American Option pricing under Multiscale model using Monte Carlo and Least-square approach

by

Tiffany Bart Adde
Kadek Maya Sri Puspita

Kandidatarbete i matematik / tillämpad matematik
Bachelor thesis in mathematics / applied mathematics

Current version:
19th June 2017

Project name:
American Option pricing under Multiscale model using Monte Carlo and Least-square approach

Authors:
Tiffany Bart Adde and Kadek Maya Sri Puspita

Supervisor:
Jan Röman

Co-supervisor:
Linus Carlsson

Reviewer:
Richard Bonner

Examiner:
Anatoliy Malyarenko

Comprising:
15 ECTS credits
Abstract

In the finance world, option pricing techniques have become an appealing topic among researchers, especially for pricing American options. Valuing this option involves more factors than pricing the European style one, which makes it more computationally challenging. This is mainly because the holder of American options has the right to exercise at any time up to maturity. There are several approaches that have been proved to be efficient and applicable for maximizing the price of this type of options. A common approach is the Least squares method proposed by Longstaff and Schwartz. The purpose of this thesis is to discuss and analyze the implementation of this approach under the Multiscale Stochastic Volatility model. Since most financial markets show randomly variety of volatility, pricing the option under this model is considered necessary. A numerical study is performed to present that the Least-squares approach is indeed effective and accurate for pricing American options.
Acknowledgments

We would like to express our gratitudes to our supervisors, Linus Carlsson and Jan Röman for their patience, guidance and motivation. Their input truly helped us through our time of research and writing of this thesis. We would also like to thank our examiner Anatoliy Malyarenko for his informative feedback.
## Contents

### Introduction
1.1 Literature review .................................................. 1
1.2 Research question .................................................. 2
1.3 Methodology ......................................................... 2

### Derivative Pricing
2.1 American options .................................................. 3
2.2 The Binomial tree ................................................... 3
2.3 Black and Scholes model ........................................... 5
2.4 Monte Carlo methods ................................................ 6
2.5 Least-Squares approach ............................................ 7
2.5.1 Numerical example for the Least-Square approach ........ 8

### American option pricing under Multiscale model using Monte Carlo methods
3.1 Multiscale Stochastic Volatility model .......................... 13
3.2 Simulation of Multiscale Stochastic Volatility model .......... 14

### Results
4.1 A Numerical Implementation of LSM ............................. 16
4.2 A Numerical Example of Monte Carlo simulation under Multiscale Volatility Model ................................................ 17
4.3 A Numerical Example of Monte Carlo simulation and Least-Square Approach under Multiscale Volatility Model .............. 21

### Conclusion
................................................................. 23

### Appendix 1
................................................................. 26
List of Figures

2.1 Binomial Tree ......................................................... 5
4.2 The figure shows a fast mean-reverting stochastic process. We see that the process goes from about 0.25 up to 0.55, which is proven to be the fast one, or in other words, the movements of the volatility are very extreme. This being compare to a slow mean-reverting process. The thicker line symbolizes the mean result for all simulations. ......................................................... 18
4.3 The figure shows the stock price under the fast mean-reverting process. The graph shows that, indeed, the prices are more concentrated to one point than the one in Figure 4.5. The thicker line symbolizes the mean result for all simulations. ......................................................... 18
4.4 The figure shows a slow mean-reverting process. We can see that compare to the $V_1$ from Figure 4.2 the difference between the highest value and the lowest is much lower as it only goes from 0.37 to 0.44. This stochastic process has less drastic changes than $V_2$. The thicker line symbolizes the mean result for all simulations. ......................................................... 19
4.5 This figure is the graph of stock price under slow mean-reverting process. Compared to the MSVM with the fast mean-reverting process, this figure shows less concentrated paths. The thicker line symbolizes the mean result for all simulations. ......................................................... 19
4.6 The figure shows both slow and fast mean-reverting processes using Monte Carlo simulation. The thicker line symbolizes the mean result for all simulations. ......................................................... 20
4.7 The figure shows the stock price under both fast and slow mean-reverting processes. This shows less concentrated paths than the fast mean-reverting (see Figure 4.3), however more than the slow mean-reverting one (see Figure 4.5). The thicker line symbolizes the mean result for all simulations. ......................................................... 20
## List of Tables

2.1 Stock price .................................................. 9
2.2 Value of the American put option at $t = 3$ ......................... 9
2.3 Early exercise decision at $t = 2$ .................................. 11
2.4 Cash flow of the American put option at $t = 3$ ..................... 11
2.5 Early exercise decision at $t = 1$ .................................. 12
2.6 Cash flow ......................................................... 12
4.7 Results of Numerical example for American Put option ............ 16
4.8 Results of Numerical example for American Call option .......... 17
4.9 Results for American Put option with MSVM under LSM .......... 21
4.10 Results for American Call option with MSVM under LSM ........ 22
Introduction

Determining the price of American style options, with the ability to exercise early, has been a major task in derivatives finance. The chances are big that these type of derivatives can be found in all financial markets, including, equity, foreign exchange, agriculture, insurance, etc. The classical model, the binomial options pricing model, is commonly used due to its simplicity and applicability to handle a variety of conditions compare to other models. The method of this model is by creating a graph that resembles a tree, which is often called the binomial tree. The model traces the evolution of the option’s important underlying variables in discrete time. The drawback is that this model becomes less practical in option pricing that involves multiple factors. Based on this reason, several numerical techniques have been developed over the years, for instance, the Least-square Monte Carlo approach introduced by Longstaff and Schwartz [14]. In order to obtain the optimal exercise strategy that maximizes the payoff, the simulation methods solve the problem given the stochastic nature of the underlying stock price. Although computationally time-consuming, the Monte Carlo simulation can converge to the solution faster for higher number of simulations.

In recent years, several researchers have considered using the stochastic volatility model to approximate the price of an option. The models are developed to eliminate the assumption of constant volatility by the Black-Scholes model [2]. Over the life of the derivative, the underlying volatility does get affected by the changes in the price level of the underlying security. By allowing the price to differ, this improves the accuracy for the calculations of this model. In this thesis, we consider a multiscale stochastic volatility model studied in [4]. This model consists of three stochastic variables and volatilities which modulated by changes occurring on fast or slow time scales.

1.1 Literature review

The use of Monte Carlo simulation for pricing American option is first attempted by Tilley [16]. His approach uses the backward algorithm and bundling technique which apply to the valuation of a single asset American put option. Even though he accomplished to show that there exists a useful algorithm for valuing American option in a path simulation, another problem arose when pricing an option which was based on several underlying assets. To address this problem, Broadie and Glasserman [3] developed a method which generates two estimators that converge to the true price. However, their estimators are limited under two decisions only, exercise or not exercise the option. Based on the early exercise features in American options, Longstaff and Schwartz in [14] developed a practical Monte Carlo method by apply-
The existence of a skew or the smile curve of the option’s implied volatility has been proven to be true in many studies. By applying the stochastic volatility models proposed by researchers, for example [10], [12], one can capture the movements of the volatility. In [8], Christoffersen et al. show the purpose of using two stochastic volatilities model is that one could capture the random movements on the asset prices and the random behaviour of market risks better, unlike using only one stochastic volatility. Chiarella and Ziveyi in [5] presented a numerical integration technique for pricing an American call option under two-factor stochastic volatilities of mean reversion type. By applying the Fourier and Laplace transforms, they are able to present an approximate formula for pricing American option. They also analyzed the effects of varying the volatilities of instantaneous variance on both the early exercise boundary and the corresponding option prices. For an overview of the recent developments in the use of multiscale volatility model for pricing American option, Agarwal et al. [1] use perturbative expansion techniques to approximate the price of a single asset put and the optimal exercise boundary. In their paper, they also stated that the fast and slow-scale approximations are equally accurate if the scaling parameter value is equal to unity. Canhanga et al. in [4] uses similar model studied in [5] for valuing European options. They analyzed the applicability and similarity between the three different methods, the asymptotic expansion method, Fourier and Laplace, and Monte Carlo simulation in order to find the true value of European options.

1.2 Research question

In this thesis, we study the application of appropriate models for pricing American options. By analyzing the performance of this models and then comparing the pricing results, we answer the following questions: How applicable the Multiscale Stochastic Volatility model is in programming and how to value the American options under this model using the Least-squares Monte Carlo approach.

1.3 Methodology

In order to implement this research, the models applied are the binomial pricing model and the Least-square Monte Carlo under the Multiscale Stochastic Volatility model. We use the price from the binomial as our reference price. Then by collecting other important parameters, including, the initial asset price, strike price, time to maturity, etc., we begin our simulation which is performed throughout in Python 3.6.

The thesis is organized in the following structure: An introduction to the important features of American options and theoretical specifications of the various option pricing models are given in the next section. Section 3 contains a brief introduction of the Multiscale Stochastic model for pricing American put options. The result of the simulation and comparison of the models is presented in Section 4. Finally, the conclusion is provided in the last section.
2.1 American options

An option is a contract that gives the holder the right but not the obligation to buy (call) or sell (put) the security or asset. The most common traded options are European and American options. In contrast with European option, an American option is an option that can be exercised at any time, before or at the maturity date. Due to its ability to exercise early, pricing American options is considered challenging by researcher according to Longstaff and Schwartz in [14]. As oppose to a call option, a put option is considered in the money when $S < K$, at the money when $S = K$ and out of the money when $S > K$, where $S$ is the stock price and $K$ is the strike price. If the option is out-of-the-money at time $t$, then it is better not to exercise. However, if the option is in-the-money, the holder has to consider whether or not they want to wait to exercise the option since the payoff might be even bigger in the future. Alternatively, exercise the option and lose this opportunity. The payoff of American option can be calculated by using the following formulas:

$$C(0, t) = 0 \text{ for all } t \in [0, T] \text{ and } C(S, T) = \max\{S - K, 0\},$$

$$P(\infty, t) = 0 \text{ and } P(S, T) = \max\{K - S, 0\},$$

where $T$ is the maturity date, $C$ denotes the call price and $P$ the put price. According to Hull in [11], the value of American and European calls are the same under no dividends. However, American put is worth more than European put options.

2.2 The Binomial tree

In order to find the value of an American option the most common used model is the Binomial tree. The binomial options pricing model which originates from the Black-Scholes-Merton model is introduced by Cox, Ross and Rubinstein in 1979, see [7]. This model is considered as one of the easiest ways for pricing American options. This technique constructs a diagram showing different possible intrinsic values that an option may follow at different nodes or time periods. One way of valuing the American option involves a backward algorithm which requires us to work from the maturity date to the starting point of the option’s life. It is important to check at each node whether exercise the option early is optimal or not. At each node, the
value of the option depends on the probability of increasing or decreasing the underlying asset by a certain percentage amount. Let us imagine that we have a stock price at time 0, where there is a certain probability that option can go up or down which can be calculated by using the formula bellow:

\[ u = e^{\sigma \sqrt{\Delta t}}, \]
\[ d = e^{-\sigma \sqrt{\Delta t}}, \]
\[ p = \frac{e^{\rho \Delta t} - d}{u - d}, \]

where \( \sigma \) is the volatility, \( \Delta t \) is the length of the time step on a binomial tree, \( u \) and \( d \) are the parameters of up and down movements and \( p \) is the probability under the risk-neutral world measure.

There can be many time steps which is indicated by the length of the time steps \( \Delta t \). For example if there is two up movements the value of the option will be denoted by \( f_{uu} \), one up and one down movement by \( f_{ud} \) and two down movement by \( f_{dd} \). The value of the option of one step binomial tree is calculated as follows:

\[ f = e^{-r\Delta t}[pf_u + (1 - p)f_d]. \tag{2.1} \]

If more time steps come in consideration we repeat the Equation (2.1) and get for a two movements binomial tree:

\[ f_u = e^{-r\Delta t}[pf_{uu} + (1 - p)f_{ud}], \tag{2.2} \]
\[ f_d = e^{-r\Delta t}[pf_{ud} + (1 - p)f_{dd}], \tag{2.3} \]
\[ f = e^{-r\Delta t}[pf_u + (1 - p)f_d]. \tag{2.4} \]

In order to calculate the real value of the option we substitute Equations (2.2) and (2.3) in (2.4) and get:

\[ f = e^{-2r\Delta t}[p^2 f_{uu} + 2p(1 - p)f_{ud} + (1 - p)^2 f_{dd}]. \]

Figure 2.1 shows how the Binomial Tree looks like.
2.3 Black and Scholes model

Black-Scholes is the primary pricing model that are widely used to calculate the fair price of option, based on six variables such as volatility, type of option, underlying stock price, time, strike price, and risk-free rate.

This model is effectively used to determine the price of a European option, which is an option that must be held to maturity. For pricing the European option with no dividend, the Black-Scholes-Merton(1973) formula is given by:

\[
C(S, t) = S N(d_1) - Ke^{-r(T-t)} N(d_2),
\]

\[
P(S, t) = Ke^{-r(T-t)} N(-d_2) - SN(-d_1),
\]

where \(N(.)\) is the cumulative distribution, \((T - t)\) is time to maturity and \(d_1, d_2\) are given by

\[
d_1 = \frac{ln(S/K) + (r + \sigma^2/2)(T - t)}{\sigma \sqrt{T - t}},
\]

\[
d_2 = \frac{ln(S/K) + (r - \sigma^2/2)(T - t)}{\sigma \sqrt{T - t}},
\]

which can also be written as

\[
d_2 = d_1 - \sigma \sqrt{T - t}.
\]

The Black-Scholes formula is not suitable for pricing American option because it relies on the assumption that there exist only one predetermined maturity date.
2.4 Monte Carlo methods

In mathematical finance, the Monte Carlo simulation is introduced by Boyle in 1977. Monte Carlo becomes attractive for valuing complex financial derivatives which helps to evaluate high-dimensional integrals. This can be shown by using the Monte Carlo method to estimate the value of an integral \( f \), mathematically it can be written as [9]:

\[
\alpha = \int_0^1 f(x)dx.
\]

We factorized the function \( f(x) \), then the expected value is given by

\[
\alpha = E[a(x)] = \int_0^1 a(x) \rho(x) dx.
\]

From the density \( \rho(x) \), we generate an i.i.d sample \( x_1, x_2, \ldots, x_n \) to estimate the expected value as

\[
\hat{\alpha} = \frac{1}{n} \sum_{i=1}^{n} a(x_i),
\]

where \( n \) is the number of random points. By the law of large numbers, suppose that \( a(x) \) is integrable, then

\[
\hat{\alpha} \longrightarrow \alpha \text{ almost surely as } n \longrightarrow \infty.
\]

However, if \( a(x) \) is also square-integrable, we define

\[
\sigma^2 = \text{Var}[a(x)] = \int_0^1 (a(x) - \alpha)^2 \rho(x) dx,
\]

which gives

\[
\frac{\sqrt{n}(\hat{\alpha} - \alpha)}{\sigma} \longrightarrow \mathcal{N}(0,1).
\]

This means that the error \( \hat{\alpha} - \alpha \) in the Monte Carlo estimate is normally distributed with mean 0 and standard deviation \( \sigma^2 / \sqrt{n} \), which gives the convergence\(^1\) rate \( O(n^{-1/2}) \). This shows that the methods have a slow convergence but do not change as the dimension increases. Based on this reason, the Monte Carlo methods are not very attractive in evaluating integrals in one dimension, but as dimensions increase so do their attractiveness.

In general, Monte Carlo method works well for valuing European options. The method shows the evolution of underlying asset prices, interest rate and other important factors of the security by simulating as many sample paths as desired under the stochastic processes. Pricing the European options is determined only by the stock price at a given starting point, maturity time, constant interest rate and volatility. However, pricing American options also require the information of option value at the intermediate times between the simulation start time and

\(^1\)Here, the convergence means that the Monte Carlo simulations will never give an exact answer, but increasingly good approximations.
the option maturity time which in this method is harder to obtain. Since pricing this option
involves backward algorithm, the Monte Carlo methods become computationally complex.

To achieve a more accurate approximation of valuing complicated path-dependent option,
where it is necessary to divide a time interval $[0, T]$ into subintervals, we have the following
illustration. Assume that the stock price grows in time according to a stochastic differential
equation given bellow

$$dS(t) = \mu S(t)dt + \sigma S(t)dW(t),$$

with $\mu$ a risk-free rate and $W(t)$ a standard Brownian motion that follows the properties [15]:

- $W(0) = 0$.
- If $r \leq s \leq t \leq u$, $W$ has independent increments, which means that $W(u) - W(t)$ and $W(s) - W(r)$ is independent.
- $W(t) - W(s)$ is normally distributed with mean 0 and variance $\sqrt{t-s}, N(0, \sqrt{t-s})$.
- $W(t)$ has continuous paths which can be proven by using the Kolmogorov’s continuity theo-

Generalizing the Equation (2.5), gives the following dynamics

$$dS(t) = \mu S(t)dt + \sigma(S(t))S(t)dW(t).$$

This equation shows that $\sigma$ is dependant on the stock price $S(t)$, as explained in [9]. The
approximation of Equation (2.6) gives us the following

$$S(t + \Delta t) = S(t) + \mu S(t)\Delta t + \sigma(S(t))S(t)W(t),$$

where $\Delta t = T/n$, since we consider $n$ to be the number of time steps. The Wiener process
$W(T)$ is distributed as $\sqrt{\Delta t}Z$ where $Z$ is a standard normal random variable. The process then
can be simulated by the following equation

$$S(t + \Delta t) = S(t) + \mu S(t)\Delta t + \sigma(S(t))S(t)\sqrt{\Delta t}Z.$$

### 2.5 Least-Squares approach

The Least Squares Monte Carlo (LSM) approach, which is developed by Longstaff and Schwartz,
see [14], is popularly used to deal with the problems of pricing American options mentioned
above. Their approach uses least-squares analysis to find the optimal option price by compar-
ing the value of continuation with the immediate exercise value at each time an early exercise
has to be made. This involves regressing the discounted future cash flows found from continu-
ation on a finite set of functions in the underlying stock prices. In other words, the intention
of the Least-Squares method is to find paths of approximation that find the optimal exercising
point in order to maximize the value of the American option.
Assuming an American option is alive within the time horizon \([0, T]\). Within the time horizon, early exercise can only be done at the \(J\) discrete times \(0 < t_1 \leq t_2 \leq \ldots \leq t_J = T\). At exercise time \(t_k\), if the payoff of immediate exercise is positive and higher than the value of continuation, that option should be exercised immediately. This continuation value is the remaining value of the option that cannot be exercised until after \(t_k\), which can also be expressed as conditional expectation of the option payoff under the risk-neutral pricing measure \(Q\).²

The following algorithm is based on the Longstaff and Schwartz approach. Introducing the notation \(C_{ij}\) as the cashflow vector at the last timestep \(t_J\). We determine the cashflow, which is the payoff of a call option, in other words, has zero continuation value, of each simulation \(i\) by the following formula:

\[
C_{i, t_J} = \max(S_{i, t_J} - K, 0).
\]

We then estimate the exercise value by selecting the stock prices at time-step \(t_{J-1}\), which has the following condition

\[
\max(S_{i, t_{J-1}} - K, 0) > 0.
\]

By regressing the discounted future cash-flows realized from continuing onto a finite set of basis function of our stock price, we can then obtain the continuation value for path \(i\) with values \(S_{i, t_n}\) at time \(t_n\) [14]:

\[
F_{i, t_{n-1}} = \sum_{j=0}^{\infty} a_j(t_n) L_j(S),
\]

where \(a_j\) are constant coefficients and \(L_j\) is the set of basis functions. The regression is performed by using all the values from all of the paths with the basic function of any degrees polynomial. We can then exercise the American option at an early stage when it fulfills the condition bellow

\[
C_{i, t_{n-1}} > F_{i, t_{n-1}}.
\]

Further, by working backward, we obtain the early exercise at each time-step. This lets us to build the value matrix from the cash flow vectors \(C_{i, t_n}\), by concatenating the cashflow vectors \(C_{i, t_n}\), where \(n = 1, \ldots, J\). The option value is then decided by discounting each cash flow back to the starting point and calculating the mean of the results.

### 2.5.1 Numerical example for the Least-Square approach

Longstaff and Schwartz presented in their paper [14] a simple numerical example in order to understand the Least-Square method. This is considered a simple numerical example as there only is 8 paths when the number of paths should be higher to resemble reality. Using the same example as in [14], in this section we are going to mathematically show an in-depth explanation of the approach.

²\(Q\) measure is a probability measure which is derived from the assumption that the present value of the financial assets is equal to their future discounted expected payoffs at the risk free rate.
Let us assume that we have the collected data as shown in Table 2.1. We consider an American put option where the strike price is 1.10 and the risk-free rate is 6% with no-dividend.

Table 2.1: Stock price

<table>
<thead>
<tr>
<th>Path</th>
<th>$t = 0$</th>
<th>$t = 1$</th>
<th>$t = 2$</th>
<th>$t = 3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.00</td>
<td>1.09</td>
<td>1.08</td>
<td>1.34</td>
</tr>
<tr>
<td>2</td>
<td>1.00</td>
<td>1.16</td>
<td>1.26</td>
<td>1.54</td>
</tr>
<tr>
<td>3</td>
<td>1.00</td>
<td>1.22</td>
<td>1.07</td>
<td>1.03</td>
</tr>
<tr>
<td>4</td>
<td>1.00</td>
<td>.93</td>
<td>.97</td>
<td>.92</td>
</tr>
<tr>
<td>5</td>
<td>1.00</td>
<td>1.11</td>
<td>1.56</td>
<td>1.52</td>
</tr>
<tr>
<td>6</td>
<td>1.00</td>
<td>.76</td>
<td>.77</td>
<td>.90</td>
</tr>
<tr>
<td>7</td>
<td>1.00</td>
<td>.92</td>
<td>.84</td>
<td>1.01</td>
</tr>
<tr>
<td>8</td>
<td>1.00</td>
<td>.88</td>
<td>1.22</td>
<td>1.34</td>
</tr>
</tbody>
</table>

From Table 2.1, we focus on the stock price at $t = 2$ and collect the stock prices that are in-the-money which means $K - S_t > 0$. In other words, the “in-the-money” stock prices, denoted as $S_t$, are 1.08, 1.07, .97, .77, and .84, which lay on paths 1, 3, 4, 6 and 7. At time $t = 3$, we would then have the value for these paths as shown in Table 2.2.

Table 2.2: Value of the American put option at $t = 3$

<table>
<thead>
<tr>
<th>Path</th>
<th>$\max(0, K - S_3)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>.00</td>
</tr>
<tr>
<td>3</td>
<td>.07</td>
</tr>
<tr>
<td>4</td>
<td>.18</td>
</tr>
<tr>
<td>6</td>
<td>.20</td>
</tr>
<tr>
<td>7</td>
<td>.09</td>
</tr>
</tbody>
</table>

The values of the option in $t = 2$ are $0.00e^{-0.06 \cdot 1}, 0.07e^{-0.06 \cdot 1}, 0.18e^{-0.06 \cdot 1}, 0.20e^{-0.06 \cdot 1}, 0.09e^{-0.06 \cdot 1}$ which we denote as $V$. We now assume that for these paths, we have an approximation in the following form

$$V = a + bS + cS^2,$$

where $a, b, c$ are constants. In order to find these constants, we have to find the minimum error for all the five paths.

$$\min_{a,b,c} \sum_{i=1}^{5} (V_i - a - bS_i - cS_i^2)^2. \quad (2.7)$$

We denote Equation (2.7) as $f(a, b, c)$. To find the minimum error, we ought to differentiate the function $f$ with respect to $a, b, c$ and equal it to 0.
\[
\frac{\partial f}{\partial a} = -2 \sum_{i=1}^{5} (V_i - a - bS_i - cS_i^2) = 0, \\
\frac{\partial f}{\partial b} = -2 \sum_{i=1}^{5} S_i (V_i - a - bS_i - cS_i^2) = 0, \\
\frac{\partial f}{\partial c} = -2 \sum_{i=1}^{5} S_i^2 (V_i - a - bS_i - cS_i^2) = 0.
\]

This leads to the following system of linear equations

\[
5a + b \sum_{i=1}^{5} S_i + c \sum_{i=1}^{5} S_i^2 = \sum_{i=1}^{5} V_i, \\
a \sum_{i=1}^{5} S_i + b \sum_{i=1}^{5} S_i^2 + c \sum_{i=1}^{5} S_i^3 = \sum_{i=1}^{5} V_i S_i, \\
a \sum_{i=1}^{5} S_i^2 + b \sum_{i=1}^{5} S_i^3 + c \sum_{i=1}^{5} S_i^4 = \sum_{i=1}^{5} V_i S_i^2.
\]

Given the following matrix rules

\[
A \begin{pmatrix} a \\ b \\ c \end{pmatrix} = B,
\]

where \( A \) is

\[
\begin{pmatrix}
5 & \sum_{i=1}^{5} S_i & \sum_{i=1}^{5} S_i^2 \\
\sum_{i=1}^{5} S_i & \sum_{i=1}^{5} S_i^2 & \sum_{i=1}^{5} S_i^3 \\
\sum_{i=1}^{5} S_i^2 & \sum_{i=1}^{5} S_i^3 & \sum_{i=1}^{5} S_i^4
\end{pmatrix},
\]

and \( B \) is

\[
\begin{pmatrix}
\sum_{i=1}^{5} V_i \\
\sum_{i=1}^{5} V_i S_i \\
\sum_{i=1}^{5} V_i S_i^2
\end{pmatrix}.
\]

This gives us

\[
\begin{pmatrix}
5 & \sum_{i=1}^{5} S_i & \sum_{i=1}^{5} S_i^2 \\
\sum_{i=1}^{5} S_i & \sum_{i=1}^{5} S_i^2 & \sum_{i=1}^{5} S_i^3 \\
\sum_{i=1}^{5} S_i^2 & \sum_{i=1}^{5} S_i^3 & \sum_{i=1}^{5} S_i^4
\end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix}
\sum_{i=1}^{5} V_i \\
\sum_{i=1}^{5} V_i S_i \\
\sum_{i=1}^{5} V_i S_i^2
\end{pmatrix}.
\]

Data from Table 2.1 has the solution \( a = -1.070 \), \( b = 2.983 \) and \( c = -1.813 \) and our equation for the value became

\[
3\text{The code used to solve this can be found in Mathews, J. H., & Fink, K. D. (2005). Numerical methods using MATLAB. New Delhi: Prentice-Hall of India. p. 274-275.}
\]
\[ V = -1.070 + 2.983S - 1.813S^2. \]

Hence, we get the cash flow of continuing and exercising at \( t = 2 \) as shown in Table 2.3. (OTM=out of the money).

Table 2.3: Early exercise decision at \( t = 2 \)

<table>
<thead>
<tr>
<th>Path</th>
<th>Exercise: ( K - S(t_2) )</th>
<th>Continuation: ( V(S(t_2)) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>.02</td>
<td>.0369</td>
</tr>
<tr>
<td>2</td>
<td>OTM</td>
<td>OTM</td>
</tr>
<tr>
<td>3</td>
<td>.03</td>
<td>.0461</td>
</tr>
<tr>
<td>4</td>
<td>.13</td>
<td>.1176</td>
</tr>
<tr>
<td>5</td>
<td>OTM</td>
<td>OTM</td>
</tr>
<tr>
<td>6</td>
<td>.33</td>
<td>.1520</td>
</tr>
<tr>
<td>7</td>
<td>.26</td>
<td>.1565</td>
</tr>
<tr>
<td>8</td>
<td>OTM</td>
<td>OTM</td>
</tr>
</tbody>
</table>

From Table 2.3, we can see that for the paths 4, 6 and 7 early exercise values are bigger than the values of continuing. Therefore, we decide to exercise at those points. Then the cash flow for \( t = 3 \) are shown in Table 2.4. Observe that the cash flow is 0 when the option is exercised on the previous time steps.

Table 2.4: Cash flow of the American put option at \( t = 3 \)

<table>
<thead>
<tr>
<th>Path</th>
<th>( \max(0, K - S_3) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>.00</td>
</tr>
<tr>
<td>2</td>
<td>.00</td>
</tr>
<tr>
<td>3</td>
<td>.07</td>
</tr>
<tr>
<td>4</td>
<td>.00</td>
</tr>
<tr>
<td>5</td>
<td>.00</td>
</tr>
<tr>
<td>6</td>
<td>.00</td>
</tr>
<tr>
<td>7</td>
<td>.00</td>
</tr>
<tr>
<td>8</td>
<td>.00</td>
</tr>
</tbody>
</table>

Now we look at Table 2.1, where \( t = 1 \). We see that the paths 1, 4, 6, 7 and 8 are in the money. As for \( t = 2 \), we found the equation of the value for continuing by using the same formula as Equation (2.7). We get

\[ V = 2.038 - 3.335S + 1.356S^2. \]

Table 2.5 shows that the values of decision for the early exercise and the values of continuing when \( t = 1 \). We see that the exercise values at paths 4, 6, 7 and 8 are higher than the
continuing, hence, we decide to early exercise. We show the cash flow for $t = 1$, $t = 2$ and $t = 3$ in Table 2.6.

Table 2.5: Early exercise decision at $t = 1$

<table>
<thead>
<tr>
<th>Path</th>
<th>Exercise: $K - S(t_1)$</th>
<th>Continuation: $V(S(t_1))$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.01</td>
<td>0.0139</td>
</tr>
<tr>
<td>2</td>
<td>OTM</td>
<td>OTM</td>
</tr>
<tr>
<td>3</td>
<td>OTM</td>
<td>OTM</td>
</tr>
<tr>
<td>4</td>
<td>0.17</td>
<td>0.1092</td>
</tr>
<tr>
<td>5</td>
<td>OTM</td>
<td>OTM</td>
</tr>
<tr>
<td>6</td>
<td>0.34</td>
<td>0.2866</td>
</tr>
<tr>
<td>7</td>
<td>0.18</td>
<td>0.1175</td>
</tr>
<tr>
<td>8</td>
<td>0.22</td>
<td>0.1533</td>
</tr>
</tbody>
</table>

Table 2.6: Cash flow

<table>
<thead>
<tr>
<th>Path</th>
<th>$t = 1$</th>
<th>$t = 2$</th>
<th>$t = 3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0.07</td>
</tr>
<tr>
<td>4</td>
<td>0.17</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0.34</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0.18</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0.22</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

The value of the option become

\[
\frac{0.07e^{-0.06\cdot3} + 0.17e^{-0.06\cdot1} + 0.34e^{-0.06\cdot1} + 0.18e^{-0.06\cdot1} + 0.22e^{-0.06\cdot1}}{8} = 0.1144. \tag{2.8}
\]

Equation (2.8) says that the value of the option under LSM is 0.1144, which is higher than $K - S(t_0) = 1.1 - 1 = 0.1$, hence it is not optimal to exercise early.
American option pricing under Multiscale model using Monte Carlo methods

3.1 Multiscale Stochastic Volatility model

The Multiscale Stochastic Volatility model (MSVM) approaches a linear approximation that has a component of the well-known Black-Scholes SDE. As opposed to the Black-Scholes formula, the MSVM does not assume that there is a constant volatility. It has also been proven that a single volatility does not fit the real world observed financial data. Indeed, this model contains two correlated volatilities, one moving fast and the other moving slow. In the finance world, the difference between a fast mean-reverting and a slow mean-reverting volatility is significant. With respect to the return of an option, if we assume that a participant has a long position in the case that the volatility is fast mean-reverting, she could then expect that the possibilities to get high return to quickly disappear. Hence, she would try to sell the option in order to keep the high return. On the other hand, if she has a short position, she will most certainly try to keep the option and wait for the price to get lower, therefore, earn a higher profit. If the participant assumes that the mean-reversion is slow, in the case where she is in a long position, she will surely try to keep the option and wait for the price to go up and receive a higher return. In contrast to the long position, when the participant is in a short one she will try to sell the option as fast as possible in order to keep a higher benefit.

We apply the following model as used in [5] and [4]:

\[ dS = \mu Sdt + \sqrt{V_1}SdW_1 + \sqrt{V_2}SdW_2, \]  
\[ dV_1 = \frac{1}{\epsilon} (\theta_1 - V_1)dt + \sqrt{\frac{1}{\epsilon}} \xi_1 \sqrt{V_1}dW_3, \]  
\[ dV_2 = \delta (\theta_2 - V_2)dt + \sqrt{\delta} \xi_2 \sqrt{V_2}dW_4, \]

where \( \mu \) is the expected return, \( V_1 \) and \( V_2 \) are the variance processes, and \( \xi_1 \) and \( \xi_2 \) are constants. All the Wiener processes are uncorrelated except \( dW_1 \) together with \( dW_3 \) are correlated with correlation coefficient \( \rho_{13} \) and \( dW_2 \) together with \( dW_4 \) are correlated with correlation coefficient \( \rho_{24} \). The variance processes \( V_1, V_2 \) are mean reverting processes with reversion rates of \( \frac{1}{\epsilon} \) and \( \delta \) and long run average of \( \theta_1, \theta_2 \). The terms \( \sqrt{\frac{1}{\epsilon}} \xi_1 \) and \( \sqrt{\delta} \xi_2 \) are the so called vol-vols for the processes \( V_1 \) and \( V_2 \).
We assume, $0 < \varepsilon << 1$ and $0 < \delta << 1$, which implies that the process $V_1$ is the fast mean-reverting and the process $V_2$ is slow mean-reverting.

### 3.2 Simulation of Multiscale Stochastic Volatility model

The problem with the Multiscale Stochastic Volatility model (MSVM) is that it follows a Square-root diffusion. Hence, in order to simulate MSVM we first simplified Equations (3.9), (3.10) and (3.11) by eliminating the slow mean-reverting stochastic process. The equation then become

$$dS = \mu S dt + \sqrt{V_1} S dW_1$$

$$dV_1 = \frac{1}{\varepsilon} (\theta - V_1) dt + \sqrt{\frac{1}{\varepsilon}} \xi \sqrt{V_1} dW_3.$$  \hspace{1cm} (3.13)

These has the same parameter as the Heston model [10]

$$dS(t) = \mu S(t) dt + \sqrt{V(t)} dW_1(t),$$  \hspace{1cm} (3.14)

$$dV(t) = \alpha (b - V(t)) dt + \sigma \sqrt{V(t)} dW_2(t),$$  \hspace{1cm} (3.15)

where $\alpha$ is the rate of mean-reverting of the Cox Ingersoll Ross process (CIR) [6], $b$ is the mean value over time, $\sigma$ is the volatility of the CIR process and $W_1$ and $W_2$ are part of a two-dimensional Brownian motion. Cox, Ingersoll and Ross proposed a model of interest rate dynamic that is as follows

$$dr(t) = \alpha (b - r(t)) dt + \sigma \sqrt{r(t)} dW(t),$$  \hspace{1cm} (3.16)

where in this case $W$ is a standard one-dimensional Brownian motion. Cox, Ingersoll and Ross suggest that if $r(0) > 0$ then $r(t)$ will never be negative if $2\alpha b \geq \sigma^2$[9]. An Euler discretization indicate the following simulation

$$r(t_{i+1}) = r(t_i) + \alpha (b - r(t_i)) \Delta t + \sigma \sqrt{r(t_i)} \Delta W_{i+1}. $$  \hspace{1cm} (3.17)

This being done, we had to assign the correlation between $W_1$ and $W_2$. We used a simplified Cholesky decomposition method that says if $W_1$ and $W_2$ has a correlation of $\rho$ we set $W_1 = z_1$ and $W_2 = \rho z_1 + \sqrt{1 - \rho^2} z_2$, where $z_1$ and $z_2$ are two random variables that are normally distributed with mean 0 and variance $\sqrt{\Delta t}$, see [13].
Algorithm 1 The algorithm for the correlation path

set $W_1 = Z_1$

for $i = 1, ..., n - 1$
    Generating $Z_1$
    Generating $Z_2$
    term1 = $\rho W_1$
    term2 = $\sqrt{1 - \rho^2} Z_2$
    $W_2(i) = \text{term1} + \text{term2}$
    Return $W_1, W_2$
end for

$Z_1$ and $Z_2$ are two brownian motions

Algorithm 2 The algorithm for the volatility path

set $V(0) = \text{initial volatility}$

for $i = 1, ..., T$
    Generate $Z_2$
    drift = $a(\mu - V(i-1)) \Delta t$
    randomness = $\sqrt{\max(V(i-1), V(0)) \Delta t Z_2(i-1)}$
    $V(i) = \max(V(i-1), V(0)) + \text{drift} + \text{randomness}$
    Return $\sqrt{\Delta t Z_2}, V$
end for

Algorithm 3 The algorithm for the asset price levels

set $S(0) = \text{initial stock price}$

for $i = 1, ..., T$
    drift = $\mu \cdot S(i-1) \cdot \Delta t$
    vol = $CIR(i-1) \cdot S(i-1) \cdot HCCP(i-1)$
    $S(i) = S(i-1) + \text{drift} + \text{vol}$
    Return $S, CIR$
end for

where $CIR$ denotes the Cox-Ingersoll-Ross process and $HCCP$ denotes the constructed correlated path of Heston model.
Results

In this section, we present the result of some numerical examples by implementing the approaches from the previous sections to price the American options. All the graphs and calculations are performed in Python 3.6. The codes used can be found in Appendix 1.

4.1 A Numerical Implementation of LSM

We study and analyze the accuracy of the Least-squares Monte Carlo proposed by Longstaff and Schwartz [14]. The accuracy is controlled by using different parameters and comparing the result from both Binomial Tree (BT) and Least-squares Monte Carlo (LSM) approaches. The fixed parameters used are \( K = 95, \mu = 0.05 \) and \( n = 50 \). The results we got from the BT and LSM with different \( S_0, \sigma \) and \( T \) are presented in the table below. The number of simulations we used in LSM was \( 10^3 \) and the degree of polynomial is 5.

Table 4.7: Results of Numerical example for American Put option

<table>
<thead>
<tr>
<th>( S_0 )</th>
<th>( \sigma )</th>
<th>( T )</th>
<th>BT</th>
<th>LSM</th>
<th>BT–LSM</th>
</tr>
</thead>
<tbody>
<tr>
<td>80</td>
<td>0.2</td>
<td>1</td>
<td>15.186</td>
<td>15.164</td>
<td>0.022</td>
</tr>
<tr>
<td>80</td>
<td>0.2</td>
<td>2</td>
<td>15.715</td>
<td>15.583</td>
<td>0.132</td>
</tr>
<tr>
<td>80</td>
<td>0.4</td>
<td>1</td>
<td>20.321</td>
<td>20.130</td>
<td>0.191</td>
</tr>
<tr>
<td>80</td>
<td>0.4</td>
<td>2</td>
<td>23.474</td>
<td>23.121</td>
<td>0.353</td>
</tr>
<tr>
<td>90</td>
<td>0.2</td>
<td>1</td>
<td>8.151</td>
<td>8.095</td>
<td>0.056</td>
</tr>
<tr>
<td>90</td>
<td>0.2</td>
<td>2</td>
<td>9.523</td>
<td>9.374</td>
<td>0.149</td>
</tr>
<tr>
<td>90</td>
<td>0.4</td>
<td>1</td>
<td>15.164</td>
<td>14.945</td>
<td>0.219</td>
</tr>
<tr>
<td>90</td>
<td>0.4</td>
<td>2</td>
<td>19.060</td>
<td>18.542</td>
<td>0.518</td>
</tr>
<tr>
<td>100</td>
<td>0.2</td>
<td>1</td>
<td>4.014</td>
<td>3.896</td>
<td>0.118</td>
</tr>
<tr>
<td>100</td>
<td>0.2</td>
<td>2</td>
<td>5.650</td>
<td>5.506</td>
<td>0.144</td>
</tr>
<tr>
<td>100</td>
<td>0.4</td>
<td>1</td>
<td>11.211</td>
<td>10.953</td>
<td>0.258</td>
</tr>
<tr>
<td>100</td>
<td>0.4</td>
<td>2</td>
<td>15.446</td>
<td>15.155</td>
<td>0.291</td>
</tr>
<tr>
<td>110</td>
<td>0.2</td>
<td>1</td>
<td>1.838</td>
<td>1.732</td>
<td>0.106</td>
</tr>
<tr>
<td>110</td>
<td>0.2</td>
<td>2</td>
<td>3.260</td>
<td>3.135</td>
<td>0.125</td>
</tr>
<tr>
<td>110</td>
<td>0.4</td>
<td>1</td>
<td>8.209</td>
<td>7.983</td>
<td>0.226</td>
</tr>
<tr>
<td>110</td>
<td>0.4</td>
<td>2</td>
<td>12.504</td>
<td>12.319</td>
<td>0.185</td>
</tr>
</tbody>
</table>
Table 4.8: Results of Numerical example for American Call option

<table>
<thead>
<tr>
<th>$S_0$</th>
<th>$\sigma$</th>
<th>$T$</th>
<th>BT</th>
<th>LSM</th>
<th>BT–LSM</th>
</tr>
</thead>
<tbody>
<tr>
<td>80</td>
<td>0.2</td>
<td>1</td>
<td>2.785</td>
<td>2.614</td>
<td>0.171</td>
</tr>
<tr>
<td>80</td>
<td>0.2</td>
<td>2</td>
<td>6.631</td>
<td>6.390</td>
<td>0.241</td>
</tr>
<tr>
<td>80</td>
<td>0.4</td>
<td>1</td>
<td>8.988</td>
<td>8.713</td>
<td>0.275</td>
</tr>
<tr>
<td>80</td>
<td>0.4</td>
<td>2</td>
<td>15.593</td>
<td>15.395</td>
<td>0.198</td>
</tr>
<tr>
<td>90</td>
<td>0.2</td>
<td>1</td>
<td>6.973</td>
<td>6.844</td>
<td>0.129</td>
</tr>
<tr>
<td>90</td>
<td>0.2</td>
<td>2</td>
<td>12.071</td>
<td>11.629</td>
<td>0.442</td>
</tr>
<tr>
<td>90</td>
<td>0.4</td>
<td>1</td>
<td>14.180</td>
<td>13.797</td>
<td>0.383</td>
</tr>
<tr>
<td>90</td>
<td>0.4</td>
<td>2</td>
<td>21.745</td>
<td>21.350</td>
<td>0.395</td>
</tr>
<tr>
<td>100</td>
<td>0.2</td>
<td>1</td>
<td>13.335</td>
<td>12.967</td>
<td>0.368</td>
</tr>
<tr>
<td>100</td>
<td>0.2</td>
<td>2</td>
<td>18.960</td>
<td>18.376</td>
<td>0.584</td>
</tr>
<tr>
<td>100</td>
<td>0.4</td>
<td>1</td>
<td>20.448</td>
<td>19.946</td>
<td>0.502</td>
</tr>
<tr>
<td>100</td>
<td>0.4</td>
<td>2</td>
<td>28.508</td>
<td>27.870</td>
<td>0.638</td>
</tr>
<tr>
<td>110</td>
<td>0.2</td>
<td>1</td>
<td>21.364</td>
<td>20.900</td>
<td>0.464</td>
</tr>
<tr>
<td>110</td>
<td>0.2</td>
<td>2</td>
<td>26.933</td>
<td>26.557</td>
<td>0.376</td>
</tr>
<tr>
<td>110</td>
<td>0.4</td>
<td>1</td>
<td>27.587</td>
<td>27.297</td>
<td>0.29</td>
</tr>
<tr>
<td>110</td>
<td>0.4</td>
<td>2</td>
<td>35.776</td>
<td>35.100</td>
<td>0.676</td>
</tr>
</tbody>
</table>

As we can see from Table 4.7 and Table 4.8, the difference between the price of American options from the two models are minimal. Hence, we assume that the Least-Square Monte Carlo is accurate and we now proceed to apply it to Multiscale Stochastic Volatility model.

### 4.2 A Numerical Example of Monte Carlo simulation under Multiscale Volatility Model

In this subsection, we apply and analyze the Monte Carlo simulation into the Multiscale Stochastic Volatility model. The result is achieved by investigating the variance of the fast mean-reverting first, which is denoted by $V_1$. Later on, we apply the same simulation to investigate the slow mean-reverting, $V_2$. This simulation produce the matrices that will be used in LSM under multiscale stochastic volatility model to find the optimal price of American put option in Section 4.3. The data we used are $\mu = 0.06$, $\varepsilon = 0.01$, $\delta = 0.1$, $\theta_1 = 0.4$, $\theta_2 = 1$, $\xi_1 = 0.1$, $\xi_2 = 1$, $V_{10} = 0.3$ and $V_{20} = 0.3$. The number of time steps $n$ was 10 and the number of simulations $m$ was 100. The small number of simulations was taken in order to see clearer on the graph the results we could get. However, the number of simulations should be higher in order to have a closer result to reality. The results of the simulations can be seen in figures 4.2–4.7.
Figure 4.2: The figure shows a fast mean-reverting stochastic process. We see that the process goes from about 0.25 up to 0.55, which is proven to be the fast one, or in other words, the movements of the volatility are very extreme. This being compare to a slow mean-reverting process. The thicker line symbolizes the mean result for all simulations.

Figure 4.3: The figure shows the stock price under the fast mean-reverting process. The graph shows that, indeed, the prices are more concentrated to one point than the one in Figure 4.5. The thicker line symbolizes the mean result for all simulations.
Figure 4.4: The figure shows a slow mean-reverting process. We can see that compare to the $V_1$ from Figure 4.2 the difference between the highest value and the lowest is much lower as it only goes from 0.37 to 0.44. This stochastic process has less drastic changes than $V_2$. The thicker line symbolizes the mean result for all simulations.

Figure 4.5: This figure is the graph of stock price under slow mean-reverting process. Compared to the MSVM with the fast mean-reverting process, this figure shows less concentrated paths. The thicker line symbolizes the mean result for all simulations.
Figure 4.6: The figure shows both slow and fast mean-reverting processes using Monte Carlo simulation. The thicker line symbolizes the mean result for all simulations.

Figure 4.7: The figure shows the stock price under both fast and slow mean-reverting processes. This shows less concentrated paths than the fast mean-reverting (see Figure 4.3), however more than the slow mean-reverting one (see Figure 4.5). The thicker line symbolizes the mean result for all simulations.
4.3 A Numerical Example of Monte Carlo simulation and Least-Square Approach under Multiscale Volatility Model

This section is to show the Least-Squares approach (LSM) under Multiscale Stochastic Volatility model. The degree of polynomial for the LSM is 5. We used the same data as in previous section, however, we used various data for $S_0$, $K$ and $m$.

<table>
<thead>
<tr>
<th>$m$</th>
<th>$S_0$</th>
<th>$K$</th>
<th>Value of the option</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10^1$</td>
<td>90</td>
<td>90</td>
<td>4.757191</td>
</tr>
<tr>
<td>$10^2$</td>
<td>90</td>
<td>90</td>
<td>2.908279</td>
</tr>
<tr>
<td>$10^3$</td>
<td>90</td>
<td>90</td>
<td>2.867147</td>
</tr>
<tr>
<td>$10^4$</td>
<td>90</td>
<td>90</td>
<td>2.870963</td>
</tr>
<tr>
<td>$10^1$</td>
<td>90</td>
<td>100</td>
<td>11.687731</td>
</tr>
<tr>
<td>$10^2$</td>
<td>90</td>
<td>100</td>
<td>10.737611</td>
</tr>
<tr>
<td>$10^3$</td>
<td>90</td>
<td>100</td>
<td>9.979482</td>
</tr>
<tr>
<td>$10^4$</td>
<td>90</td>
<td>100</td>
<td>10.204631</td>
</tr>
<tr>
<td>$10^1$</td>
<td>100</td>
<td>90</td>
<td>0.937536</td>
</tr>
<tr>
<td>$10^2$</td>
<td>100</td>
<td>90</td>
<td>0.208727</td>
</tr>
<tr>
<td>$10^3$</td>
<td>100</td>
<td>90</td>
<td>0.411637</td>
</tr>
<tr>
<td>$10^4$</td>
<td>100</td>
<td>90</td>
<td>0.379972</td>
</tr>
<tr>
<td>$10^1$</td>
<td>100</td>
<td>100</td>
<td>4.674494</td>
</tr>
<tr>
<td>$10^2$</td>
<td>100</td>
<td>100</td>
<td>3.188882</td>
</tr>
<tr>
<td>$10^3$</td>
<td>100</td>
<td>100</td>
<td>3.355485</td>
</tr>
<tr>
<td>$10^4$</td>
<td>100</td>
<td>100</td>
<td>3.272534</td>
</tr>
<tr>
<td>$10^1$</td>
<td>100</td>
<td>110</td>
<td>11.269590</td>
</tr>
<tr>
<td>$10^2$</td>
<td>100</td>
<td>110</td>
<td>10.951877</td>
</tr>
<tr>
<td>$10^3$</td>
<td>100</td>
<td>110</td>
<td>10.374541</td>
</tr>
<tr>
<td>$10^4$</td>
<td>100</td>
<td>110</td>
<td>10.218680</td>
</tr>
<tr>
<td>$10^1$</td>
<td>100</td>
<td>120</td>
<td>25.218587</td>
</tr>
<tr>
<td>$10^2$</td>
<td>100</td>
<td>120</td>
<td>21.285015</td>
</tr>
<tr>
<td>$10^3$</td>
<td>100</td>
<td>120</td>
<td>19.812217</td>
</tr>
<tr>
<td>$10^4$</td>
<td>100</td>
<td>120</td>
<td>19.865982</td>
</tr>
</tbody>
</table>

From Table 4.9, it is seen that the value of the American put options almost always become smaller as the number of simulations increase. We know that it is optimal to exercise when the value of the put option is lower than $K - S_0$ [11]. Hence, we can also see when $S_0 = 90$, $K = 100$ and $m = 10^3$ it is optimal to exercise early since the value of the option is lower than 10. This could also be seen when $S_0 = 100$, $K = 120$ and $m = 10^3, 10^4$. 

21
We also try to implement the approach to price the American call options. However, we see that the value of the option is always higher than $S_0 - K$, which means that it is never optimal to exercise early.
Conclusion

This thesis provides an overview of Multiscale Stochastic Volatility model for pricing American options. We have studied the application of several approaches, which are the Binomial tree and Least-squares Monte Carlo, on non-dividend American stock options. We have presented the results obtained by Python for the numerical implementations. In order to achieve that, we had to determine if the Least-squares Monte Carlo (LSM) approach was accurate by comparing it with a simpler model, the Binomial tree. Even though the approximated price was always higher for the LSM, we concluded that the results of the two models were much alike. This being done, we then constructed the simulation using Monte Carlo under Multiscale Stochastic Volatility model (MSVM) with the help of Heston model and interest rates model proposed by Cox, Ingersoll and Ross. The reason being that we found the same problematic as the processes include the square root diffusion for our $dV_1$ and $dV_2$.

Moreover, we apply the LSM into this model to find whether it is optimal to exercise the option or not at an early stage. The results we got from this simulation informed us when we should exercise early for an American put option as the price from LSM is lower than $K - S_0$. Although there should be more examples, the result we got for the American call options shows that it is better to exercise at maturity.

For further study, we would like to improve the accuracy of our experiment with real data numbers. The purpose being to investigate if our model is applicable to the real world of finance.
Bibliography


Appendix 1

Python code for Binomial Tree

"""
Binomial Tree for European/American Put/Call option
"""
import numpy as np

def BinomialTree(type, S0, K, mu, sigma, T, N, american="true"):
    # we improve the previous tree by checking for early exercise
    # for american options

    # calculate delta T
    deltaT = T / N

    # up and down factor will be constant for the tree so we
    # calculate outside the loop
    u = np.exp(sigma * np.sqrt(deltaT))
    d = 1.0 / u

    # to work with vector we need to init the arrays using numpy
    fs = np.asarray([0.0 for i in range(N + 1)])

    # we need the stock tree for calculations of expiration values
    fs2 = np.asarray([(S0 * u**j * d**(N - j)) for j in range(N + 1)])

    # we vectorize the strikes as well so the expiration check
    # will be faster
    fs3 = np.asarray([float(K) for i in range(N + 1)])

    # rates are fixed so the probability of up and down are fixed.
    # this is used to make sure the drift is the risk free rate
a = np.exp(mu * deltaT)
p = (a - d) / (u - d)
oneMinusP = 1.0 - p

# Compute the leaves, f_{N, j}
if type == "C":
    fs[:] = np.maximum(fs2 - fs3, 0.0)
else:
    fs[:] = np.maximum(-fs2 + fs3, 0.0)

# calculate backward the option prices
for i in range(N-1, -1, -1):
    fs[:i] = np.exp(-mu * deltaT) * (p * fs[i:1] + oneMinusP * fs[:i])
    fs2[:] = fs2[:] * u

    if american == 'true':
        # Simply check if the option is worth more alive or dead
        if type == "C":
            fs[:] = np.maximum(fs[:], fs2[:] - fs3[:])
        else:
            fs[:] = np.maximum(fs[:], -fs2[:] + fs3[:])

    # print fs
    return fs

print("Price: \%f" % BinomialTree('P', 80, 95, 0.05, 0.2, 1, 50))
Python code for pricing American put options using the LSM Approach

The codes below is based on the codes constructed by Jesus Perez Colino in https://github.com.

```python
import numpy as np

class AmericanOptionsLSMC:
S0 : float : initial stock/index level
K : float : strike price
T : float : time to maturity (in year fractions)
n : int : grid or granularity for time (in number of total points)
mu : float : constant risk-free short rate
div : float : dividend yield
sigma : float : volatility factor in diffusion term
m : number of simulation
"

    def __init__(self, option_type, S0, K, T, n, mu, div, sigma, m):
        try:
            self.option_type = option_type
            assert isinstance(option_type, str)
            self.S0 = float(S0)
            self.K = float(K)
            assert T > 0
            self.T = float(T)
            assert n > 0
            self.n = int(n)
            assert mu >= 0
            self.mu = float(mu)
            assert div >= 0
            self.div = float(div)
            assert sigma > 0
            self.sigma = float(sigma)
            assert m > 0
            self.m = int(m)
        except ValueError:
            print('Error passing Options parameters')
```

28
if option_type != 'call' and option_type != 'put':
    raise ValueError("Error: option_type not valid. Enter 'call' or 'put'")

if S0 < 0 or K < 0 or T <= 0 or mu < 0 or div < 0 or sigma < 0:
    raise ValueError("Error: Negative inputs not allowed")

LSMC=AmericanOptionsLSMC(option_type='put',
S0=80,
K=95,
T=1,
n=50,
mu=0.05,
div=0,
sigma=0.2,
m=1000)

def MCprice_matrix(self, seed = 123):
    """Returns MC price matrix rows: time columns: price-path simulation""

    time_unit = LSMC.T / float(LSMC.n)
    np.random.seed(seed)
    MCprice_matrix = np.zeros((LSMC.n + 1, LSMC.m), dtype=np.float64)
    MCprice_matrix[0,:] = LSMC.S0
    for t in range(1, LSMC.n + 1):
        brownian = np.random.standard_normal(LSMC.m / 2)
        brownian = np.concatenate((brownian, -brownian))
        MCprice_matrix[t, :] = (MCprice_matrix[t - 1, :]* np.exp((LSMC.mu - LSMC.sigma ** 2 / 2.) * time_unit +
                            LSMC.sigma * brownian * np.sqrt(time_unit)))
    return MCprice_matrix

a = MCprice_matrix(LSMC)
print(a)

def MCpayoff(self):
    """Returns the inner-value of American Option""

    MCpricematrix=np.array(MCprice_matrix(LSMC, seed = 123))
    print(MCpricematrix)
    if LSMC.option_type == 'call':
        payoff = np.maximum(MCpricematrix - LSMC.K,
                            np.zeros((LSMC.n + 1, LSMC.m), dtype=np.float64))
    else:
        payoff = np.maximum(MCpricematrix, np.zeros((LSMC.n + 1, LSMC.m), dtype=np.float64))
```python
payoff = np.maximum(LSMC.K - MCpricematrix, np.zeros((LSMC.n + 1, LSMC.m), dtype=np.float64))
return payoff

def value_vector(self):
    time_unit = LSMC.T / float(LSMC.n)
    MC_payoff=MCpayoff(LSMC)
    MCpricematrix=MCprice_matrix(LSMC, seed = 123)
    discount = np.exp(-LSMC.mu * time_unit)
    value_matrix = np.zeros_like(MC_payoff)
    value_matrix[-1, :] = MC_payoff[-1, :]
    for t in range(LSMC.n - 1, 0, -1):
        regression = np.polyfit(MCpricematrix[t, :], value_matrix[t + 1, :] * discount, 5)
        continuation_value = np.polyval(regression, MCpricematrix[t, :])
        value_matrix[t, :] = np.where(MC_payoff[t, :] > continuation_value, MC_payoff[t, :],
                                      value_matrix[t + 1, :] * discount)
    return value_matrix[1, :] * discount

def price(self):
    valuevector=value_vector(LSMC)
    return np.sum(valuevector) / float(LSMC.m)

print('price:%f' %price(LSMC))

Monte Carlo simulation under the Multiscale Stochastic Volatility Model

The codes bellow is based on the codes constructed by Stuart Reid in http://nbviewer.jupyter.org.

import math
import numpy
import random
import numpy.random as nrand
import matplotlib.pyplot as plt
```
class ModelParameters:
    
    """
    Encapsulates model parameters
    """

def __init__(self, 
    all_s0, all_time, all_delta, all_sigma1, all_sigma2, 
    gbm_mu, 
    all_r0=0.0, cir_rho13=0.0, cir_rho24=0.0, 
    heston_a1=0.0, heston_a2=0.0, heston_theta1=0.0, 
    heston_theta2=0.0, 
    heston_vol01=0.0, heston_vol02=0.0):
    
    # This is the starting asset value
    self.all_s0 = all_s0
    # This is the amount of time to simulate for
    self.all_time = all_time
    # This is the delta, the rate of time e.g. 1/252 = daily
    self.all_delta = all_delta
    # This is the volatility of the stochastic processes
    self.all_sigma1 = all_sigma1
    # This is the volatility of the stochastic processes
    self.all_sigma2 = all_sigma2
    # This is the annual drift factor for geometric brownian motion
    self.gbm_mu = gbm_mu
    # This is the correlation between the wiener processes of the
    Heston model
    self.cir_rho13 = cir_rho13
    # This is the correlation between the wiener processes of the
    Heston model
    self.cir_rho24 = cir_rho24
    # This is the rate of mean reversion for volatility in the
    Heston model
    self.heston_a1 = heston_a1
    # This is the rate of mean reversion for volatility in the
    Heston model
    self.heston_a2 = heston_a2
    # This is the long run average volatility for the Heston model
    self.heston_theta1 = heston_theta1
    # This is the long run average volatility for the Heston model
    self.heston_theta2 = heston_theta2
    # This is the starting volatility value for the Heston model
    self.heston_vol01 = heston_vol01
    # This is the starting volatility value for the Heston model
    self.heston_vol02 = heston_vol02
mp = ModelParameters(all_s0=100,
    all_time=100,
    all_delta=0.00396825396,
    all_sigma1=1,
    all_sigma2=0.316,
    gbm_mu=0.06,
    cir_rho13=0.5,
    cir_rho24=0.5,
    heston_a1=100,
    heston_a2=0.1,
    heston_theta1=0.4,
    heston_theta2=1,
    heston_vol01=0.3,
    heston_vol02=0.3)

paths = 100

def plot_stochastic_processes(processes, title):
    """
    This method plots a list of stochastic processes with a specified title.
    :return: plots the graph of the two
    """
    plt.style.use(['ggplot'])
    fig, ax = plt.subplots(1)
    fig.suptitle(title, fontsize=16)
    ax.set_xlabel('Number of time steps')
    ax.set_ylabel('Simulated Asset Price')
    x_axis = numpy.arange(0, len(processes[0]), 1)
    for i in range(len(processes)):
        plt.plot(x_axis, processes[i], linewidth=0.5)
    mean = numpy.mean(numpy.array(stochastic_volatility_examples), axis=0)
    plt.plot(numpy.arange(0, mp.all_time, 1), mean, 'p-', linewidth=4)
    plt.savefig(title, format='pdf')
    plt.show()

def heston_construct_correlated_path1(param, brownian_motion_one):
    """
    This method is a simplified version of the Cholesky decomposition method for just two assets. It does not make use of matrix algebra and is therefore quite easy to implement.
    :param param: model parameters object
    """
: return: a correlated brownian motion path

# We do not multiply by sigma here, we do that in the Heston model
sqrt_delta = math.sqrt(param.all_delta)
# Construct a path correlated to the first path
brownian_motion_three = []
for i in range(param.all_time - 1):
    term_one = param.cir_rh013 * brownian_motion_one[i]
    term_two = math.sqrt(1 - math.pow(param.cir_rh013, 2.0)) *
                random.normalvariate(0, sqrt_delta)
    brownian_motion_three.append(term_one + term_two)
return numpy.array(brownian_motion_one),
                 numpy.array(brownian_motion_three)

def heston_construct_correlated_path2(param, brownian_motion_two):
    ""
    This method is a simplified version of the Cholesky decomposition method
    for just two assets. It does not make use
    of matrix algebra and is therefore quite easy to implement.
    :param param: model parameters object
    :return: a correlated brownian motion path
    ""
# We do not multiply by sigma here, we do that in the Heston model
sqrt_delta = math.sqrt(param.all_delta)
# Construct a path correlated to the second path
brownian_motion_four = []
for i in range(param.all_time - 1):
    term_one = param.cir_rh024 * brownian_motion_two[i]
    term_two = math.sqrt(1 - math.pow(param.cir_rh024, 2.0)) *
                random.normalvariate(0, sqrt_delta)
    brownian_motion_four.append(term_one + term_two)
return numpy.array(brownian_motion_two),
                 numpy.array(brownian_motion_four)

def cox_ingersoll_ross_heston1(param):
    ""
    This method returns the rate levels of a mean-reverting cox
    inger s oll ross process. It is used to model interest rates as well as
    stochastic volatility in the Heston model. Because the returns between
    the underlying and the stochastic volatility should be correlated
    we pass a correlated Brownian motion process into the method
    from which the interest rate levels are constructed. The other
correlated process is used in the Heston model
param param: the model parameters objects
:return: the interest rate levels for the CIR process

# We don't multiply by sigma here because we do that in heston
sqrt_delta_sigma1 = math.sqrt(param.all_delta) * param.all_sigma1
brownian_motion_volatility = nrnd.normal(loc=0, scale=sqrt_delta_sigma1, size=param.all_time)
a1, theta1, zero1 = param.heston_a1, param.heston_theta1, param.heston_vol01
volatilities1 = [zero1]
for i in range(1, param.all_time):
    drift = a1 * (theta1 - volatilities1[i-1]) * param.all_delta
    randomness = param.all_sigma1 * math.sqrt(max(volatilities1[i-1], zero1)) * brownian_motion_volatility[i-1]
    volatilities1.append(max(volatilities1[i-1], zero1) + drift + randomness)
return numpy.array(brownian_motion_volatility),
numpy.array(volatilities1)

def cox_ingersoll_ross_heston2(param):

    This method returns the rate levels of a mean-reverting cox ingersoll ross process. It is used to model interest rates as well as stochastic volatility in the Heston model. Because the returns between the underlying and the stochastic volatility should be correlated we pass a correlated Brownian motion process into the method from which the interest rate levels are constructed. The other correlated process is used in the Heston model
:param param: the model parameters objects
:return: the interest rate levels for the CIR process

# We don't multiply by sigma here because we do that in heston
sqrt_delta_sigma2 = math.sqrt(param.all_delta) * param.all_sigma2
brownian_motion_volatility = nrnd.normal(loc=0, scale=sqrt_delta_sigma2, size=param.all_time)
a2, theta2, zero2 = param.heston_a2,
param.heston_theta2, param.heston_vol02
volatilities2 = [zero2]
for i in range(1, param.all_time):
    drift = a2 * (theta2 - volatilities2[i-1]) * param.all_delta
    randomness = param.all_sigma2 * math.sqrt(max(volatilities2[i-1], 0.05)) *
brownian_motion_volatility[i - 1]
volatilities2.append(max(volatilities2[i - 1], 0.05) + drift
+ randomness)
return numpy.array(brownian_motion_volatility),
numpy.array(volatilities2)

def heston_model_levels1(param):
    ""
    NOTE -- this method is dodgy! Need to debug!
    The Heston model is the geometric brownian motion model with
    stochastic volatility. This stochastic volatility is given by
    the cox ingersoll ross process. Step one on this method is to
    construct two correlated GBM processes. One is used for the
    underlying asset prices and the other is used for the stochastic
    volatility levels
:param param: model parameters object
:return: the prices for an underlying following a Heston process
    ""
    assert isinstance(param, ModelParameters)
    # Get two correlated brownian motion sequences for the volatility
    # parameter and the underlying asset
    # brownian_motion_market,
    # brownian_motion_vol = get_correlated_paths_simple(param)
brownian, cir_process1 = cox_ingersoll_ross_heston1(param)
brownian, brownian_motion_market1 =
heston_construct_correlated_path1(param, brownian)

heston_market_price_levels = [param.all_s0]
for i in range(1, param.all_time):
    drift = param.gbm_mu * heston_market_price_levels[i - 1] *
    param.all_delta
    voll = cir_process1[i - 1] * heston_market_price_levels[i - 1] *
    brownian_motion_market1[i - 1]
    heston_market_price_levels.append(heston_market_price_levels[i - 1] +
    drift + voll)
return numpy.array(heston_market_price_levels),
numpy.array(cir_process1)

stochastic_volatility_examples = []
for i in range(paths):
    stochastic_volatility_examples.append(
    heston_model_levels1(mp)[0])
plot_stochastic_processes(stochastic_volatility_examples,
stochastic_volatility_examples = []
for i in range(paths):
    stochastic_volatility_examples.append(
        heston_model_levels1(mp)[1])
plot_stochastic_processes(stochastic_volatility_examples, 
"Heston volatility level with v1")

def heston_model_levels2(param):
    ""
    NOTE — this method is dodgy! Need to debug!
    The Heston model is the geometric brownian motion model with
    stochastic volatility. This stochastic volatility is given by
    the cox ingersoll ross process. Step one on this method is to
    construct two correlated GBM processes. One is used for the
    underlying asset prices and the other is used for the stochastic
    volatility levels
    :param param: model parameters object
    :return: the prices for an underlying following a Heston process
    ""
    assert isinstance(param, ModelParameters)
    # Get two correlated brownian motion sequences for the volatility
    # parameter and the underlying asset
    # brownian motion vol
    # brownian_motion_vol = get_correlated_paths_simple(param)
brownian, cir_process2 = cox_ingersoll_ross_heston2(param)
brownian, brownian_motion_market2 = heston_construct_
correlated_path2(param, brownian)

heston_market_price_levels = [param.all_s0]
for i in range(1, param.all_time):
    drift = param.gbm_mu * heston_market_price_levels[i - 1] * 
    param.all_delta
    vol2 = cir_process2[i - 1] * 
    heston_market_price_levels[i - 1] * 
    brownian_motion_market2[i - 1]
    heston_market_price_levels.append( 
        heston_market_price_levels[i - 1] + drift + vol2)
return numpy.array(heston_market_price_levels), 
    numpy.array(cir_process2)
stochastic_volatility_examples = []
for i in range(paths):
    stochastic_volatility_examples.append(heston_model_levels2(mp)[0])
plot_stochastic_processes(stochastic_volatility_examples,
    "Heston price level with \nu2")

stochastic_volatility_examples = []
for i in range(paths):
    stochastic_volatility_examples.append(heston_model_levels2(mp)[1])
plot_stochastic_processes(stochastic_volatility_examples,
    "Heston volatility level with \nu2")

def MSVM(param):
    ""
    NOTE – this method is dodgy! Need to debug!
The Heston model is the geometric brownian motion model with stochastic volatility. This stochastic volatility is given by the cox ingersoll ross process. Step one on this method is to construct two correlated GBM processes. One is used for the underlying asset prices and the other is used for the stochastic volatility levels
:param param: model parameters object
:return: the prices for an underlying following a Heston process
    ""
    assert isinstance(param, ModelParameters)
    # Get two correlated brownian motion sequences for the volatility
    # parameter and the underlying asset
    # brownian_motion_market, brownian_motion_vol =
    # get_correlated_paths_simple(param)
    brownian, cir_process1 = cox_ingersoll_ross_heston1(param)
    brownian, brownian_motion_market1 =
        heston_construct_correlated_path1(param, brownian)
    brownian, cir_process2 = cox_ingersoll_ross_heston2(param)
    brownian, brownian_motion_market2 =
        heston_construct_correlated_path2(param, brownian)

    heston_market_price_levels = [param.all_s0]
    for i in range(1, param.all_time):
        drift = param.gbm_mu * heston_market_price_levels[i - 1] *
            param.all_delta
        vol1 = cir_process1[i - 1] *
heston_market_price_levels[i - 1] *
brownian_motion_market1[i - 1]

vol2 = cir_process2[i - 1] *
heston_market_price_levels[i - 1] *
brownian_motion_market2[i - 1]

heston_market_price_levels.append(
    heston_market_price_levels[i - 1] + drift + vol1 + vol2)

return numpy.array(heston_market_price_levels),
    numpy.array(cir_process1), numpy.array(cir_process2)

stochastic_volatility_examples = []
for i in range(path):
    stochastic_volatility_examples.append(MSM(mp)[0])
plot_stochastic_processes(stochastic_volatility_examples ,
    "MSVM price level with v1 & v2")

stochastic_volatility_examples = []
for i in range(path):
    stochastic_volatility_examples.append(MSM(mp)[1])
stochastic_volatility_examples.append(MSM(mp)[2])
plot_stochastic_processes(stochastic_volatility_examples ,
    "MSVM volatility level with v1 & v2")