ALTERNATIVE APPROACHES TO THE LEAST SQUARE METHOD IN AMERICAN OPTION PRICING

by

Tunç Utku
Bachelor of Engineering, Galatasaray University, 2016

PROJECT SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE IN FINANCE

In the Master of Science in Finance Program of the Faculty of Business Administration

© Tunç Utku 2019
SIMON FRASER UNIVERSITY
Fall 2019

All rights reserved. However, in accordance with the Copyright Act of Canada, this work may be reproduced, without authorization, under the conditions for Fair Dealing. Therefore, limited reproduction of this work for the purposes of private study, research, criticism, review and news reporting is likely to be in accordance with the law, particularly if cited appropriately.
Approval

Name: Tunç Utku

Degree: Master of Science in Finance

Title of Project: Alternative Approaches to The Least Square Method in American Option Pricing

Supervisory Committee:

______________________________
Professor Andrey Pavlov
Senior Supervisor

______________________________
Dr. Phil Goddard
Second Reader

Date Approved: ___________________________
Abstract

This paper introduces alternative methods to least square method (LSM) implemented by Longstaff-Schwartz in 2001 to enhance the pricing of American options with Monte Carlo Simulation. The goal is to provide evidence that alternative methods provide more precise pricing compared to least square method.

The alternative methods include various machine learning (ML) algorithms classified as regression models: artificial neural networks (ANN), decision trees, support vector machines (SVM), stochastic gradient descent (SGD), isotonic regression, and Gaussian Process Regression (GPR).

As a part of the calibration process, real market data used for the Merton jump diffusion model and CIR model. Finally, the paper compares errors with each ML algorithms to arrive to the most appropriate algorithm.

Keywords: American option pricing, Monte Carlo Simulation, LSM, calibration, stochastic interest rate, jump-diffusion, CIR, ML, Longstaff-Schwartz
Acknowledgments

Foremost, I would like to express my sincere gratitude to my advisors Dr. Phil Goddard and Prof. Andrey Pavlov, for their support not only during the research phase but also throughout the MSc Finance program where I equipped with the fundamental knowledge and desire to pursue this paper.
Table of Contents

Approval ................................................................................................................................. vii
Abstract................................................................................................................................. viii
Acknowledgments .................................................................................................................. ix
Table of Contents .................................................................................................................... x
List of Figures .......................................................................................................................... xi
List of Tables ............................................................................................................................ xii

1: Introduction .......................................................................................................................... 1

2: Least-Square Method for American Option Pricing ............................................................. 2
  2.1 Valuation Framework ...................................................................................................... 2
  2.2 LSM Algorithm ............................................................................................................. 2

3: Alternative Methods for American Option Pricing (Scikit-learn, 2019) ........................... 4
  3.1 Supervised Learning ..................................................................................................... 4
    3.1.1 Artificial Neural Networks (“ANN”) ................................................................. 4
    3.1.2 Decision Trees .................................................................................................... 5
    3.1.3 Support Vector Machines ................................................................................... 7
    3.1.4 Stochastic Gradient Descent .............................................................................. 8
    3.1.5 Isotonic Regression ............................................................................................. 9
    3.1.6 Gaussian Process Regression ............................................................................. 10

4: Calibration and Pricing ....................................................................................................... 12
  4.1 Methodology and Framework ....................................................................................... 12
    4.1.1 Discretization ...................................................................................................... 13
  4.2 Calibration ..................................................................................................................... 14
    4.2.1 Data Gathering and Preparation ......................................................................... 15
    4.2.2 Calibration Results ............................................................................................ 16
  4.3 Simulation ..................................................................................................................... 17
  4.4 Pricing ........................................................................................................................... 19
    4.4.1 QuantLib ............................................................................................................ 19

5: Conclusion .......................................................................................................................... 20
  5.1 Evaluation ..................................................................................................................... 21

6: Next Steps ........................................................................................................................... 22

Appendices .............................................................................................................................. 23
  Appendix A: Script for American Option Pricing ............................................................... 23
  Appendix B: Functions for Calibration ............................................................................. 27
  Appendix C: Functions for Calibration ............................................................................. 30
  Appendix C: Function for Valuation ................................................................................. 34
  Appendix C: Quantlib Function ......................................................................................... 37

Works Cited .............................................................................................................................. 38
List of Figures

Figure 1: Artificial Neural Networks Illustration .................................................................................. 4
Figure 2: Decision Tree Illustration ...................................................................................................... 6
Figure 3: Support Vector Machines Illustration ..................................................................................... 7
Figure 4: Stochastic Gradient Descent Illustration ................................................................................. 9
Figure 5: Isotonic Regression Illustration ............................................................................................. 10
Figure 6: Gaussian Process Regression Illustration ............................................................................... 11
Figure 7: Spot Curve Generated by B-Spline .......................................................................................... 16
Figure 8: Illustration of Uncorrelated and Correlated Data ................................................................... 17
Figure 9: Interest Rates Paths Generated by CIR Model ....................................................................... 18
Figure 10: Index Paths Generated by Merton’s Jump Model ................................................................. 18
Figure 11: Index Distribution after Simulation ....................................................................................... 19
Figure 12: Option Prices with Various Strike Prices .............................................................................. 20
Figure 13: Average Errors by Each Method ........................................................................................... 20
Figure 14: Durations by Each Method .................................................................................................... 21
List of Tables

Table 1: Bond Yields used in CIR Calibration ................................................................. 15
Table 2: Calibration Parameters for CIB ......................................................................... 16
Table 3: Calibration Parameters for Merton’s Jump Diffusion ........................................ 17
1: Introduction

In contrast to European options, American options give the option holder the flexibility to exercise the option any time before the maturity. Compared to various numerical methods for option pricing, Monte Carlo simulation is an appropriate method to price American options since they are path depended. However, Monte Carlo simulation requires a lot of computational as well as pricing challenges.

Longstaff-Schwartz introduced a pricing method (Longstaff & Schwartz, 2001) to overcome above-mentioned challenges by integrating the least square method in each time step to calculate expected payoffs in the Monte Carlo Simulation. This provides computational advantages; however, it introduces approximation error from least-square calculation. In order to decrease the error and provide more accurate pricing, this paper explores alternative machine learning algorithms.

The paper consists of several sections. Section 2 covers the explanation of the pricing methodology of Longstaff-Schwartz. Section 3 introduces alternative methods to the least square methods such as neural networks, decision trees, support vector machines, stochastic gradient descents, isotonic regression, and Gaussian Process Regression. Section 4 elaborates on the calibration methods and simulation of the option with market data. Section 5 gives the prices calculated with eight methods including the original method which is least squares and compares the methods by the error with actual market prices. Finally, section 6 proposes further steps to enhance the existing project and eventually enable to calculate more accurate prices.
2: Least-Square Method for American Option Pricing

The least square method is introduced by Longstaff-Schwartz in their paper published in 2001.

2.1 Valuation Framework


\[ V_0 = E_0^Q (B_0(T)V_T) \]  
\[ V_t(s) = \max [h_t(s), C_t(s)] \]

\( E \) is the expectation operator and \( Q \) is the unique risk-neutral probability measure. In addition, \( h_t(s) \) is the maximum payoff and \( C_t(s) \) is the expected payoff. The major insight of Longstaff-Schwartz is to estimate the continuation values \( \hat{C}_{t,i} \) by ordinary least-squares regression

\[ \hat{C}_{t,i} = \sum_{d=1}^{d} \alpha_{d,t}^* * b_d(S_{t,i}) \]

2.2 LSM Algorithm

The objective of the LSM algorithm is to provide a pathwise approximation to the optimal stopping rule that maximizes the value of the American option. The LSM
approach uses least squares to approximate the conditional expectation function at $t_{k-1}$, $t_{k-2}$, $t_{k-3} \ldots t_1$. The minimization is given below.

$$
\min_{\alpha_{t_1}, \ldots, \alpha_{t_D}} \frac{1}{I} \sum_{i=1}^{I} (Y_{t,i} - \sum_{d=1}^{D} \alpha_{d,t} \ast b_d(S_{t,i}))^2
$$

(4)

The quality of the regression can be improved upon when restricting the paths involved in the regression to those where the option is in-the-money.
3: Alternative Methods for American Option Pricing (Scikit-learn, 2019)

3.1 Supervised Learning

Supervised learning is a learning model built to make a prediction, given an unforeseen input instance. A supervised learning algorithm takes a known set of input datasets and its known responses to the data (output) to learn the regression/classification model (Shobha & Rangaswamy, 2018). Supervised learning algorithms consist of 2 different techniques: linear regression and classification. Due to the structure of American options, regression techniques are appropriate for valuation.

3.1.1 Artificial Neural Networks (“ANN”)

Given a set of features, $X = x_1, x_2, ..., x_m$, and a target $y$, it can learn a non-linear function approximator for regression. In contrast to logistic regression, neural networks has one or more non-linear layers, called hidden layers. Figure below illustrates the concept of ANN.

![Artificial Neural Networks Illustration](image)
The leftmost layer, known as the input layer, consists of a set of neurons \( \{ x_1, x_2, \ldots, x_m \} \) representing the input features. Each neuron in the hidden layer transforms the values from the previous layer with a weighted linear summation \( w_1 x_1 + w_1 x_1 + w_1 x_1 + \cdots + w_m x_m \), followed by a non-linear activation function. The output layer receives the values from the last hidden layer and transforms them into output values.

The advantages of Multi-layer Perceptron are:

- Capability to learn non-linear models.
- Capability to learn models in real-time (on-line learning)

The disadvantages of Multi-layer Perceptron (MLP) include:

- MLP with hidden layers has a non-convex loss function where there exists more than one local minimum. Therefore different random weight initializations can lead to different validation accuracy.
- MLP requires tuning a number of hyper parameters such as the number of hidden neurons, layers, and iterations.

3.1.2 Decision Trees¹

Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. For instance, in the example below, decision trees learn from data to approximate a sine curve.

with a set of if-then-else decision rules. The deeper the tree, the more complex the decision rules and the fitter the model.

Figure 2: Decision Tree Illustration

The advantages of the decision tree method include:

- Simple to interpret and visualized
- It doesn’t need any normalization, addition of dummy variables, or other complex data preparation stages.
- Model validation and explainability are higher compared to other methods.

The disadvantages of the decision trees method include:

- Small variations in data can cause the regression change significantly, which can cause instability.
- Higher chance of over fitting in comparison to other methods.
3.1.3 Support Vector Machines

The support-vector machine constructs a hyperplane or set of hyperplanes that can be used for regression (saedsayad, 2019). Intuitively, a good separation is achieved by the hyperplane that has the largest distance to the nearest training-data point of any class (so-called functional margin), since, in general, the larger the margin, the lower the generalization error of the classifier. An example of a support vector machine regression is given below.

Figure 3: Support Vector Machines Illustration

\[ y_i = \langle w, x_i \rangle + b + \varepsilon \]

\[ y_i = \langle w, x_i \rangle + b - \varepsilon \]

\[ \min \frac{1}{2} \|w\|^2 + C \sum_{i=1}^{N} (\xi_i + \xi_i^*) \]
The constraints:

\[ y_i - wx_i - b \leq \xi_i + \xi_i^* \]  

\[ wx_i + b - y_i \leq \xi_i + \xi_i^* \]  

\[ \xi_i, \xi_i^* \geq 0 \]

Advantages of Support Vector Machines are:

- Effective in high dimensional spaces
- Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.

The disadvantages of support vector machines:

- Overfitting in case the number of features is much greater than the number of samples

### 3.1.4 Stochastic Gradient Descent

In stochastic gradient descent, a few samples are selected randomly instead of the whole data set for each iteration. In Gradient Descent, there is a term called “batch” which denotes the total number of samples from a dataset that is used for calculating the gradient for each iteration. In typical Gradient Descent optimization, like Batch Gradient Descent, the batch is taken to be the whole dataset. Although, using the whole dataset is really useful for getting to the minima in a less noisy or less random manner, but the problem arises when our datasets get really huge.

The advantages of Stochastic Gradient Descent are:
- Efficiency.
- Ease of implementation (lots of opportunities for code tuning).

The disadvantages of Stochastic Gradient Descent include:

- SGD requires a number of hyperparameters such as the regularization parameter and the number of iterations.
- SGD is sensitive to feature scaling.

An example of SGD is given below.

![Figure 4: Stochastic Gradient Descent Illustration](image)

### 3.1.5 Isotonic Regression

Isotonic regression fits a non-decreasing function data by solving the following equation:

\[
\min \sum_i w_i (y_i - \hat{y}_i)^2
\]  

(9)

\[
\hat{y}_{min} = \hat{y}_1 \leq \hat{y}_2 \leq ... \leq \hat{y}_n \leq \hat{y}_{max}
\]  

(10)

A comparison between linear regression and isotonic regression is given below.
The algorithm sweeps through the data looking for violations of the monotonicity constraint. When it finds one, it adjusts the estimate to the best possible fit with constraints. Sometimes it also needs to modify previous points to make sure the new estimate does not violate the constraints.²

The advantage of isotonic regression is:

- Fast and simple

The disadvantage of isotonic regression is:

- One or several points at the ends of the interval are sometimes noisy

3.1.6 **Gaussian Process Regression**

Gaussian Processes (GP) are a generic supervised learning method designed to solve regression and probabilistic classification problems.

² [http://fa.bianp.net/blog/2013/isotonic-regression/](http://fa.bianp.net/blog/2013/isotonic-regression/)
The advantages of Gaussian processes are:

- The prediction interpolates the observations (at least for regular kernels).
- The prediction is probabilistic (Gaussian) so that one can compute empirical confidence intervals and decide based on those if one should refit (online fitting, adaptive fitting) the prediction in some region of interest.
- Versatile: different kernels can be specified. Common kernels are provided, but it is also possible to specify custom kernels.

The disadvantages of Gaussian processes include:

- They are not sparse, i.e., they use the whole samples/features information to perform the prediction.
- They lose efficiency in high dimensional spaces – namely when the number of features exceeds a few dozens.

An illustration of the GPR process is given below.

*Figure 6: Gaussian Process Regression Illustration*
4: Calibration and Pricing

Calibration and pricing of the American option consist of Merton’s jump-diffusion model (Merton, 1976) as well as CIR stochastic interest rate models (Cox, Ingersoll, & Ross, 1985). First of all, in order to use the models in a Monte Carlo simulation, discretization is needed. The next step is calibrating the market data, and the final step is to simulate asset and interest rate paths.

4.1 Methodology and Framework

The framework consists of 2 parts. The first part includes Merton’s jump-diffusion model, which allow to reflect the discontinuities in returns. The model incorporates Poisson distributions to simulate log-normally distributed jumps in returns. The diffusion equation is given below.

\[ dS = Sr \, dt + S \sigma \, Z + J \, dP \] (11)

In the above-given equation dP is a Poisson process, and J represents the magnitude of a jump. In addition, J is assumed to be log-normally distributed with \( \log(J) \sim N(\mu, \sigma_J^2/2) \). The definition of \( \mu \) is given in equation 11. The Poisson process defines the probability of exactly \( m \) jumps occurring over the time interval \( dt \). The equation of the Poisson distribution is given below.

\[ P(M, \lambda dt) = \frac{e^{-\lambda dt} (\lambda dt)^m}{m!} \] (12)

Regarding the stochastic interest rate, the model uses Cox, Ingersoll and Ross with the following dynamics.
\[ dr_t = kr(\theta r - r_t)dt + \sigma r\sqrt{r_t}dW_t \]  

(13)

### 4.1.1 Discretization

Monte Carlo simulation requires discretization (Hilpisch, 2015) for simulating interest rates as well as index paths. The discretization of Merton’s jump-diffusion and CIR model is given below.

\[ S_{t+dt} = S_t e^{(\bar{u} dt + \sigma \sqrt{\Delta t} \epsilon)} e^{(m \mu_j + \sigma_j \int_{i=1}^{m} \epsilon_i^j)} \]  

(14)

\[ \mu_j = r_j - \sigma^2/2 \]  

(15)

\[ \hat{\mu}_j = \hat{r}_j - \sigma_j^2/2 \]  

(16)

\[ \hat{r} = r - \lambda(e^{\hat{r}_j} - 1) \]  

(17)

\[ \tilde{r}_t = \tilde{r}_s + k_r(\theta_r - \tilde{r}_r)\Delta t + \sigma_r \sqrt{\tilde{r}_r} \sqrt{\Delta t} z_t^2 \]  

(18)

\[ r_t = \tilde{r}_t^+ \]  

(19)

\[ \bar{r}_t \equiv (r_t + r_s)/2 \]  

(20)

The meanings of the variables and parameters are:

- \( S_t \) index level at date \( t \)
- \( r_t \) risk-less short rate at date \( t \)
- \( k_r \) speed of adjustment of \( r_t \)
• $\theta_r$ the long-term average of the short rate

• $\sigma_r$ volatility coefficient of the short rate

• $Z_t$ standard Brownian motions

4.2 Calibration

The calibration enables to incorporate market indicators to the simulation process. The calibration process consists of using market data and fitting the best model by running an optimizer and finding the optimum parameters to be used in the Monte Carlo simulation.

The calibration of the CIR model (Hilpisch, 2015) for stochastic interest rates from zero-coupon prices to get $\kappa r, \theta r, \sigma r$. To calibrate the CIR model to the forward rates, the MSE of the market-implied, and model implied forward rate curve at selected discrete points in time is minimized (Brandimarte, 2006). The optimization function is given below.

$$\min_{\alpha} \frac{1}{M} \sum_{m=0}^{M} (f(0, m\Delta t) - f^{CIR85}(0, t; \alpha))^2$$

$$(21)$$

$$f^{(CIR85)}(0, t; \alpha) = \frac{K_r \theta_r (e^{\gamma t} - 1)}{2\gamma + (K_r + \gamma)(e^{\gamma t} - 1)} + \frac{r_0}{2\gamma + (K_r + \gamma)(e^{\gamma t} - 1)} \frac{4\gamma^2 e^{\gamma t}}{(2\gamma + (K_r + \gamma)(e^{\gamma t} - 1))^2}$$

$$\gamma \equiv \sqrt{K^2_r + 2\sigma^2_r}$$

$$(22)$$

Calibration of Merton Jump model for the jumps and index prices from historical index prices to get $r_j, \hat{r}_j, \sigma, \sigma_j, \lambda$. Maximum likelihood estimation is a common technique
used for calibrating models to market data. To simplify the maximum likelihood function, Bernoulli discrete distribution (Ball & Torous, 1983) can be used, which is given below.

\[
\min = (1 - \lambda) \phi(x; \hat{\mu}, \sigma^2) + \lambda \phi(x; \hat{\mu} + \mu_j, \sigma^2 + \sigma_j^2) \quad (24)
\]

4.2.1 Data Gathering and Preparation

In terms of data gathering, zero-coupon bond data, as well as OIS data, used to calibrate CIR interest rate parameters. Yields of the bonds and their duration are given below.

<table>
<thead>
<tr>
<th>Date</th>
<th>1d</th>
<th>1w</th>
<th>1m</th>
<th>3m</th>
<th>6m</th>
<th>12m</th>
<th>24m</th>
</tr>
</thead>
<tbody>
<tr>
<td>15-11-2019</td>
<td>1.54</td>
<td>1.59</td>
<td>1.56</td>
<td>1.57</td>
<td>1.59</td>
<td>1.54</td>
<td>1.61</td>
</tr>
</tbody>
</table>

In addition, the SPY index price (SPDR S&P 500 ETF) used to calibrate Merton’s jump-diffusion model parameters. The index prices are exported from Yahoo Finance. The time period for the index prices is 1 year starting on November 16, 2018 to November 15, 2019.

Finally, to compare generated options prices with the options traded on the market, data on SPY call and put options are used. The market data is taken from Yahoo Finance. For market quotes, the settlement date is selected as November 15, and for expiry date is one year from that day.
4.2.2 Calibration Results

From given government bonds traded in the market, the spot curve has been generated by using an interpolation method called “B-spline”. Curve building is an essential part of the calibration of the CIR model. The spot curve is given below.

![Figure 7: Spot Curve Generated by B-Spline](image)

CIR calibration runs an optimization as mentioned above and finds the optimal values for $k_r, \theta_r$, and $\sigma_r$ which are given in the table below.

<table>
<thead>
<tr>
<th>Table 2: Calibration Parameters for CIB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variables</td>
</tr>
<tr>
<td>Values</td>
</tr>
</tbody>
</table>

Calibration of Merton’s jump-diffusion model also runs an optimization and calculates the optimum values for $r_j, \tilde{r}_j, \sigma, \sigma_j$, and $\lambda$ which are given below.
Table 3: Calibration Parameters for Merton’s Jump Diffusion

<table>
<thead>
<tr>
<th>Variables</th>
<th>$r_j$</th>
<th>$\hat{r}_j$</th>
<th>$\sigma$</th>
<th>$\sigma_j$</th>
<th>$\lambda$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Values</td>
<td>0.1313</td>
<td>0.1189</td>
<td>0.2134</td>
<td>0.1786</td>
<td>6.16708e-06</td>
</tr>
</tbody>
</table>

4.3 Simulation

As mentioned in the introduction, the Monte Carlo simulation is an efficient and flexible method to evaluate financial models and derivative pricing formulas numerically. The first step when valuing derivative instruments via MCS is to discretize the stochastic differential equations (SDE) that govern the dynamics of a given model.

As mentioned in the above sections, the simulation includes two different random numbers that are used to simulate the index price paths as well as interest rate paths. Since there is a correlation between the interest rates and index prices, Cholesky decomposition is used to generate correlated paths. The Cholesky decomposition transforms a vector of uncorrelated normally distributed random variables to correlated normally distributed random variables. The distribution figure of uncorrelated and correlated data is given below.

Figure 8: Illustration of Uncorrelated and Correlated Data
The next step of the valuation is running the simulation. The Monte Carlo Simulation for interest rates, which are based on the CIR model, consists of 100 paths and 252-time steps. The generated interest rate paths are given below.

Figure 9: Interest Rates Paths Generated by CIR Model

The Monte Carlo Simulation for S&P index prices, which are based on Merton’s jump-diffusion model, consists of 100 paths and 252-time steps. The generated index paths are given below.

Figure 10: Index Paths Generated by Merton’s Jump Model
As a part of the valuation framework, the Monte Carlo simulation is based on log-normally distributed index prices. The histogram showing the distribution of the final index prices is given below.

![Histogram of Final Stock Prices](image)

**Figure 11: Index Distribution after Simulation**

4.4 Pricing

To compare the consistency of each ML algorithms, several strike prices are chosen. For American call option pricing, 41 strike prices are used, which are between 250 to 350. For American put option pricing, 38 strike prices are used, which are between 250 to 350. The valuation framework is based on the methodology given in Langstaff-Schwartz paper; however, the payoff estimation differs per ML algorithm.

4.4.1 QuantLib

To compare the performance of the ML algorithms with existing models, a custom built-in function (Balaraman, 2017) used, which is written by Goutham Balaraman. The custom function computes the value of an American option by a Binomial Engine using the CRR approach.
5: Conclusion

The prices calculated for eight methods, including the least-square method and Quanlib function, are given below. The spot prices for the same option is also included.

*Figure 12: Option Prices with Various Strike Prices*

The difference between the spot price and the calculated option price for a strike price is the estimated error. To generalize the methods, this paper uses the average for each strike price. The average error between the spot price as well eight methods are given below.

*Figure 13: Average Errors by Each Method*
Finally, to assess the performance of each method, the processing time is calculated between the beginning and the ending of each method. The durations are given below.

![Duration by Method](image)

**Figure 14: Durations by Each Method**

5.1 Evaluation

Based on the average errors, Quantlib function generates a more consistent price. Following Quantlib function, the most appropriate method is isotonic regression among ML methods. In terms of timing, the GPR requires the most computational power since there is a significant timing difference between other methods.

Based on the results given above, there is a significant difference between the prices generated with ML algorithms (including LSM) and market quotes. A re-evaluation of the pricing structure used in the project could increase the consistency of the prices generated.
6: Next Steps

The next steps consist of 2 parts. The first part includes the improvements in the existing model used as a part of this paper. The second part covers the additional steps to take to provide more consistency.

As mentioned in the conclusion section, there is a gap between calculated prices and actual prices. A re-evaluation of the framework applied in the model, as well as the initial points assigned to variables during the calibration process would increase the consistency of the final prices.

Secondly, instead of the CIR model, Brennan and Schwartz two factor interest rate model (Brennan & Schwartz, 1982) provides an alternative framework to simulate interest rates. With that perspective, it is also possible to value not only index options but also derivatives that are based on interest rates, such as interest rate futures options.

Finally, dynamic delta hedging is a perfect method to hedge against price changes of a derivative instrument. Therefore calculating the price sensitivities with alternative ML algorithms constitutes another potential research area.
Appendices

Appendix A: Script for American Option Pricing

1. # Import Libraries
2. import numpy as np
3. import matplotlib.pyplot as plt
4. import time
5. import pandas as pd
6. 
7. # Import necessary functions and variables
8. from Simulation import AssetPaths, SRD_generate_paths, random_number_generator_custom
9. from Calibration import FitJumpDiffusionModel, CIR_calibration
10. from Calibration import r0, index_returns, S0, f, rho # Variables
11. from Valuation import option_pricing
12. from Quantlib import quantlibprice
13. 
14. 
15. Option Specs
16. 
17. 
18. # "put" or "call"
19. option_type = "put"
20. 
21. # Variance reduction techniques: anti_paths or halton
22. variance_reduction = "anti_paths"
23. 
24. 
25. Model Inputs
26. 
27. 
28. 
29. 
30. # Set stock generation parameters
31. T = 1 # maturity
32. dt = 1/252 # set dt as one day
33. nPaths = 10000 # number of paths
34. steps = int(T/dt) # number of steps
35. 
36. 
37. 
38. Model Calibration
39. 
40. 
41. 
42. 
43. # Calibration of Jump Parameters
44. param = np.zeros(5)
45. param = FitJumpDiffusionModel(index_returns, dt)
46. 
47. 
48. # Calibration of Interest Rate Parameters
49. kappa_r, theta_r, sigma_r = CIR_calibration()
50. 
51. 
52. Simulation
53. 
54. 

# Create Random Numbers
```python
rand = random_number_generator_custom(nPaths,steps,variance_reduction)
```

# Simulate interest rates (CIR Model)
```python
r = SRD_generate_paths(r0, kappa_r, theta_r, sigma_r, T, steps, nPaths, rand, rho)
```

# Simulate Stock Prices
```python
S = AssetPaths(S0,param,dt,steps,nPaths,rand,rho)
```

""" Valuation """
```python
# Set Strike Prices
if option_type == "put":
    Market_Data = pd.read_csv("SPDR_Put.csv")
elif option_type == "call":
    Market_Data = pd.read_csv("SPDR_Call.csv")

# Create variable arrays
n_options = Market_Data.shape[0]
price = np.zeros((9,n_options))
times = np.zeros((8,n_options))
price_diff = np.zeros((8,n_options))

# "LS" Least Square
# "NN" Neural Networks
# "DT" Decision Tree
# "SVM" Support vector machines
# "ISO" Isotonic Regression
# "GPR" Gaussian Processes Regression
method_type = ["Spot","Quantlib","LS", "NN", "DT", "SVM", "SGD", "ISO", "GPR"]

# Value options with various regression techniques and strike prices
for idx in range(Market_Data.shape[0]):
    # Set strike price for each option
    K = Market_Data["Strike"][idx]
    market_vol = Market_Data["Volatility"][idx]
    Market_Quote = Market_Data["Last Price"][idx]
    price[0,idx] = Market_Quote

    # Price the option with a built-in function in Quantlib
    start_time = time.time()
    price[1,idx] = quantlibprice(float(K),option_type,float(market_vol))
    price_diff[0,idx] = np.absolute(Market_Quote - price[1,idx])
    end_time = time.time()
    times[0,idx] = end_time - start_time

    # Price the option price with each regression method
    for mdx in range(len(method_type)-2):
        # Price the option given asset & interest rate paths and other parameters
        start_time = time.time()
price[mdx+2,idx] = option_pricing(S,K,r,dt,steps,nPaths,option_type,method_type[mdx+2])
end_time = time.time()

# Calculate time to execute each valuation method
times[mdx+1,idx] = end_time - start_time

# Calculate the difference between market quote and custom prices
price_diff[mdx+1,idx] = np.absolute(Market_Quote - price[mdx+2,idx])

# Calculate average price error and time for each regression method
Average_Errors = np.mean(price_diff, axis=1)
Average_Duration = np.mean(times, axis=1)

"""Visualization of the interest rate, stock paths and code results
"""

# Plot Spot Rate Curve
plt.plot(f)
plt.xlabel('Maturity')
plt.ylabel('Yield')
plt.title("Spot Curve")
plt.show()

# Plot Interest Rates Paths
plt.plot(r)
plt.xlabel('Time')
plt.ylabel('Interest Rates')
plt.title("Interest Rate Paths")
plt.show()

# Plot Stock Prices
plt.plot(S)
plt.xlabel('Time')
plt.ylabel('Index')
plt.title("Index Paths")
plt.show()

# Plot the Distribution of Final Stock Prices
plt.hist(S[-1])
plt.xlabel('Price')
plt.ylabel('Amount')
plt.title("Distribution of Final Stock Prices")
plt.show()

# Plot Average Duration
plt.bar(method_type[1:],Average_Duration)
plt.xlabel('Methods')
plt.ylabel('Duration')
plt.title("Duration by Method")
plt.show()

# Plot Average Errors
plt.bar(method_type[1:],Average_Errors)
plt.xlabel('Methods')
plt.ylabel('Error')
plt.title("Error by Method")
plt.show()

# Plot Calculated Prices
```python
174. Df_Price = pd.DataFrame(price, index = method_type, columns = Market_Data["Strike"]).iloc[:,0])
175. plt.plot(Df_Price.T)
176. plt.legend(method_type)
177. plt.xlabel('Strike Price')
178. plt.ylabel('Option Price')
179. plt.title('Option Price by Strike Price')
180. plt.show()
```
Appendix B: Functions for Calibration

1. # Import Libraries
2. import numpy as np
3. import scipy.interpolate as sci
4. import math
5. import pandas as pd
6. from scipy.optimize import minimize
7. from scipy.stats import norm
8. from scipy.optimize import fmin
9. 
10. ""
11. Input for Calibrations
12. ""
13. 
14. 
15. # Input for CIR Calibrations
16. 
18. 
19. 
20. 
21. 
22. zero_rates = r_list
23. r0 = r_list.iloc[0] # 0.0 # set to zero
24. 
25. 
26. tck = sci.splrep(t_list, zero_rates) # cubic splines
27. 
28. 
29. 
30. 
31. f = ts_list + de_list * tn_list
32. 
33. # Import S&P 500 SPDR Values
34. 
35. SPY = pd.read_csv("SPY.csv")
36. 
37. 
38. 
39. Correlation Between Index and overnight LIBOR
40. ""
41. 
42. 
43. 
44. 
45. 
46. 
47. 
48. Jump Calibration
49. ""
50. 
51. # Calibration of Jump Diffusion Model
52. def FitJumpDiffusionModel(X,dt):
53. 
54. 
55. 
56.
# start alpha and sig at their values if there are no jumps
x0[0] = np.mean(X)/dt  # mu
x0[1] = np.std(X)  # sig

# alphaJ and sigJ are intialized from the returns
x0[2] = x0[0]  # alphaJ
x0[3] = x0[1]  # sigJ
x0[4] = 0  # lambda

# Perform an optimization to get the model parameters
boundary=[[np.finfo(float).eps,math.inf],[np.finfo(float).eps,math.inf],
          [np.finfo(float).eps,1-np.finfo(float).eps]]
llf = lambda params:localMaximumLikelihoodCostFcn(params,X,dt)
params = minimize(llf,x0,bounds=boundary).x
return params

def localMaximumLikelihoodCostFcn(param,X,dt):
    # To help with the explanation, break out the parameters
    r=param[0]
sig=param[1]
alphaJ = param[2]
sigJ = param[3]
lambda_1 = param[4]

mu = r - sig*sig/2
muJ = alphaJ - lambda_1*lambda_1/2

# Use the formulation given by MacDonald (2nd Ed. p641)
k = np.exp(alphaJ)-1
cost = -np.sum(np.log(lambda_1*norm.pdf(X,(mu-lambda_1)*dt+muJ,
                            sig*sqrt(sig**2*dt+sigJ**2))+(1-lambda_1)*norm.pdf(X,(mu-lambda_1)*dt,
                            sig*sqrt(dt))))
return cost

""" Interest Rate Calibration """
def CIR_forward_rate(opt):
    """ Function for forward rates in CIR85 model. """
    Parameters
    =========
kappa_r: float
        mean-reversion factor
    theta_r: float
        long-run mean
    sigma_r: float
        volatility factor

    Returns
forward_rate: float

...  

kappa_r, theta_r, sigma_r = opt
t = tn_list
g = np.sqrt(kappa_r ** 2 + 2 * sigma_r ** 2)
sum1 = ((kappa_r * theta_r * (np.exp(g * t) - 1)) /
(2 * g + (kappa_r + g) * (np.exp(g * t) - 1)))
sum2 = r0 * (((4 * g ** 2 * np.exp(g * t)) /
(2 * g + (kappa_r + g) * (np.exp(g * t) - 1)) ** 2)
forward_rate = sum1 + sum2
return forward_rate

def CIR_error_function(opt):
    """ Error function for CIR85 model calibration. ""
    kappa_r, theta_r, sigma_r = opt
    if 2 * kappa_r * theta_r < sigma_r ** 2:
        return 100
    if kappa_r < 0 or theta_r < 0 or sigma_r < 0.001:
        return 100
    forward_rates = CIR_forward_rate(opt)
    MSE = np.sum((f - forward_rates) ** 2) / len(f)
    # print opt, MSE
    return MSE

def CIR_calibration():
    opt = fmin(CIR_error_function, [0.3, 0.04, 0.1],
                xtol=0.00001, ftol=0.00001,
                maxiter=500, maxfun=1000)
    return opt
Appendix C: Functions for Calibration

1. `import` numpy as np
2. `import` math
3. ```
4. # Functions for random number generation
5. def HaltonSequence(n,b):
6. # This function generates the first n numbers in Halton’s low discrepancy sequence with base b
7.    hs=np.zeros(n)
8.    for idx in range(0,n):
9.        hs[idx] = Halton_SingleNumber(idx+1,b)
10.       return hs
11. def Halton_SingleNumber(n,b):
12. # This function shows how to calculate the nth number in Halton’s low discrepancy sequence.
13.     n0 = n
14.     hn = 0
15.     f = 1/b
16.     while n0 > 0:
17.         n1 = math.floor(n0/b)
18.         r = n0-n1*b
19.         hn = hn + f*r
20.         f = f/b
21.         n0 = n1
22.     return hn
23. def random_number_generator_custom(nPaths,steps,variance_reduction="anti_paths"):
24.     ''' Function to generate random variables with Halton Sequence
25.     Parameters
26.     =========
27.     nPaths: float
28.     Number of Paths
29.     steps: float
30.     Number of Steps
31.     Returns
32.     ======
33.     matrix: NumPy array
34.     Random numbers
35.     ...'
36.     if variance_reduction=="anti_paths":
37.         rand = np.random.normal(size=(2,steps,int(nPaths)))
38.     elif variance_reduction=="halton":
39.         rand = np.zeros((2, steps, nPaths), dtype=np.float)```
for jdx in range(0,2):
    # Normally distributed random number generator
    base1 = 2
    base2 = 7
    # generate two Halton sequences
    hs1 = HaltonSequence(int(np.ceil(nPaths*steps)/2)),base1)
    hs2 = HaltonSequence(int(np.ceil((nPaths*steps)/2)),base2)
    # Use the simple Box-Mueller approach to create normal variates
    R = np.sqrt(-2*np.log(hs1))
    Theta = 2*math.pi*hs2
    P = R*np.sin(Theta)
    Q = R*np.cos(Theta)
    # The complete sequence is
    N = np.concatenate((P,Q),axis=0)
    rand[0]=N
return rand

def generate_cholesky(rho_rs):
    ''' Function to generate Cholesky matrix.
    Parameters
    ===========
    rho: float
        correlation between index level and short rate
    Returns
    =======
    matrix: NumPy array
        Cholesky matrix
    ...'''
    covariance = np.zeros((2, 2), dtype=np.float)
    covariance[0] = [1.0, rho_rs]
    covariance[1] = [rho_rs, 1.0]
    cho_matrix = np.linalg.cholesky(covariance)
    return cho_matrix

def AssetPaths(S0,param,dt,steps,nPaths,rand,rho_rs):
    ''' Function to generate paths for the underlying asset with only jumps
    Parameters
    ===========
    S0: float
        initial value of asset
    param: float
    dt: float
        long-run mean
    steps: float
        volatility factor
    nPaths: float
        volatility factor
    Returns
    =======
    S: NumPy array
        simulated asset paths
r = param[0]
sig = param[1]
alphaJ = param[2]
sigJ = param[3]
lambda_1 = param[4]

# Calculate the Drift Factor
mu = r - sig*sig/2
muJ = alphaJ - lambda_1*lambda_1/2

# Use the formulation given by MacDonald (2nd Ed. p641)
k = np.exp(alphaJ)-1

# Create Random Numbers for expNotJ part
ran = np.zeros((steps, nPaths), dtype=np.float)
cho_matrix = generate_cholesky(rho_rs)

for t in range(1, steps+1):
    ran[t-1, :] = np.dot(cho_matrix, rand[:, t-1])[1]

# Do the non-jump component
expNotJ = np.cumprod(np.exp((muJ-lambda_1*k)*dt+sig*np.sqrt(dt)*
    ran), axis=0)

# Do the jump component
m = np.random.poisson(lambda_1*dt,(steps, nPaths))
expJ = np.exp(m*(muJ+sigJ*np.random.normal(scale=np.maximum(1,np.sqrt(m)),size=(steps, nPaths))))

returns = expNotJ * expJ

# Merge paths with initial stock value
S = S0*(np.concatenate((np.ones((1, nPaths)),returns), axis=0))

return S

---

def SRD_generate_paths(x0, kappa, theta, sigma, T, steps, nPaths, rand, rho_rs):
    ''' Function to simulate Square-Root Diffusion (SRD/CIR) process.

Parameters
==========
x0: float
    initial value
kappa: float
    mean-reversion factor
theta: float
    long-run mean
sigma: float
    volatility factor
T: float
    final date/time horizon
steps: float
    number of time steps
nPaths: float
    number of paths

Returns
-------
x: NumPy array

...simulated variance paths

dt = T / steps

x = np.zeros((steps, nPaths), dtype=np.float)
x[0] = x0

xh = np.zeros_like(x)
xh[0] = x0

sdt = np.sqrt(dt)

cho_matrix = generate_cholesky(rho_rs)

for t in range(1, steps):
    ran = np.dot(cho_matrix, rand[:, t-1])
    xh[t] = (xh[t-1] + kappa * (theta - np.maximum(0, xh[t-1])) * dt +
             np.sqrt(np.maximum(0, xh[t-1])) * sigma * ran[0] * sdt)
    x[t] = np.maximum(0, xh[t])

return x
1. import numpy as np
2. 3. from sklearn.neural_network import MLPRegressor
4. from sklearn.tree import DecisionTreeRegressor
5. from sklearn import svm,linear_model
6. from sklearn.isotonic import IsotonicRegression
7. from sklearn.gaussian_process import GaussianProcessRegressor
8. 9. def option_pricing(S,K,r,dt,steps,nPaths,option_type,method_type):
10. 11. # Calculate intrinsic values
12. if option_type=="put":
13.   cFlows = np.maximum(K-S[1:,:],0)
14. elif option_type=="call":
15.   cFlows = np.maximum(S[1:,:]-K,0)
16. 17. # Calculate the discount factor for each time step
18. disc = np.exp(-r*dt)
19. 20. # Loop backwards in time filling the cash flows matrix
21. for idx in range(steps-2,-1,-1):
22. 23. # Determine which cashflows to regress
24. mask = cFlows[idx,:]>0
25. 26. # Form the Y and X columns to be regressed
27. Xdata = np.extract(mask,S[idx+1])
28. Ydata = np.extract(mask,cFlows[idx+1])*np.extract(mask,disc[idx+1])
29. 30. if method_type =="LS":
31.   32.   if len(Xdata) == 1:
33.     coeffs = Ydata
34.   elif len(Xdata) == 0:
35.     coeffs = []
36.   else:
37.     # Do the regression, in this case using a quadratic fit
38.     coeffs = np.polyfit(Xdata,Ydata,np.minimum(2,len(Xdata)-1))
39. 40. # Calculate potential payoff from "backwards induction"
41. payoffs = np.polyval(coeffs,Xdata)
42. 43. elif method_type =="NN":
44.   45.   clf = MLPRegressor(verbose=False,hidden_layer_sizes=(20,20,20,20),
46.   warm_start=False, early_stopping=True, activation = 'relu',
47.   shuffle =False, tol=0.000001, max_iter = 15)
48. 49.   if len(Xdata) == 1:
50.     payoffs = Ydata
51.   elif len(Xdata) == 0:
52.     payoffs = []
53.   else:
54.     clf.fit(Xdata.reshape(-1, 1),Ydata)
55.     payoffs = clf.predict(Xdata.reshape(-1, 1))
56. 57. elif method_type =="DT":
58. 59. 60.
clf = DecisionTreeRegressor(max_depth=None, min_samples_split=2, min_samples_leaf=1,
min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, presort=False)

if len(Xdata) == 1:
    payoffs = Ydata
elif len(Xdata) == 0:
    payoffs = []
else:
    clf.fit(Xdata.reshape(-1, 1), Ydata)
    payoffs = clf.predict(Xdata.reshape(-1, 1))

elif method_type == "SVM":
    clf = svm.SVR()
    if len(Xdata) == 1:
        payoffs = Ydata
    elif len(Xdata) == 0:
        payoffs = []
    else:
        clf.fit(Xdata.reshape(-1, 1), Ydata)
        payoffs = clf.predict(Xdata.reshape(-1, 1))

elif method_type == "SGD":
    clf = linear_model.SGDRegressor(max_iter=1000, tol=1e-3)
    if len(Xdata) == 1:
        payoffs = Ydata
    elif len(Xdata) == 0:
        payoffs = []
    else:
        clf.fit(Xdata.reshape(-1, 1), Ydata)
        payoffs = clf.predict(Xdata.reshape(-1, 1))

elif method_type == "ISO":
    clf = IsotonicRegression()
    if len(Xdata) == 1:
        payoffs = Ydata
    elif len(Xdata) == 0:
        payoffs = []
    else:
        clf.fit(Xdata, Ydata)
        payoffs = clf.predict(Xdata)

elif method_type == "GPR":
    clf = GaussianProcessRegressor()
    if len(Xdata) == 1:
        payoffs = Ydata
    elif len(Xdata) == 0:
        payoffs = []
    else:
        clf.fit(Xdata.reshape(-1, 1), Ydata)
        payoffs = clf.predict(Xdata.reshape(-1, 1))

# Find location(s) where early exercise is optimal
eeLoc = np.extract(mask, cFlows[idx, :]) > payoffs

# Update the cash flow matrix to account for early exercise
mask[mask] = eeLoc # These locations get exercised early

cFlows[idx, :] = mask * cFlows[idx, :] # These are the cash flows

cFlows[idx+1:, :] = cFlows[idx+1:, :] * np.logical_not(mask)

# Discount each cash flow and average
oPrice = np.matmul(np.cumprod(disc.mean(1)), np.transpose((np.sum(cFlows, axis=1) / nPaths)))

return oPrice
Appendix C: Quantlib Function

```python
# option data
maturity_date = ql.Date(15, 11, 2020)
spot_price = 311.790009
dividend_rate = 0

if optiontype == "call":
    option_type = ql.Option.Call
elif optiontype == "put":
    option_type = ql.Option.Put

risk_free_rate = 0.0154
day_count = ql.Actual365Fixed()
calendar = ql.UnitedStates()

calculation_date = ql.Date(15, 11, 2019)
ql.Settings.instance().evaluationDate = calculation_date

payoff = ql.PlainVanillaPayoff(option_type, strike_price)
settlement = calculation_date

am_exercise = ql.AmericanExercise(settlement, maturity_date)
american_option = ql.VanillaOption(payoff, am_exercise)

spot_handle = ql.QuoteHandle(ql.SimpleQuote(spot_price)
flat_ts = ql.YieldTermStructureHandle(ql.FlatForward(calculation_date, risk_free_rate, day_count)
dividend_yield = ql.YieldTermStructureHandle(ql.FlatForward(calculation_date, dividend_rate, day_count)
flat_vol_ts = ql.BlackVolTermStructureHandle(ql.BlackConstantVol(calculation_date, calendar, volatility, day_count)

bsm_process = ql.BlackScholesMertonProcess(spot_handle, dividend_yield,
flat_ts,
flat_vol_ts)

steps = 200
binomial_engine = ql.BinomialVanillaEngine(bsm_process, "crr", steps)
american_option.setPricingEngine(binomial_engine)
return american_option.NPV()
```
Works Cited