AMSI
VACATION
RESEARCH
SCHOLARSHIPS
2017-2018

Numerical Optimisation Applied to Monte Carlo Algorithms for Finance

Phillip Luong

Supervised by Professor Hans De Sterck, Professor Gregoire Loeper, and Dr Ivan Guo

Monash University

Vacation Research Scholarships are funded jointly by the Department of Education and Training and the Australian Mathematical Sciences Institute.

Australian Government
Department of Education and Training

AMSI AUSTRALIAN MATHEMATICAL SCIENCES INSTITUTE

1 Introduction

Options are financial contracts that offers the right (but not the obligation) to buy (call option) or sell (put option) an asset at an agreed price. Options are often used to create safer financial strategies, to minimise the risk of losing a lot of money. In Financial Mathematics, calculating the price of these options quickly is important.

There are many different types of options, depending on the asset being sold and the terms of the two parties. Differences between option types will include *Strike Price* and when the option may be exercised. The *Strike Price* refers to the agreed price to pay for the asset. An example of a class of options are *European Options*, which can only be exercised at a single time point in the future (known as the *Time of Expiry*). In contrast, *American Options* may be exercised at any time point between the time of purchase and the time of expiry. In finance, the nuances to the agreed-upon option makes the contract itself difficult to accurately value. In addition to the demand to quickly value the current price of such options, it is imperative to create a method which accurately models the future price of an option. In this project, we wish to find a method to construct a model to correctly price the value of these options with the use of Monte Carlo Simulations and Optimisation techniques.

Previously, many researchers have been able to model the price of various types of options. The price of a *European option* may be calculated via the Black-Scholes Partial Differential Equation (1.1) (Black and Scholes, 1973). This model calculates the price of a European option, U, which will change depending on the price of an asset, $x \in \mathbb{R}^+$ (dollars), and the time of purchase, $t \in [0, T]$ (years). Furthermore, we define the *Strike Price*, K, *Volatility*, σ, *Risk-Free Rate*, r, and *Time of Expiry*, T. The *Volatility* is the standard deviation of the price of the asset over time, and the *Risk-Free Rate* is the theoretical rate of an investment with zero risk (for example, in a savings account). The value of European options is calculated by solving the following Partial Differential Equation:

$$\frac{\partial U}{\partial t} + \frac{1}{2}\sigma^2 x^2 \frac{\partial^2 U}{\partial x^2} + rx\frac{\partial U}{\partial x} = rU. \tag{1.1}$$

We apply the following boundary conditions:

We consider the price of an asset cannot decrease below \$0. If the price of an asset is originally worth \$0, we say that it's option is worth \$0 for all $t \in [0, T]$. In cases like this, we would theoretically purchase the asset, rather than purchasing the option, since it can only increase in value. So we have

$$u(0, t) = 0 \, \forall \, t \in [0, T].$$

The price of the option will converge towards the price of the asset for very expensive assets (assuming the Strike Price remains the same):

$$u(x, t) \to x \, \text{as} \, x \to \infty.$$

At the time of expiry, the value of an option is only non-zero if the asset is worth more than the Strike Price. In this case, the value of the option is:

$$u(x, T) = (x - K)^+ := \max(x - K, 0).$$

Solving Equation 1.1 analytically gives us the following solution:

$$U(x, t) = x\Phi(z) - Ke^{-r(T-t)}\Phi(z - \sigma\sqrt{T - t})), \tag{1.2}$$

$$\text{where} \, z = \frac{1}{\sigma\sqrt{T - t}} \left[\ln\left(\frac{x}{K}\right) + (T - t)\left(r + \frac{\sigma^2}{2}\right)\right],$$

and $\Phi(z)$ is the Normal Cumulative Distribution function evaluated at z. In this report, we shall refer to equation (1.2), as the Black-Scholes function.

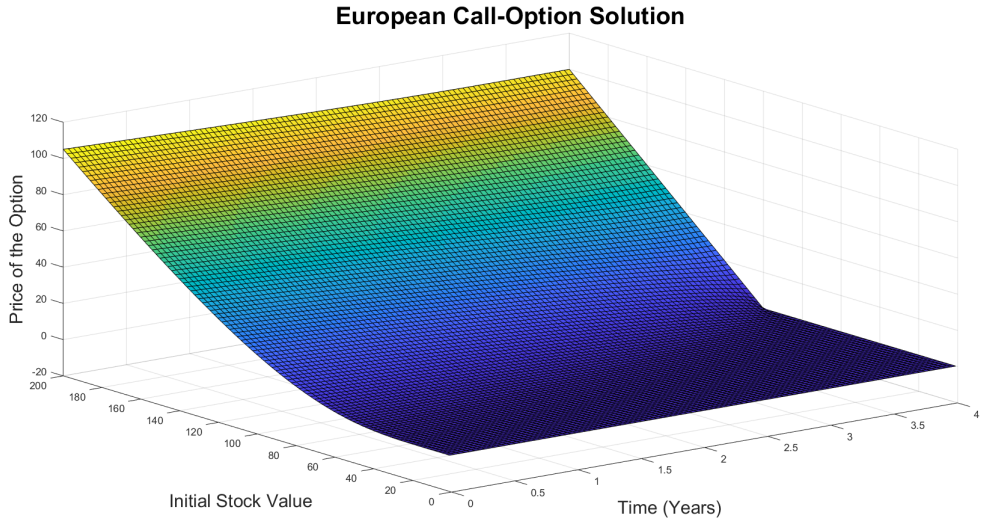

Figure 1.1 Exact plot of the Black-Scholes PDE, with boundary condition,

$$u(x,T) = (x-K)^+.$$

Alternatively, we apply the *Call-Spread Boundary*, a function composed of a difference of two European Options with different Strike Prices (K_1 and K_2)

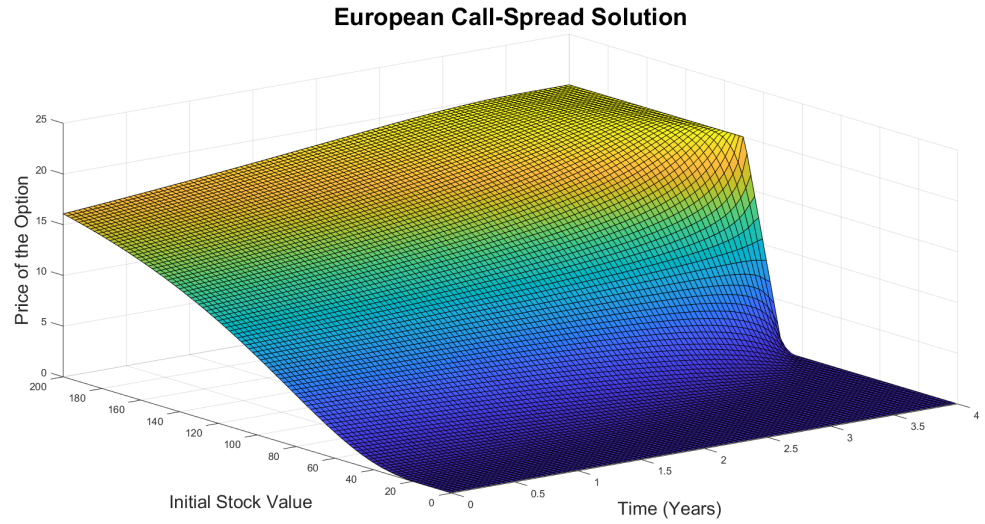

Figure 1.2 Exact plot of the Black-Scholes PDE (Equation 1.1), with the call-spread function as the boundary condition,

$$u(x,T) = (x-k_1)^+ - u(x-k_2)^+.$$

The goal of this project is to model the value of the European Call option, and Call-Spread Function at any time point $\hat{t} \in [0, T]$ using a basis of polynomials, a basis of Black-Scholes functions, and a combination of the two. This will be completed by applying optimisation techniques such as the least-squares method.

2 Methodology

2.1 Simulation of Asset and Option Prices

Firstly, we simulated the price of a particular stock using Monte Carlo Simulations. For these simulations, we fix the initial stock price ($x_0 = 100$), risk-free rate ($r = 0$), volatility ($\sigma = 0.3$), and time until expiry ($T = 4$). We wish to use this data to model the price of the call option at $\hat{t} = 2$. We calculate the logarithm of the stock price to ensure that the value of x_i remains positive at every time step (Equation 2.1).

$$\ln(x_{i+1}) = \ln(x_i) + rdt - \frac{\sigma^2}{2}dt + \sigma dW_t \tag{2.1}$$

Here, W_t denotes a Weiner Process, with time increments normally distributed $\mathcal{N}(0, dt)$ and $dt = T/\#$ steps (years). In our simulations, we use 1001 time steps.

Next, we use the data to model the price of $U(x, \hat{t}|\vec{c})$ at time, $\hat{t} \in [0, T]$, and where $\vec{c}$ is the vector containing all the parameters to the model. Here, we utilise the *Longstaff-Schwartz method*. The *Longstaff-Schwartz method* uses the simulated asset prices at time, $\hat{t}$, and the value of $U(x, T|\vec{c})$ to calculate the expected value of $f(x; \vec{c}) = U(x, \hat{t})$ (Longstaff and Schwartz, 2001). The results for a sample of these simulations are calculated by using equation (2.1).

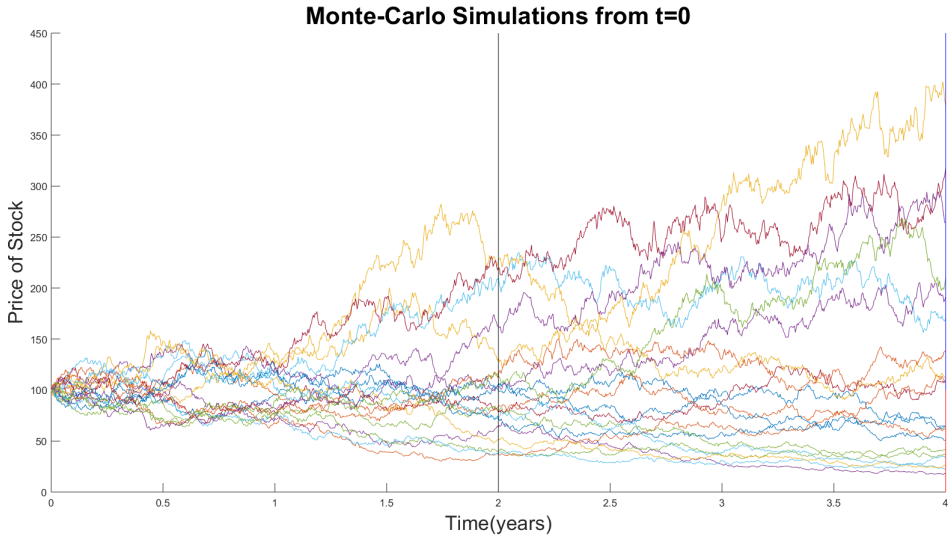

Figure 2.1: Depicting the 20 Monte Carlo Simulations of the stock price over 4 years

The first 20 simulations of the 100,000 Monte-Carlo simulations. Simulations have the parameters, $x_0 = 100, r = 0, v = 0.3, T = 4, \sigma = 0.3$. The line at $\hat{t} = 2$ is present to show our aim of the project. In this particular case, we attempt to model the price of the a call option, meaning that at time, T, the call option of the asset is worth $\$(x - K)^+$, where $K = 100$.

We calculate the price of y_i at the time $t = 4$, using the value of the European Option (Equation 2.2), and the value of the call-spread function (Equation 2.3) at the time of expiry.

$$y_i = U(x_i(T), T) = (x - K)^+ \tag{2.2}$$

$$y_i = U(x_i, T) = (x_i(T) - K_1)^+ - (x_i(T) - K_2)^+ \tag{2.3}$$

After these calculations, we generate a set of data-points; where (x_i, y_i) form a pair representing the price of the asset at time, $\hat{t}$, and the price of $U(x_i(T), T)$ respectively.

2.2 Modelling the Price of Option Prices via the Least Squares Method

Next, we use the data-points to optimise for $\vec{c}$, where $\vec{c}$ minimises

$$ss(x_i;\vec{c}) = \sum_{i\in I}(y_i - f(x_i(\hat{t});\vec{c}))^2 \tag{2.4}$$

via the least-squares method (Equation 2.4). Here, I is the set of all simulated data points $\{(x_i, y_i)\}$, each x_i is the stock value at $t = \hat{t}$, and $ss(x_i;\vec{c})$ is the sum of squares formula given the data points. Furthermore, $f(x_i(\hat{t});\vec{c})$ is the basis function that we are using to model the price of the option (either (2.6), (2.7) or (2.8)), and y_i is dependent on the formula we are evaluating, whether it is the value of the call option (2.2) or the call-spread function (2.3).

The models that we wish to fit the model are

- **A p-degree polynomial**

$$f(x|\vec{c}) = \sum_{i=0}^{p} c_i x^i \tag{2.5}$$

  where $\vec{c} = [c_0 \dots c_p]$ are the coefficients to each polynomial term.

- **A single Black-Scholes formula (with a constant c_0)**

$$f(x_i(\hat{t})|\vec{c}) = c_0 + c_1 \text{bls}(x, K, r, t, \sigma) \tag{2.6}$$

  where $\vec{c} = [c_0\, c_1\, K\, \sigma]$ are the constant and coefficients of the Black-Scholes function, and the parameter of the varying Strike Price and Volatility.

- **A sum of two Black-Scholes formulae and a linear equation**

$$f(x|\vec{c}) = c_0 + c_1 x + c_2 \text{bls}(x, K_1, r, t, \sigma_1) + c_3 \text{bls}(x, K_2, r, t, \sigma_2) \tag{2.7}$$

  where $\vec{c} = [c_0\, c_1\, c_2\, c_3\, K_1\, K_2\, \sigma_1\, \sigma_2]$ are the constant and coefficients to the Black-Scholes functions, and the parameters to the Strike Prices and Volatilities.

We find the optimal value of $\vec{c}$, via three methods using MATLAB's inbuilt linear solvers, '\' and *lscov*, and the non-linear solver, *fminunc* (MATLAB, 2017).

2.2.1 The Linear Solvers, lscov and '\'

The linear solver, *lscov* solves the matrix equation

$$A\vec{c} = \vec{y}, \tag{2.8}$$

where A is a $m \times n$ matrix, $\vec{c}$ is a $n \times 1$ vector, and $\vec{y}$ is a $m \times 1$ vector. In our case, $m = 10^6$ and $n = p+1$, where p denotes the order of the polynomial. Since the dimensions $\vec{c}$ and $\vec{y}$ differ, we multiply A^T on both sides in order to solve for $\vec{c}$. This is what the "\" operator solves. In addition to solving for $\vec{c}$, *lscov* simplifies the problem by performing a QR decomposition on A first.

$$\vec{c} = (A^T A)^{-1} A^T \vec{y},$$

These problems were primarily effective to deduce $\vec{c}$ when we modelled the data to $f(x|\vec{c}) = \sum_{i=0}^{p} c_i x^i$. However, since bases (2.6) and (2.7) contained non-linear parameters, we cannot just use lscov or '\' to find $\vec{c}$.

2.2.2 The Non-Linear Solver, fminunc

Non-linear solvers are necessary in this research project to find parameters located in the Black-Scholes functions, like in bases (2.6) and (2.7). The inbuilt MATLAB non-linear least squares solver is fminunc, which relies on the Quasi-Newton Method or the Trust Region Method, to find the optimal $\vec{c} \in \mathbb{R}$ from an initial guess, $\vec{c_0}$. While both methods use gradient information, they have differences in their algorithms. As both methods are inbuilt solvers in MATLAB, the specific details to the algorithms are not fully provided here (MATLAB, 2017).

The Quasi-Newton Method is a line-search method which involves performing a line search to find the step length by estimating a Hessian Matrix to calculate the step direction. In MATLAB, it is possible to either supply a gradient function derived analytically, or to allow it to numerically estimate the gradient at each step (MATLAB, 2017). Such options give changes to the overall algorithm which may affect the rate of convergence.

In contrast, the Trust Region Method first selects a step size before choosing a stepping direction. In this case, the step-size is dependent on the gradient, meaning it is necessary for us to supply the gradient function. The gradient functions are all provided in Appendix A.

In addition, to accelerate the non-linear optimisation processes, we embedded *lscov* into our non-linear solvers, minimising the objective function over the coefficients. In this case, the coefficients, $c_0, c_1, \ldots$ become a function of the non-linear parameters, $K_1, \sigma_1, K_2, \ldots$. Details are supplied in Appendix A. We consider the Black-Scholes Problem with basis (2.7), where $c_0(K, \sigma)$ and $c_1(K, \sigma)$ are optimally solved via

$$\vec{c}(K, \sigma) = (A^T A)^{-1} A^T \vec{y}, \tag{2.9}$$

where A is the matrix

$$\begin{bmatrix} 1 & \text{bls}(x_1, K, \sigma) \\ \vdots & \vdots \\ 1 & \text{bls}(x_{100,000}, K, \sigma) \end{bmatrix}. \tag{2.10}$$

Then, the sum of squares problem is

$$ss(c_0(K, \sigma), c_1(K, \sigma), K, \sigma) = \sum_{i \in I} (y_i - (c_0(K, \sigma) + c_1(K, \sigma)\text{bls}(K, \sigma))^2 := g(K, \sigma). \tag{2.11}$$

We compared the number of iterations required to find K and σ with the Quasi-newton method without a supplied gradient. Due to time constraints, we were only able to implement this method to model a single Call Option using basis (2.6). In addition, we also ended up approximating the gradient functions as

$$\frac{\partial g(K, \sigma)}{\partial K} = \nabla_{\vec{c}}(ss(\vec{c}, K, \sigma)) \frac{\partial \vec{c}(K, \sigma)}{\partial K} + \frac{\partial ss(\vec{c}, K, \sigma)}{\partial K} \approx \frac{\partial ss(\vec{c}, K, \sigma)}{\partial K}, \tag{2.12}$$

and

$$\frac{\partial g(K, \sigma)}{\partial \sigma} = \nabla_{\vec{c}}(ss(\vec{c}, K, \sigma)) \frac{\partial \vec{c}(K, \sigma)}{\partial \sigma} + \frac{\partial ss(\vec{c}, K, \sigma)}{\partial \sigma} \approx \frac{\partial ss(\vec{c}, K, \sigma)}{\partial \sigma}, \tag{2.13}$$

where $\vec{w} = [K\, \sigma]$.

3 Computational Results and Discussion

3.1 Polynomial Estimates of the Black-Scholes Equation

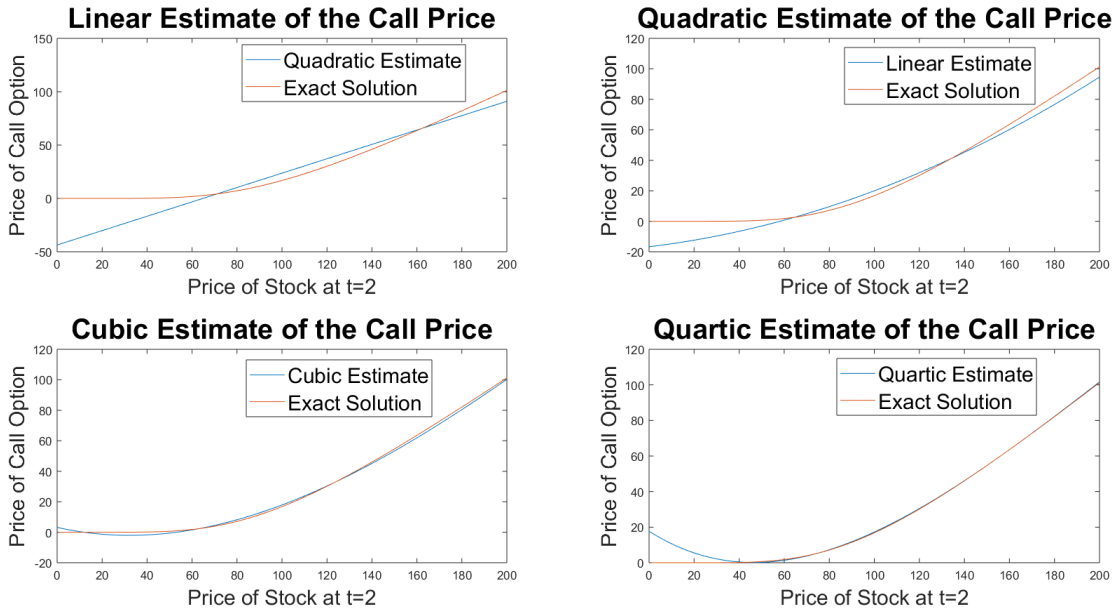

Figure 3.1: Estimates of the Black-Scholes solution using Polynomials (Equation 2.5) of Increasing Order obtained via *lscov*

Table 3.1: Coefficients of the Polynomial Estimates obtained via lscov

Basis	c_0	c_1	c_2	c_3	c_4
Linear	-43.6770	0.6736	N/A	N/A	N/A
Quadratic	-16.7120	0.1771	0.0019	N/A	N/A
Cubic	3.1865	-0.3321	0.0055	-7.0×10^{-6}	N/A
Quartic	17.6953	-0.8061	0.0104	-2.6×10^{-5}	2.2×10^{-10}

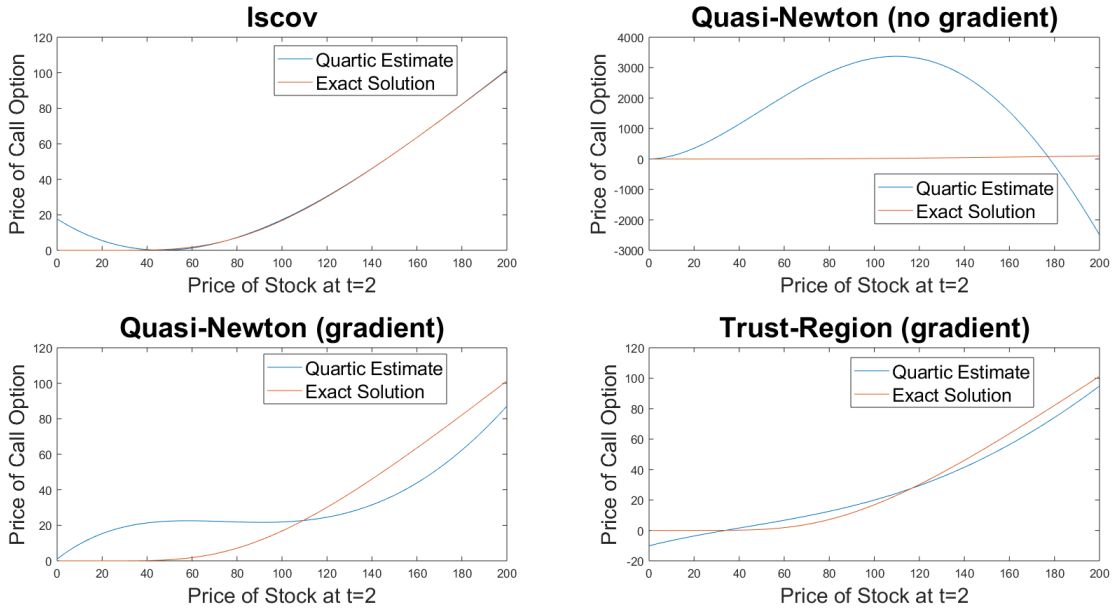

Figure 3.2: Comparison of the Quartic Estimates provided by Different Optimisation Methods

$$f(x|\vec{c}) = c_0 + c_1x + c_2x^2 + c_3x^3 + c_4x^4$$

Table 3.2: Coefficients of the Quartic Estimates via different Optimisation Methods

Method	c_0	c_1	c_2	c_3	c_4
lscov	17.6953	-0.8061	0.0104	-2.6×10^{-5}	2.2×10^{-10}
\	17.6953	-0.8061	0.0104	-2.6×10^{-5}	2.2×10^{-10}
Initial Guess	1	1	1	1	1
Quasi-Newton (no gradient given)	1.0000	0.9999	0.9918	-0.0081	1.4×10^{-5}
Quasi-Newton (gradient given)	0.9998	0.9837	-0.0151	-8.6×10^{-5}	-1.2×10^{-7}
Trust Region	-10.1346	0.3717	-0.0030	2.7×10^{-5}	-4.1×10^{-8}

Figure 3.2 shows that the linear methods (such as lscov), are providing better estimates than the non-linear methods (both Quasi-Newton Methods and the Trust-Region Method).

As expected, increasing the order of the polynomial, results in a better estimation of the Black Scholes equation (Figure 3.1). However, these results tend to diverge at the lower and

upper end of the domain. Interestingly, as we increase the order of the polynomial, the parameter values will start to differ (Table 3.2). There were slight (but negligible) differences between the linear methods, however, the Quasi-Newton method and Trust Region method gave very different results. Since these parameters obtained by the non-linear methods give a higher objective value than the linear methods, we needed to investigate why these methods were not working. We suspect that this is due to the ill-conditioning of the problem, however, further work will be needed to verify this.

3.2 Estimates using the non-linear Bases of the Black-Scholes Equation

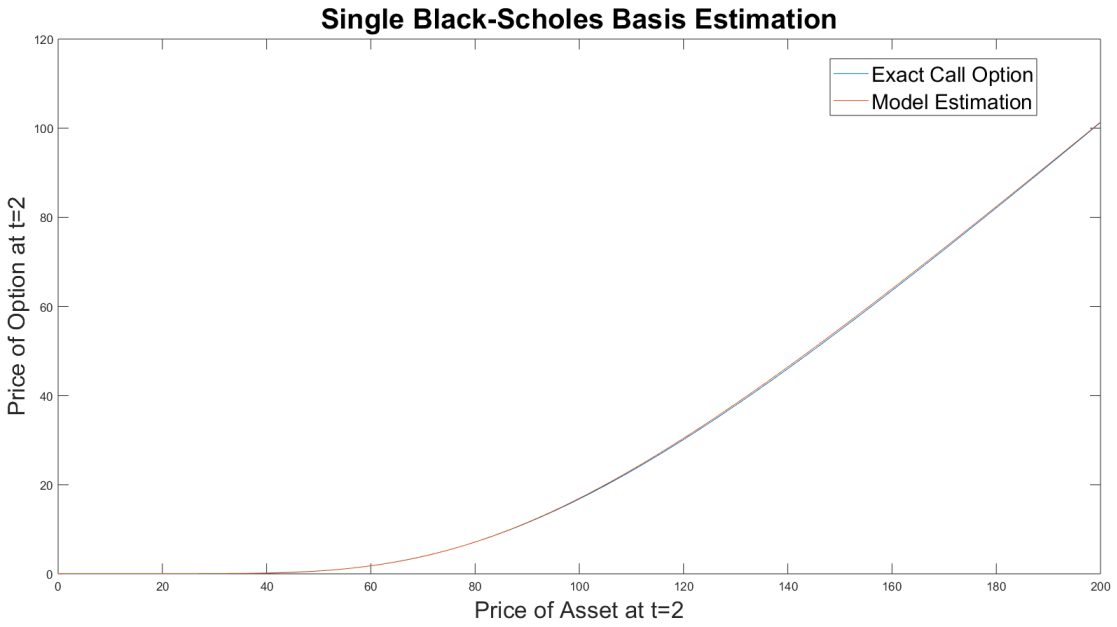

Figure 3.3: Estimate of the European Option using a single Black-Scholes Formula Basis

$$f(x|\vec{c}) = c_0 + c_1 \mathrm{bls}(x, K, r, t, \sigma).$$

Table 3.3: The Coefficients of the European Option using a single Black-Scholes Formula Basis

Method	c_0	c_1	K	σ
Expected Solution	0	1	100	0.3
Initial Guess	0.5	0.5	75	0.001
Quasi-Newton (no gradient given)	0.0639	0.9824	97.7721	0.2895
Quasi-Newton (gradient given)	0.0639	0.9824	97.7720	0.2895
Trust Region	-0.9192	0.8846	85.13	0.2312

With the initial guess, $[c_0, c_1, K, \sigma] = [0.5, 0.5, 75, 0.001]$, Figure 3.3 show that both the Quasi-Newton and Trust Region methods both estimate the price of a call option accurately. However, the Quasi-Newton method is highly dependent on the initial guess, where an initial guess of $K = 65$ will cause the algorithm to find incorrect parameter values.

For the non-linear algorithms, we only allowed a maximum of 100 iterations the Quasi-Newton and Trust Region. Although the results for the trust-region method appear not to converge, if we increase the number of iterations, these results appear to eventually converge. Additionally, increasing number of data points also reduces the number of iterations required to reach the optimal coefficient values.

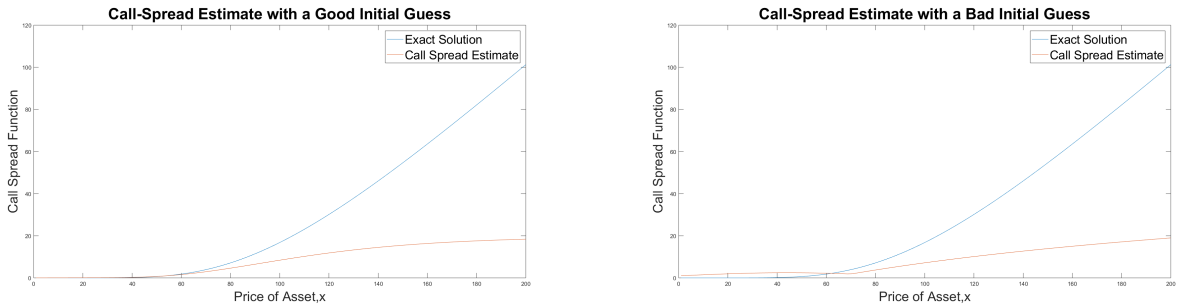

Figure 3.4: Estimates for the Call-Spread function using basis (2.7)

$$f(x|\vec{c}) = c_0 + c_1 x + c_2 \text{bls}(x, K_1, r, t, \sigma_1) + c_3 \text{bls}(x, K_2, r, t, \sigma_2).$$

For this data, we set K_1, K_2, σ_1, and σ_2 to 90, 110, 0.3, and 0.3 respectively. Then using the basis (2.7) we used fminunc to find $\vec{\hat{c}} = [c_0, c_1, c_2, c_3, K_1, K_2, \sigma_1, \sigma_2]$.
The first curve uses a 'good' initial guess (Table 3.4). In contrast the second curve was given

a 'bad' initial guess (Table 3.4), which estimated parameters further away than the actual value.

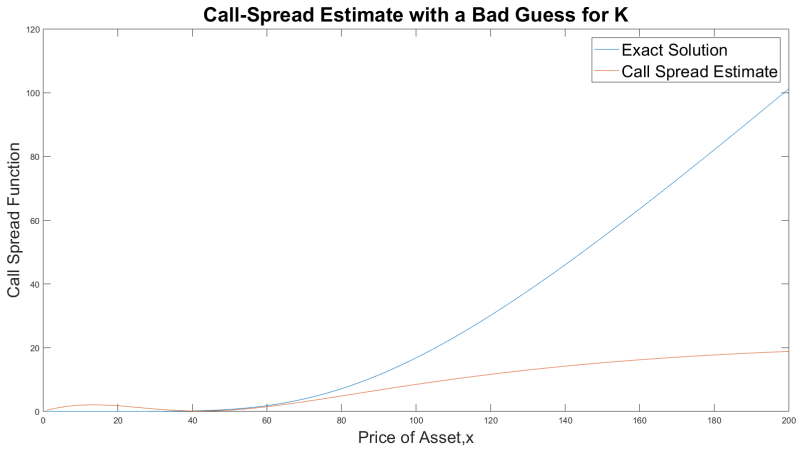

Figure 3.5: Estimates for the Call-Spread function with starting strike estimates $K_1 = 60$ and $K_2 = 140$

Table 3.4: Coefficients of the Call-Spread function using basis (2.7)

Initial Guess	c_0	c_1	c_2	c_3	K_1	K_2	σ_1	σ_2
Expected Solution	0	0	1	-1	90	110	0.3	0.3
'Good' Guess Initial	0.25	0.25	0.75	-0.75	95	105	0.24	0.36
'Good' Guess Result	0.0155	0.0024	1.838	-1.837	94.8286	105.1892	0.2867	0.2854
'Bad' Guess Initial	1	1	0	0	70	130	0.001	0.6
'Bad' Guess Result	0.9735	0.0506	0.2448	-0.2489	69.9974	130.0003	0.0000	0.6089
Figure (3.5) Initial	0	0	1	-1	50	150	0.3	0.3
Figure (3.5) Result	0.09727	0.3052	0.7476	-1.200	50.0652	149.9850	0.2950	1.1540

This method used the Quasi-Newton Method with an estimated gradient function for a maximum of 100 iterations. In all cases, the method ended before reaching the maximum number of 100 iterations.

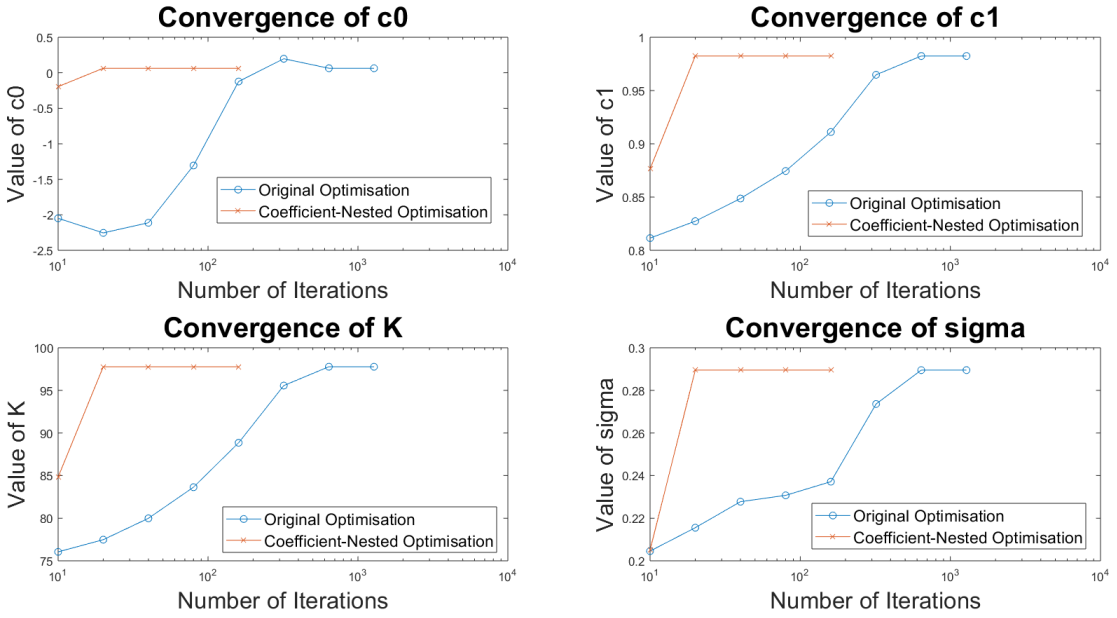

Figure 3.6: Convergence of Model Parameters using non-linear Optimisation vs the lscov nested non-linear Optimisation (2.11) plotted on a semi-log scale

Table 3.5: The Progression of the Parameters for increasing iterations

# Iterations	c_0	c_1	K	σ
Expected Solution	0	1	100	0.3
10 (NL)	-2.0491	0.8115	76.0518	0.2046
20 (NL)	-2.2542	0.8274	77.4784	0.2155
40 (NL)	-2.1120	0.8487	79.9863	0.2277
80 (NL)	-1.3054	0.8745	83.6391	0.2307
160 (NL)	-0.1227	0.9111	88.8512	0.2372
320 (NL)	0.1978	0.9648	-95.5620	0.2736
640 (NL)	0.0639	0.9824	97.7720	0.2895
1280 (NL)	0.0639	0.9824	-97.7720	0.2895
10 (nested NL)	-0.1949	0.8764	84.8305	0.2051
20 (nested NL)	0.0639	0.9824	97.7720	0.2895
40 (nested NL)	0.0639	0.9824	97.7720	0.2895

Note: NL refers to non-linear

In Figure 3.5, we only changed the initial values of K_1 (60) and K_2 (140), to observe the change in K when using the non-linear methods. We notice that the value of K_1 and K_2 doesn't move too much, and instead, is compensated by changing the values of σ_1 and σ_2. This could be troublesome, as it results in inaccuracies for lower and greater values of stock (shown on Figure 3.)). As such, increasing the number of unknown parameters results in a decrease in the performance of these methods, requiring better 'initial guesses' and more iterations. It is possible for us to reduce the amount of iterations by nesting the linear optimisation process in the non-linear solvers, as we now explain for a call option.

In Figure 3.6, we used the Quasi-Newton Method with no supplied gradient function, the single Black-Scholes basis ($f(x|\vec{c}) = c_0 + c_1\text{bls}(x, K, r, t, \sigma)$) and an initial guess of $[K, \sigma] = [75, 0.001]$. Clearly, the Nested non-linear optimisation required far less iterations than the original method (20 opposed to 640) for all parameters. The nested method shows great promise to optimise other functions, such as the Call-Spread function.

4 Future Directions

Further work in this project will be taken to look at extending the optimisation process to value more complex option types such as *American Options* and *Bermudan Options*. It would be interesting to also consider other types of spread functions such as the *Butterfly Spread* Boundary Condition, *Down-and-in Payoff* function and the *Discrete Down-and-out barrier put* function. These functions will affect the amount of Black-Scholes formulae required to accurately construct a model.

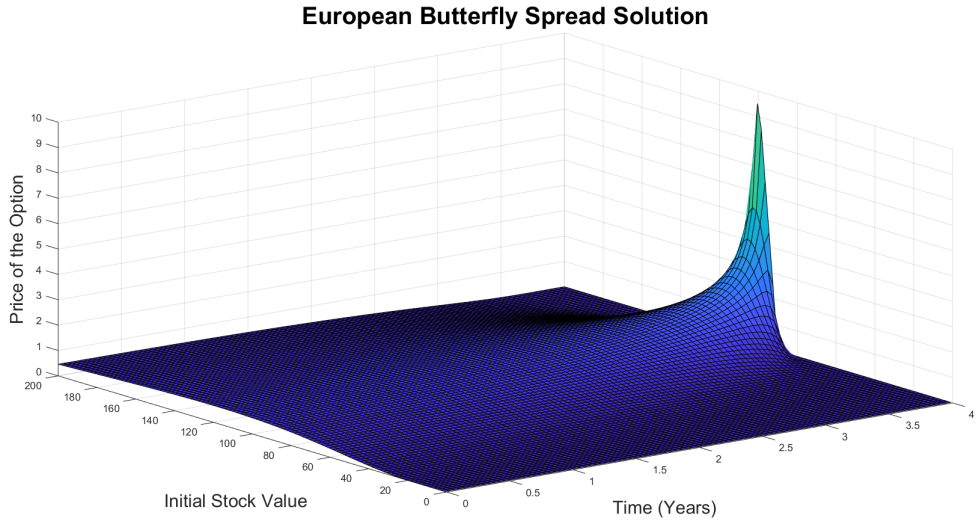

Figure 4.1: Exact Plot of the Black-Scholes PDE with the Butterfly-Spread Boundary Condition

$$U(x,T) = (x-K_1)^+ - 2(x - \frac{K_1+K_2}{2})^+ + (x-K_2)^+ \tag{4.1}$$

As we sometimes experienced issues with the strike price, K, and volatility parameter, σ, reaching negative values which make no sense and break the model; it may be necessary to approach this minimisation problem as a constrained problem, rather than the current unconstrained problem. In MATLAB, this would mean changing the method to *fmincon* rather than *fminunc* and then supplying the necessary constraints. Alternatively, we may substitute $K = e^c$ and $\sigma = e^\theta$ (where c and θ are parameters that span the Real Line) in order to ensure the two parameters remain positive.

In addition, in these cases, it may be beneficial to add a penalty term to our objective function, to ensure that the Strike Prices of the independent Black-Scholes formulae remain distant from each other. An example penalty function is

$$\text{penalty}_{i,j} = \frac{\gamma_{i,j}}{|K_i - Kj|}. \tag{4.2}$$

for some constant $\gamma_{i,j}$.

We may also include other sets of bases. Longstaff and Schwartz have previously used a set of weighted Laguerre polynomials and have suggested potentially using other basis polynomials

to model the price of options (Longstaff and Schwartz, 2001). This is something that can also be explored .

In this research project, we used standard least squares to create an estimate of the price of the option. This could be done by providing a better weighting for the more important regions of the curve.

Appendix A The Least Squares Gradient Functions

Note here that $\tau = T - t$, and knowing that

$$\frac{\partial z}{\partial K} = -\frac{1}{K\sigma\sqrt{\tau}} \quad \text{and} \quad \frac{\partial z}{\partial \sigma} = -\frac{1}{\sigma^2\sqrt{\tau}}\ln(\frac{x}{K}) + \sqrt{\tau}(\frac{1}{2} - \frac{r}{\sigma^2}),$$

we obtain the following partial derivatives

$$\frac{\partial \text{bls}}{\partial K} = -\frac{x}{K\sigma\sqrt{\tau}}\phi(z) - e^{-r\tau}\Phi(z - \sigma\sqrt{\tau}) + \frac{e^{-r\tau}}{\sigma\sqrt{\tau}}\phi(z - \sigma\sqrt{\tau}), \text{and} \tag{A.1}$$

$$\frac{\partial \text{bls}}{\partial \sigma} = x\frac{\partial z}{\partial \sigma}\phi(z) - Ke^{-r\tau}\frac{\partial z}{\partial \sigma}\phi(z - \sigma\sqrt{\tau}), \tag{A.2}$$

$$\nabla_{\vec{c}}(ss(\vec{c}; x)) = -2\sum_{i \in I}(\nabla_{\vec{c}}(f(\vec{c}; x_i))\,(y_i - f(\vec{c}; x_i)), \tag{A.3}$$

where $\nabla_{\vec{c}}(f(\vec{c}; x_i))$ is dependent on the basis function we are using.

The Gradient functions of the Sum of Squares formula are an $N \times 1$ vector (where there are N parameters) consisting of the partial derivatives of the Sum of Squares with respect to each parameter .

$$\nabla_{\vec{c}} ss = \begin{bmatrix} \frac{\partial ss}{\partial c_1} \\ \frac{\partial ss}{\partial c_2} \\ \vdots \\ \frac{\partial ss}{\partial c_N} \end{bmatrix}$$

For the n-degree polynomial cases (2.5), the gradient vector entries follow a general formula

$$\frac{\partial ss}{\partial c_j} = \sum_{i \in I} jx^{j-1}(y_i - (\sum_{j=0}^{n} c_i x^i)). \tag{A.4}$$

If the basis function is a cubic function

$$f(x|\vec{c}) = \sum_{i=0}^{3} c_i x^i,$$

we would get the following gradient function

$$\nabla_{\vec{c}}(ss(\vec{c}; x_i : i \in I)) = -2 \begin{bmatrix} \sum_{i \in I}(y_i - (c_0 + c_1 x_i + c_2 x_i^2 + c_3 x_i^3)) \\ c_1 \sum_{i \in I}(y_i - (c_0 + c_1 x_i + c_2 x_i^2 + c_3 x_i^3)) \\ 2c_2 \sum_{i \in I} x_i (y_i - (c_0 + c_1 x_i + c_2 x_i^2 + c_3 x_i^3)) \\ 3c_3 \sum_{i \in I} x_i^2 (y_i - (c_0 + c_1 x_i + c_2 x_i^2 + c_3 x_i^3)) \end{bmatrix}. \tag{A.5}$$

If we consider basis (2.6):

$$f(x|\vec{c}) = c_0 + c_1 bls(x, K, r, t, \sigma), \tag{A.6}$$

we obtain the following gradient function

$$\nabla_{\vec{c}}(ss(\vec{c}; x_i : i \in I)) = -2 \begin{bmatrix} \sum_{i \in I}(y_i - f(x_i|\vec{c})) \\ \sum_{i \in I} \text{bls}(x_i, K, \sigma)(y_i - f(x_i|\vec{c}))) \\ c_1 \sum_{i \in I} \frac{\partial \text{bls}}{\partial K}(x_i, K, \sigma)(y_i - f(x_i|\vec{c})) \\ c_1 \sum_{i \in I} \frac{\partial \text{bls}}{\partial \sigma}(x_i, K, \sigma)(y_i - f(x_i|\vec{c})) \end{bmatrix}. \tag{A.7}$$

When we consider the Black-Scholes Problem, where $c_0(K, \sigma)$ and $c_1(K, \sigma)$ are optimally solved via

$$\vec{c}(K, \sigma) = (A^T A)^{-1} A^T \vec{y}, \tag{A.8}$$

and A is the matrix

$$\begin{bmatrix} 1 & \text{bls}(x_1, K, \sigma) \\ \vdots & \vdots \\ 1 & \text{bls}(x_N, K, \sigma) \end{bmatrix},$$

we obtain the following gradient function

$$\nabla_{\vec{c}}(ss(k, \sigma; x_i : i \in I)) = -2 \begin{bmatrix} \sum_{i \in I} \frac{\partial \text{ss}}{\partial K}(x_i, K, \sigma)(y_i - f(x_i|\vec{c})) \\ \sum_{i \in I} \frac{\partial \text{ss}}{\partial \sigma}(x_i, K, \sigma)(y_i - f(x_i|\vec{c})). \end{bmatrix}. \tag{A.9}$$

Consequently, the gradient functions become

$$\frac{\partial g(K, \sigma)}{\partial K} = \nabla_{\vec{c}}(ss(\vec{c}, K, \sigma)) \frac{\partial \vec{c}(K, \sigma)}{\partial K} + \frac{\partial ss(\vec{c}, K, \sigma)}{\partial K}$$

and

$$\frac{\partial g(K, \sigma)}{\partial \sigma} = \nabla_{\vec{c}}(ss(\vec{c}, K, \sigma)) \frac{\partial \vec{c}(K, \sigma)}{\partial \sigma} + \frac{\partial ss(\vec{c}, K, \sigma)}{\partial \sigma},$$

where the Partial Derivatives of $\vec{c}$ obtained by

$$\frac{\partial \vec{c}(K,\sigma)}{\partial K} = (A^T A)^{-1}(\frac{\partial A^T}{\partial K} - \frac{\partial (A^T A)}{\partial K}), \tag{A.10}$$

and

$$\frac{\partial \vec{c}(K,\sigma)}{\partial \sigma} = (A^T A)^{-1}(\frac{\partial A^T}{\partial \sigma} - \frac{\partial (A^T A)}{\partial \sigma}). \tag{A.11}$$

If we consider basis (2.7)

$$f(x|\vec{c}) = c_0 + c_1 x + c_2 \text{bls}(x, K_1, r, t, \sigma_1) + c_3 \text{bls}(x, K_2, r, t, \sigma_2)$$

we obtain the following gradient function

$$\nabla_{\vec{c}}(ss(\vec{c}; x_i : i \in I)) = -2 \begin{bmatrix} \sum_{i \in I}(y_i - f(x_i|\vec{c})) \\ \sum_{i \in I} x_i(y_i - f(x_i|\vec{c})) \\ \sum_{i \in I} \text{bls}(x_i, K_1, \sigma_1)(y_i - f(x_i|\vec{c})) \\ \sum_{i \in I} \text{bls}(x_i, K_2, \sigma_2)(y_i - f(x_i|\vec{c})) \\ c_2 \sum_{i \in I} \frac{\partial \text{bls}}{\partial K_1}(x_i, K_1, \sigma_1)(y_i - f(x_i|\vec{c})) \\ c_2 \sum_{i \in I} \frac{\partial \text{bls}}{\partial K_2}(x_i, K_2, \sigma_2)(y_i - f(x_i|\vec{c})) \\ c_3 \sum_{i \in I} \frac{\partial \text{bls}}{\partial \sigma_1}(x_i, K_1, \sigma_1)(y_i - f(x_i|\vec{c})) \\ c_3 \sum_{i \in I} \frac{\partial \text{bls}}{\partial \sigma_2}(x_i, K_2, \sigma_2)(y_i - f(x_i|\vec{c})) \end{bmatrix}. \tag{A.12}$$

References

Black, F. and Scholes, M. (1973), 'The pricing of options and corporate liabilities', *Journal of political economy* **81**(3), 637–654.

Longstaff, F. A. and Schwartz, E. S. (2001), 'Valuing american options by simulation: a simple least-squares approach', *The review of financial studies* **14**(1), 113–147.

MATLAB (2017), *version 9.3.0.713579 (R2017b)*, The MathWorks Inc., Natick, Massachusetts.