C}{\partial v\partial x} \\ &+ \frac{1}{2}\sigma^2 v\frac{\partial^2 C}{\partial v^2} - rC + [\kappa(\theta - v) - \lambda v]\frac{\partial C}{\partial v} = 0. \end{aligned} \tag{34}$$

Replace C by equation (17), we have

$$\begin{aligned}&\frac{1}{2}v\frac{\partial^2 P_j}{\partial x^2}+\rho\sigma v\frac{\partial^2 P_j}{\partial x\partial v}+\frac{1}{2}\sigma^2 v\frac{\partial^2 P_j}{\partial v^2}+(r+u_j v)\frac{\partial P_j}{\partial x}\\&+(a-b_j v)\frac{\partial P_j}{\partial v}+\frac{\partial P_j}{\partial t}=0\end{aligned}\tag{35}$$

for $j = 1, 2$, where

$$u_1=\tfrac{1}{2},\ u_2=-\tfrac{1}{2},\ a=k\theta,\ b_1=k+\lambda-\rho\sigma,\ b_2=k+\lambda$$

Our next goal is to solve P_1 and P_2. Instead of solving P_1 and P_2 directly, we consider the characteristic function [34].

Definition 2.1. The characteristic function provides an alternative way for describing a random variable x. It is defined as the Fourier transform of the density function $g(x)$ of the random variable x:

$$f(\phi)=E(e^{ix\phi})=\int_{-\infty}^{\infty}e^{ix\phi}g(x)dx.\tag{36}$$

The following theorem shows us one way to find P_1 and P_2, if their characteristic functions are given respectively.

Theorem 2.1. (Inversion Theorem Gil-Pelaez 1951)

$$Pr(x>k)=\frac{1}{2}+\frac{1}{\pi}\int_0^{\infty}Re(\frac{e^{-iuk}f(u)}{iu})du$$

where $f(u)$ is the characteristic function of the random variable x.

Remark. $P_j = Pr^j(lnS_T > ln\mathrm{k}) = \frac{1}{2}+\frac{1}{\pi}\int_0^{\infty}Re(\frac{e^{-i\phi ln\mathrm{k}}f_j(\phi,x,v,T)}{i\phi})d\phi$

where $j = 1, 2$, and $f_j(\phi, x, v, T)$ is the characteristic function of $X_T = lnS_T$.

Kendall and Stuart(1977) [24] established that the integral above converges.

Note that $f_j(\phi) = E(e^{i\phi lnS_T}) = E(e^{i\phi lnS_T}|\mathcal{F}_t)$. The Feynman-kac Theorem gives us the PDEs that f_j satisfies. In fact, f_j will follow the PDE in Equation (35). For future use, we list the multi-dimensional Feynman-Kac Theorem, not just the bivariate version, as following[34].

Theorem 2.2. (Feynman-Kac Theorem)

Suppose that the vector $\tilde{x}_t = \begin{pmatrix} x_1(t) \\ \vdots \\ x_n(t) \end{pmatrix}$ *follows the n-dimensional stochastic process*

$$d\tilde{x}_t = \begin{pmatrix} \mu_1(\tilde{x}_t, t) \\ \vdots \\ \mu_n(\tilde{x}_t, t) \end{pmatrix} dt + \begin{pmatrix} \sigma_{11}(\tilde{x}_t, t) & \cdots & \sigma_{1m}(\tilde{x}_t, t) \\ \vdots & \ddots & \vdots \\ \sigma_{n1}(\tilde{x}_t, t) & \cdots & \sigma_{nm}(\tilde{x}_t, t) \end{pmatrix} \begin{pmatrix} dw_1^Q(t) \\ \vdots \\ dw_m^Q(t) \end{pmatrix}. \tag{37}$$

$\tilde{W}_t^Q = \begin{pmatrix} w_1^Q(t) \\ \vdots \\ w_m^Q(t) \end{pmatrix}$ *is a vector of m-dimensional independent Q-Brownian motion.*

$\tilde{\sigma}(\tilde{x}_t, t) = \begin{pmatrix} \sigma_{11}(\tilde{x}_t, t) & \cdots & \sigma_{1m}(\tilde{x}_t, t) \\ \vdots & \ddots & \vdots \\ \sigma_{n1}(\tilde{x}_t, t) & \cdots & \sigma_{nm}(\tilde{x}_t, t) \end{pmatrix}$ *is the volatility matrix of size n by m.*

$\tilde{\mu}(\tilde{x}_t, t) = \begin{pmatrix} \mu_1(\tilde{x}_t, t) \\ \vdots \\ \mu_n(\tilde{x}_t, t) \end{pmatrix}$ *is n-dimensional.*

Define the Heston generator Λ *as*

$$\Lambda = \sum_{i=1}^{n} \mu_i \frac{\partial}{\partial x_i} + \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} (\tilde{\sigma}\tilde{\sigma}^T)_{ij} \frac{\partial^2}{\partial x_i \partial x_j}. \tag{38}$$

The Feynman-kac theorem states that the partial differential equation of a function $u(\tilde{x}_t, t)$ *given by*

$$\frac{\partial u}{\partial t} - r(\tilde{x}_t, t)u + \Lambda u = 0 \tag{39}$$

and with the terminal condition $u(\tilde{x}_T, T) = f(\tilde{x}_T, T)$, has the solution

$$u(\tilde{x}_t, t) = E^Q[e^{-\int_t^T r(\tilde{x}_s, s)ds)} f(\tilde{x}_T, T)|F_t] \tag{40}$$

Hence, choose $\tau = T - t$, then PDE for the characteristic function is

$$\begin{aligned} &\frac{1}{2}v\frac{\partial^2 f_j}{\partial x^2} + \rho\sigma v\frac{\partial^2 f_j}{\partial x\partial v} + \frac{1}{2}\sigma^2 v\frac{\partial^2 f_j}{\partial v^2} + (r + u_j v)\frac{\partial f_j}{\partial x} \\ &+ (a - b_j v)\frac{\partial f_j}{\partial v} - \frac{\partial f_j}{\partial \tau} = 0, \end{aligned} \tag{41}$$

subject to

$$f_j(\phi, x, v, T) = e^{i\phi lnS_T} \tag{42}$$

2.4 Solving the Characteristic Function

In this section, we will derive the closed form solution to the characteristic function. We assume that the characteristic function has the following form

$$f_j(\phi, x, v, T)) = \exp(C_j(\tau, \phi) + D_j(\tau, \phi)v + i\phi x). \tag{43}$$

Substitute f_j in Equation(41) by Equation(43) and drop the f_j terms, we get

$$v(-\frac{\partial D_j}{\partial \tau} + \rho\sigma i\phi D_j - \frac{1}{2}\phi^2 + \frac{1}{2}\sigma^2 D_j^2 + u_j i\phi - b_j D_j) - \frac{\partial C_j}{\partial \tau} + ri\phi + aD_j = 0. \tag{44}$$

Then, it is equivalent to solve the following two ODEs

$$\frac{\partial D_j}{\partial \tau} = \rho\sigma i\phi D_j - \frac{1}{2}\phi^2 + \frac{1}{2}\sigma^2 D_j^2 + u_j i\phi - b_j D_j, \tag{45}$$

$$\frac{\partial C_j}{\partial \tau} = ri\phi + aD_j, \tag{46}$$

subject to

$$D_j(0, \phi) = 0, \tag{47}$$

$$C_j(0, \phi) = 0. \tag{48}$$

Equation (45) has the type of Riccati Equation [34]. The solution for D_j is

$$D_j(\tau, \phi) = \frac{b_j - \rho\sigma i\phi + d_j}{\sigma^2}\Big(\frac{1 - e^{d_j\tau}}{1 - g_j e^{d_j\tau}}\Big), \tag{49}$$

where

$$d_j = \sqrt{(\rho\sigma i\phi - b_j)^2 - \sigma^2(2u_j i\phi - \phi^2)} \tag{50}$$

$$g_j = \frac{b_j - \rho\sigma i\phi + d_j}{b_j - \rho\sigma i\phi - d_j} \tag{51}$$

$$u_1 = \tfrac{1}{2},\ u_2 = -\tfrac{1}{2},\ a = k\theta,\ b_1 = k + \lambda - \rho\sigma,\ b_2 = k + \lambda.$$

We integrate both sides of Equation (46), we have

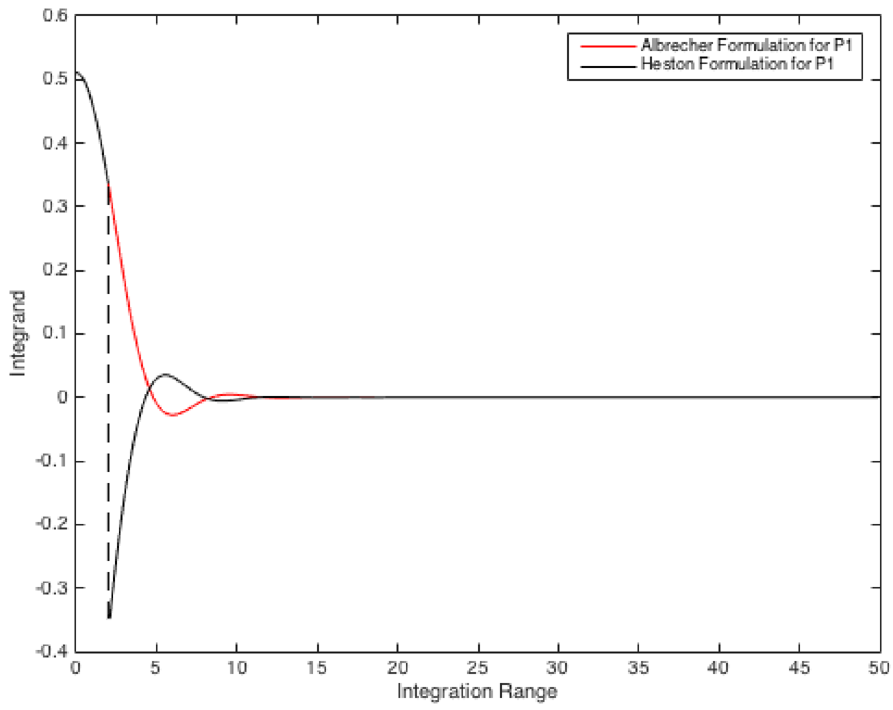
$$C_j(\tau, \phi) = ri\phi\tau + \frac{a}{\sigma^2}\Big[(b_j - \rho\sigma i\phi + d_j)\tau - 2\ln\Big(\frac{1 - g_j e^{d_j\tau}}{1 - g_j}\Big)\Big] \tag{52}$$

To ensure that v_t will be always strictly positive, parameters have to satisfy the Feller's condition [15].

Theorem 2.3 (Feller's condition)**.** *If the parameters obey the following condition, then the process v_t is strictly positive:*

$$2k\theta > \sigma^2. \tag{53}$$

Albrecher [1] suggests a slightly different form for D_j. However, this change leads to a much better behaved integrand, which creats less error for numerical integration. The reason is the branching problem in the complex analysis. The better performence can be seen in the following picture.



S = 30, K = 20, T = 5, r = 0.001, $\kappa = 1.4$,

$\theta = 0.05$, $\sigma = 0.5$, $v_0 = 0.05$, $\rho = -0.8$.

Figure 2.2: Albrecher vs Heston

Albrecher's new formular is equivalent to the original one. What he did was to multiply $exp(-d_j\tau)$ in the top and bottom of D_j. In Albrecher's formulation,

$$D_j(\tau,\phi) = \frac{b_j - \rho\sigma i\phi + d_j}{\sigma^2}\Big(\frac{1-e^{-d_j\tau}}{1-m_je^{-d_j\tau}}\Big), \tag{54}$$

where

$$m_j = \frac{1}{g_j}. \tag{55}$$

As a result,

$$C_j = ri\phi\tau + \frac{k\theta}{\sigma^2}\Big[(b_j - \rho\sigma i\phi - d_j)\tau - 2\ln\Big(\frac{1-m_je^{-d_j\tau}}{1-m_j}\Big)\Big]. \tag{56}$$

Albrecher proved that the Heston's formulation is unstable under some conditions and the new version is stable under the full parameter space. However, both versions cause no problem if the Feller's condition is exactly satisfied. since the modification by Albrecher always works, we will keep using his formulation.

In summary, the closed-form solution of the Heston model is:

$$C(S,v,t) = S_tP_1 - Ke^{-r(T-t)}P_2$$

$$P_j = \frac{1}{2} + \frac{1}{\pi}\int_0^\infty Re(\frac{e^{-i\phi lnk}f_j(\phi,x,v,T)}{i\phi})d\phi$$

$$f_j(\phi;x_t,v_t) = \exp(C_j(\tau,\phi) + D_j(\tau,\phi)v_t + i\phi x_t)$$

$$D_j(\tau,\phi) = \frac{b_j - \rho\sigma i\phi + d_j}{\sigma^2}\Big(\frac{1-e^{-d_j\tau}}{1-m_je^{-d_j\tau}}\Big),$$

$$C_j = ri\phi\tau + \frac{k\theta}{\sigma^2}\Big[(b_j - \rho\sigma i\phi - d_j)\tau - 2\ln\Big(\frac{1-m_je^{-d_j\tau}}{1-m_j}\Big)\Big].$$

where

$$d_j = \sqrt{(\rho\sigma i\phi - b_j)^2 - \sigma^2(2u_j i\phi - \phi^2)}$$

$$g_j = \frac{b_j - \rho\sigma i\phi + d_j}{b_j - \rho\sigma i\phi - d_j}$$

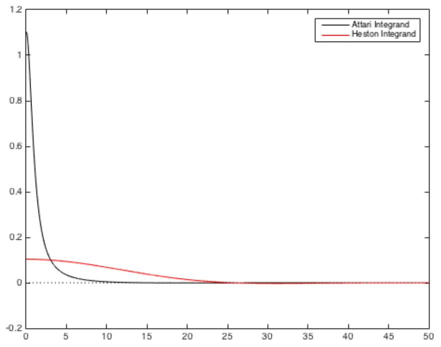
$$m_j = \frac{1}{g_j}$$

$$u_1 = \tfrac{1}{2},\ u_2 = -\tfrac{1}{2},\ a = k\theta,\ b_1 = k + \lambda - \rho\sigma,\ b_2 = k + \lambda$$

2.5 Numerical Integration

As long as we have the integrand, the only problem left in pricing is to numerically approximate the integral. It is not hard to observe that the integrand dampens to zero as ϕ increases. Although we are supposed to integrate the integrand from 0 to ∞, the dampening property allows us to just look at a finite range $[0, M]$, where M is a properly selected large number.

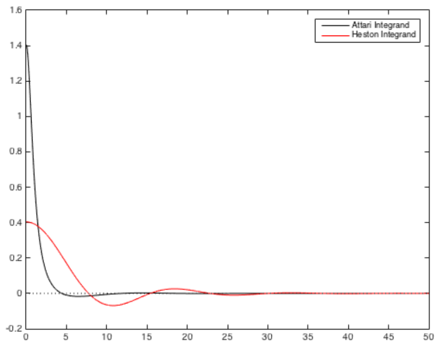
We will apply Gauss-Lobatto quadrature for computing the integral. It can be modified for integrals over any finite integration range $[a, b]$. The good point about the Guass-Lobatto quadrature is that the endpoints of the interval are included. Therefore, we must select a small number for the left endpoint a, instead of 0.



S = 20, K = 18, T = 0.1, r = 0.01, $\kappa = 1.4$,

$\theta = 0.05$, $\sigma = 0.3$, $v_0 = 0.05$, $\rho = -0.8$

Figure 2.3: Attari vs Heston Example 1



S = 30, K = 20, T = 1/12, r = 0.01, $\kappa = 1.4$,

$\theta = 0.05$, $\sigma = 0.25$, $v_0 = 0.05$, $\rho = -0.8$

Figure 2.4: Attari vs Heston Example 2

For some parameter sets, the integrand does not decay as fast as we expect.

Therefore, we have to choose some large upper bound for the numerical integration, which will increase the computation. Attari [2] gave us an alternate formula for the call price under the Heston model. The integrand under Attari's formulation decays faster. However, as you can see in Figure (2.3) and Figure (2.4), Attari's integrand is steeper around 0, which is also an issure in numerical integration. Attari's formular for the call option is:

$$C(K) = S - \frac{1}{2}Ke^{-r\tau} - \frac{Ke^{-r\tau}}{\pi}\int_0^\infty A(u)du$$

where

$$A(u) = \frac{(R_2(u) + \frac{I_2(u)}{u})cos(uh) + (I_2(u) - \frac{R_2(u)}{u})sin(uh)}{1+u^2},$$

$$\varphi(u) = f_2(u) \cdot exp(-iu(lnS + r\tau)) = R_2(u) + iI_2(u),$$

$$h = \frac{Ke^{-r\tau}}{S}.$$

2.6 The Drawbacks of the Heston Model

It is already a big improvement that Heston model does not have a constant volatility. This is also one of the reasons why the Heston model becomes one of the most well-known stochastic volatility models. However, at short maturity, the Heston implied volatility still does not fit the market Implied volatility too well, although evidence shows that the Heston model fits the market implied volatility very well under longer maturities. In the following chapter, we will introduce the double Heston model to overcome the problem when we have short maturity [34].

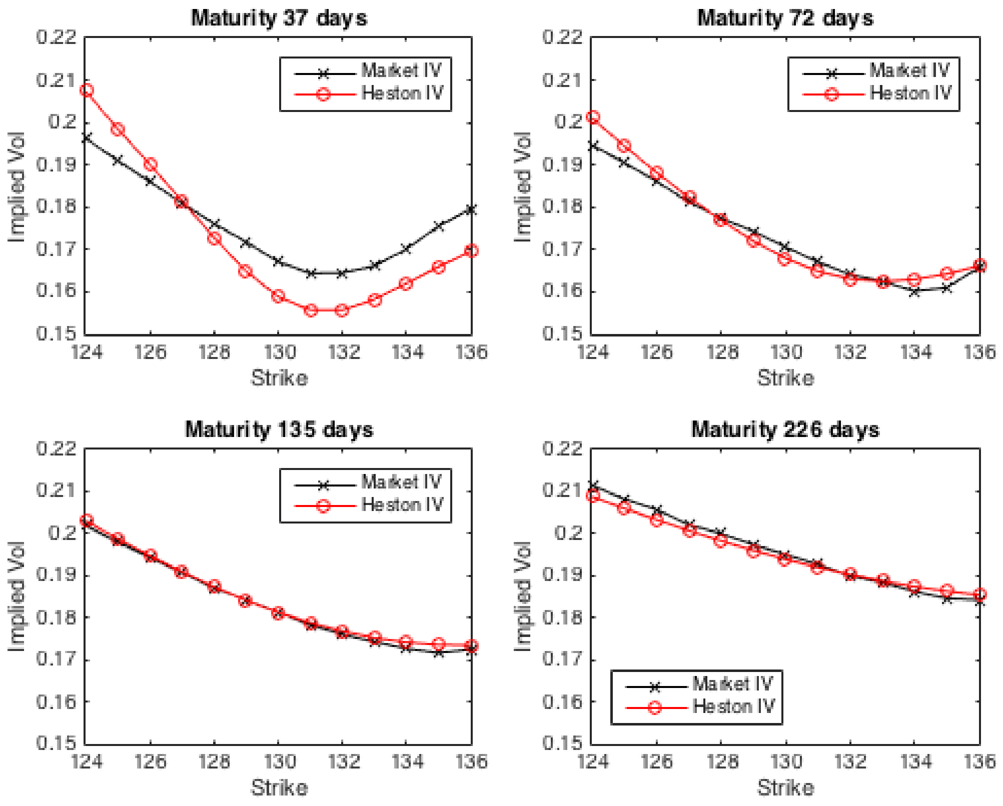


Figure 2.5: Implied Volatility of Heston Model

CHAPTER 3
THE DOUBLE HESTON MODEL

3.1 The Derivation of the Double Heston Model

The Heston model was created to overcome the Black-Sholes model's shortcoming of having constant volatility. However, the Heston model fails to fit the implied volatility smile very well sometimes. Such failure happens often when we have a short maturity. Christoffersen [11] introduced the double Heston model, which is just a simple extension of the original Heston model. Suppressing the t subscript for notational simplicity, the double Heston model can be summarized as following [34]:

$$dS = rSdt + \sqrt{v_1}SdW_1 + \sqrt{v_2}SdW_2, \tag{1}$$

$$dv_1 = \kappa_1(\theta_1 - v_1)dt + \sigma_1\sqrt{v_1}dW_3, \tag{2}$$

$$dv_2 = \kappa_2(\theta_2 - v_2)dt + \sigma_2\sqrt{v_2}dW_4, \tag{3}$$

and

$$E[dW_1dW_3] = \rho_1 dt, \tag{4}$$

$$E[dW_2dW_4] = \rho_2 dt, \tag{5}$$

$$E[dW_1dW_2] = E[dW_3dW_4] = E[dW_1dW_4] = E[dW_2dW_3] = 0. \tag{6}$$

We attempt to use the multi-dimensional Feynman-Kac Theorem to derive the PDE for the double Heston model. In order to get the operator for the double Heston model, we need to rewrite the SDEs. Set $x = lnS$, then we have

$$dx = (r - \frac{1}{2}(v_1 + v_2))dt + \sqrt{v_1}dW_1 + \sqrt{v_2}dW_2. \tag{7}$$

To construct a multi-dimensional independent $Q - Brownian$ motion , we set

$$W_1 = B_1 \tag{8}$$

$$W_2 = B_2 \tag{9}$$

$$W_3 = \rho_1 B_1 + \sqrt{1-\rho_1^2}B_3 \tag{10}$$

$$W_4 = \rho_2 B_2 + \sqrt{1-\rho_2^2}B_4 \tag{11}$$

Set $\tilde{x}_t = (x, v_1, v_2)$, then we can get the volatility matrix

$$\tilde{\sigma}(\tilde{x}_t, t) = \begin{pmatrix} \sqrt{v_1} & \sqrt{v_2} & 0 & 0 \\ \sigma_1\sqrt{v_1}\rho_1 & 0 & \sigma_1\sqrt{v_1(1-\rho_1^2)} & 0 \\ 0 & \sigma_2\sqrt{v_2}\rho_2 & 0 & \sigma_2\sqrt{v_2(1-\rho_2^2)} \end{pmatrix}, \tag{12}$$

and the drift

$$\mu = \begin{pmatrix} r - \frac{1}{2}(v_1+v_2) \\ \kappa_1(\theta_1 - v_1) \\ \kappa_2(\theta_2 - v_2) \end{pmatrix}. \tag{13}$$

As a result, the generator for the double Heston model is

$$\begin{aligned} \Lambda = [r - \frac{1}{2}(v_1+v_2)]\frac{\partial}{\partial x} + \kappa_1(\theta_1 - v_1)\frac{\partial}{\partial v_1} + \kappa_2(\theta_2 - v_2)\frac{\partial}{\partial v_2} \\ + \frac{1}{2}(v_1+v_2)\frac{\partial^2}{\partial x^2} + \rho_1\sigma_1 v_1\frac{\partial^2}{\partial x \partial v_1} + \rho_2\sigma_2 v_2\frac{\partial^2}{\partial x \partial v_2} \\ + \frac{1}{2}\sigma_1^2 v_1\frac{\partial^2}{\partial v_1^2} + \frac{1}{2}\sigma_2^2 v_2\frac{\partial^2}{\partial v_2^2} \end{aligned} \tag{14}$$

Again, Duffie [14] claimed that the characteristic function for $(x_T, v_{1,T}, v_{2,T})$ has the following form

$$\begin{aligned}
&f(\phi, \phi_1, \phi_2, x_t, v_{1,t}, v_{2,t}) \\
&= E[exp(i\phi x_T + i\phi_1 v_{1,T} + i\phi_2 v_{2,T})|\mathcal{F}_t] \\
&= exp[A(\tau, \phi, \phi_1, \phi_2) + B_0(\tau, \phi, \phi_1, \phi_2)x_t \\
&+ B_1(\tau, \phi, \phi_1, \phi_2)v_{1,t} + B_2(\tau, \phi, \phi_1, \phi_2)v_{2,t}].
\end{aligned} \tag{15}$$

Therefore, we have

$$\frac{\partial A}{\partial t} = -\mathbf{M}_0^T \boldsymbol{\beta} - \frac{1}{2}\boldsymbol{\beta}^T \mathbf{H_0} \boldsymbol{\beta}, \tag{16}$$

$$\frac{\partial B_0}{\partial t} = -\mathbf{M}_1^T \boldsymbol{\beta} - \frac{1}{2}\boldsymbol{\beta}^T \mathbf{H_1} \boldsymbol{\beta}, \tag{17}$$

$$\frac{\partial B_1}{\partial t} = -\mathbf{M}_2^T \boldsymbol{\beta} - \frac{1}{2}\boldsymbol{\beta}^T \mathbf{H_2} \boldsymbol{\beta}, \tag{18}$$

$$\frac{\partial B_2}{\partial t} = -\mathbf{M}_3^T \boldsymbol{\beta} - \frac{1}{2}\boldsymbol{\beta}^T \mathbf{H_3} \boldsymbol{\beta}, \tag{19}$$

where

$$\mathbf{M}_0 = \begin{pmatrix} r \\ \kappa_1\theta_1 \\ \kappa_2\theta_2 \end{pmatrix}, \mathbf{M}_1 = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \mathbf{M}_2 = \begin{pmatrix} -\frac{1}{2} \\ -\kappa_1 \\ 0 \end{pmatrix}, \mathbf{M}_3 = \begin{pmatrix} -\frac{1}{2} \\ 0 \\ -\kappa_2 \end{pmatrix},$$

$$\boldsymbol{\beta}^T = (B_0, B_1, B_2)$$

$$\mathbf{H}_0 = \mathbf{H}_1 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \mathbf{H}_2 = \begin{pmatrix} 1 & \sigma_1\rho_1 & 0 \\ \sigma_1\rho_1 & \sigma_1^2 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \mathbf{H}_3 = \begin{pmatrix} 1 & 0 & \sigma_2\rho_2 \\ 0 & 0 & 0 \\ \sigma_2\rho_2 & 0 & \sigma_2^2 \end{pmatrix}.$$

The boundary conditions are:

$$B_0(0,\phi,\phi_1,\phi_2) = i\phi_0, B_1(0,\phi,\phi_1,\phi_2) = i\phi_1,$$

$$B_2(0,\phi,\phi_1,\phi_2) = i\phi_2, A(0,\phi,\phi_1,\phi_2) = 0.$$

To get the characteristic function of the x_T, We set $\phi_1 = \phi_2 = 0$. Replace ϕ_0 by ϕ, we immediately get $B_0(\tau,\phi,\phi_1,\phi_2) = i\phi$. Then, we get

$$\frac{\partial A}{\partial \tau} = ri\phi + \kappa_1\theta_1 B_1 + \kappa_2\theta_2 B_2, \tag{20}$$

$$\frac{\partial B_1}{\partial \tau} = \frac{1}{2}\sigma_1^2 B_1^2 - (\kappa_1 - i\phi\rho_1\sigma_1)B_1 + \frac{1}{2}\phi(\phi+i), \tag{21}$$

$$\frac{\partial B_2}{\partial \tau} = \frac{1}{2}\sigma_2^2 B_2^2 - (\kappa_2 - i\phi\rho_2\sigma_2)B_2 + \frac{1}{2}\phi(\phi+i). \tag{22}$$

Solve Equation (76) - (78) respect to the boundary conditions $B_1(0) = 0$, $B_2(0) = 0$ and $A(0) = 0$, we have

$$B_j(\tau,\phi) = \frac{\kappa_j - \rho_j\sigma_j\phi i + d_j}{\sigma_j^2}[\frac{1-e^{d_j\tau}}{1-g_j e^{d_j\tau}}], \tag{23}$$

$$A(\tau,\phi) = r\phi i\tau + \sum_{j=1}^{2}\frac{\kappa_j\theta_j}{\sigma_j^2}[(\kappa_j - \rho_j\sigma_j\phi i + d_j)\tau - 2ln(\frac{1-g_j e^{d_j\tau}}{1-g_j})], \tag{24}$$

where

$$g_j = \frac{k_j - \rho_j\sigma_j\phi i + d_j}{k_j - \rho_j\sigma_j\phi i - d_j}, \tag{25}$$

$$d_j = \sqrt{(\kappa_j - \rho_j\sigma_j\phi i)^2 + \sigma_j^2\phi(\phi+i)}. \tag{26}$$

Again, in order to have a better behaved integrand, we set $c_j = \frac{1}{g_j}$. We have

$$B_j(\tau,\phi) = \frac{\kappa_j - \rho_j\sigma_j\phi i - d_j}{\sigma_j^2}[\frac{1-e^{-d_j\tau}}{1-c_je^{-d_j\tau}}], \tag{27}$$

$$A(\tau,\phi) = r\phi i\tau + \sum_{j=1}^{2}\frac{\kappa_j\theta_j}{\sigma_j^2}[(\kappa_j - \rho_j\sigma_j\phi i - d_j)\tau - 2ln(\frac{1-c_je^{-d_j\tau}}{1-c_j})]. \tag{28}$$

To summarize the double Heston model, the formulation for the call price under the double Heston model is presented as following:

$$C(S,v,t) = S_tP_1 - Ke^{-r(T-t)}P_2 \tag{29}$$

$$P_1 = \frac{1}{2} + \frac{1}{\pi}\int_0^\infty Re(\frac{e^{-i\phi lnK}f(\phi - i, x, v_1, v_2)}{i\phi Se^{r\tau}})d\phi \tag{30}$$

$$P_2 = \frac{1}{2} + \frac{1}{\pi}\int_0^\infty Re(\frac{e^{-i\phi lnK}f(\phi, x, v_1, v_2)}{i\phi})d\phi \tag{31}$$

$$f(\phi; x_t, v_1, v_2) = \exp(A(\tau,\phi) + i\phi x_t + B_1(\tau,\phi)v_1 + B_2(\tau,\phi)v_2) \tag{32}$$

3.2 The Implied Volatility from the Double Heston Model

Compared to the implied volatility from the Heston model, the implied volatility from the double Heston model is much closer to the market implied volatility, no matter given a short or longer maturity [34].

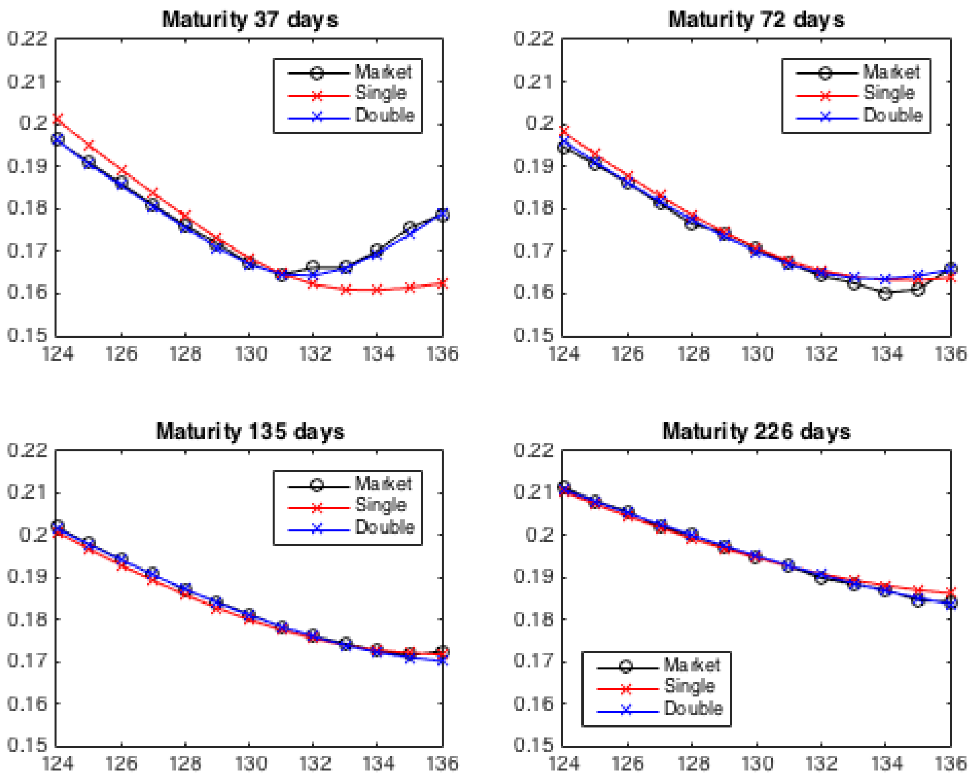


Figure 3.1: Implied Volatility of the Double Heston Model

CHAPTER 4
CALIBRATION FOR THE DOUBLE HESTON MODEL

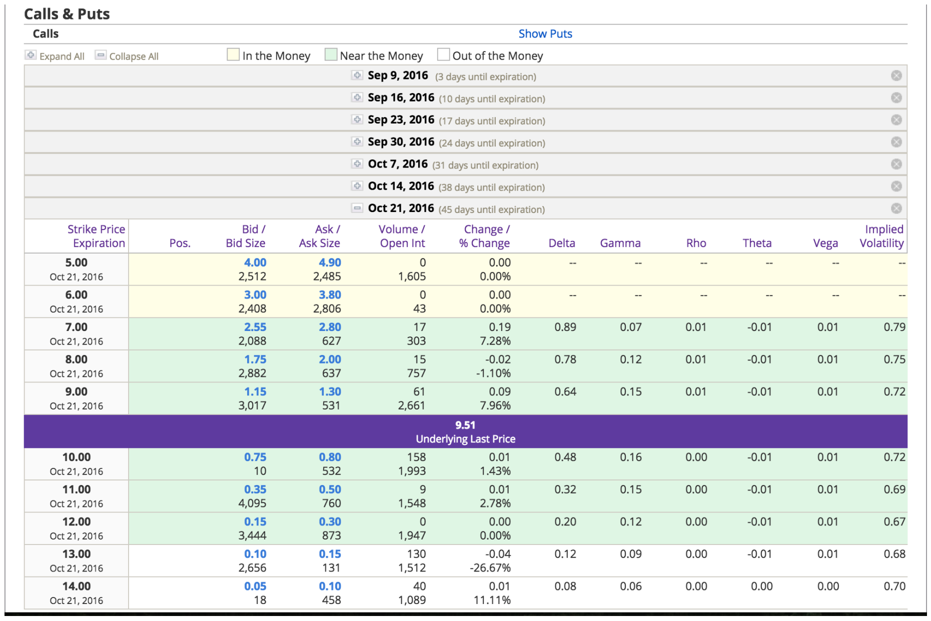
In the last three chapters, we introduced the closed form solution of the Heston model and the double Heston model. However, we have not introduced the calibration of our models. The parameter estimation plays a key role in the financial industry. Hedge funds and trading companies are all putting great effort on increasing the speed and accuracy of parameter estimation. In 2010, financial institutions, including high-speed algorithmic trading firms in the Wall Street, paid billions of dollars to the company called Spread Networks to build a line, running 827 miles from Chicago to Carteret in New Jersey. The line allows the quantitative traders in New York City to get the most fresh data from Chicago Mercantile Exchange immediately. In October 2012, Spread Networks announced that the estimated roundtrip time was around 13 milliseconds. Now, you see how competitive the battles between financial firms in the Wall Street are. The sooner the better. The traders do not want to be late for even a millisecond and sometimes they would use some unstable algorithm in practice for trading as long as the algorithm is fast enough.

All trading firms has their own trading software to help them make trading decisions. To calibrate models takes the most time in trading software development. We have to refresh our model, the parameters, as new data comes in. On the other hand, we also want to keep the accuracy at the same time. In this Chapter, we are going to discuss the calibration problem for the double Heston model.

There are several types of parameter estimation methods existed for being considered. The most straight forward one is the Monte Carlo (MC) simulation. However, the MC simulation is slow since it usually generates thousands of possible paths of the stock price and then take the average. Besides the MC technic, two most common ways utilized in the financial industry are the non-gradient and the gradient-based optimization algorithms. The gradient-based algorithm makes full use of the information from the computed gradient of the objective function (loss function) and thus it is usually faster than the non-gradient method, whose speed does not benefit from the information about the derivatives of the objective function. No matter which ways will be used, these tools are all with objective functions. The first section of this chapter will present the most common used objective functions when estimating parameters of the Heston and the double Heston model. The optimization algorithms will be discussed in the sections followed.

4.1 The Objective Function

The goal of calibrating the double Heston model is to find the proper parameter estimates so that the prices or the implied volatilities from the model are as close as possible to the real market data. The European call option can be viewed as a function of parameters. To illurstrate this point, it is good to have a look at the trading screen of one call option. As shown in the following picture, at different maturities and strikes, we have the corresponding call prices and implied volatilities.



Calls & Puts

Calls — Show Puts

Expand All — Collapse All — In the Money — Near the Money — Out of the Money

Sep 9, 2016 (3 days until expiration)

Sep 16, 2016 (10 days until expiration)

Sep 23, 2016 (17 days until expiration)

Sep 30, 2016 (24 days until expiration)

Oct 7, 2016 (31 days until expiration)

Oct 14, 2016 (38 days until expiration)

Oct 21, 2016 (45 days until expiration)

Strike Price / Expiration	Pos.	Bid / Bid Size	Ask / Ask Size	Volume / Open Int	Change / % Change	Delta	Gamma	Rho	Theta	Vega	Implied Volatility
5.00 Oct 21, 2016		4.00 2,512	4.90 2,485	0 1,605	0.00 0.00%	--	--	--	--	--	--
6.00 Oct 21, 2016		3.00 2,408	3.80 2,806	0 43	0.00 0.00%	--	--	--	--	--	--
7.00 Oct 21, 2016		2.55 2,088	2.80 627	17 303	0.19 7.28%	0.89	0.07	0.01	-0.01	0.01	0.79
8.00 Oct 21, 2016		1.75 2,882	2.00 637	15 757	-0.02 -1.10%	0.78	0.12	0.01	-0.01	0.01	0.75
9.00 Oct 21, 2016		1.15 3,017	1.30 531	61 2,661	0.09 7.96%	0.64	0.15	0.01	-0.01	0.01	0.72
9.51 Underlying Last Price											
10.00 Oct 21, 2016		0.75 10	0.80 532	158 1,993	0.01 1.43%	0.48	0.16	0.00	-0.01	0.01	0.72
11.00 Oct 21, 2016		0.35 4,095	0.50 760	9 1,548	0.01 2.78%	0.32	0.15	0.00	-0.01	0.01	0.69
12.00 Oct 21, 2016		0.15 3,444	0.30 873	0 1,947	0.00 0.00%	0.20	0.12	0.00	-0.01	0.01	0.67
13.00 Oct 21, 2016		0.10 2,656	0.15 131	130 1,512	-0.04 -26.67%	0.12	0.09	0.00	-0.01	0.01	0.68
14.00 Oct 21, 2016		0.05 18	0.10 458	40 1,089	0.01 11.11%	0.08	0.06	0.00	0.00	0.00	0.70

Figure 4.1: The Trading Interface of the Call Option of an Oil ETF

Recall that the market call price is determined by the trading activities of everyone involved, therefore we use $C^M(\tau, K)$ to represent the market price. For the price derived from the model, like the double Heston, parameters are the additional elements to achieve the price. So, we use $C(\tau, K, \Theta)$ to represent the model price. The Θ is the collection of κ, θ, σ, v_0, and ρ in the Heston model. In the double Heston model, $\Theta = (\kappa_1, \kappa_2, \theta_1, \theta_2, \sigma_1, \sigma_2, v_{01}, v_{02}, \rho_1, \rho_2)$. Similarly, $IV^M(\tau, K)$ represents the market implied volatility and $IV(\tau, K, \Theta)$ represents the model implied volatility.

For a set of N maturities τ_i and a set of M strikes K_j, we define the following two objective functions:

$$F_{price}(\Theta) = \frac{1}{NM} \sum_{\tau_i, K_j}^{N,M} (C^M(\tau_i, K_j) - C(\tau_i, K_j, \Theta))^2, \tag{1}$$

and

$$F_{IV}(\Theta) = \frac{1}{NM}\sum_{\tau_i, K_j}^{N,M}(IV^M(\tau_i, K_j) - IV(\tau_i, K_j, \Theta))^2, \tag{2}$$

The object of the calibration is to find the parameter estimates which minimizes the objective function $F_{price}(\Theta)$ or $F_{IV}(\Theta)$, under parameter constraints, like the feller's condition. $F_{IV}(\Theta)$ is more meaningful since the implied volatility plays the most important role in derivative trading. However, $F_{IV}(\Theta)$ is much more expensive to calculate since root-finding algorithm must be applied to find the implied volatility. DE

4.2 A Non-gradient Method: The Differential Evolution Algorithm

The Differential Evolution (DE) algorithm was first introduced by Storn and Price for global optimization in 1995 [39]. The DE algorithm is a population-based algorithm using the following operators: crossover, mutation and selection. As a popular heuristic algorithm, DE's advantages are its simple structure, ease of use, speed and robustness [39]. The main idea of the DE algorithm is that it keeps producing better candidates for the solution of the optimization problem by updating the existing candidate members. The flow of the DE algorithm can be determined as follows (and as shown in Figure 1.3).

Step 1: confirm the dimension of each the solution vector, which is usually the collection of parameters to be optimized. The dimension of the solution vector is equal to the number of key parameters that will be optimized by the DE algorithm. For example, the dimension of the solution vector for the Heston model is five and the dimenison of the solution vector for the double Heston model is ten. By optimizing the solution vector, DE algorithm generates the optimized value of all key parameters.

Step 2: set up the objective (fitness) function and the stopping criterion. The objective function, as we defined in section above, is used to judge whether the solution vector is good or not. The objective function can not be determined until the problem is understood very well. Because the DE algorithm is based on a cycle mechanism, it will not stop until the stopping criterion is met.

Step 3: initialize N, the number of solution vectors (candidates for the best estimates) and the dimension D of each solution vector. We also initialize the generation counter G as $G = 1$. Define the solution vectors as $x_i^j(G)$, where $i = 1, 2, ..., N$ and $j = 1, 2, ..., D$. Calculate the value of the objective function on each solution vector.

Step 4: mutation operation: mutate solution vectors based on the following equation

$$y_i(G+1) = x_{r1} + F[x_{r2}(G) - x_{r3}(G)],\ i = 1, 2, ..., N$$

where $r1$,$r2$ and $r3$ are mutually different integers and they are randomly selected from $\{1, 2, ..., i-1, i+1, ..., N\}$. $F > 0$ is a certain parameter called a mutation factor, which controls the amplification of the differential variation $(x_{r2}(G) - x_{r3}(G))$. F usually assumes a value from the range$[0, 2]$.

Step 5: crossover operation: following the mutation operation, the crossover operation helps increase the diversity of mutative solution vectors. Define

$$u_i^j(G+1) = \begin{cases} y_i^j(G+1) & \text{if} \quad rand_j \leq CR \ \text{ or } j = rand_i, \\ x_i^j(G), & \text{else,} \end{cases}$$
$$i = 1, 2, ..., N, \quad j = 1, 2, ..., D,$$

where $rand_j$ is a random number taken from $(0, 1)$, and $rand_i \in \{1, 2, ..., D\}$ is also a random number. $CR \in [0, 1]$ is the crossover constant.

Step 6: selection: according to the equation below, we decide the solution vectors of the next generation

$$x^i(G+1) = \begin{cases} u_i(G+1) & \text{if} \quad F_x(u_i(G+1)) < F_x(x_i(G)), \\ x_i(G), & \text{else,} \end{cases}$$

where F_x is the objective function . $F_x(u_i(G+1)) < F_x(x_i(G))$ means $u_i(G+1)$ is a better solution than $x_i(G)$

Step 7: loop to Step 4 until the stopping criterion is satisfied.

4.2.1 Choosing CR, F, and N

The choices of CR, NP and F depend on the specific problem being solved. These values directly influence the performance of the DE algorithm. Setting proper values for CR, NP and F is usually difficult [25][22]. We have some good guidelines

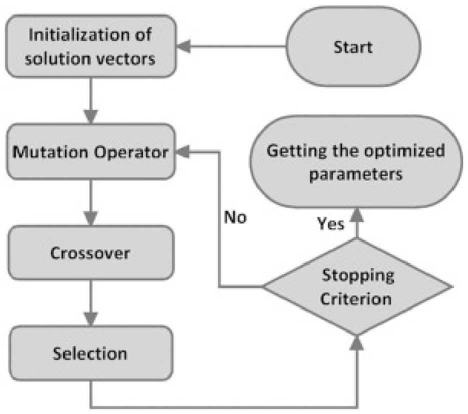

Initialization of solution vectors
Start
Mutation Operator
Getting the optimized parameters
No
Yes
Crossover
Stopping Criterion
Selection

Figure 4.2: Flow of the DE Algorithm

for setting them [25]. NP should be approximately $5 - 10$ times the value of the dimension of the solution vector. It is effective to let F lie in $[0.4, 1]$. A good range for CR is $[0.1, 1]$.

4.2.2 Accelerating the Pricing

Fiodar Kilin [26] pointed out that the most time-consuming part of the calibration is the unavoidable evaluation of the characteristic function of the double Heston model. Therefore, Fiodar Kilin used a caching technique to decrease the number of evaluations of the characteristic function as much as possible. The caching technique significantly cuts the amount of recalculations of the characteristic function.

Take the double Heston for example and let us have a look at equations (29) - (32) in Chapter 3. To approximate P_2 by using the Gauss-Lobatto quadrature, we first need to generate the weights $\{w_i\}_{i=1}^{M}$of the abscissas $\{x_i\}_{i=1}^{M}$, then the integral

$\int_a^b G(x)dx$ can be numerically approximated by $\sum_{j=1}^{N} w_j G(x_j)$. The integrand is the real part of $\frac{e^{-i\phi lnK} f(x_i,\Theta,\tau)}{i\phi}$. For a fixed strike price K, we have to evaluate the characteristic function $f(x_i, \Theta, \tau)$ at each x_i. Because $f(x_i, \Theta, \tau)$ does not depend on the value of K, we can reuse the value of $f(x_i, \Theta, \tau)$ at each x_i for different strike prices, as long as other parameters keep the same. Fiodar Kilin's paper future suggested that some time-consuming parts of the characteristic function does not depend on the maturity and the strike price, and thus we can reuse the value of those parts whenever only the maturity and the strike price change. As the algorithm was not given in his paper, this present paper will apply his future suggestion to the double Heston model, and investigate the performance. Taking computing P_2 for example, the future optimized algorithm can be summarized as Algorithm 4.1.

Algorithm 4.1 Caching technique to accelerate the numerical integration

Input: Parameters Θ, abscissas$\{x_i\}$, weights$\{w_i\}$, strike prices $K[n]$ and expires $\tau[n]$.

Output: Probabilities $P_2[\tau, K]$ from double Heston model.

```
for each $x_i$ do
  evaluate $c_j$ and $d_j$ at $x_i$; {% those parts do not depend on $\tau$ and $K$}
end for
for each $\tau_i$ in $\tau[n]$ do
  for each $x_i$ do
    evaluate the characteristic function at each $\tau_i$ and $x_i$;
  end for
  for  each $K_i$ in $K[n]$ do
    evaluate $P_2[\tau_i, K_i]$ at $K_i$;
  end for
end for
return  $P_2[\tau, K]$.
```

4.3 A Gradient-based Method: The Levenberg-Marquardt Method

The Levenberg-Marquardt method is recommended for solving curve fitting problem, especially when the objective function is in the form of the sum of squares. This algorithm changes between the steepest descent method and the Gauss-Newton method [30][31], by adjusting the a damping factor. The steepest descent method has the advantage that it is guaranteed to find the minimum through numerous times of iterations as long as it exists [28], since it always goes along non-increasing directions. However, the steepest descent method may take too much iterations to reach the minimum. The Gauss-Newton method is famous for its ability to find the minimum within small number of iterations, although the computation is heavy in each iteration. However, in some cases especially when the initial guess is very far off the final minimum, the Gauss-Newton method is not stable. The Levenberg-Marquardt was designed to inherit the stability from the steepest descent method and to obtain the fast convergence from the Newton method. Cui [13] recently applied the Levenberg-Marquardt Method to the calibration of the Heston model. They claimed that their calibration algorithm for the Heston model is faster than all previous existed models. In the Chapter 6, we are going to extend their algorithm to the double Heston model by using the technique of Algorithmic Differentiation. To finish this chapter, we present the Levenberg-Marquardt Method for the double Heston model in the following.

Let $C_D(\Theta, K_i, \tau_i)$ be the price from the double Heston model. Again, $C^M(K_i, \tau_i))$ represents the market price. The calibration problem can be viewed as

$$\min_{\Theta \in R^m} f(\Theta), \tag{3}$$

where

$$r_i(\Theta) = C_D(\Theta, K_i, \tau_i) - C^M(K_i, \tau_i)), \tag{4}$$

$$\tilde{r}(\Theta) = [r_1(\Theta), r_2(\Theta), ..., r_n(\Theta)]^T, \tag{5}$$

$$f(\Theta) = \frac{1}{2}\tilde{r}^T(\Theta)\tilde{r}(\Theta). \tag{6}$$

We define the Jacobian matrix of the vector $\tilde{r}(\Theta)$ as $J \in R^{m\times n}$, with elements

$J_{ij} = \frac{\partial C_D(\Theta, K_j, \tau_j)}{\partial \Theta_i}$. Then the Levenberg-Marquardt method can be summarized as Algorithm 4.2.

Algorithm 4.2 Levenberg-Marquard method for double Heston model

```
1: Initialize values for $\Theta$, the damping factor $\mu$, as well as the adjusting constant $a_u$
   and $a_d$. Evaluate $\tilde{r}$ and $J$;
2: Calculate $g = J^T J + \mu I$, and $\nabla f = J^T \tilde{r}$;
3: Evaluate $\Theta_{new} = \Theta - g^{-1} \nabla f$, and calculate $f(\Theta_{new})$;
4: If $f(\Theta_{new}) < f(\Theta)$: $\Theta = \Theta_{new}$, $\tilde{r} = \tilde{r}_{new}$ and $\mu = \mu \cdot a_d$.
   Otherwise: keep the old $\Theta$ and $\mu = \mu \cdot a_u$;
5: Check the stop criteria $||\tilde{r}|| < \epsilon_1$ or $||J \cdot \mathbf{e}|| < \epsilon_2$ or $\frac{||\Theta_{new} - \Theta||}{||\Theta||} < \epsilon_3$.
   If yes, stop; If no, go to step 2.
```

CHAPTER 5
SENSITIVITY ANALYSIS OF THE DOUBLE HESTON MODEL

In this chapter, we will discuss the sensitivities of the double Heston model to all its parameters. As mentioned in Chapter 4, to apply gradient methods to the calibration of the double Heston model, we have to run sensitive analysis on parameters $\Theta = (\kappa_1, \kappa_2, \theta_1, \theta_2, \sigma_1, \sigma_2, \sqrt{v_{01}}, \sqrt{v_{02}}, \rho_1, \rho_2)$. We have ten parameters here and it is not easy to cheaply compute all the first order partial derivatives, since the double Heston model itself is already very complicated. Obviously, it will cost more time to find the second order partial derivatives.

Also, we need to run sensitive analysis on other parameters: τ, S, r, v_{01}, and v_{02}. The sensitivity of the model price to these parameters are usually denoted by Greek letters and we call them Greeks. Greeks are the essential risk measures in options strategies. Insurance companies and banks hedge their assets mainly based on the value of Greeks of their portfolios. Greeks are also very useful for derivative traders.Trading options without an understanding of the Greeks is extremely dangerous.

In Section 5.1, we will introduce the most common Greeks in financial market. In Section 5.2, we will introduce the Algorithmic Differentiation, which is a powerful algorithm to conduct exact derivative calculation.

5.1 Greeks of the Double Heston Model

Let C_D be the model price derived for the double Heston Model. Define $v_1 = \sqrt{v_{01}}$ and Define $v_2 = \sqrt{v_{02}}$. Then, the most common greeks are defined in the Table 5.1. For other purposes of uses, values of many other Greeks, e.g., Lambda, Charm, Vera, Color and Zomma, might be needed but not very often.

Rouah[34] provided the semi-analytical expressions for some Greeks of the double Heston. Based on a different formulation of the price, Cui [13] recently provided the semi-analytical expressions for parameters set Θ of the Heston model, unfortunately not the double Heston model. However, it is still expensive to evaluate all those partial derivatives together. This problem can be ideally solved by the Algorithmic Differentiation, which is the topic of the next section.

Table 5.1: Double Heston Greeks

Greek	Definition	Greek	Definition
Delta	$\frac{\partial C_D}{\partial S}$	Gamma	$\frac{\partial^2 C_D}{\partial S^2}$
Rho	$\frac{\partial C_D}{\partial r}$	Theta	$-\frac{\partial C_D}{\partial \tau}$
Vega11	$\frac{\partial C_D}{\partial v_1}$	Vega12	$\frac{\partial C_D}{\partial v_2}$
Vanna1	$\frac{\partial^2 C_D}{\partial S \partial v_1}$	Vanna2	$\frac{\partial^2 C_D}{\partial S \partial v_2}$
Volga 1	$\frac{\partial^2 C_D}{\partial v_1^2}$	Volga2	$\frac{\partial^2 C_D}{\partial v_2^2}$

5.2 Algorithmic Differentiation

The Algorithmic Differentiation (also known as the Automatic Differentiation or AD) refers a family of techniques to numerically evaluate the derivatives (Jacobian, Hessian, etc) of a function, specified by a computer program which might contain thousands of lines. AD relies on the main idea that the function $f(x)$ defined by a computer program, regardless of its complexity, is simply a composition of elementary functions (x, sin(x), e^x, etc), and basic arithmetic (addition, multiplication, etc). What AD dose is applying all rules in calculus (quotient rule, product rule, especially chain rule, etc) repeatedly to all elementary functions and operations involved in the function. AD generates computer code without the need for deriving explicit formulas by hand, by making full use of the software representation of the function, and overloading basic operators and elementary functions.

In comparison to symbolic differentiation and finite difference methods, the chain rule based AD yields partial derivatives accurately and cheaply. Theoretical complexity results show that the reverse mode of AD produces the gradient vector at no more than five times the cost of evaluating the original scalar function [16]. Using the forward mode, the Jacobian can be calculated, on average, with operation count of no more than five times the operation count of the original function [5]. In practice, this bound has been abseved to be much lower [40]. In addition, AD also has the accuracy up to the machine precision, and has no truncation errors.

AD is not symbolic differentiation. AD is much more efficient than symbolic differentiation in differentiating algorithms. Consider the function programmed by the following code:

```
function f(x)
  y = x;
  for i = 1:10^5
    y = cos(x + y);
  end
return y
```

Symbolic differentiation would take massive memory to store, and much more time to compute the huge expression of the derivative. In contrast, AD can differentiate the function above in about the same time of evaluating the original function.

AD does not belong to finite-difference methods either. Derivatives are usually computed by some variant of the finite difference approach like

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}, \quad h \textit{ small}. \tag{1}$$

However, this method is inefficient because it requirs $\mathcal{O}(n)$ evaluations of n-dimensional function $f(x)$ to compute the gradient $\nabla f(x)$. In addition, we have to choose some small value for h, which will cause approximation error. In contrast, in the best case, the exact gradient of $f(x)$ can be evaluated at the cost of $\mathcal{O}(1)$ evaluations of $f(x)$ itself.

The AD has two modes: the forward mode and the adjoint mode. The two methods may have substantially different costs, based on the number of inputs and outputs of the function. Forward mode is more efficient when you have fewer arguments than outputs. In contrast, the reverse mode (also named adjoint mode) is much better when the number of arguments is much larger than that of the outputs. One successful application of the adjoint mode in financial industry is the Credit Valuation Adjustment (CVA) modeling [8]. The CVA model [42] contains over 600 parameters. For a CVA evaluation taking 10 seconds, adjoint approach produces the full set of sensitivities in less than 40 seconds, while finite differences require approximately 1 hour and 40 minutes [19]. In this present thesis, the adjoint mode will be applied to the calibration of the double Heston model, since the objective function is a one-dimensional function with 10 parameters. To illustrate the difference between the reverse mode and the forward node, two baby examples from Griewank's book [16] will be presented in the following subsection.

5.2.1 Forward Mode vs Reverse Mode

Consider computing $\frac{\partial f}{\partial x}$, where $f(x, y)$ is given as

$$f(x,y) = \big[sin(\frac{x}{y}) + \frac{x}{y} - e^y\big] * \big[\frac{x}{y} - e^y\big].$$

The computer program evaluation steps for $f(x, y)$ are listed in Table 5.2. The forward mode of AD is described as Table 5.3 and $\dot{m}_i$ represents the derivative of the ith intermediate variable respect to the input x. The adjoint mode of AD is given in Table 5.4 and $\bar{m}_i$ represents the derivative of the final output f respect to the ith intermediate variable.

Table 5.2: Evaluate $f(x, y)$

at $(x, y) = (1.5000, 0.5000)$

$m_{-1} = x$	$= 1.5000$	
$m_0 = y$	$= 0.5000$	
$m_1 = m_{-1}/m_0$	$= 1.5000/0.5000$	$= 3.0000$
$m_2 = sin(m_1)$	$= sin(3.000)$	$= 0.1411$
$m_3 = e^{m_0}$	$= exp(0.5000)$	$= 1.6487$
$m_4 = m_1 - m_3$	$= 3.0000 - 1.6487$	$= 1.3513$
$m_5 = m_2 + m_4$	$= 0.1411 + 1.3513$	$= 1.4924$
$m_6 = m_5 * m_4$	$= 1.4924 * 1.3513$	$= 2.0167$
$y = m_6$	$= 2.0167$	

Table 5.3: Forward Mode to Find $\frac{\partial f}{\partial x}$ at $(x, y) = (1.5000, 0.5000)$

$m_{-1} = x$	$= 1.5000$	
$\dot{m}_{-1} = \dot{x}$	$= 1.0000$	
$m_0 = y$	$= 0.5000$	
$\dot{m}_0 = \dot{y}$	$= 0.0000$	
$m_1 = m_{-1}/m_0$	$= 1.5000/0.5000$	$= 3.0000$
$\dot{m}_1 = (\dot{m}_{-1} - m_1 * \dot{m}_0)/m_0$	$= 1.0000/0.5000$	$= 2.0000$
$m_2 = sin(m_1)$	$= sin(3.000)$	$= 0.1411$
$\dot{m}_2 = cos(m_1) * \dot{m}_1$	$= -0.9900 * 2.0000$	= -1.9800
$m_3 = e^{m_0}$	$= exp(0.5000)$	$= 1.6487$
$\dot{m}_3 = m_3 * \dot{m}_0$	$= 1.6478 * 0.0000$	$= 0.0000$
$m_4 = m_1 - m_3$	$= 3.0000 - 1.6487$	$= 1.3513$
$\dot{m}_4 = \dot{m}_1 - \dot{m}_3$	$= 2.0000 - 0.0000$	$= 2.0000$
$m_5 = m_2 + m_4$	$= 0.1411 + 1.3513$	$= 1.4924$
$\dot{m}_5 = \dot{m}_2 + \dot{m}_4$	$= -1.9800 + 2.0000$	$= 0.0200$
$m_6 = m_5 * m_4$	$= 1.4924 * 1.3513$	$= 2.0167$
$\dot{m}_6 = \dot{m}_5 * m_4 + m_5 * \dot{m}_4$	$= 0.0200 * 1.3513 + 1.4924 * 2.0000$	$= 3.0118$
$y = m_6$	$= 2.0167$	
$\dot{y} = \dot{m}_6$	$= 3.0118$	

Table 5.4: Backward Mode to Find $\frac{\partial f}{\partial x}$ and $\frac{\partial f}{\partial y}$ at $(x, y) = (1.5000, 0.5000)$

$m_{-1} = x = 1.5000$
$m_0 = y = 0.5000$
$m_1 = m_{-1}/m_0 = 1.5000/0.5000 = 3.0000$
$m_2 = sin(m_1) = sin(3.000) = 0.1411$
$m_3 = e^{m_0} = exp(0.5000) = 1.6487$
$m_4 = m_1 - m_3 = 3.0000 - 1.6487 = 1.3513$
$m_5 = m_2 + m_4 = 0.1411 + 1.3513 = 1.4924$
$m_6 = m_5 * m_4 = 1.4924 * 1.3513 = 2.0167$
$f = m_6 = 2.0167$
$\bar{m}_6 = \bar{f} = 1.0000$
$\bar{m}_5 = \bar{m}_6 * m_4 = 1.0000 * 1.3513 = 1.3513$
$\bar{m}_4 = \bar{m}_6 * m_5 = 1.0000 * 1.4924 = 1.4924$
$\bar{m}_4 = \bar{m}_4 + \bar{m}_5 = 1.4924 + 1.3513 = 2.8437$
$\bar{m}_2 = \bar{m}_5 = 1.3513$
$\bar{m}_3 = -\bar{m}_4 = -2.8437$
$\bar{m}_1 = \bar{m}_4 = 2.8437$
$\bar{m}_0 = \bar{m}_3 * m_3 = -2.8437 * 1.6487 = -4.6884$
$\bar{m}_1 = \bar{m}_1 + \bar{m}_2 * cos(m_1) = 2.8437 + 1.3513 * (-0.9900) = 1.5059$
$\bar{m}_0 = \bar{m}_0 - \bar{m}_1 * m_1/m_0 = -4.6884 - 1.5059 * 3.000/0.5000 = -13.7239$
$\bar{m}_{-1} = \bar{m}_1/m_0 = 1.5059/0.5000 = 3.0118$
$\bar{y} = \bar{m}_0 = -13.7239$
$\bar{x} = \bar{m}_{-1} = 3.0118$

As shown in the Table 5.3, the forward mode of AD just simply applies chain rule mechanically to each line in the Table 5.2. For the adjoint mode, things becomes a little bit more complicated. The reverse mode of AD first evaluates the function forwardly (forward sweep), as same as Table 5.2. Then, the adjoint mode of AD computes the derivative of the final output with respect to each intermediate variable reversely (reverse sweep). Keane and Nair [23] summarized the general trace of the adjoint mode of AD for scaled functions in their book. Let $\{x_i\}_{i=1}^{N}$ be the collection of all the input variables, y be the output, and $\{m_i\}_{i=1-N}^{L}$ be all the intermediate variables $\{m_i\}_{i=1-N}^{L}$. Let $\bar{m}_i$ be the derivative of the final output with respect to the ith intermediate variable m_i. Let u_i be the collection of all the intermediate variables that m_i **directly** depends on ($j \prec i$), and thus we set $m_i = \varphi_i(u_i)$. Therefore the adjoint mode of AD can be generally described as the following Table 5.5.

Table 5.5: Adjoint Recursion

$\bar{m}_i$	$\leftarrow$	0	$i = 1-n, \cdots, L$
m_{i-N}	$\leftarrow$	x_i	$i = 1, \cdots, N$
m_i	$\leftarrow$	$\varphi_i(u_i)$	$i = 1, \cdots, L$
y	$\leftarrow$	m_L	
$\bar{m}_L$	$\leftarrow$	1.000	
$\bar{m}_j$	$\leftarrow$	$\bar{m}_j + \bar{m}_i \cdot \frac{\partial}{\partial m_j}\varphi_i(u_i)$ for $j \prec i$	$j = L-1, \cdots, 1-N$
$\bar{x}_i$	$\leftarrow$	$\bar{m}_{i-N}$	$i = N, \cdots, 1$

CHAPTER 6
APPLICATIONS AND NUMERICAL RESULTS

In this Chapter, we will introduce our new hybrid calibration model, and list the numerical results. The computations were performed on a MacBook Pro with a 2.4 GHz Intel Core i5 processor, 8 GB of RAM and OS X EI Capitan version 10.11.6. All algorithms were coded in C++ using Xcode version 7.3.1. The C++ boost library [35] was used to conduct matrix operations.

In section 4.2, we discussed the algorithms that can improve the speed of pricing. Choose $S = 61.9$, $r = 0.03$, and $q = 0$. We set the parameters as $[\kappa_1, \kappa_2, \theta_1, \theta_2, \sigma_1, \sigma_2, v_{01}, v_{02}, \rho_1, \rho_2] = [0.9, 1.2, 0.1, 0.15, 0.1, 0.2, 0.36, 0.49, -0.5, -0.5]$. To generate 100 option prices, we choose ten different strike prices and ten different expiration dates. For the Gauss-lobatto quadrature, we choose $a = 10^{-8}$, $b = 100$ and $N = 32, 64, 128..$ Repeat the experiment 3000 times, the average computational costs under different schemes are listed in the following Table 6.1.

Table 6.1: Computational Cost of the Double Heston Model (in ms)

	N = 32	N = 64	N = 128
Original algorithm	14.647	28.472	57.410
Kilin's algorithm	2.316	4.491	8.969
Algorithm 4.1	1.968	3.827	7.677

According to the Table 6.1, Kilin's algorithm can also significantly accelerate the double Heston pricing model, although his idea was designed for the original Heston model. By using our algorithm 4.1, on average, the time of computation can be reduced by 15% further (compared to the algorithm mentioned in Kilin's paper). Kilin proved that his scheme of the direct integration with the caching technique outperforms the FFT methods [2] [9][10], and should be the best choice for practical applications. Therefore, we should follow the algorithm 4.1 for the pricing purpose in our study.

In the following discussion about the calibration for the double Heston model, we refer to finding parameter estimates minimizing the loss function $F_{price}(\Theta)$ (equation (1) in 4.1). It would be the most ideal case if we could utilize gradient-based methods. Gradient-based methods require, and make full use of the computation of Hessians matrix or Jacobian matrix, which improves the rate of convergence. Achieving a fast and accurate computation of the Jacobian matrix is one big obstacle to applying gradient-based method to the double Heston model. Classical finite difference methods do not work well for the double Heston model. The automatic algorithmic differentiation will be used to compute the Jacobian matrix in our study. Again, we generate 100 option prices, with ten different strikes and maturities. As ten parameters need to be calibrated in the double Heston model, the Jacobian matrix is a 100 by 10 matrix. Using operator overloading and expression templates, many existed software libraries [3][18] are able to translate the algorithm code to the adjoint code, which enables algorithms to be differentiated automatically. Although the run time of applying such libraries may be a little bit slower than the code written by hand since the translating process is involved, but it is much faster in terms of the user time.

Unfortunately, the pricing algorithm of the double Heston model is full of operations of complex numbers. So far, there is no fast, and stable software libraries that also has a good functionality for differentiating those algorithms which contain many operations of complex numbers. However, a function of complex numbers can always be implemented as the combination of two functions of double numbers such that one function computes the real part and the other one computes the imaginary part. This present thesis implemented the double Heston model in the way that is compatible with AD libraries. As different AD libraries are achieved in many different ways, the efficiency of each library varies a lot. This thesis uses the following two popular libraries, the FADBAD and Adept, to differentiate the pricing algorithm of the double Heston model. We will also compare the result to that from the finite difference method. Keep using the some setting for parameters Θ mentioned above, and repeat the experiment for 500 times, the average computational costs under different schemes are listed in the following Table 6.2.

Table 6.2: Computational Cost of the Jacobian Matrix of the Double Heston Model with Respect to Parameters Θ (in ms)

	N = 32	N = 64	N = 128
Adept	76.58	154.50	300.98
FADBAD	543.58	1067.25	2087.72
Finite Difference	291.97	571.38	1124.68

As shown in Table 6.2, the way of implementing the AD library affects the computation efficiency a lot. The Adept library is much faster than the FADBAD library, although they all belong to the ADD libraries. Also, the Adept library beats the classical finite difference method. In fact, the efficiency is not the only advantage of applying the AD to the double Heston model. Another advantage is that the AD is more accurate and stable. Fix the maturity as one year ($\tau = 1.0$), we generate five prices using strikes $K = [55, 60, 61.9, 65, 70]$. Therefore, we have a 5 by 10 Jacobian matrix. The results are listed in the following four pages.

.

$$N = 128$$

Jacobian matrix from the Adept library:

$$\begin{pmatrix}
-0.952888 & 4.49315 & -0.248976 & -2.32506 & 8.68017 & -1.02313 & 5.48439 & -0.370774 & 2.45301 & 7.60949 \\
-1.01162 & 4.78587 & -0.376694 & -2.98736 & 9.23644 & -1.07975 & 5.83096 & -0.523554 & 3.19575 & 8.07537 \\
-1.03211 & 4.88065 & -0.424808 & -3.21921 & 9.41599 & -1.09874 & 5.94253 & -0.580772 & 3.45838 & 8.22446 \\
-1.05719 & 5.01686 & -0.502067 & -3.57481 & 9.67333 & -1.12236 & 6.1021 & -0.669935 & 3.86299 & 8.43661 \\
-1.09081 & 5.19139 & -0.621925 & -4.09136 & 10.0013 & -1.15245 & 6.30457 & -0.806698 & 4.45657 & 8.70304
\end{pmatrix}.$$

Jacobian matrix from the FADBAD library:

$$\begin{pmatrix}
-0.960115 & 4.49315 & -0.250831 & -2.32467 & 8.68017 & -1.0244 & 5.48439 & -0.371409 & 2.45325 & 7.60949 \\
-1.01885 & 4.78587 & -0.378549 & -2.98697 & 9.23644 & -1.08102 & 5.83096 & -0.524189 & 3.19598 & 8.07537 \\
-1.03582 & 4.88065 & -0.426663 & -3.21886 & 9.41599 & -1.09747 & 5.94253 & -0.580137 & 3.45814 & 8.22446 \\
-1.0607 & 5.01686 & -0.502067 & -3.57479 & 9.67333 & -1.12363 & 6.1021 & -0.67057 & 3.86323 & 8.43661 \\
-1.10156 & 5.19139 & -0.623781 & -4.09097 & 10.0013 & -1.15489 & 6.30457 & -0.807332 & 4.45681 & 8.70304
\end{pmatrix}.$$

Jacobian matrix from the finite difference method with $\Delta = 10^{-7}$:

$$\begin{pmatrix}
-0.703281 & 3.45141 & 1.57722 & 18769.3 & 6.72555 & -0.805793 & 4.28171 & 18764.7 & 18757.2 & 6.04158 \\
-0.833974 & 4.06811 & 1.43075 & 18768.3 & 7.9195 & -0.947326 & 5.03742 & 18764.6 & 18758.1 & 7.09386 \\
-0.883679 & 4.30343 & 1.36976 & 18768 & 8.37466 & -1.00083 & 5.32526 & 18764.5 & 18758.5 & 7.49392 \\
-0.964412 & 4.68657 & 1.26412 & 18767.4 & 9.11523 & -1.08737 & 5.79332 & 18764.5 & 18759.1 & 8.14361 \\
-1.09275 & 5.29801 & 1.07855 & 18766.5 & 10.2958 & -1.22402 & 6.53878 & 18764.3 & 18760.2 & 9.1762
\end{pmatrix}.$$

$$N = 64$$

Jacobian matrix from the Adept library:

$$\begin{pmatrix}
-0.944002 & 4.49539 & -0.297464 & -2.32487 & 8.68451 & -1.02665 & 5.48355 & -0.36242 & 2.45301 & 7.60836 \\
-1.00306 & 4.78829 & -0.429373 & -2.98683 & 9.24111 & -1.08322 & 5.83019 & -0.51607 & 3.19574 & 8.07433 \\
-1.02874 & 4.88313 & -0.479079 & -3.21857 & 9.42079 & -1.10078 & 5.94179 & -0.572984 & 3.45814 & 8.22346 \\
-1.04895 & 5.01945 & -0.558936 & -3.57402 & 9.67833 & -1.1207 & 6.1014 & -0.660782 & 3.86205 & 8.43566 \\
-1.08291 & 5.19414 & -0.682985 & -4.09023 & 10.0066 & -1.15072 & 6.30394 & -0.798415 & 4.45563 & 8.70218
\end{pmatrix}.$$

Jacobian matrix from the FADBAD library:

$$\begin{pmatrix}
-0.958845 & 4.49539 & -0.304886 & -2.32338 & 8.68451 & -1.02197 & 5.48355 & -0.36242 & 2.45301 & 7.60836 \\
-1.04681 & 4.78829 & -0.444217 & -2.98394 & 9.24111 & -1.07346 & 5.83019 & -0.513531 & 3.19481 & 8.07433 \\
-1.04358 & 4.88313 & -0.486501 & -3.21716 & 9.42079 & -1.09101 & 5.94179 & -0.570445 & 3.4572 & 8.22346 \\
-1.07786 & 5.01945 & -0.566358 & -3.57246 & 9.67833 & -1.12109 & 6.1014 & -0.663321 & 3.86299 & 8.43566 \\
-1.09775 & 5.19414 & -0.690407 & -4.08867 & 10.0066 & -1.15111 & 6.30394 & -0.800954 & 4.45657 & 8.70218
\end{pmatrix}.$$

Jacobian matrix from the finite difference method with $\Delta = 10^{-7}$:

$$\begin{pmatrix}
-0.544997 & 3.45417 & 4.66965 & 74481.2 & 6.73087 & -0.660913 & 4.27902 & 74479.4 & 74471 & 6.03789 \\
-0.671256 & 4.07104 & 4.09358 & 74480.4 & 7.92516 & -0.798267 & 5.0348 & 74479.1 & 74472 & 7.09026 \\
-0.719276 & 4.30642 & 3.86933 & 74480.1 & 8.38044 & -0.850184 & 5.32267 & 74479 & 74472.4 & 7.49036 \\
-0.797261 & 4.68967 & 3.49734 & 74479.6 & 9.12123 & -0.934132 & 5.79077 & 74478.8 & 74473 & 8.14011 \\
-0.921164 & 5.30129 & 2.88215 & 74478.7 & 10.3022 & -1.06661 & 6.5363 & 74478.5 & 74474 & 9.17279
\end{pmatrix}.$$

$$N = 32$$

Jacobian matrix from the Adept library:

$$\begin{pmatrix} -0.913298 & 4.49895 & -0.498963 & -2.3277 & 8.69169 & -1.02171 & 5.47414 & -0.325321 & 2.45633 & 7.59575 \\ -0.976929 & 4.79414 & -0.648526 & -2.9827 & 9.25262 & -1.06001 & 5.82292 & -0.473187 & 3.19021 & 8.06457 \\ -1.02456 & 4.88967 & -0.704801 & -3.21222 & 9.43359 & -1.07776 & 5.93512 & -0.531581 & 3.45114 & 8.2145 \\ -1.02727 & 5.02694 & -0.795214 & -3.56444 & 9.69295 & -1.10452 & 6.09553 & -0.624133 & 3.85395 & 8.4278 \\ -1.06561 & 5.2029 & -0.935888 & -4.07617 & 10.0237 & -1.15676 & 6.29906 & -0.774988 & 4.44864 & 8.69571 \end{pmatrix}.$$

Jacobian matrix from the FADBAD library:

$$\begin{pmatrix} -1.02892 & 4.49895 & -0.528651 & -2.32208 & 8.69169 & -1.00139 & 5.47414 & -0.315165 & 2.45258 & 7.59575 \\ -0.976929 & 4.79414 & -0.648526 & -2.98238 & 9.25262 & -1.06158 & 5.82292 & -0.483343 & 3.19396 & 8.06457 \\ -1.02456 & 4.88967 & -0.704801 & -3.21159 & 9.43359 & -1.11838 & 5.93512 & -0.551894 & 3.45864 & 8.2145 \\ -1.02727 & 5.02694 & -0.795214 & -3.56413 & 9.69295 & -1.12483 & 6.09553 & -0.634289 & 3.8577 & 8.4278 \\ -1.12186 & 5.2029 & -0.935888 & -4.07617 & 10.0237 & -1.15676 & 6.29906 & -0.774988 & 4.44864 & 8.69571 \end{pmatrix}.$$

Jacobian matrix from the finite difference method $\Delta = 10^{-7}$:

$$\begin{pmatrix} 0.0774589 & 3.47134 & 16.7739 & 293392 & 6.75967 & -0.0901775 & 4.27202 & 293402 & 293390 & 6.02352 \\ -0.0258882 & 4.05902 & 14.5378 & 293392 & 7.89997 & -0.206228 & 4.99409 & 293401 & 293391 & 7.03225 \\ -0.0660459 & 4.28768 & 13.6799 & 293392 & 8.34294 & -0.250832 & 5.27423 & 293401 & 293392 & 7.42249 \\ -0.13192 & 4.6637 & 12.2714 & 293392 & 9.07065 & -0.323509 & 5.73405 & 293400 & 293392 & 8.06185 \\ -0.237623 & 5.27071 & 9.97863 & 293391 & 10.2437 & -0.439009 & 6.47434 & 293399 & 293393 & 9.0884 \end{pmatrix}.$$

$$N = 128$$

Jacobian matrix from the finite difference method $\Delta = 10^{-5}$:

$$\begin{pmatrix} -130.096 & 3.45142 & 1.57719 & 193.123 & 6.72555 & -0.805837 & 4.28171 & 0.890552 & 181.037 & 6.04158 \\ -141.99 & 4.06811 & 1.43076 & 192.198 & 7.9195 & -0.947371 & 5.03742 & 0.801364 & 182.003 & 7.09386 \\ -146.509 & 4.30343 & 1.36978 & 191.841 & 8.37466 & -1.00088 & 5.32526 & 0.760823 & 182.379 & 7.49392 \\ -153.883 & 4.68657 & 1.26417 & 191.255 & 9.11523 & -1.08742 & 5.79332 & 0.687228 & 182.999 & 8.1436 \\ -165.774 & 5.29801 & 1.07864 & 190.307 & 10.2958 & -1.22407 & 6.53878 & 0.55036 & 184.013 & 9.17619 \end{pmatrix}.$$

Jacobian matrix from the finite difference method $\Delta = 10^{-7}$:

$$\begin{pmatrix} -0.703281 & 3.45141 & 1.57722 & 18769.3 & 6.72555 & -0.805793 & 4.28171 & 18764.7 & 18757.2 & 6.04158 \\ -0.833974 & 4.06811 & 1.43075 & 8768.3 & 7.9195 & -0.947326 & 5.03742 & 18764.6 & 18758.1 & 7.09386 \\ -0.883679 & 4.30343 & 1.36976 & 18768 & 8.37466 & -1.00083 & 5.32526 & 18764.5 & 18758.5 & 7.49392 \\ -0.964412 & 4.68657 & 1.26412 & 18767.4 & 9.11523 & -1.08737 & 5.79332 & 18764.5 & 18759.1 & 8.14361 \\ -1.09275 & 5.29801 & 1.07855 & 18766.5 & 10.2958 & -1.22402 & 6.53878 & 18764.3 & 18760.2 & 9.1762 \end{pmatrix}.$$

Jacobian matrix from the finite difference method with $\Delta = 10^{-9}$:

$$\begin{pmatrix} -0.719391 & 3.45143 & 1.60789 & 1.87638e+06 & 6.72556 & -0.809036 & 4.28175 & 0.888931 & 1.87637e+06 & 6.04161 \\ -0.850021 & 4.06813 & 1.461 & 1.87638e+06 & 7.91951 & -0.950363 & 5.03748 & 0.799652 & 1.87637e+06 & 7.09392 \\ -0.89975 & 4.30339 & 1.39978 & 1.87638e+06 & 8.37465 & -1.0038 & 5.32526 & 0.759018 & 1.87637e+06 & 7.4939 \\ -0.980469 & 4.68654 & 1.29388 & 1.87638e+06 & 9.1152 & -1.09021 & 5.7933 & 0.685333 & 1.87637e+06 & 8.14359 \\ -1.10872 & 5.29803 & 1.10793 & 1.87638e+06 & 10.2959 & -1.22659 & 6.53882 & 0.548416 & 1.87637e+06 & 9.17621 \end{pmatrix}.$$

The results show that it is accurate and stable to apply the AD library to the double Heston model. In contrast, the finite difference method is unstable. When the Gauss-lobatto quadrature node number N changes from 32 to 128, the Jacobian matrix should not change too much, since the double Heston model price does not change a lot. This is illustrated in the Jacobian matrix derived from the Adept and FADBAD library. Also, the finite difference method is too sensitive to the change of step size Δ. When the step size varies from 10^{-5} to 10^{-9}, the derivatives of the price respect to changes of some parameters dramatically. For each parameter, a particular step size should be chosen since a common step size might not work for every parameter in Θ. We do see NaN exceptions in the some columns of the Jacobian matrix for some particular parameter estimates. This might be caused by the properties of the double Heston model. In our study we reset the value as 1.0 if any value in the Jacobian matrix turns out to be NaN. Although this is not the best way to deal with the NaN issue, however we can still accelerate the calibration process significantly.

The AD libraries helped us overcome the difficulty of finding the Jacobian matrix. The next question is: what kind of gradient-based optimization method should we use? Cui [13] suggested the Levenberg-Marquardt method. Cui's study applied the Levenberg-Marquardt method to calibrate the Heston model. In her study, she claimed that the levenberg-Marquardt method does not have the problem of multiple local minima reported in previous research. Unfortunately, this is at least not the case for the double Heston model. We use the data of S&P 500 options on

April 13, 2012, with S = 137.14, q = 0.0068, and r = 0.001.

Table 6.3: S&P 500 Options on April 13, 2012

	K = 120	K = 125	K = 130	K = 135
τ = 0.12328	17.540	12.890	8.537	4.690
τ = 0.26849	18.480	14.124	10.041	6.437
τ = 0.71507	21.137	17.188	13.510	10.162
τ = 0.95432	22.365	18.547	15.045	11.761

To illustrate the problem of the local minima, we try several initial values for κ_1, θ_1, σ_1, ρ_1, v_{01}, κ_2, θ_2, σ_2, ρ_2, and v_{02}. Set $\epsilon_1 = 10^{-5}$, $\epsilon_2 = 10^{-10}$, $\epsilon_3 = 10^{-10}$, $a_d = 0.5$, and $a_u = 2$. Set the maximum iteration number to be 3000. For the Gauss-lobatto quadrature, we choose $a = 10^{-8}$, $b = 100$ and $N = 64$. When the initial values are [1.62252, 0.125338, 1.48448, -0.803741, 0.0231058, 3, 0.01, 0.813268, 0.475087, 0.00881475], the Levenberg-Marquardt method conducts 75 evaluations of the Jacobian matrix, and it stops with $\frac{||\Theta_{new}-\Theta||}{||\Theta||} < 10^{-10}$. Evaluating the Jacobian matrix costs most of the time. When we use the FADBAD library to calculate the Jacobian matrix, the whole process takes 18.3 seconds, while the time is reduced to about 3 seconds using the Adep library. The loss error is 0.000445, which is pretty good. However, with initial values [0.997568, 0.32112, 1.79195, 0.794992, 0.186382, 2.18037, 0.67025, 0.255969, -0.998349, 1.9827], the Levenberg-Marquardt method conducts 30 evaluations of the Jacobian matrix, and it stops with $\frac{||\Theta_{new}-\Theta||}{||\Theta||} < 10^{-10}$. However, the loss error is 1.23214, which is not good. With initial values [1.39178, 1.16608, 0.61754, 0.325766, 0.0245606, 1.652, 0.0396852, 0.154538, -0.863593, 0.219294], the Levenberg-Marquardt method conducts 50 evaluations of the Jacobian matrix, and it stops with $\frac{||\Theta_{new}-\Theta||}{||\Theta||} < 10^{-10}$. The loss error is 0.0169041, which is good.

Thousands more examples can be given to demonstrate that applying the Levenberg-Marquardt method to the double Heston model, suffers from the problem of local minima. However, this does not mean that we should throw the Levenberg-Marquardt method away from the double Heston model. It will still be great if we could locally apply the Levenberg-Marquardt to accelerate the convergent rate of the calibration process. The heuristic optimization method is applied in our study, in order to locate a local area the Levenberg-Marquardt method has a better convergence behavior. In the present thesis, we present the hybrid model by mixing the Differential Evolution algorithm and Levenberg-Marquardt method. Our new model has a much better performance than the single Differential Evolution algorithm, in the sense of accuracy and efficiency.

To illustrate, in the following examples, for the Differential Evolution algorithm, we set CR = 0.7, F = 0.8, NP = 150, and the maximum iteration number to be 2000. Keep using the data in Table 6.3, we first employ the DE algorithm, and stop it when the loss error is less than 0.1. When the DE algorithm finishes, we then use the parameter estimates derived from the DE algorithm as the initial values for the Levenberg-Marquardt method. Repeat the experiment for 50 times, on average, the DE algorithm takes 118.70 seconds to reduce the loss error under 0.1. On average, the Levenberg-Marquardt method with the FADBAD takes 25.481 seconds (less than 5 seconds with the Adept) to reduce the loss error to 0.011. In addition, we also need to explore how long the DE algorithm will take to reduce the loss error to 0.011. This time, we stop the DE algorithm when the loss error is less than 0.011. It turns out that the DE algorithm takes 267.89 seconds to a achieve a loss error under 0.011. Furthermore, on average, the Levenberg-Marquardt method with the FADBAD takes 8.4 seconds (less than 2 seconds with the Adept) to reduce the loss error to 0.0016. As you can see, the Levenberg-Marquardt method accelerated the convergence rate, and significantly enhanced the accuracy.

Instead of just doing the application of the double Heston model, our study tends to provide a general practical idea to accelerate the calibration of financial models, by combining the automatic algorithmic differentiation algorithms, heuristic optimization algorithms, and gradient-based optimization methods. Such idea can be used in many other financial models (Monte Carlo simulations).

The good point of the Levenberg-Marquardt method is that it does not make the move unless the loss error is decreased. It is always not harmful to at least have a try, since the Levenberg-Marquardt method is much more computational cheap than heuristic methods. For heuristic methods, there existed so many models, and it is just hard to judge which one is better than others. The choice totally depends on the particular problem being solved. The common advantage of heuristic models is that the local minima are almost sure guaranteed. Under this principle, most heuristic optimization methods (e.g. PSO, GA, and the Nelder-Mead method) can be bundled with gradient-based methods (like the Levenberg-Marquardt method), as long as the goal is to find a relatively satisfactory local minimum. Also, it is not necessary to apply the heuristic method first and then the gradient-based method. Both types of methods can be used interchangeably multiple times in order to achieve a faster calibration. It is also worthwhile to mention that the AD algorithm is just a tool to calculate the Jacobian matrix, if it is not required. It is possible to derive the analytic derivatives for many models. However, as fast AD libraries (like Adept) are only 10-25% slower than carefully handwritten code [18], it is still very efficient and saves lots of time in the sense of user time. By the way, AD libraries can also be used to compute the Greeks.

CHAPTER 7
CONCLUSION AND FUTURE WORK

Derivative pricing plays a core role in financial modeling. This present thesis does a detailed study on the pricing methodology under the risk neutral measure, the sensitivity analysis, and the calibration of the double Heston model, which is an extension of the Heston model (a popular stochastic volatility pricing model for equity derivatives). With the idea of Kilin's Strike Vector computation method, which can be further optimized, we reduce the pricing time by about 15%. Our study made another contribution by conducting the sensitivity analysis using the Automatic Differentiation (AD) algorithm. As our study demonstrates, the AD algorithm is a faster and more stable tool than the finite difference methods. We use the AD algorithm to conquer the barrier of computing the Jacobian matrix of the double Heston model with respect to 10 parameters. Lastly, our novel hybrid model enables us to accelerate the calibration of the double Heston model. This new hybrid model combines the heuristic method (Differential Evolution algorithm) and the gradient-based algorithm (Levenberg-Marquardt method), and it significantly enhances the efficiency and accuracy of the calibration process using the DE algorithm only.

More than just doing an example of the double Heston model, our study provides a novel idea, and (more importantly) a practical tool, in order to accelerate the calibration of financial models, by combining the AD algorithms, heuristic optimization algorithms, and gradient-based optimization methods. Such an idea can be used in many other financial models (like Monte Carlo simulations). There are several topics that can be further investigated related to our work. It might be possible to explicitly derive the analytic Jacobian matrix. This will not improve the speed of the calibration a lot, since computing the Jacobian matrix is not the most time-consuming part of the whole calibration process. However, the analytic Jacobian matrix is more stable. Another future advance could involve designing how to use the heuristic and the gradient-based methods interchangeably, multiple times, in order to achieve a faster calibration. To me, the most meaningful work would be parameters' dimension reduction. It will be perfect if we could reduce the number of parameters (to be calibrated) to the extent that the double Heston model is a convex function of all remaining parameters and will not have the problem of local minima anymore.

APPENDIX A
SELECTED CODE: THE ADEPT VERSION FOR THE DOUBLE HESTON MODEL

```
void doublehestonadept(const int& Gauss_N, const double& a, const double& b, const adouble*
    weights, const adouble* nodes, const adouble* tau, const int& tau_size, const adouble* K,
    const int& K_size, const adouble& r, const adouble& q, const adouble& S, const adouble*
    parameters, adouble* HestonC)
{
    adouble x = log(S);
    adouble kappa1 = parameters[0];
    adouble theta1 = parameters[1];
    adouble sigma1 = parameters[2];
    adouble rho1 = parameters[3];
    adouble v01 = parameters[4];
    adouble kappa2 = parameters[5];
    adouble theta2 = parameters[6];
    adouble sigma2 = parameters[7];
    adouble rho2 = parameters[8];
    adouble v02 = parameters[9];
    adouble* P2 = new adouble[tau_size * K_size];

    for(int i = 0; i < tau_size; i++)
    {
        for(int j = 0; j < K_size; j++)
        {
            P2[i * K_size + j] = 0;
        }
    }
    adouble* real_c1 = new adouble[Gauss_N + 1];
    adouble* imag_c1 = new adouble[Gauss_N + 1];
    adouble* real_c2 = new adouble[Gauss_N + 1];
    adouble* imag_c2 = new adouble[Gauss_N + 1];
    adouble* real_d1 = new adouble[Gauss_N + 1];
    adouble* imag_d1 = new adouble[Gauss_N + 1];
    adouble* real_d2 = new adouble[Gauss_N + 1];
    adouble* imag_d2 = new adouble[Gauss_N + 1];
    for(int i = 0; i < Gauss_N + 1; i++)
    {
        real_c1[i] = 0;
        imag_c1[i] = 0;
        real_c2[i] = 0;
        imag_c2[i] = 0;
        real_d1[i] = 0;
        imag_d1[i] = 0;
        real_d2[i] = 0;
        imag_d2[i] = 0;
    }


    adouble phi_R = 0.0;
    adouble phi_I = 0.0;
    adouble R_temp = 0.0;
    adouble I_temp = 0.0;
    adouble temp_top_R = 0.0;
    adouble temp_top_I = 0.0;
    adouble temp_bot_R = 0.0;
    adouble temp_bot_I = 0.0;
    for(int i = 0; i < Gauss_N + 1; ++i)
    {
        phi_R = nodes[i];
        R_temp = kappa1 * kappa1 − (rho1 * rho1 − 1) * sigma1 * sigma1 * (phi_R * phi_R − phi_I *
             phi_I) + 2 * kappa1 * rho1 * sigma1 * phi_I − sigma1 * sigma1 * phi_I;
        I_temp = −2 * (rho1 * rho1 − 1) * sigma1 * sigma1 * phi_R * phi_I + sigma1 * sigma1 *
             phi_R − 2 * kappa1 * rho1 * sigma1 * phi_R;
        real_d1[i] = real_squred_complex(R_temp, I_temp);
        imag_d1[i] = imag_squred_complex(R_temp, I_temp);
        R_temp = kappa2 * kappa2 − (rho1 * rho1 − 1) * sigma2 * sigma2 * (phi_R * phi_R − phi_I *
             phi_I) + 2 * kappa2 * rho2 * sigma2 * phi_I − sigma2 * sigma2 * phi_I;
        I_temp = −2 * (rho2 * rho2 − 1) * sigma2 * sigma2 * phi_R * phi_I + sigma2 * sigma2 *
             phi_R − 2 * kappa2 * rho1 * sigma2 * phi_R;
        real_d2[i] = real_squred_complex(R_temp, I_temp);
        imag_d2[i] = imag_squred_complex(R_temp, I_temp);

        // c
        temp_top_R = kappa1 + real_d1[i] + rho1 * sigma1 * phi_I;
        temp_top_I = imag_d1[i] − rho1 * sigma1 * phi_R;
        temp_bot_R = kappa1 − real_d1[i] + rho1 * sigma1 * phi_I;
        temp_bot_I = −imag_d1[i] − rho1 * sigma1 * phi_R;
        real_c1[i] = real_quotient_complex(temp_bot_R, temp_bot_I, temp_top_R, temp_top_I);
        imag_c1[i]= imag_quotient_complex(temp_bot_R, temp_bot_I, temp_top_R, temp_top_I);

        temp_top_R = kappa2 + real_d2[i] + rho2 * sigma2 * phi_I;
        temp_top_I = imag_d2[i] − rho2 * sigma2 * phi_R;
        temp_bot_R = kappa2 − real_d2[i] + rho2 * sigma2 * phi_I;
        temp_bot_I = −imag_d2[i] − rho2 * sigma2 * phi_R;

        real_c2[i] = real_quotient_complex(temp_bot_R, temp_bot_I, temp_top_R, temp_top_I);
        imag_c2[i] = imag_quotient_complex(temp_bot_R, temp_bot_I, temp_top_R, temp_top_I);
```

```
}
adouble* real_f = new adouble[tau_size * (Gauss_N + 1)];
adouble* imag_f = new adouble[tau_size * (Gauss_N + 1)];
for(int i = 0; i < tau_size * (Gauss_N + 1); i++)
{
    real_f[i] = 0;
    imag_f[i] = 0;
}
adouble a1 = 0.0;
adouble l1 = 0.0;
adouble a2 = 0.0;
adouble l2 = 0.0;
adouble B_real_temp = 0.0;
adouble B_imag_temp = 0.0;
adouble real_B1 = 0.0;
adouble imag_B1 = 0.0;
adouble real_B2 = 0.0;
adouble imag_B2 = 0.0;
adouble G_top_real = 0.0;
adouble G_top_imag = 0.0;
adouble G_bot_real = 0.0;
adouble G_bot_imag = 0.0;
adouble real_G1 = 0.0;
adouble imag_G1 = 0.0;
adouble real_logG1 = 0.0;
adouble imag_logG1 = 0.0;
adouble real_G2 = 0.0;
adouble imag_G2 = 0.0;
adouble real_logG2 = 0.0;
adouble imag_logG2 = 0.0;
adouble real_A1 = 0.0;
adouble imag_A1 = 0.0;
adouble real_A2 = 0.0;
adouble imag_A2 = 0.0;
adouble real_A = 0.0;
adouble imag_A = 0.0;
adouble real_exp = 0.0;
adouble imag_exp = 0.0;

for(int i = 0; i < tau_size; i++)
{
    for(int j = 0; j < Gauss_N + 1; j++)
    {
        phi_R = nodes[j];
        // B1
        a1 = -real_d1[j] * tau[i];
        l1 = -imag_d1[j] * tau[i];
        temp_top_R = 1.0 - cos(l1) * exp(a1);
        temp_top_I = -sin(l1) * exp(a1);
        temp_bot_R = 1.0 - real_c1[j] * exp(a1) * cos(l1) + imag_c1[j] * exp(a1) * sin(l1);
        temp_bot_I = -real_c1[j] * sin(l1) * exp(a1)  - imag_c1[j] * cos(l1) * exp(a1);
        B_real_temp = real_quotient_complex(temp_top_R, temp_top_I, temp_bot_R, temp_bot_I);
        B_imag_temp = imag_quotient_complex(temp_top_R, temp_top_I, temp_bot_R, temp_bot_I);
        real_B1 = 1 / sigma1 / sigma1 * ((kappa1 + rho1 * sigma1 * phi_I - real_d1[j]) *
            B_real_temp + (imag_d1[j] + rho1 * sigma1 * phi_R) * B_imag_temp);
        imag_B1 = 1 / sigma1 / sigma1 * ((kappa1 + rho1 * sigma1 * phi_I - real_d1[j]) *
            B_imag_temp + (-imag_d1[j] - rho1 * sigma1 * phi_R) * B_real_temp);

        // B2
        a2 = -real_d2[j] * tau[i];
        l2 = -imag_d2[j] * tau[i];
        temp_top_R = 1.0 - cos(l2) * exp(a2);
        temp_top_I = -sin(l2) * exp(a2);
        temp_bot_R = 1.0 - real_c2[j] * exp(a2) * cos(l2) + imag_c2[j] * exp(a2) * sin(l2);
        temp_bot_I = -real_c2[j] * sin(l2) * exp(a2) - imag_c2[j] * cos(l2) * exp(a2);
        B_real_temp = real_quotient_complex(temp_top_R, temp_top_I, temp_bot_R, temp_bot_I);
        B_imag_temp = imag_quotient_complex(temp_top_R, temp_top_I, temp_bot_R, temp_bot_I);
        real_B2 = ((kappa2 + rho2 * sigma2 * phi_I - real_d2[j]) * B_real_temp + (imag_d2[j] +
             rho2 * sigma2 * phi_R) * B_imag_temp) / sigma2 / sigma2;
        imag_B2 = ((kappa2 + rho2 * sigma2 * phi_I - real_d2[j]) * B_imag_temp + (-imag_d2[j]
            - rho2 * sigma2 * phi_R) * B_real_temp) / sigma2 / sigma2;
        G_top_real = 1 - real_c1[j] * exp(a1) * cos(l1) + imag_c1[j] * exp(a1) * sin(l1);
        G_top_imag = -real_c1[j] * sin(l1) * exp(a1)  - imag_c1[j] * cos(l1) * exp(a1);
        G_bot_real = 1 - real_c1[j];
        G_bot_imag = -imag_c1[j];
        real_G1 = real_quotient_complex(G_top_real, G_top_imag, G_bot_real, G_bot_imag);
        imag_G1 = imag_quotient_complex(G_top_real, G_top_imag, G_bot_real, G_bot_imag);
        real_logG1 = real_log_complex(real_G1, imag_G1);
        imag_logG1 = imag_log_complex(real_G1, imag_G1);

        // log(G2) = ln[(1.00 - c * exp(-d * T)) / (1.00 - c)];
        G_top_real = 1 - real_c2[j] * exp(a2) * cos(l2) + imag_c2[j] * exp(a2) * sin(l2);
        G_top_imag = -real_c2[j] * sin(l2) * exp(a2)  - imag_c2[j] * cos(l2) * exp(a2);
        G_bot_real = 1 - real_c2[j];
        G_bot_imag = -imag_c2[j];
        real_G2 = real_quotient_complex(G_top_real, G_top_imag, G_bot_real, G_bot_imag);
        imag_G2 = imag_quotient_complex(G_top_real, G_top_imag, G_bot_real, G_bot_imag);
        real_logG2 = real_log_complex(real_G2, imag_G2);
        imag_logG2 = imag_log_complex(real_G2, imag_G2);

```

```
            // A_1
            real_A1 = (kappa1 + rho1 * sigma1 * phi_I - real_d1[j]) * tau[i] - 2 * real_logG1;
            imag_A1 = (-imag_d1[j] - rho1 * sigma1 * phi_R) * tau[i] - 2 * imag_logG1;

            // A_2
            real_A2 = (kappa2 + rho2 * sigma2 * phi_I - real_d2[j]) * tau[i] - 2 * real_logG2;
            imag_A2 = (-imag_d2[j] - rho2 * sigma2 * phi_R) * tau[i] - 2 * imag_logG2;
            // A
            real_A = -(r - q) * phi_I * tau[i] + (real_A1 * kappa1 * theta1 / sigma1 / sigma1 +
                real_A2 * kappa2 * theta2 / sigma2 / sigma2);
            imag_A = (r - q) * phi_R * tau[i] + (imag_A1 * kappa1 * theta1 / sigma1 / sigma1 +
                imag_A2 * kappa2 * theta2 / sigma2 / sigma2);
            real_exp = real_A - x * phi_I + real_B1 * v01 + real_B2 * v02;
            imag_exp = imag_A + x * phi_R + imag_B1 * v01 + imag_B2 * v02;
            real_f[i * (Gauss_N + 1) + j] = exp(real_exp) * cos(imag_exp);
            imag_f[i * (Gauss_N + 1) + j] = exp(real_exp) * sin(imag_exp);
        }
    }
    adouble phi = 0.0;
    adouble temp = 0.0;
    adouble lnk = 0.0;
    for(int i = 0; i < tau_size; i++)
    {
        for(int k = 0; k < K_size; k++)
        {
            lnk = log(K[k]);
            for(int j = 0; j < (Gauss_N + 1); j++)
            {
                phi = nodes[j];
                temp_top_R = cos(phi * lnk) * real_f[i * (Gauss_N + 1) + j] + sin(phi * lnk) *
                    imag_f[i * (Gauss_N + 1) + j];
                temp_top_I = cos(phi * lnk) * imag_f[i * (Gauss_N + 1) + j] - sin(phi * lnk) *
                    real_f[i * (Gauss_N + 1) + j];
                temp_bot_R = 0.0;
                temp_bot_I = phi;
                temp = real_quotient_complex(temp_top_R, temp_top_I, temp_bot_R, temp_bot_I) *
                    weights[j];
                P2[i * K_size + k] = P2[i * K_size + k] + temp;
            }
            P2[i * K_size + k] = (b - a) / 2.0 * P2[i * K_size + k];
            P2[i * K_size + k] = 0.50 + P2[i * K_size + k] / M_PI;
        }
    }
    adouble* P1 = new adouble[tau_size * K_size];
    for(int i = 0; i < tau_size; i++)
    {
        for(int j = 0; j < K_size; j++)
        {
            P1[i * K_size + j] = 0;
        }
    }
    //some variables for future use: to store temporary value
    phi_I = -1.0000;
    //find c_j and d_j, which are not related to tau and K
    for(int i = 0; i < Gauss_N + 1; ++i)
    {
        // d_j
        phi_R = nodes[i];
        R_temp = kappa1 * kappa1 - (rho1 * rho1 - 1) * sigma1 * sigma1 * (phi_R * phi_R - phi_I *
             phi_I) + 2 * kappa1 * rho1 * sigma1 * phi_I - sigma1 * sigma1 * phi_I;
        I_temp = -2 * (rho1 * rho1 - 1) * sigma1 * sigma1 * phi_R * phi_I + sigma1 * sigma1 *
             phi_R - 2 * kappa1 * rho1 * sigma1 * phi_R;
        real_d1[i] = real_squred_complex(R_temp, I_temp);
        imag_d1[i] = imag_squred_complex(R_temp, I_temp);
        R_temp = kappa2 * kappa2 - (rho1 * rho1 - 1) * sigma2 * sigma2 * (phi_R * phi_R - phi_I *
             phi_I)
        + 2 * kappa2 * rho2 * sigma2 * phi_I - sigma2 * sigma2 * phi_I;
        I_temp = -2 * (rho2 * rho2 - 1) * sigma2 * sigma2 * phi_R * phi_I + sigma2 * sigma2 *
             phi_R - 2 * kappa2 * rho1 * sigma2 * phi_R;
        real_d2[i] = real_squred_complex(R_temp, I_temp);
        imag_d2[i] = imag_squred_complex(R_temp, I_temp);
        // c_j
        temp_top_R = kappa1 + real_d1[i] + rho1 * sigma1 * phi_I;
        temp_top_I = imag_d1[i] - rho1 * sigma1 * phi_R;
        temp_bot_R = kappa1 - real_d1[i] + rho1 * sigma1 * phi_I;
        temp_bot_I = -imag_d1[i] - rho1 * sigma1 * phi_R;
        real_c1[i] = real_quotient_complex(temp_bot_R, temp_bot_I, temp_top_R, temp_top_I);
        imag_c1[i]= imag_quotient_complex(temp_bot_R, temp_bot_I, temp_top_R, temp_top_I);

        temp_top_R = kappa2 + real_d2[i] + rho2 * sigma2 * phi_I;
        temp_top_I = imag_d2[i] - rho2 * sigma2 * phi_R;
        temp_bot_R = kappa2 - real_d2[i] + rho2 * sigma2 * phi_I;
        temp_bot_I = -imag_d2[i] - rho2 * sigma2 * phi_R;
        real_c2[i] = real_quotient_complex(temp_bot_R, temp_bot_I, temp_top_R, temp_top_I);
        imag_c2[i] = imag_quotient_complex(temp_bot_R, temp_bot_I, temp_top_R, temp_top_I);
    }

    for(int i = 0; i < tau_size; i++)
    {
        for(int j = 0; j < Gauss_N + 1; j++)
```

```
{
    phi_R = nodes[j];

    // B1
    a1 = -real_d1[j] * tau[i];
    l1 = -imag_d1[j] * tau[i];
    temp_top_R = 1.0 - cos(l1) * exp(a1);
    temp_top_I = -sin(l1) * exp(a1);
    temp_bot_R = 1.0 - real_c1[j] * exp(a1) * cos(l1) + imag_c1[j] * exp(a1) * sin(l1);
    temp_bot_I = -real_c1[j] * sin(l1) * exp(a1)  - imag_c1[j] * cos(l1) * exp(a1);
    B_real_temp = real_quotient_complex(temp_top_R, temp_top_I, temp_bot_R, temp_bot_I);
    B_imag_temp = imag_quotient_complex(temp_top_R, temp_top_I, temp_bot_R, temp_bot_I);
    real_B1 = 1 / sigma1 / sigma1 * ((kappa1 + rho1 * sigma1 * phi_I - real_d1[j]) *
         B_real_temp + (imag_d1[j] + rho1 * sigma1 * phi_R) * B_imag_temp);
    imag_B1 = 1 / sigma1 / sigma1 * ((kappa1 + rho1 * sigma1 * phi_I - real_d1[j]) *
         B_imag_temp + (-imag_d1[j] - rho1 * sigma1 * phi_R) * B_real_temp);

    // B2
    a2 = -real_d2[j] * tau[i];
    l2 = -imag_d2[j] * tau[i];
    temp_top_R = 1.0 - cos(l2) * exp(a2);
    temp_top_I = -sin(l2) * exp(a2);
    temp_bot_R = 1.0 - real_c2[j] * exp(a2) * cos(l2) + imag_c2[j] * exp(a2) * sin(l2);
    temp_bot_I = -real_c2[j] * sin(l2) * exp(a2) - imag_c2[j] * cos(l2) * exp(a2);
    B_real_temp = real_quotient_complex(temp_top_R, temp_top_I, temp_bot_R, temp_bot_I);
    B_imag_temp = imag_quotient_complex(temp_top_R, temp_top_I, temp_bot_R, temp_bot_I);
    real_B2 = ((kappa2 + rho2 * sigma2 * phi_I - real_d2[j]) * B_real_temp + (imag_d2[j] +
          rho2 * sigma2 * phi_R) * B_imag_temp) / sigma2 / sigma2;
    imag_B2 = ((kappa2 + rho2 * sigma2 * phi_I - real_d2[j]) * B_imag_temp  + (-imag_d2[j]
          - rho2 * sigma2 * phi_R) * B_real_temp) / sigma2 / sigma2;

    //log(G1)
    G_top_real = 1 - real_c1[j] * exp(a1) * cos(l1) + imag_c1[j] * exp(a1) * sin(l1);
    G_top_imag = -real_c1[j] * sin(l1) * exp(a1)  - imag_c1[j] * cos(l1) * exp(a1);
    G_bot_real = 1 - real_c1[j];
    G_bot_imag = -imag_c1[j];
    real_G1 = real_quotient_complex(G_top_real, G_top_imag, G_bot_real, G_bot_imag);
    imag_G1 = imag_quotient_complex(G_top_real, G_top_imag, G_bot_real, G_bot_imag);
    real_logG1 = real_log_complex(real_G1, imag_G1);
    imag_logG1 = imag_log_complex(real_G1, imag_G1);

    // log(G2) = ln[(1.00 - c * exp(-d * T)) / (1.00 - c)];
    G_top_real = 1 - real_c2[j] * exp(a2) * cos(l2) + imag_c2[j] * exp(a2) * sin(l2);
    G_top_imag = -real_c2[j] * sin(l2) * exp(a2)  - imag_c2[j] * cos(l2) * exp(a2);
    G_bot_real = 1 - real_c2[j];
```

```
            G_bot_imag = -imag_c2[j];
            real_G2 = real_quotient_complex(G_top_real, G_top_imag, G_bot_real, G_bot_imag);
            imag_G2 = imag_quotient_complex(G_top_real, G_top_imag, G_bot_real, G_bot_imag);
            real_logG2 = real_log_complex(real_G2, imag_G2);
            imag_logG2 = imag_log_complex(real_G2, imag_G2);

            // A_1
            real_A1 = (kappa1 + rho1 * sigma1 * phi_I - real_d1[j]) * tau[i] - 2 * real_logG1;
            imag_A1 = (-imag_d1[j] - rho1 * sigma1 * phi_R) * tau[i] - 2 * imag_logG1;

            // A_2
            real_A2 = (kappa2 + rho2 * sigma2 * phi_I - real_d2[j]) * tau[i] - 2 * real_logG2;
            imag_A2 = (-imag_d2[j] - rho2 * sigma2 * phi_R) * tau[i] - 2 * imag_logG2;

            // A
            real_A = -(r - q) * phi_I * tau[i] + (real_A1 * kappa1 * theta1 / sigma1 / sigma1 +
                real_A2 * kappa2 * theta2 / sigma2 / sigma2);
            imag_A = (r - q) * phi_R * tau[i] + (imag_A1 * kappa1 * theta1 / sigma1 / sigma1  +
                imag_A2 * kappa2 * theta2 / sigma2 / sigma2);
            real_exp = real_A - x * phi_I + real_B1 * v01 + real_B2 * v02;
            imag_exp = imag_A + x * phi_R + imag_B1 * v01 + imag_B2 * v02;
            real_f[i * (Gauss_N + 1) + j] = exp(real_exp) * cos(imag_exp);
            imag_f[i * (Gauss_N + 1) + j] = exp(real_exp) * sin(imag_exp);
        }
    }

    for(int i = 0; i < tau_size; i++)
    {
        for(int k = 0; k < K_size; k++)
        {
            lnk = log(K[k]);
            for(int j = 0; j < Gauss_N + 1; j++)
            {
                phi = nodes[j];
                temp_top_R = cos(phi * lnk) * real_f[i * (Gauss_N + 1) + j] + sin(phi * lnk) *
                    imag_f[i * (Gauss_N + 1) + j];
                temp_top_I = cos(phi * lnk) * imag_f[i * (Gauss_N + 1) + j] - sin(phi * lnk) *
                    real_f[i * (Gauss_N + 1) + j];
                temp_bot_R = 0.0;
                temp_bot_I = phi * S * exp((r - q) * tau[i]);
                temp = real_quotient_complex(temp_top_R, temp_top_I, temp_bot_R, temp_bot_I) *
                    weights[j];
                P1[i * K_size + k] = P1[i * K_size + k] + temp;
            }
            P1[i * K_size + k] = (b - a) / 2.0 * P1[i * K_size + k];
```

```
                P1[i * K_size + k] = 0.50 + P1[i * K_size + k] / M_PI;
            }
        }

        for(int i = 0; i < tau_size; i++)
        {
            for(int j = 0; j < K_size; j++)
            {
                HestonC[i * K_size + j] = S * exp(-q * tau[i]) * P1[i * K_size + j] - K[j] * exp(-r *
                     tau[i]) * P2[i * K_size + j];
            }
        }
        delete[] real_c1;
        delete[] imag_c1;
        delete[] real_c2;
        delete[] imag_c2;
        delete[] real_d1;
        delete[] imag_d1;
        delete[] real_d2;
        delete[] imag_d2;
        delete[] real_f;
        delete[] imag_f;
        delete[] P1;
        delete[] P2;
    }
```

APPENDIX B
SELECTED CODE: THE LEVENBERG-MARQUARDT ALGORITHM

```
void make_matrix_from_pointer(matrix<double>& A, double* a, int row, int col)
{
    for(int i = 0; i < row; i++)
    {
        for(int j = 0; j < col; j++)
        {
            A(i,j) = a[i * col + j];
        }
    }
}

double norm_2(const matrix<double>& A, int col)
{
    double sum = 0.0;
    for(int i = 0; i < col; i++)
    {
        sum = sum + A(i, 0) * A(i, 0);
    }
    return sum;
}

void function_fvec(const double* parameters, matrix<double>& fi)
{
    double* result = doubleheston(set_Gauss_N, set_a, set_b, set_weights, set_nodes, set_tau,
         set_tau_size, set_K, set_K_size, set_r, set_q, set_S, parameters);
    for(int i = 0; i < set_tau_size; i++)
    {
        for(int j = 0; j < set_K_size; j++)
        {
            fi(i * set_K_size + j, 0) = result[i * set_K_size + j] - set_MktPri[i * set_K_size + j
                ];
        }
    }
    delete[] result;
    result = NULL;
}

void function_jac_Ad(const double* parameter, matrix<double>& fi, matrix<double> &jac)
{
    B<double>* parameters = new B<double>[10];
    for(int i = 0; i < 10; i++)
    {
        parameters[i] = parameter[i];
    }
    B<double>* result = ADdoubleheston(AD_Gauss_N, AD_a, AD_b, AD_weights, AD_nodes, AD_tau,
         AD_tau_size, AD_K, AD_K_size, AD_r, AD_q, AD_S, parameters);
    for(int i = 0; i < AD_tau_size; i++)
    {
        for(int j = 0; j < AD_K_size; j++)
        {
            fi(i * set_K_size + j, 0) = result[i * set_K_size + j].x() - set_MktPri[i * set_K_size
                + j];
        }
    }
    for(int i = 0; i < AD_tau_K_size; i++)
    {
        result[i].diff(i, AD_tau_K_size);
    }
    for(int i = 0; i < AD_tau_K_size; i++)
    {
        for(int j = 0; j < 10; j++)
        {
            jac(i, j) = parameters[j].d(i);
            if(!(jac(i,j) == jac(i,j)))
            {
               jac(i, j) = 1;
            }
           cout << jac(i, j) << " ";
        }
       cout<<endl;
    }
    delete[] parameters;
    parameters = NULL;
    delete[] result;
    result = NULL;
}

void function_jac_Adep(adept::Stack& stack,const double* parameter, matrix<double>& fi, matrix<
     double> &jac)
{
    double* jacc = new double[set_tau_K_size * 10];
    adouble adept_S;
    adouble adept_r;
    adouble adept_q;
    adept_S.set_value(set_S);
    adept_q.set_value(set_q);
    adept_r.set_value(set_r);
    std::vector<adouble> adept_nodes(set_Gauss_N + 1);
    adept::set_values(&adept_nodes[0], set_Gauss_N + 1, set_nodes);
    std::vector<adouble> adept_weights(set_Gauss_N + 1);
    adept::set_values(&adept_weights[0], set_Gauss_N + 1, set_weights);
    std::vector<adouble> adept_tau(set_tau_size);
    adept::set_values(&adept_tau[0], set_tau_size, set_tau);
    std::vector<adouble> adept_K(set_K_size);
    adept::set_values(&adept_K[0], set_K_size, set_K);
    std::vector<adouble> input(10); // Vector of active input variables
    adept::set_values(&input[0], 10, parameter); // Initialize adouble inputs
    std::vector<adouble> y(set_tau_K_size); // Create vector of active output variables
    stack.new_recording();
    doublehestonadept(set_Gauss_N, set_a, set_b, &adept_weights[0], &adept_nodes[0],
                      &adept_tau[0], set_tau_size, &adept_K[0], set_K_size,
                      adept_r, adept_q, adept_S, &input[0], &y[0]);
    stack.independent(&input[0], 10); // Identify independent variables
    stack.dependent(&y[0], set_tau_K_size); // Identify dependent variables
    stack.jacobian(jacc); // Compute & store Jacobian in jac
    for(int i = 0; i < set_tau_size; i++)
    {
        for(int j = 0; j < set_K_size; j++)
        {
            fi(i * set_K_size + j, 0) = y[i * set_K_size + j].value();
        }
    }
    for(int i = 0; i < set_tau_K_size; i++)
    {
        for(int j = 0; j < 10; j++)
        {
            jac(i, j) = jacc[j * set_tau_K_size + i];
            if(!(jac(i,j) == jac(i,j)))
            {
                jac(i, j) = 1;
            }
        }
    }
    delete[] jacc;
}

bool InvertMatrix(const matrix<double>& input, matrix<double>& inverse)
{
    typedef permutation_matrix<std::size_t> pmatrix;
    // create a working copy of the input
    matrix<double> A(input);
    // create a permutation matrix for the LU-factorization
    pmatrix pm(A.size1());
    // perform LU-factorization
    double res = lu_factorize(A, pm);
    if (res != 0)
    {
        cout << "no inverse" << endl;
        return false;
    }
    // create identity matrix of "inverse"
    inverse.assign(identity_matrix<double> (A.size1()));
    // backsubstitute to get the inverse
    lu_substitute(A, pm, inverse);
    return true;
}

double LM(adept::Stack& stack, double* parameters) // parameters passed by reference
{
    clock_t start = 0;
    clock_t end = 0;
```

```
double  Jacobian_time = 0.0;
int count = 1;
//initialization
double* parameters_new = new double[10];
matrix<double> solution(10, 1); // 10 X 1 matrix
matrix<double> solution_new(10, 1);// 10 X 1 matrix
make_matrix_from_pointer(solution, parameters, 10, 1);

int maxI = 800;
double e1 = 0.0004;
double e2 = 1e-9;
double e3 = 1e-5;
double temp = 0.0;
double condition1 = 10.0; // store the value of ||r||
double condition2 = 10.0; // store ||delta theta|| / theta
double mu = 1;
const double mu_up = 2;
const double mu_down = 0.5;

bool I_flag = 1; // check if (J^T * J + mu * I) has inverse

// r
matrix<double> r(set_tau_K_size, 1);
matrix<double> r_new(set_tau_K_size, 1);
double r_norm_new = 0.0;

//J and J^T
matrix<double> J_T(set_tau_K_size, 10); //

//function_jac_Adep(stack, parameters, r, J_T);
function_jac_Ad(parameters, r, J_T); // find J_T
matrix<double> J = trans(J_T);// 10 X 16 matrix
double r_norm = norm_2(r, set_tau_K_size);
identity_matrix<double> I(10); // 10 X 10
matrix<double> temp_I(10, 10);
matrix<double> grad_f(set_tau_K_size, 1);
matrix<double> change(10, 1);
double delta_F = 0.0;
for(int k = 0; k < maxI; k++)
{
    condition1 = r_norm / set_tau_K_size;
    if(condition1 < e1)
    {
        cout << "Exist with condition 1" << endl;
        cout << "Iterations = " << k << endl;
        cout << "LM reduced the error to: "<< r_norm / set_tau_K_size << endl;
        cout << "Jacobian evaluations = " << count << endl;
        cout << "jacobian time = " <<(double)Jacobian_time / CLOCKS_PER_SEC * 1000.0 << endl;
        for(int i = 0; i < 10; i++)
        {
            cout << solution(i,0) << " ";
        }
        cout << endl;
        return r_norm / set_tau_K_size;
    }
    Inverse:
    I_flag = InvertMatrix(prod(J, J_T) + mu * I, temp_I);
    if(!I_flag)
    {
        cout<< " no inverse with mu = " << mu << endl;
        mu = mu * mu_up;
        goto Inverse;
    }
    grad_f = 1 * prod(J, r); // 10 X 1;
    change = -prod(temp_I, grad_f); // 10 X 1
    condition2 = norm_2(change, 10) / norm_2(solution, 10);
    if(condition2 < e2)
    {
        cout << "Iterations = " << k << endl;
        cout << "LM reduced the error to: "<< r_norm / set_tau_K_size << endl;
        cout << "Jacobian evaluations = " << count << endl;
        cout<< "Exist with condition 2 and mu = " << mu << endl;
        cout << "jacobian time = " << (double)Jacobian_time / CLOCKS_PER_SEC * 1000.0<< endl;
        cout << "parameters = ";
        for(int i = 0; i < 10; i++)
        {
            cout << solution(i,0) << " ";
        }
        cout << endl;
        return r_norm / set_tau_K_size;
    }
    solution_new = solution + change;
    for(int i = 0; i < 10; i++)
    {
        if(solution_new(i,0) > boundright[i])
        {
            solution_new(i,0) = boundright[i];
        }
        if(solution_new(i,0) < boundleft[i])
        {
            solution_new(i,0) = boundleft[i];
        }
    }

    for(int i = 0; i < 10; i++)
    {
        parameters_new[i] = solution_new(i, 0);
    }
    function_fvec(parameters_new, r_new);
    r_norm_new = norm_2(r_new, set_tau_K_size);
    delta_F = r_norm - r_norm_new;
    if(delta_F > 0)
    {
        start = clock();
        //function_jac_Adep(stack, parameters_new, r, J_T);
        function_jac_Ad(parameters_new, r, J_T);
        for(int row = 0;  row < set_tau_K_size; row++)
        {
            for(int col = 0; col < 10; col++)
            {
                temp = temp + abs(J_T(row, col));
            }
            if(temp < e3)
            {
                cout << "Iterations = " << k << endl;
                cout << "LM reduced the error to: "<< r_norm / set_tau_K_size << endl;
                cout << "Jacobian evaluations = " << count << endl;
                cout<< "Exist with condition 3 and infity norm = " << temp << endl;
                cout << "jacobian time = " << (double)Jacobian_time / CLOCKS_PER_SEC *
                     1000.0<< endl;
                cout << "parameters = ";
                for(int i = 0; i < 10; i++)
                {
                    cout << solution(i,0) << " ";
                }
                cout << endl;
                return r_norm / set_tau_K_size;
            }
            else
            {
                temp = 0;
            }
        }
        end = clock();
        mu = mu * mu_down;
        for(int i = 0; i < 10; i++)
        {
            parameters[i] = parameters_new[i];
            solution(i,0) = parameters_new[i];
        }
        r_norm = r_norm_new;
        cout<< count << ": error =  "<< r_norm / set_tau_K_size << endl;
        count = count + 1;
        Jacobian_time = Jacobian_time + (double)end - (double)start;
        cout << "jacobian time = " << (double)Jacobian_time / CLOCKS_PER_SEC * 1000.0 << endl;
    }
    else
    {
        mu = mu * mu_up;
    }
}
delete[] parameters_new;
return r_norm / set_tau_K_size;
}
```

APPENDIX C
SELECTED CODE: THE DE ALGORITHM

```
double* DE(const int& Gauss_N, const double& a, const double& b, const double* weights, const
    double* nodes, const double* tau, const int& tau_size, const double* K, const int& K_size,
    const double* data, const double& r, const double& q, const double& S)
{
    //Differential Algorithm setting
    int dim_s = 10; // dimension of solution
    int G_max = 2000; // Generations
    int totalNP = 150; // population members
    double CR = 0.7; // croseover ratio
    double CR_min = 0.7;
    double CR_max = 0.7;
    double F = 0.8; // Threshold
    double F_min = 0.8;
    double F_max = 0.8;
    double error = 0.2;
    double** population = new double*[totalNP];
    double** population_new = new double*[totalNP];
    double* best = new double[dim_s];
    for(int i = 0; i < totalNP; ++i)
    {
        population_new[i] = new double[dim_s];
        population[i] = new double[dim_s];
```

```
}
double value_new = 0.0;
double value_old = 0.0;
int a1 = 0;
int a2 = 0;
int a3 = 0;
double temp = 0.0;
int temp_dim = 0;
std::random_device rand_dev;
std::mt19937 re(rand_dev());
uniform_real_distribution<double> unif(0, 1);
uniform_int_distribution<int> unif_int_NP(0, totalNP - 1);
uniform_int_distribution<int> unif_int_dimension(0, dim_s - 1);
for(int i = 0; i < totalNP; i++)
{
    for(int j = 0; j < dim_s; j++)
    {
        population[i][j] = boundleft[j] + unif(re) * (boundright[j] - boundleft[j]);
        population_new[i][j] = -1;
    }
}
 for(int k = 0; k < G_max; k++)
{
    F = F_min + (1 - (double)k / (double)G_max) * (F_max - F_min);
    CR = CR_min + (1 - (double)k / (double)G_max) * (CR_max-CR_min);
    for(int i = 0; i < totalNP; i++)
    {
        a1 = unif_int_NP(re);
        a2 = unif_int_NP(re);
        a3 = unif_int_NP(re);
        while (a1 == a2 || a1 == a3 || a2 == a3)
        {
            a1 = unif_int_NP(re);
            a2 = unif_int_NP(re);
            a3 = unif_int_NP(re);
        }
        temp_dim = unif_int_dimension(re);
        for(int j = 0; j < dim_s; j++)
        {
            temp = unif(re);
            if(temp <= CR || j == temp_dim)
                population_new[i][j] = population[a1][j] + F * (population[a2][j] - population
                    [a3][j]);
            else
                population_new[i][j] = population[i][j];
        }

        for(int j = 0; j< dim_s; j++)
        {
            if(population_new[i][j] > boundright[j])
            {
                population_new[i][j] = boundright[j] + unif(re) * (population[i][j]-
                    boundright[j]);
            }
            if(population_new[i][j] < boundleft[j])
            {
                population_new[i][j] = boundleft[j] + unif(re) * (population[i][j] - boundleft
                    [j]);
            }
        }
    }

    for(int i = 0; i < totalNP; i++)
    {
        value_new = fin(Gauss_N,  a,  b, weights, nodes,
                        tau, tau_size, K, K_size, data,
                        r, q, S,
                        population_new[i]);

        value_old = fin(Gauss_N,  a,  b, weights, nodes,
                        tau, tau_size, K, K_size, data,
                        r, q, S, population[i]);
        if(value_new < value_old)
        {
            if(value_new < error)
            {
                for(int s = 0; s < dim_s; s++)
                {
                    best[s] = population_new[i][s];
                }
                cout << "(break)The error of DE is equal to: " << fin(Gauss_N,  a,  b, weights
                    , nodes, tau, tau_size, K, K_size, data, r, q, S, best) <<endl;
                cout << "Iteration = " << k << endl;
                return best;
            }
            for(int j = 0; j < dim_s; j++)
            {
                population[i][j] = temp;
                population[i][j] = population_new[i][j];
                population_new[i][j] = temp;
                }
            }
        }
    }
    double* value = new double[totalNP];
    for(int i = 0; i < totalNP; i++)
    {
        value[i] = fin(Gauss_N, a, b, weights, nodes, tau, tau_size, K, K_size, data, r, q, S,
            population[i]);
    }
    int best_index = 0;
    for(int i = 0; i < totalNP; i++)
    {
        if(value[i] < value[best_index])
        {
            best_index = i;
        }
    }
    for(int i = 0; i < dim_s; i++)
    {
        best[i] = population[best_index][i];
    }
    cout << "The error of DE is equal to: " << fin(Gauss_N, a, b, weights, nodes, tau, tau_size,
        K, K_size, data, r, q, S, population[best_index]) <<endl;
    for(int i = 0; i < totalNP; i++)
    {
        delete[] population[i];
        delete[] population_new[i];
    }
    delete[] population;
    delete[] population_new;
    return best;
}
```

REFERENCES

[1] Hansjörg Albrecher, Philipp Mayer, Wim Schoutens, and Jurgen Tistaert. The little heston trap. *Wilmott Magazine*, pages 83–92, 2007.

[2] Mukarram Attari. Option pricing using fourier transforms: A numerically efficient simplification. *Available at SSRN 520042*, 2004.

[3] Claus Bendtsen and Ole Stauning. Fadbad, a flexible c++ package for automatic differentiation. *Department of Mathematical Modelling, Technical University of Denmark*, 1996.

[4] Nicholas H Bingham and Rüdiger Kiesel. *Risk-neutral valuation: Pricing and hedging of financial derivatives.* Springer Science & Business Media, 2013.

[5] Christian H Bischof, Martin H Bucker, Paul Hovland, Uwe Naumann, and Jean Utke. *Advances in Automatic Differentiation. Lecture Notes in Computational Science and Engineering.* Springer, 2008.

[6] Fischer Black and Myron Scholes. The pricing of options and corporate liabilities. *The journal of political economy*, pages 637–654, 1973.

[7] Douglas T Breeden. An intertemporal asset pricing model with stochastic consumption and investment opportunities. *Journal of financial Economics*, 7(3):265–296, 1979.

[8] Luca Capriotti, Shinghoi Jacky Lee, and Matthew Peacock. Real time counterparty credit risk management in monte carlo. *Risk Magazine (in press)*, 2011.

[9] Peter Carr and Dilip Madan. Option valuation using the fast fourier transform. *Journal of computational finance*, 2(4):61–73, 1999.

[10] Kyriakos Chourdakis. Option pricing using the fractional fft. *Journal of Computational Finance*, 8(2):1–18, 2004.

[11] Peter Christoffersen, Steven Heston, and Kris Jacobs. The shape and term structure of the index option smirk: Why multifactor stochastic volatility models work so well. *Management Science*, 55(12):1914–1932, 2009.

[12] John C Cox, Jonathan E Ingersoll Jr, and Stephen A Ross. A theory of the term structure of interest rates. *Econometrica: Journal of the Econometric Society*, pages 385–407, 1985.

[13] Yiran Cui, Sebastian del Bano Rollin, and Guido Germano. Full and fast calibration of the heston stochastic volatility model. *arXiv preprint arXiv:1511.08718*, 2015.

[14] Darrell Duffie, Jun Pan, and Kenneth Singleton. Transform analysis and asset pricing for affine jump-diffusions. *Econometrica*, 68(6):1343–1376, 2000.

[15] Willliam Feller. *An introduction to probability theory and its applications (Vol.2)*, volume 2. Wiley, New York, 1966.

[16] Andreas Griewank and Andrea Walther. *Evaluating derivatives: principles and techniques of algorithmic differentiation*. Siam, 2008.

[17] Steven L Heston. A closed-form solution for options with stochastic volatility with applications to bond and currency options. *Review of financial studies*, 6(2):327–343, 1993.

[18] Robin J Hogan. Fast reverse-mode automatic differentiation using expression templates in c++. *ACM Transactions on Mathematical Software (TOMS)*, 40(4):26, 2014.

[19] Cristian Homescu. Adjoints and automatic (algorithmic) differentiation in computational finance. *Available at SSRN 1828503*, 2011.

[20] John Hull and Alan White. The pricing of options on assets with stochastic volatilities. *The journal of finance*, 42(2):281–300, 1987.

[21] Kiyosi Itô. 109. stochastic integral. *Proceedings of the Imperial Academy*, 20(8):519–524, 1944.

[22] René Jursa and Kurt Rohrig. Short-term wind power forecasting using evolutionary algorithms for the automated specification of artificial intelligence models. *International Journal of Forecasting*, 24(4):694–709, 2008.

[23] Andy Keane and Prasanth Nair. *Computational approaches for aerospace design: the pursuit of excellence*. John Wiley & Sons, 2005.

[24] Maurice G Kendall, Alan Stuart, and JK Ord. *The advanced theory of statistics*, volume 3. London, 1968.

[25] MH Khademi, MR Rahimpour, and A Jahanmiri. Differential evolution (de) strategy for optimization of hydrogen production, cyclohexane dehydrogenation and methanol synthesis in a hydrogen-permselective membrane thermally coupled reactor. *international journal of hydrogen energy*, 35(5):1936–1950, 2010.

[26] Fiodar Kilin. Accelerating the calibration of stochastic volatility models. 2006.

[27] Fima C Klebaner et al. *Introduction to stochastic calculus with applications*, volume 57. World Scientific, 2005.

[28] David G Luenberger. *Introduction to linear and nonlinear programming*, volume 28. Addison-Wesley Reading, MA, 1973.

[29] Thomas Mikosch. *Elementary stochastic calculus with finance in view.* World scientific, 1998.

[30] Jorge J Moré. The levenberg-marquardt algorithm: implementation and theory. In *Numerical analysis*, pages 105–116. Springer, 1978.

[31] Jorge Nocedal and Stephen Wright. *Numerical optimization.* Springer Science & Business Media, 2006.

[32] Bernt Oksendal. *Stochastic differential equations: an introduction with applications.* Springer Science & Business Media, 2013.

[33] Eckhard Platen and Nicola Bruti-Liberati. *Numerical solution of stochastic differential equations with jumps in finance*, volume 64. Springer Science & Business Media, 2010.

[34] Fabrice D Rouah. *The Heston Model and its Extensions in Matlab and C.* John Wiley & Sons, 2013.

[35] Boris Schäling. *The boost C++ libraries.* Boris Schäling, 2011.

[36] Louis O Scott. Option pricing when the variance changes randomly: Theory, estimation, and an application. *Journal of Financial and Quantitative analysis*, 22(04):419–438, 1987.

[37] Steven E Shreve. *Stochastic calculus for finance II: Continuous-time models*, volume 11. Springer Science & Business Media, 2004.

[38] Elias M Stein and Jeremy C Stein. Stock price distributions with stochastic volatility: an analytic approach. *Review of financial Studies*, 4(4):727–752, 1991.

[39] Rainer Storn and Kenneth Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11(4):341–359, 1997.

[40] Mohit Tawarmalani and Nikolaos V Sahinidis. *Convexification and global optimization in continuous and mixed-integer nonlinear programming: theory, algorithms, software, and applications*, volume 65. Springer Science & Business Media, 2002.

[41] James B Wiggins. Option values under stochastic volatility: Theory and empirical estimates. *Journal of financial economics*, 19(2):351–372, 1987.

[42] Steven H Zhu and Michael Pykhtin. A guide to modeling counterparty credit risk. *GARP Risk Review, July/August*, 2007.