The Use of Numerical Methods in Solving Pricing Problems for Exotic Financial Derivatives with a Stochastic Volatility

Rachael England

September 6, 2006

Declaration

I confirm that this work is my own and the use of all other material from other sources has been properly and fully acknowledged.

Acknowledgements

I would like to thank Doctor Peter K. Sweby for his supervision and patience during the course of this dissertation, and Philip McCabe for his help and advice regarding the financial and practical part of this work. I would also like to thank the Engineering and Physical Sciences Research Council and ABN AMRO for sponsoring and supporting this project, as well as various friends, family and work colleagues for their encouragement and willingness to listen.

Abstract

We firstly implement and analyse the variable θ method for the Black-Scholes model, which is a one dimensional parabolic partial differential equation. This method is then applied to various financial instruments; firstly to European Swaptions, in order to compare the results to accepted market prices; and secondly as a pricing method for exotic derivatives. However, the Black-Scholes contains known biases; in order to rectify this problem, we then apply the same process to the Heston model. The Heston model allows the derivative price to depend also upon the volatility of the underlying asset price, and assumes that the volatility also follows a stochastic process, rather than the constant value assumed by the Black-Scholes model. This results in a two dimensional partial differential equation, which we solve using a similar θ method. We also compare the results of this to the results from the Black-Scholes model. Finally, we examine an extension to the Heston model whereby the derivative price is also assumed to depend upon the price of a bond; this allows the model to incorporate the implicit value of the stochastic interest rate. A similar numerical method is applied here, and the algorithm for solving the resulting difference equations is found to be too inefficient to apply; we therefore suggest the use of an alternative algorithm method, known as the alternating direction implicit method, or of a different type of numerical method, such as a finite element scheme.

Contents

1 Introduction 6
1.1 Definition of Assets . . . 6
1.2 Definition of a Bond . . . 6
1.3 Definition of a Financial Derivative . . . 7
1.4 Vanilla Options . . . 7
1.5 Exotic Derivatives . . . 8
1.6 Definition of Arbitrage . . . 9
1.7 The Black-Scholes Model . . . 10
1.8 Motivation . . . 11

2 Terminal and Boundary Conditions 12
2.1 European Call Option . . . 12
2.2 European Put Option . . . 13
2.3 European Call Swaption . . . 14
2.4 European Put Swaption . . . 14
2.5 Compound Options . . . 15
2.6 Digital Option . . . 15
2.7 Volatility Boundary Conditions . . . 16

3 Numerical Solution of the Black-Scholes Equation 17
3.1 The Theta Method . . . 17
3.2 Invertibility of the Scheme . . . 19
3.3 Accuracy . . . 20
3.4 Stability . . . 21
3.5 Examples of Schemes . . . 21

4 Inclusion of Stochastic Volatility 22
4.1 Obtaining the Partial Differential Equation . . . 22
4.2 The Theta Scheme . . . 23
4.3 Invertibility of the Scheme . . . 31
4.4 Accuracy . . . 32
4.5 Stability . . . 35
4.6 Examples of Schemes . . . 35

5 Further Work: Inclusion of Bond Price 36
5.1 Derivation of Partial Differential Equation . . . 36
5.2 Theta Method . . . 37
5.3 Alternating Direction Implicit Method . . . 38
5.4 Finite Element Method . . . 38
5.5 Finite Volume Method . . . 38
5.6 Comparison with Monte Carlo Simulations . . . 38

6 Algorithms for solving the equations **40**
6.1 Matrix Inversion 40
6.2 Black-Scholes Solver 41
6.3 Heston Model Solver 42

7 Results **43**
7.1 Comparison with Stella Christodoulous Results 44
7.2 Variation of Boundaries and Step Sizes 45
7.3 Market Information for Volatilities and Forward Rates, and Prices for European Swaptions 46
7.4 Pricing of Knock-in Caps and Comparison with Market Data 47
7.5 Discussion of Results 48

8 Summary and Conclusion **51**

A Appendix **53**

1 Introduction

In this section we examine the general financial background and terms, and the motivation for this project. Much of this information is based upon the book 'Options, Futures, & Other Derivatives'[1].

Companies buy and sell assets and financial derivatives in the financial markets. The buyer and seller must come to an agreement over the prices of these financial instruments, and while intuition and knowledge of the current market values are used to refine the price, a pricing model is required for the initial price. A popular model for this is the Black-Scholes[2] model; however, there are known biases in this model, since it assumes that the volatility of the market is deterministic. We shall examine a different model, called the Heston[4] model, which does not make this assumption, and compare the results alongside the results according to the Black-Scholes model, comparing both to current market values. However, this model is difficult to solve analytically, particularly for exotic derivatives; numerical methods are therefore required.

1.1 Definition of Assets

Companies such as investment banks buy and sell both assets and financial derivatives in order to make money. Examples of assets include;

1. Shares: Shares in a company are nominally worth the value of the company divided by the number of shares. If the company does well and its value increases, the value of an individual share then also increases; conversely if the company does badly, the value decreases.
2. Commodities: Commodities are physical objects or substances which can be directly bought or sold, such as gold or orange juice.
3. Bonds: A bond is a promise by the writer of the bond (usually a company, called a corporate bond, or the government, called a gilt bond) to pay the holder a specified amount of money at a certain time, called the maturity. In buying a bond, one is gambling on the change in the general interest rate to increase the bond's worth; there is also a chance the writer will be unable to pay the bond when the maturity time arrives.

1.2 Definition of a Bond

A bond is a promise by a company or government (the *issuer*) to pay the buyer (*holder*) of the bond a certain percentage, called the *coupon rate*, of its redemption price at certain fixed intervals of time, usually three months. The redemption price is agreed upon at the time that a bond is first issued, although the bond itself may later be bought and sold at different prices, and is the price at which the company will later pay the holder to return the bond at the maturity time t_T.

The cash flow for anyone who buys a bond therefore looks like this:

- Time $t = t_0$ - the holder of the bond pays some price P to the issuer
- Time $t = t_1$ - the holder receives DR from the issuer, where D is the coupon rate and R is the redemption price
- Time $t = t_2$ - the holder receives DR from the issuer $\vdots \vdots$
- Time $t = t_{T-1}$ - the holder receives DR from the issuer
- Time $t = t_T$ - the holder receives DR + R from the issuer.

1.3 Definition of a Financial Derivative

Financial derivatives are products whose value is based upon the value of some underlying asset.

As an example, suppose a company wishes to buy 1,000 computers in three months time. The current price is 1,000 per computer, but they expect the price to increase. How does the company protect itself from this possibility?

There are three ways of doing this.

1. They could buy the computers now. This prevents the company from investing their money in the meantime.
2. They could make an arrangement to buy the computers in three months time for a price agreed upon now. This is called a forward contract.
3. They could buy the right, but not the obligation, to buy the computers in three months time for a price agreed upon now. This is called an option.

The last two possibilities are financial derivatives, and the price of each contract depends upon the price of the underlying asset (i.e. the price of the computers).

1.4 Vanilla Options

Vanilla options are a very popular type of financial derivative, and typical market prices are therefore easily found and regulated. There are two basic types of vanilla options:

1. Call Option: A call option gives the buyer, or *holder*, the right, but not the obligation, to buy the underlying asset from the seller of the option by a certain date (called the *maturity date*) for a certain price (called the *strike price*). Suppose the strike price is given by E and the final value of the asset price at the maturity date T is S_T. Then it can be seen that the final payoff at time T is equal to $S_T - E$ if E is less than S_T; i.e. this is the extra profit made by the holder by buying at price E instead of the current market price. However, if E is greater than S_T, then the holder will not exercise the option as the asset can be bought more cheaply at the current market price. The end payoff is this case is therefore zero.

2. Put Option: A call option gives the holder the right, but not the obligation, to sell the underlying asset to the seller of the option by a maturity date for a certain strike price. Suppose the strike price is given by E and the final value of the asset price at the maturity date T is S_T. Then it can be seen that the final payoff at time T is equal to $E - S_T$ if E is greater than S_T; i.e. this is the extra profit made by the holder by selling the asset at price E instead of the current market price. However, if E is less than than S_T, then the holder will not exercise the option as the asset can be sold for a greater value at the current market price. The end payoff is this case is therefore zero.

A vanilla option which can be exercised only at the maturity date is called a European option; if it can be exercised at any time up to the maturity date, it is called an American option.

1.5 Exotic Derivatives

Exotic derivatives refer to derivatives which are non-standard, and there are far fewer trades than for vanilla options. It is therefore more difficult to find a typical market price, and the payoff equations are often complicated. This makes pricing them more difficult to do, as an analytical solution to the various pricing models cannot always be found. This project will therefore focus upon the pricing of exotic derivatives using numerical methods.

Examples of exotic derivatives include:

1. Package: A package is a portfolio containing varying amounts of vanilla options, forward contracts, money, and the underlying assets. The payoff is relatively easy to calculate, as it will be a linear combination of the things from which it is constructed.

2. Chooser: A chooser option enables its holder to decide whether the option is a call or a put upon a certain date (not the maturity date of the final option). The price can be given as the maximum price of the underlying call and the underlying put, and can therefore also be calculated in terms of standard prices.

3. Digital Option: A digital option is worth 1 (either a unit value of cash or a unit of the underlying asset) upon certain conditions, and 0 otherwise. For instance, a cash-or-nothing call provides the holder with a payoff of 1 in cash if the underlying asset value is above a certain value at the maturity date, else the holder receives nothing; an asset-or-nothing call is similar, but provides the holder with a unit of the underlying asset (making a payoff of S_T) if the asset price is above a certain value.

4. Compound Option: A compound option is a vanilla option upon a second vanilla option. Consider a compound call-upon-call option. The holder of this derivative has bought the right, but not the obligation, to buy for a price E_1 at the maturity time T_1 the right, but not the obligation, to buy the underlying asset at time T_2 for a price E_2.

5. Barrier Option: The payoff of a barrier option depends upon whether or not the asset price has ever reached a certain value during a certain period of time. An example of this is the down-and-out barrier call, where the option is worth the same as a standard vanilla call provided the asset price did not fall below a barrier price over a certain period of time, else it is worth nothing. Similarly, there exists down-and-in options, up-and-out options, and up-and-in options.

6. Interest Rate Swaps: An interest rate swap is an agreement between two parties to pay each other a series of interest payments on a previously agreed amount, called the *principal* amount. These payments will be based upon different interest rates. For example; an exchange between a fixed rate, which is based upon the current predictions for the interest rates, and a floating rate, which will change along with the general market interest rate. In this example, if the market interest rate increases, then the second party will have to pay more than the first; if it decreases, then the opposite is true. We note from this definition that interest rate swaps are equivalent to an exchange of bonds. In the example given, the equivalent swap would be a standard fixed-rate bond as defined earlier swapped with the principal amount.

7. Swap Option (*Swaption*): A swap option gives the holder the right, but not the obligation, to enter into a specified interest rate swap at a certain maturity date T. Consider the fixed/floating interest rate swap mentioned above. Suppose a party has bought a swaption for the right to swap a fixed interest rate of x for a floating rate. If at time T the general market fixed rate exchange for a floating is less that x then the party will not exercise their swaption; if however the market rate is greater than x, then the holder will exercise their right. If we consider the bond equivalence suggested above, we can see that the payoff on such a swaption would be equal to that of a call option on the fixed rate bond with a strike price equal to the principal.

As before, a derivative which can be exercised only at the maturity date is called European; if it can be exercised at any time up to the maturity date, it is said to be American. We shall concentrate on European Swaptions for the purposes of this research.

1.6 Definition of Arbitrage

An important concept used for the pricing of financial derivatives is that of arbitrage. Arbitrage is defined as the chance to make money without the risk of loss. This breaks down into two different scenarios; making an immediate profit with no risk of future loss, and no immediate cost of future loss but the possibility of future gain. It is assumed that arbitrage does not exist in the market. In practice, such opportunities do in fact sometimes arise, but vanish quickly as market demand drives up the price.

This leads to what is known as the 'law of one price': if there exists two securities, both with the same payoff, then the securities must have the same price. If this is not the case, then an investor could buy the cheaper and sell the more expensive, thus making an immediate profit with no future cost.

Suppose we have an investment with a certain payoff K at time T. Suppose also that there exists a general risk-free interest rate r in the market. Then if an amount equal to Ke^{-rT} is invested in the risk-free security, it will also be worth K at time T. So by the law of one price, the original investment must also have price Ke^{-rT}, and must also grow at the risk free rate r.

1.7 The Black-Scholes Model

One model which is commonly used to calculate the price of financial derivatives is the Black-Scholes model[2]. This model was developed by Fischer Black and Myron Scholes in 1973, and the key idea behind it is that it is possible to develop a riskless portfolio of one derivative and an amount of the asset; by the assumption of no arbitrage, this portfolio must therefore grow at the riskless rate.

Black and Scholes began by assuming that the underlying asset price grows according to the equation

$$dS = \mu dt + \sigma dW \tag{1}$$

where S represents the price of the asset, t represents the time, and W represents a random term with a Wiener process distribution. Hence, the rate of change of the asset price is proportional to some growth term μ with respect to time plus some random normalised term which is proportional to the volatility represented by σ.

Let $U = U(S, t)$ represent the value of the derivative at time t according to the price of the underlying asset. Then by Taylor's theorem

$$\begin{array}{rcl} dU & = & U(t + dt, S + dS) - U(t, S) \\ & = & \frac{\partial U}{\partial t}(t, S)dt + \frac{\partial U}{\partial S}(t, S)dS + \frac{1}{2}\frac{\partial^2 U}{\partial S^2}(t, S)(dS)^2 + O(dt^{\frac{3}{2}}). \end{array} \tag{2}$$

By equation (1) dS^2 can be calculated as

$$\begin{array}{rcl} (dS)^2 & = & (\mu dt + \sigma dW)^2 \\ & = & \mu^2 dt^2 + 2\mu\sigma dt dW + \sigma^2 dW^2. \end{array}$$

Noting that $dW^2 = dt$ we can discard terms of order $dt^{\frac{3}{2}}$ and higher to get

$$(dS)^2 = \sigma^2 dt. \tag{3}$$

Equations (1) and (3) can then be substituted back into (2) to get

$$dU = \sigma \frac{\partial U}{\partial S} dW + (\frac{\partial U}{\partial t} + \mu \frac{\partial U}{\partial S} + \frac{1}{2}\sigma^2 \frac{\partial^2 U}{\partial S^2})dt. \tag{4}$$

We can now construct a portfolio by buying one derivative and selling Δ lots of the underlying asset. Then the value of the portfolio Π is given by

$$\Pi = U - \Delta S. \tag{5}$$

The rate of change of the value of the portfolio can therefore be given by

$$d\Pi = dU - \Delta dS$$

and substituting (1) and (4) into this equation gives

$$\begin{aligned} d\Pi &= \sigma \tfrac{\partial U}{\partial S} dW + (\tfrac{\partial U}{\partial t} + \mu \tfrac{\partial U}{\partial S} + \tfrac{1}{2}\sigma^2 \tfrac{\partial^2 U}{\partial S^2})dt - \Delta(\mu dt + \sigma dW) \\ &= \sigma(\tfrac{\partial U}{\partial S} - \Delta)dW + (\tfrac{\partial U}{\partial t} + \mu \tfrac{\partial U}{\partial S} + \tfrac{1}{2}\sigma^2 \tfrac{\partial^2 U}{\partial S^2} - \mu\Delta)dt. \end{aligned}$$

We can set Δ equal to $\frac{\partial U}{\partial S}$ to get

$$d\Pi = (\frac{\partial U}{\partial t} + \frac{1}{2}\sigma^2 \frac{\partial^2 U}{\partial S^2})dt. \tag{6}$$

As the random term has vanished, this portfolio is risk free, and by the law of one price must therefore grow at the risk free rate. Hence

$$\begin{aligned} d\Pi &= r\Pi dt \\ &= r(U - S\tfrac{\partial U}{\partial S})dt. \end{aligned} \tag{7}$$

The two equations (6) and (7) can then be set equal, rearranged, and divided by dt to get

$$\frac{1}{2}\sigma^2 \frac{\partial^2 U}{\partial S^2} + rs\frac{\partial U}{\partial S} - rU + \frac{\partial U}{\partial t} = 0. \tag{8}$$

This is the Black-Scholes partial differential equation, and can be solved together with the payoff at expiry to obtain a solution for the value of the financial derivative.

1.8 Motivation

However the Black-Scholes equation contains known biases, as was documented by Mark Rubenstein in 1998[3]. In order to account for this, Steven L. Heston[4] suggested a different model for the movement of the underlying asset price. This model is as follows:

$$dS = \mu S dt + \sqrt{v(t)} S dW_1 \tag{9}$$

where v is this time given as the volatility. It can be seen from this that the rate of change of the asset price is assumed to be proportional to the price of the asset, and that the volatility itself may be some equation that evolves with time.

We shall use an Ornstein-Uhlenbeck[5] process to represent the volatility. This is given by

$$dv(t) = k[\theta - v(t)]dt + \sigma\sqrt{v(t)}dW_2. \tag{10}$$

2 Terminal and Boundary Conditions

In this section we shall examine different derivatives and their payoffs, as well as determining boundary conditions. These conditions will then be examined from a numerical methods perspective.

2.1 European Call Option

Let $C(S,t)$ be the value of a standard European call option with a strike price of E and a maturity time T, where S is the current value of the underlying asset, and t is the current time. Consider the payoff at time T. If the value of E is greater than the value of S at this time, then the holder will not exercise the option; the payoff in this case is therefore zero. If the value of E is less than the value of S at this time, then the holder will exercise the option, and the gain in using the option rather than the market value will be equal to $S-E$. This gives the expression for the final payoff as

$$C(S,T) = max(S-E,0); \tag{11}$$

this is our terminal condition for the models.

Allow S to approach infinity. In this case it becomes more and more likely that the holder will exercise the right to buy the asset for a price of E. E will also become small in comparison with S; this creates the boundary condition

$$C(S,t) \to S \text{ as } S \to \infty. \tag{12}$$

Allow S to approach zero. In this case it becomes less and less likely that the holder willl exercise the right to buy the asset for a price of E, and more likely that the option will expire worthless. The price will therefore become zero. This creates the boundary condition

$$C(0,t) = 0. \tag{13}$$

We now consider the use of these conditions within the context of a numerical finite difference scheme. Rather than using a terminal condition, we would prefer to take an initial condition and step forward through time. In order to achieve this, we shall transform the equation by replacing t with τ, where $\tau = T-t$. This transforms the terminal condition into the initial condition

$$C(S,0) = max(S-E,0) \tag{14}$$

where $C(S,\tau)$ is the price of the call option for the asset price S at time $T-\tau$.

A numerical scheme also cannot be solved across an infinite plane. It is also probable that the required range of asset prices is much higher than zero; in this case we would not want to extend our numerical plane back to $S=0$ as this would be inefficient. Instead, we shall choose a suitably small value S^- and a suitably large value S^+, and calculate equivalent conditions at these points. We note that these conditions must match with the initial condition at $\tau = 0$.

Previous work by Stella Christodoulou[6] has shown that Dirichlet approximations for financial derivatives produce the same effect as Neumann conditions, provided the required range of solutions for S is far enough away from the boundaries. As we are only concerned with the middle range values for S, this is acceptable; hence, Dirichlet conditions shall be used. This produces the conditions

$$\begin{aligned} C(S^-, \tau) &= 0 \\ C(S^+, \tau) &= S^+ - Ee^{-r\tau}. \end{aligned} \tag{15}$$

2.2 European Put Option

Let $P(S, t)$ be the value of a standard European put option with a strike price of E and a maturity time T, where S is the current value of the underlying asset, and t is the current time. Consider the payoff at time T. If the value of E is less than the value of S at this time, then the holder will not exercise the option; the payoff in this case is therefore zero. If the value of E is greater than the value of S at this time, then the holder will exercise the option, and the gain in using the option rather than the market value will be equal to $E - S$. This gives the expression for the final payoff as

$$P(S, T) = max(E - S, 0); \tag{16}$$

this is our terminal condition for the models.

Allow S to approach infinity as before. In this case it becomes very unlikely that the holder will exercise the right to sell the asset for a price of E when the market value is much higher; the option is therefore likely to expire worthless. This creates the boundary condition

$$P(S, t) \to 0 \text{ as } S \to \infty. \tag{17}$$

Now allow S to approach zero. In this case it becomes likely that the holder will exercise the right to sell the asset for a price of E, as the prevailing market price will also be close to zero. The payoff therefore becomes likely to approach E, a value which must then also be discounted to the current price. This creates the boundary condition

$$P(0, t) = Ee^{-r(T-t)}. \tag{18}$$

We now consider the use of these conditions within the context of a numerical finite difference scheme as before. We shall again transform the equation by replacing t with τ, where $\tau = T - t$. This transforms the terminal condition into the initial condition

$$P(S, 0) = max(E - S, 0) \tag{19}$$

where $P(S, \tau)$ is the price of the put option for the asset price S at time $T - \tau$.

As before, we shall use equivalent conditions at S^+ and S^-, where S^+ is suitably large and S^- is suitably small. These produce the Dirichlet boundary conditions

$$P(S^-,\tau) = Ee^{-r\tau} - S^- \tag{20}$$
$$C(S^+,\tau) = 0.$$

2.3 European Call Swaption

Suppose the holder of a swaption has the right, but not the obligation, to enter into an interest rate swap at time T, where the swap lasts for n years and enables the holder to pay a fixed rate R_x (which is decided at the time of issuing the swaption) once a year in exchange for receiving the floating market rate.

Consider the payoff at time T. There will be some rate R which the market at this time considers to be the equivalent of receiving the floating rate. If R is less than R_x then the holder will not exercise their right as it would be cheaper to pay the fixed rate R_x; however if R is greater than R_x, then the holder will exercise the right. This makes the payoff at each successive time interval that interest rates are exchanged equal to $max(R - R_x, 0)$.

Assume we receive a payment at time t_i of 1. Due to interest rates being present within the financial markets, the current value of this amount is in fact equal to

$$D(t_i) = e^{-r_i t_i} \tag{21}$$

where r_i is the interest applied to any sum of money at time t_i (the predicted values of the r_i are called the *spot rate*).

By applying this to each of the payoffs during the lifetime of the swap, we can calculate the expected current value of the total payoff as

$$(\sum_{i=1}^{m} D(t_i))C(R, 0, R_x) \tag{22}$$

where $C(R, t, R_x)$ is the value of a call with underlying asset R and a strike price of R_x at time t. Here R at time $t \neq T$ is taken to be the expected value at time t for the rate at which fixed interest rate payments may be exchanged for floating rate payments at time T, and is known as the *forward rate.*

We also note that as we have already used spot rates to calculate the current value of payments at different times, we may set r equal to zero when solving this call. The conditions for a call option may then be used in order to find a value for the swaption.

2.4 European Put Swaption

Similarly, suppose the holder of a swaption has the right, but not the obligation, to enter into an interest rate swap at time T, where the swap lasts for n years and enables the holder to receive a fixed rate R_x (which is decided at the time of issuing the swaption) once a year in exchange for paying the floating market rate.

Consider the payoff at time T. There will be some rate R which the market at this time considers to be the equivalent of receiving the floating rate. If R is greater than R_x then the holder will not exercise their right as they would receive more money for receiving R than for the fixed rate R_x; however if R is less than R_x, then the holder will exercise the right to receive the higher rate. This makes the payoff at each successive time interval that interest rates are exchanged equal to $max(R_x - R, 0)$.

By applying the values of $P(t_i)$ as before to each of the payoffs during the lifetime of the swap, we can calculate the expected current value of the total payoff as

$$(\sum_{i=1}^{m} D(t_i))P(R, 0, R_x) \tag{23}$$

where $P(R, t, R_x)$ is the value of a put with underlying asset R and a strike price of R_x at time t. Here R at time $t \neq T$ is taken to be the forward rate as before; we may also once again set r to zero. The conditions for a standard put option may then be used in order to find a value for the swaption.

2.5 Compound Options

Suppose an investor holds a call-upon-call option of value $ConC(S, \tau)$, where $\tau = T - t$ and T is the maturity of the call-upon-call. At time T they have the right to buy a call option for a strike price E. This call option would then given them the right to buy at time T_2 the underlying asset of value S for a strike price of E_2.

Consider the payoff at time T. If the value of the call is worth less than E, then the holder will not exercise the call-upon-call option, which will expire worthless. If the value of the call is greater than E, then the holder will exercise the right, thus obtaining a payoff of $C(S, T, E_2) - E$.

In order to solve this option numerically, it is necessary to run the model twice; the first to solve for the call in order to find the price of the call at time T with respect to S, and the second time using the same boundary conditions as in a call but with the initial condition

$$ConC(S, 0) = max(C(S, T, E_2) - E, 0). \tag{24}$$

Similar working may be used to price the call-upon-put, put-upon-put, and put-upon-call.

2.6 Digital Option

Suppose an investor holds a digital option which reaches maturity at time T and strtike E. Then by definition, the payoff of the digital option is equal to

$$Di(S, T) = \frac{max(S - E, 0)}{S - E} \tag{25}$$

.

As before, we will transform the equation using $\tau = T - t$ in order to obtain the terminal condition

$$Di(S,0) = \frac{max(S-E,0)}{S-E} \tag{26}$$

where $Di(S,\tau$ is the value of the digital option at time $T-\tau$ and S is the value of the underlying asset.

Consider the case where S approaches zero. Then the value of the digital option must approach $e^{-r\tau}$, as this is the current value of a payoff of 1. Similarly, as S approaches infinity, the digital option will approach 0. Translated into a numerical scheme, this gives the conditions

$$Di(S^-,\tau) = 0 \tag{27}$$
$$Di(S^+,\tau) = e^{-r\tau}$$

2.7 Volatility Boundary Conditions

Boundary conditions are also needed for the Heston model. For all of these derivatives, as the volatility approaches zero or infinity, the price approaches a steady state. This is reflected by using Neumann conditions instead of the Dirichlet conditions used for the S boundaries.

As with the asset price, we shall use a relatively large value v^+ in order to replicate the condition as v approaches infinity, and a relatively small value v^- to replicate the condition as v approaches zero. This gives us the condtions

$$\frac{\partial U(S,v^-,\tau)}{\partial v} = 0 \tag{28}$$
$$\frac{\partial U(S,v^+,\tau)}{\partial v} = 0$$

where $U(S,v,\tau)$ is the value of the derivative in question for an underlying asset price of S and a volatility of v at time $t = T - \tau$.

3 Numerical Solution of the Black-Scholes Equation

In this section we shall examine the use of the θ method for the numerical solution of the Black-Scholes equation. This method was implemented by Stella Christodoulou[6] in 2000, and we shall try to reproduce her results, as well as analysing the accuracy, stability, and solvability of the numerical equations.

3.1 The Theta Method

In order to numerically solve the Black-Scholes equation, we will first transform equation (8) by setting $\tau = T - t$ where T is the maturity time of the derivative. This way we can step forward through the τ variable instead of backwards through time. By doing this, equation (8) becomes

$$\frac{\partial U}{\partial \tau} = \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 U}{\partial S^2} + rS\frac{\partial U}{\partial S} - rU. \tag{29}$$

We now divide up the (S, τ) plane into discrete intervals as shown below in order to numerically solve this equation.

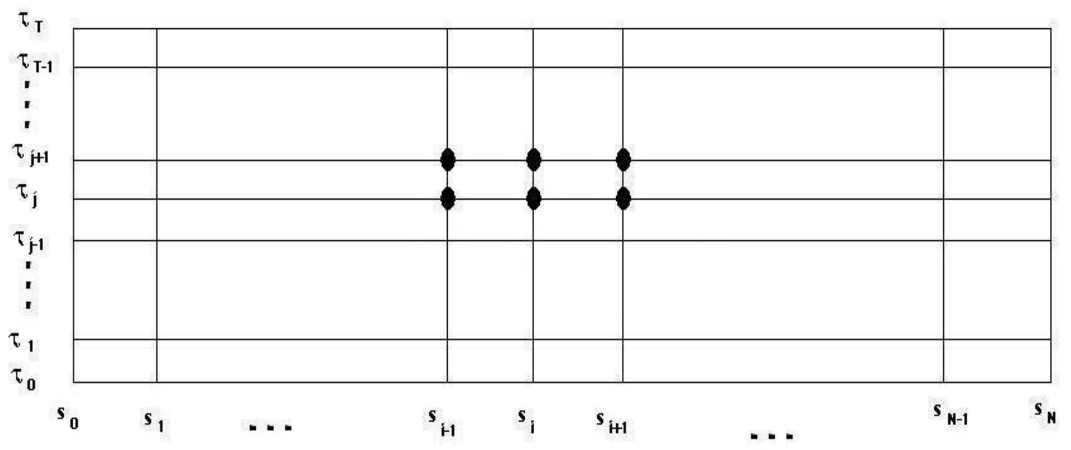


Figure 1: The discrete (S, τ) plane with the stencil marked as dots.

These discrete points can be used to approximate the differential equation (29) by taking a central difference for the S terms and a forward difference for the τ term. Here, we shall take a central difference at both time j and time $j+1$ for the S terms, and use a weighted average of the two.

Thus the approximations can be expressed as

$$\frac{\partial u}{\partial \tau} \approx \frac{U_i^{j+1} - U_i^j}{\delta\tau} \tag{30}$$

$$\frac{\partial^2 U}{\partial S^2} \approx \theta_1(\frac{U_{i-1}^{j+1} - 2U_i^{j+1} + U_{i+1}^{j+1}}{\delta S^2}) + \theta_2(\frac{U_{i-1}^{j} - 2U_i^{j} + U_{i+1}^{j}}{\delta S^2})$$

$$\frac{\partial U}{\partial S} \approx \theta_3(\frac{U_{i+1}^{j+1} - U_{i-1}^{j+1}}{2\delta S}) + \theta_4(\frac{U_{i+1}^{j} - U_{i-1}^{j}}{2\delta S})$$

$$U \approx \theta_5 U_i^{j+1} + \theta_6 U_i^j$$

where $U_i^j \approx U(S_i, \tau_j)$.

The approximations (30) can then be inserted into equation (29) to obtain the finite difference equation

$$\begin{array}{rcl} \frac{U_i^{j+1}-U_i^j}{\delta\tau} & = & \frac{1}{2}\sigma^2 S_i^2[\theta_1(\frac{U_{i-1}^{j+1}-2U_i^{j+1}+U_{i+1}^{j+1}}{\delta S^2}) + \theta_2(\frac{U_{i-1}^{j}-2U_i^{j}+U_{i+1}^{j}}{\delta S^2})] \\ & & +rS_i[\theta_3(\frac{U_{i+1}^{j+1}-U_{i-1}^{j+1}}{2\delta S}) + \theta_4(\frac{U_{i+1}^{j}-U_{i-1}^{j}}{2\delta S})] - r[\theta_5 U_i^{j+1} + \theta_6 U_i^j] \end{array} \tag{31}$$

where $\theta_1 + \theta_2 = \theta_3 + \theta_4 = \theta_5 + \theta_6 = 1$, and $0 < \theta_i < 1$ for all i.

Setting

$$\begin{array}{c} \alpha_i = \frac{1}{2}\sigma^2 S_i^2 \frac{\delta\tau}{\delta S^2} \\ \beta_i = \frac{1}{2} r S_i \frac{\delta\tau}{\delta S} \\ \gamma_i = -r\delta\tau \end{array} \tag{32}$$

and multiplying both sides by $\delta\tau$ we obtain the equation

$$\begin{array}{rcl} U_i^{j+1} - U_i^j & = & \alpha_i[\theta_1(U_{i-1}^{j+1} - 2U_i^{j+1} + U_{i+1}^{j+1}) + \theta_2(U_{i-1}^{j} - 2U_i^{j} + U_{i+1}^{j})] \\ & & +\beta_i(U_{i+1}^{j+1} - U_{i-1}^{j+1}) + \theta_4(U_{i+1}^{j} - U_{i-1}^{j})] + \gamma_i[\theta_5 U_i^{j+1} + \theta_6 U_i^j]. \end{array} \tag{33}$$

We now define the following

$$\begin{array}{c} a_i = -\alpha_i\theta_1 + \beta_i\theta_3 \\ b_i = 1 + 2\alpha_i\theta_1 - \gamma_i\theta_5 \\ c_i = -\alpha_i\theta_1 - \beta_i\theta_3 \\ a_i' = \alpha_i\theta_2 - \beta_i\theta_4 \\ b_i' = 1 - 2\alpha_i\theta_2 + \gamma_i\theta_6 \\ c_i' = \alpha_i\theta 2 + \beta_i\theta_4 \end{array} \tag{34}$$

and using (34) with (33) we obtain

$$a_i U_{i-1}^{j+1} + b_i U_i^{j+1} + c_i U_{i+1}^{j+1} = a_i' U_{i-1}^j + b_i' U_i^j + c_i' U_{i+1}^j. \tag{35}$$

It can be seen from this equation that at each time step, the scheme may be applied by solving the matrix equation

$$A\mathbf{U^{j+1}} = \mathbf{x} \tag{36}$$

where

$$A = \begin{pmatrix} b_1 & c_1 & 0 & \dots & 0 \\ a_2 & b_2 & c_2 & & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & & a_{N-2} & b_{N-2} & c_{N-2} \\ 0 & & 0 & a_{N-1} & b_{N-1} \end{pmatrix}$$

$$\mathbf{U^{j+1}} = \begin{pmatrix} U_1^{j+1} \\ U_2^{j+1} \\ \vdots \\ U_{N-1}^{j+1} \end{pmatrix}$$

$$\mathbf{x} = \begin{pmatrix} a'_1U_0^j + b'_1U_1^j + c'_1U_2^j - a_1U_0^{j+1} \\ a'_2U_1^j + b'_2U_2^j + c'_2U_3^j \\ a'_3U_2^j + b'_3U_3^j + c'_3U_4^j \\ \vdots \\ a'_{N-2}U_{N-3}^j + b'_{N-2}U_{N-2}^j + c'_{N-2}U_{N-1}^j \\ a'_{N-1}U_{N-2}^j + b'_{N-1}U_{N-1}^j + c'_{N-1}U_N^j - c_{N-1}U_N^{j+1} \end{pmatrix}$$

and N is the number of steps on the S axis, and a_i, b_i and c_i are defined as above.

3.2 Invertibility of the Scheme

If the matrix A is singular, then the finite difference equation cannot be solved. In order to ensure the invertibility of A, consider the following theorem.

Theorem 1: Suppose the matrix A is such that $|A_{(i,i)}| > \sum_{i\neq j} |A_{(i,j)}|$ (strictly diagonally dominant). Then A is non-singular.

It is therefore possible to conclude that for this case, we can guarantee the invertibility of A by ensuring that $|b_i| > |a_i| + |c_i|$.

i.e.

$$|1 + 2\alpha_i\theta_1 - \gamma\theta_5| > | - \alpha_i\theta_1 + \beta_i\theta_3| + | - \alpha_i\theta_1 - \beta_i\theta_3|.$$

By the definitions given in (32), it can be seen that α_i, $\beta_i > 0$ and $\gamma_i < 0$. Hence, this equation is true if and only if

$$1 + 2\alpha_i\theta_1 - \gamma_i\theta_5 > \alpha_i\theta_1 + \beta_i theta_3 + |-\alpha_i\theta_1 + \beta_i\theta_3|.$$

Therefore, we require one of the following two conditions to be fulfilled:

$$\alpha_i\theta_1 > \beta_i\theta_3 \tag{37}$$

or

$$\begin{gathered}\alpha_i\theta_1 < \beta_i\theta_3,\\ 1 - \gamma_i\theta_5 > 2(\beta_i\theta_3 - \alpha_i\theta_1).\end{gathered}$$

3.3 Accuracy

The truncation error Φ_i^j is a measure of the discretisation error of the scheme; i.e. the error in approximating one step of the equation. This can be measured using the equation $\Phi_i^j = L_i^j(U - U_i^j)L_i^j(U)$ where $L_i^j(U)$ represents the application of the numerical scheme to U.

In this case, this becomes

$$\begin{aligned}\Phi_i^j = & -\tfrac{1}{\delta\tau}\big(U(S_i,\tau_j+\delta\tau) - U(S_i,\tau_j)\big)\\ & +\tfrac{\sigma^2 S_i^2}{2\delta S^2}\Big[\theta_1\big(U(S_i-\delta S,\tau_j+\delta\tau) - 2U(S_i,\tau_j+\delta\tau) + U(S_i+\delta S,\tau_{j+\delta\tau})\big)\\ & +\theta_2\big(U(S_i+\delta S,\tau_j) - 2U(S_i,\tau_j) + U(S_i-\delta S,\tau_j)\big)\Big]\\ & +\tfrac{rS_i}{2\delta S}\Big[\theta_3\big(U(S_i+\delta S,\tau_j+\delta\tau) - U(S_i-\delta S,\tau_j+\delta\tau)\big) + \theta_4\big(U(S_i+\delta S,\tau_j) - U(S_i-\delta S,\tau_j)\big)\Big]\\ & -r\Big[\theta_5 U(S_i,\tau_j+\delta\tau) + \theta_6 U(S_i,\tau_j)\Big]\end{aligned}$$

and using Taylor's theorem, we can expand around $U(S_i, \tau_j)$ to obtain

$$\begin{aligned}\Phi_i^j = & -\tfrac{1}{\delta\tau}\big(U + \delta\tau U_\tau + \tfrac{\delta\tau^2}{2}U_{\tau\tau} + \ldots - U\big)\\ & \tfrac{\sigma^2 S_i^2}{2\delta S^2}\Big[\theta_1\big(U(S,\tau+\delta\tau) - \delta S U_S(S,\tau+\delta\tau) + \tfrac{\delta S^2}{2}U_{SS}(S,\tau+\delta\tau) - \tfrac{\delta S^3}{6}U_{SSS}(S,\tau+\delta\tau)\\ & +\tfrac{\delta S^4}{24}U_{SSSS}(S,\tau+\delta\tau) + \ldots - 2U(S,\tau+\delta\tau) + U(S,\tau+\delta\tau) + \delta S U_s(S,\tau+\delta\tau)\\ & +\tfrac{\delta S^2}{2}U_{SS}(S,\tau+\delta\tau) + \tfrac{\delta S^3}{6}U_{SSS}(S,\tau+\delta\tau) + \tfrac{\delta S^4}{24}U_{SSSS}(S,\tau+\delta\tau) + \ldots\big)\\ & +\theta_2\big(U - \delta S U_S + \tfrac{\delta S^2}{2}U_{SS} - \tfrac{\delta S^3}{6}U_{SSS} + \tfrac{\delta S^4}{24}U_{SSSS} + \ldots - 2U\\ & +U + \delta S U_S + \tfrac{\delta S^2}{2}U_{SS} + \tfrac{\delta S^3}{6}U_{SSS} + \tfrac{\delta S^4}{24}U_{SSSS} + \ldots\big)\Big]\\ & +\tfrac{rS_i}{2\delta S}\Big[\theta_3\big(U(S,\tau+\delta\tau) + \delta S U_S(S,\tau+\delta\tau) + \tfrac{\delta S^2}{2}U_{SS}(S,\tau+\delta\tau) + \tfrac{\delta S^3}{6}U_{SSS}(S,\tau+\delta\tau) + \ldots\\ & -U(S,\tau+\delta\tau) + \delta S U_s(S,\tau+\delta\tau) - \tfrac{\delta S^2}{2}U_{SS}(S,\tau+\delta\tau) + \tfrac{\delta S^3}{6}U_{SSS}(S,\tau+\delta\tau) - \ldots\big)\\ & +\theta_4\big(U + \delta S U_S + \tfrac{\delta S^2}{2}U_{SS} + \tfrac{\delta S^3}{6}U_{SSS} + \ldots - U\\ & +\delta S U_S - \tfrac{\delta S^2}{2}U_{SS} + \tfrac{\delta S^3}{6}U_{SSS} - \ldots\big)\Big] - r\Big[\theta_5\big(U + \delta\tau U_t + \tfrac{\delta\tau^2}{2}U_{\tau\tau} + \ldots\big) + \theta_6 U].\end{aligned}$$

Remembering that $\theta_1+\theta_2=\theta_3+\theta_4=\theta_5+\theta_6=1$ and using equation (29), this can be simplified to

$$\Phi_i^j=-\frac{\delta\tau}{2}U_{\tau\tau}+\frac{\sigma^2S^2\delta S^2}{24}U_{SSSS}+\frac{\sigma^2S^2\theta_1\delta\tau}{2}U_{SS\tau}+\frac{rS\delta S^2}{6}U_{SSS}+rS\theta-3\delta\tau U_{S\tau}-r\theta_5\delta\tau U_\tau+\text{higher order terms.} \tag{38}$$

Hence, this numerical approximation is accurate to first order time, second order asset price.

3.4 Stability

If a scheme is unstable, small errors will blow up at each application of the scheme; so the solution will not be accurate, even if truncation error is small.

As this equation is in a similar form as the diffusion equation, its stability can be calculated using fourier stability applied locally - i.e. the condition must be satisfied over all points within the solution domain.

We let $U_i^j=\Lambda^je^{-zni\delta S}$ where $z=\sqrt{-1}$ and n is an arbitrary constant, and substitute this expression into the numerical scheme. In order for the scheme to be stable, $|\Lambda|$ must be less than 1.

Therefore in this case, it is necessary to choose values of θ_i and δS, $\delta\tau$ such that

$$|a_i'e^{-zn\delta S}+b_i'+c_i'e^{zn\delta S}|<|a_ie^{-zn\delta S}+b_i+c_ie^{zn\delta S}| \tag{39}$$

for all i.

3.5 Examples of Schemes

Scheme	θ_1	θ_2	θ_3	θ_4	θ_5	θ_6	S.D.D. Interval	Stability Interval
Crank-Nicolson	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$r<\sigma^2S_i$	Unconditional
Kenneth-Vetzal	1	0	1	0	0	1	$r<\sigma^2S_i$	$\delta\tau\leq\frac{2}{r}$
Fully Implicit	1	0	1	0	1	0	$r<\sigma^2S_i$	Unconditional
Semi Implicit	1	0	0	1	1	0	Unconditional	Unconditional
Explicit 1	0	1	0	1	0	1	Unconditional	$\delta\tau\leq\frac{\sigma^2}{r^2+r\sigma^2}$
Explicit 2	0	1	0	1	1	0	Unconditional	$\delta\tau\leq\frac{\sigma^2}{r^2}$ and $\frac{\delta\tau}{\delta S^2}\leq\frac{1}{\sigma^2S_i^2}$

4 Inclusion of Stochastic Volatility

We shall now examine the Heston model and numerically solve the partial differential equation using the same θ method as before. Stella Christodoulou[6] also tried to solve this equation; however, she used transformations to create a finite difference equation which could be solved using the alternating direction implicit algorithm. We shall instead attempt to solve the differential equation over the entire (S, τ) domain simultaneously. Hence, such a transformation is not required.

4.1 Obtaining the Partial Differential Equation

Using the amended model for the asset price as suggested in the introduction, we can now perform a similar analysis to before to obtain a new differential equation.

Let $U = U(S, v, t)$ represent the value of the derivative at time t according to the price of the underlying asset. Then by Taylor's theorem

$$\begin{aligned} dU &= U(S + dS, v + dv, t + dt) - U(S, v, t) \\ &= \tfrac{\partial U}{\partial t}(S, v, t)dt + \tfrac{\partial U}{\partial S}(S, v, t)dS + \tfrac{\partial U}{\partial v}(S, v, t)dv + \tfrac{1}{2}\tfrac{\partial^2 U}{\partial S^2}(S, v, t)(dS)^2 \\ &\quad + \tfrac{1}{2}\tfrac{\partial^2 v}{\partial v^2}(S, v, t)(dv)^2 + \tfrac{\partial^2 U}{\partial S \partial v}(S, v, t)dSdv + O(dt^{\frac{3}{2}}). \end{aligned} \tag{40}$$

By equation (9) and discarding terms of order $dt^{\frac{3}{2}}$ as before, dS^2 can be calculated as

$$(dS)^2 = vS^2 dt. \tag{41}$$

Similarly, (10) can be used to obtain the equation

$$(dv)^2 = \sigma^2 v dt \tag{42}$$

and a similar process can be used which results in the expression

$$dSdv = \sigma S v dW_1 dW_2 = \sigma S v \rho dt \tag{43}$$

where ρ represents the correlation between the two Wiener processes W_1 and W_2.

Equations (9), (41), (42) and (43) can then be substituted back into (40), which can then be rearranged to get

$$dU = \sqrt{v}S\frac{\partial U}{\partial S}dW_1 + \sigma\sqrt{v}\frac{\partial U}{\partial v}dW_2 + (\frac{\partial U}{\partial t} + \mu S\frac{\partial U}{\partial S} + k[\theta - v]\frac{\partial U}{\partial v} + \frac{1}{2}S^2 v\frac{\partial^2 U}{\partial S^2} + \sigma S v \rho\frac{\partial^(2)U}{\partial S \partial v} + \frac{1}{2}\sigma^2 v\frac{\partial^2 U}{\partial V^2})dt. \tag{44}$$

We can now construct a portfolio by buying one derivative and selling Δ lots of the underlying asset. Then the value of the portfolio Π is given by

$$\Pi = U - \Delta S. \tag{45}$$

The rate of change of the value of the portfolio can therefore be given by

$$d\Pi = dU - \Delta dS$$

and substituting (9) and (44) into this equation and setting $\Delta = \frac{\partial U}{\partial S}$ as before gives

$$d\Pi = (\frac{\partial U}{\partial t} + k[\theta - v]\frac{\partial U}{\partial v} + \frac{1}{2}vS^2\frac{\partial^2 U}{\partial S^2} + \sigma S v\rho\frac{\partial^2 U}{\partial v \partial S} + \frac{1}{2}\sigma^2 v\frac{\partial^2 U}{\partial v^2})dt + \sigma\sqrt{v}\frac{\partial U}{\partial v}dW_2. \tag{46}$$

The random term in W_2 can be replaced by $-\lambda(t,S,v)\frac{\partial U}{\partial v}dt$ where lambda is called the 'price of volatility risk', and represents the extra amount of capital return that the investor in the derivative expects to gain in exchange for taking on the risk. It is a measurement of how risk averse investors are likely to be; we shall assume that $\lambda = 0$.

As the random term has now vanished, this portfolio is risk free, and by the law of one price must therefore grow at the risk free rate. Hence

$$\begin{aligned} d\Pi &= r\Pi dt \\ &= r(U - S\tfrac{\partial U}{\partial S})dt. \end{aligned} \tag{47}$$

The two equations (46) and (47) can then be set equal, rearranged, and divided by dt to get

$$\frac{1}{2}vS^2\frac{\partial^2 U}{\partial S^2} + \rho\sigma v S\frac{\partial^2 U}{\partial S \partial v} + \frac{1}{2}\sigma^2 v\frac{\partial^2 U}{\partial v^2} + rs\frac{\partial U}{\partial S} + k[\theta - v(t)]\frac{\partial U}{\partial v} - rU + \frac{\partial U}{\partial t} = 0. \tag{48}$$

This is the Heston model. It is possible to solve this analytically for financial derivatives with a simple payoff using a transformation of variables to transform this into a parabolic equation. However, it is often not possible to find a solution in this manner when dealing with exotic derivatives as the payoff equations are much more complex. Instead, we shall once again use numerical methods to solve this model, and the result may be validated by applying the schemes to the more standard options.

4.2 The Theta Scheme

As before, we will first transform equation (48) by setting $\tau = T - t$ where T is the maturity time of the derivative. This way we can step forward through the τ variable instead of backwards through time. By doing this, equation (48) becomes

$$\frac{1}{2}vS^2\frac{\partial^2 U}{\partial S^2} + \rho\sigma v S\frac{\partial^2 U}{\partial S \partial v} + \frac{1}{2}\sigma^2 v\frac{\partial^2 U}{\partial v^2} + rs\frac{\partial U}{\partial S} + k[\theta - v(t)]\frac{\partial U}{\partial v} - rU = \frac{\partial U}{\partial \tau}. \tag{49}$$

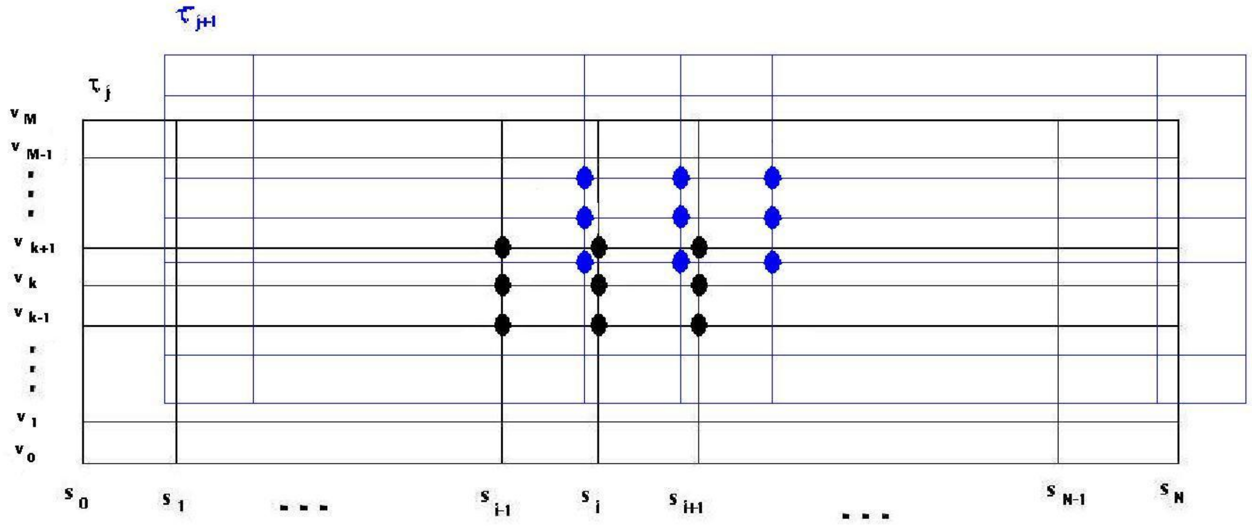


Figure 2: The discrete (S, v) plane for time levels j and $j+1$ with the stencil marked as dots.

We now divide up the (S, v, τ) plane into discrete intervals as shown below in order to numerically solve this equation.

These discrete points can be used to approximate the differential equation (49) by taking a central difference for the S and v terms and a forward difference for the τ term. Here, we shall take a central difference at both time j and time $j+1$ for the S and v terms, and use a weighted average of the two.

Thus the approximations can be expressed as

$$\frac{\partial u}{\partial \tau} \approx \frac{U_{i,k}^{j+1} - U_{i,k}^{j}}{\delta \tau} \tag{50}$$

$$\frac{\partial^2 U}{\partial S^2} \approx \theta_1 (\frac{U_{i-1,k}^{j+1} - 2U_{i,k}^{j+1} + U_{i+1,k}^{j+1}}{\delta S^2}) + \theta_2 (\frac{U_{i-1,k}^{j} - 2U_{i,k}^{j} + U_{i+1,k}^{j}}{\delta S^2})$$

$$\frac{\partial^2 U}{\partial S \partial v} \approx \theta_3 (\frac{U_{i+1,k+1}^{j+1} - U_{i-1,k+1}^{j+1} - U_{i+1,k-1}^{j+1} + U_{i-1,k-1}^{j+1}}{4\delta S \delta v}) + \theta_4 (\frac{U_{i+1,k+1}^{j} - U_{i-1,k+1}^{j} - U_{i+1,k-1}^{j} + U_{i-1,k-1}^{j}}{4\delta S \delta v})$$

$$\frac{\partial^2 U}{\partial v^2} \approx \theta_5 (\frac{U_{i,k-1}^{j+1} - 2U_{i,k}^{j+1} + U_{i,k+1}^{j+1}}{\delta v^2}) + \theta_6 (\frac{U_{i,k-1}^{j} - 2U_{i,k}^{j} + U_{i,k+1}^{j}}{\delta v^2})$$

$$\frac{\partial U}{\partial S} \approx \theta_7 (\frac{U_{i+1,k}^{j+1} - U_{i-1,k}^{j+1}}{2\delta S}) + \theta_8 (\frac{U_{i+1,k}^{j} - U_{i-1,k}^{j}}{2\delta S})$$

$$\frac{\partial U}{\partial v} \approx \theta_9 (\frac{U_{i,k+1}^{j+1} - U_{i,k-1}^{j+1}}{2\delta v}) + \theta_{10} (\frac{U_{i,k+1}^{j} - U_{i,k-1}^{j}}{2\delta v})$$

$$U \approx \theta_{11} U_{i,k}^{j+1} + \theta_{12} U_{i,k}^{j}$$

where $U_{i,k}^j \approx U(S_i, v_k, \tau_j)$.

The approximations (50) can then be inserted into equation (49) to obtain the finite difference equation

$$\begin{aligned} \frac{U_{i,k}^{j+1}-U_{i,k}^{j}}{\delta\tau} \quad = \quad & \tfrac{1}{2} v_k S_i^2 [\theta_1 (\tfrac{U_{i-1,k}^{j+1} - 2U_{i,k}^{j+1} + U_{i+1,k}^{j+1}}{\delta S^2}) + \theta_2 (\tfrac{U_{i-1,k}^{j} - 2U_{i,k}^{j} + U_{i+1,k}^{j}}{\delta S^2})] \\ & + \rho\sigma v_k S_i [\theta_3 (\tfrac{Uj+1_{i+1,k+1} - U_{i-1,k+1}^{j+1} - U_{i+1,k-1}^{j+1} + U_{i-1,k-1}^{j+1}}{4\delta S \delta v}) + \theta_4 (\tfrac{U_{i+1,k+1}^{j} - U_{i-1,k+1}^{j} - U_{i+1,k-1}^{j} + U_{i-1,k-1}^{j}}{4\delta S \delta v})] \\ & + \tfrac{1}{2}\sigma^2 V_k [\theta_5 (\tfrac{U_{i,k-1}^{j+1} - 2U_{i,k}^{j+1} + U_{i,k+1}^{j+1}}{\delta v^2}) + \theta_6 (\tfrac{U_{i,k-1}^{j} - 2U_{i,k}^{j} + U_{i,k+1}^{j}}{\delta v^2})] \\ & + r S_i [\theta_7 (\tfrac{U_{i+1,k}^{j+1} - U_{i-1,k}^{j+1}}{2\delta S}) + \theta_8 (\tfrac{U_{i+1,k}^{j} - U_{i-1,k}^{j}}{2\delta S})] \\ & + K[\theta - v_k][\theta_9 (\tfrac{U_{i,k+1}^{j+1} - U_{i,k-1}^{j+1}}{2\delta v}) + \theta_{10} (\tfrac{U_{i,k+1}^{j} - U_{i,k-1}^{j}}{2\delta v})] \\ & - r[\theta_{11} U_{i,k}^{j+1} + \theta_{12} U_{i,k}^{j}] \end{aligned} \tag{51}$$

where $\theta_1 + \theta_2 = \theta_3 + \theta_4 = \theta_5 + \theta_6 = \theta_7 + \theta_8 = \theta_9 + \theta_{10} = \theta_{11} + \theta_{12} = 1$, and $0 < \theta_i < 1$ for all i.

Setting

$$\begin{aligned} \alpha_i^k &= \tfrac{1}{2} v_k S_i^2 \tfrac{\delta\tau}{\delta S^2} \\ \beta_i^k &= \tfrac{1}{4} \rho\sigma v_k S_i \tfrac{\delta\tau}{\delta S \delta v} \\ \gamma_i^k &= \tfrac{1}{2} \sigma^2 v_k \tfrac{\delta\tau}{\delta v^2} \\ \zeta_i^k &= \tfrac{1}{2} r S_i \tfrac{\delta\tau}{\delta S} \\ \eta_i^k &= \tfrac{1}{2} K[\theta - v_k] \tfrac{\delta\tau}{\delta v} \\ \mu_i^k &= -r\delta\tau \end{aligned} \tag{52}$$

and multiplying both sides by $\delta\tau$ we obtain the equation

$$\begin{aligned} U_{i,k}^{j+1} - U_{i,k}^{j} \quad = \quad & \alpha_i^k [\theta_1 (U_{i-1,k}^{j+1} - 2U_{i,k}^{j+1} + U_{i+1,k}^{j+1}) + \theta_2 (U_{i-1,k}^{j} - 2U_{i,k}^{j} + U_{i+1,k}^{j})] \\ & + \beta_i^k [\theta_3 (Uj + 1_{i+1,k+1} - U_{i-1,k+1}^{j+1} - U_{i+1,k-1}^{j+1} + U_{i-1,k-1}^{j+1}) \\ & + \theta_4 (U_{i+1,k+1}^{j} - U_{i-1,k+1}^{j} - U_{i+1,k-1}^{j} + U_{i-1,k-1}^{j})] \\ & + \gamma_i^k [\theta_5 (U_{i,k-1}^{j+1} - 2U_{i,k}^{j+1} + U_{i,k+1}^{j+1}) + \theta_6 (U_{i,k-1}^{j} - 2U_{i,k}^{j} + U_{i,k+1}^{j})] \\ & + \zeta_i^k [\theta_7 (U_{i+1,k}^{j+1} - U_{i-1,k}^{j+1}) + \theta_8 (U_{i+1,k}^{j} - U_{i-1,k}^{j})] \\ & + \eta_i^k [\theta_9 (U_{i,k+1}^{j+1} - U_{i,k-1}^{j+1}) + \theta_{10} (U_{i,k+1}^{j} - U_{i,k-1}^{j})] + \mu_i^k [\theta_{11} U_{i,k}^{j+1} + \theta_{12} U_{i,k}^{j}]. \end{aligned} \tag{53}$$

We now define the following

$$
\begin{aligned}
a_i^k &= -\beta_i^k\theta_3\\
b_i^k &= -\alpha_i^k\theta_1+\zeta_i^k\theta_7\\
c_i^k &= \beta_i^k\theta_3\\
d_i^k &= -\gamma_i^k\theta_5+\eta_i^k\theta_9\\
e_i^k &= 1+2\alpha_i^k\theta_1+2\gamma_i^k\theta_5-\mu_i^k\theta_{11}\\
f_i^k &= -\gamma_i^k\theta_5-\eta_i^k\theta_9\\
g_i^k &= \beta_i^k\theta_3\\
h_i^k &= -\alpha_i^k\theta_1-\zeta_i^k\theta_7\\
l_i^k &= -\beta_i^k\theta_3\\
a_i^{k\prime} &= \beta_i^k\theta_4\\
b_i^{k\prime} &= \alpha_i^k\theta_2-\zeta_i^k\theta_8\\
c_i^{k\prime} &= -\beta_i^k\theta_4\\
d_i^{k\prime} &= \gamma_i^k\theta_6-\eta_i^k\theta_{10}\\
e_i^{k\prime} &= 1-2\alpha_i^k\theta_2-2\gamma_i^k\theta_6+\mu_i^k\theta_{12}\\
f_i^{k\prime} &= \gamma_i^k\theta_6+\eta_i^k\theta_{10}\\
g_i^{k\prime} &= -\beta_i^k\theta_4\\
h_i^{k\prime} &= \alpha_i^k\theta_2+\zeta_i^k\theta_8\\
l_i^{k\prime} &= \beta_i^k\theta_4
\end{aligned}
\tag{54}
$$

and using (54) with (53) we obtain

$$
\begin{aligned}
&a_i^kU_{i-1,k-1}^{j+1}+b_i^kU_{i-1,k}^{j+1}+c_i^kU_{i-1,k+1}^{j+1}+d_i^kU_{i,k-1}^{j+1}+e_j^kU_{i,k}^{j+1}+f_i^kU_{i,k+1}^{j+1}+g_i^kU_{i+1,k-1}^{j+1}+h_i^kU_{i+1,k}^{j+1}+l_i^kU_{i+1,k+1}^{j+1}\\
&=a_i^{k\prime}U_{i-1,k-1}^{j}+b_i^{k\prime}U_{i-1,k}^{j}+c_i^{k\prime}U_{i-1,k+1}^{j}+d_i^{k\prime}U_{i,k-1}^{j}+e_j^{k\prime}U_{i,k}^{j}+f_i^{k\prime}U_{i,k+1}^{j}+g_i^{k\prime}U_{i+1,k-1}^{j}+h_i^{k\prime}U_{i+1,k}^{j}+l_i^{k\prime}U_{i+1,k+1}^{j}.
\end{aligned}
\tag{55}
$$

As before, we may use Dirichlet conditions for the S plane, i.e. the values at the boundaries are known. However, the conditions for the v plane are Neumann, i.e. $\frac{\partial U}{\partial v}(S,v+,\tau)=\frac{\partial U}{\partial v}(S,v-,\tau)=0$ where $v+$ is the maximum value of v and $v-$ is the minimum value of v.

We can approximate these boundary conditions using the equations

$$\frac{\partial U}{\partial v}(S,v-,\tau)\approx\frac{U_{i,1}^j-U_{i,-1}^j}{2\delta v}=0 \tag{56}$$

and

$$\frac{\partial U}{\partial v}(S,v+,\tau)\approx\frac{U_{i,M+1}^j-U_{i,M-1}^j}{2\delta v}=0$$

for all i and k, where M is the number of steps on the v axis.

Hence, the expressions

$$U_{i,-1}^j=U_{i,1}^j$$

and

$$U_{i,M+1}^j=U_{i,M-1}^j$$

can be inserted into (55) for all i and j to obtain the equations

$$\begin{aligned} &b_i^0 U_{i-1,0}^{j+1} + (c_i^0 + a_i^0) U_{i-1,1}^{j+1} + e_j^0 U_{i,0}^{j+1} + (f_i^0 + d_i^0) U_{i,1}^{j+1} + h_i^0 U_{i+1,0}^{j+1} + (l_i^0 + g_i^0) U_{i+1,1}^{j+1} \\ &= b_i^{0\prime} U_{i-1,0}^{j} + (c_i^{0\prime} + a_i^{0\prime}) U_{i-1,1}^{j} + e_j^{0\prime} U_{i,0}^{j} + (f_i^{0\prime} + d_i^{0\prime}) U_{i,1}^{j} + h_i^{0\prime} U_{i+1,0}^{0} + (l_i^{0\prime} + g_i^{0\prime}) U_{i+1,1}^{j} \end{aligned} \tag{57}$$

and

$$\begin{aligned} &(a_i^M + c_i^M) U_{i-1,M-1}^{j+1} + b_i^M U_{i-1,M}^{j+1} + (d_i^M + f_i^M) U_{i,M-1}^{j+1} + e_j^M U_{i,M}^{j+1} + (g_i^M + l_i^m) U_{i+1,M-1}^{j+1} + h_i^M U_{i+1,M}^{j+1} \\ &= (a_i^{M\prime} + c_i^{M\prime}) U_{i-1,M-1}^{j} + b_i^{M\prime} U_{i-1,M}^{j} + (d_i^{M\prime} + f_i^{M\prime}) U_{i,M-1}^{j} + e_j^{M\prime} U_{i,M}^{j} + (g_i^{M\prime} + l_i^{M\prime}) U_{i+1,M-1}^{j} + h_i^{M\prime} U_{i+1,M}^{j} \end{aligned}$$

for the scheme at the boundaries.

As both v and S have two boundary conditions, the scheme must be solved along both directions at once. We shall do this by solving along the entire (S,τ) grid simultaneously at each successive time step. It can be seen from these equations that at each time step, this scheme may be applied by solving the matrix equation

$$A\mathbf{U^{j+1}} = B\mathbf{U^{j}} + \mathbf{x} \tag{58}$$

where

$$A = \begin{pmatrix} D_0 & E_0 & 0 & \dots & 0 \\ C_1 & D_1 & E_1 & & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & & C_{M-1} & D_{M-1} & E_{M-1} \\ 0 & ... & 0 & C_M & D_M \end{pmatrix}$$

$$B = \begin{pmatrix} D_0' & E_0' & 0 & \dots & 0 \\ C_1' & D_1' & E_1' & & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & & C_{M-1}' & D_{M-1}' & E_{M-1}' \\ 0 & ... & 0 & C_M' & D_M' \end{pmatrix}$$

$$\underline{U^j} = \begin{pmatrix} U_{1,0}^j \\ U_{2,0}^j \\ \vdots \\ U_{N-1,0}^j \\ U_{1,1}^j \\ U_{2,1}^j \\ \vdots \\ U_{N-1,1}^j \\ \vdots \\ U_{N-1,M}^j \end{pmatrix}$$

$$
\underline{x} = \begin{pmatrix}
b_1^{0\prime}U_{0,0}^j + (c_1^{0\prime} + a_1^{0\prime})U_{0,1}^j - \big(b_1^0 U_{0,0}^{j+1} + (c_1^0 + a_1^0)U_{0,1}^{j+1}\big) \\
0 \\
\vdots \\
0 \\
h_{N-1}^{0\prime}U_{N,0}^j + (l_{N-1}^{0\prime} + g_{N-1}^{0\prime})U_{N,1}^j - \big(h_{N-1}^0 U_{N,0}^{j+1} + (l_{N-1}^0 + g_{N-1}^0)U_{N,1}^{j+1}\big) \\
a_1^{1\prime}U_{0,0}^j + b_1^{1\prime}U_{0,1}^j + c_1^{1\prime}U_{0,2}^j - \big(a_1^1 U_{0,0}^{j+1} + b_1^1 U_{0,1}^{j+1} + c_1^1 U_{0,2}^{j+1}\big) \\
0 \\
\vdots \\
0g_{N-1}^{1\prime}U_{N,0}^j + h_{N-1}^{1\prime}U_{N,1}^j + l_{N-1}^{1\prime}U_{N,2}^j - \big(g_{N-1}^1 U_{N,0}^{j+1} + h_{N-1}^1 U_{N,1}^{j+1} + l_{N-1}^1 U_{N,2}^{j+1}\big) \\
\vdots \\
\vdots \\
a_1^{M-1\prime}U_{0,M-2}^j + b_1^{M-1\prime}U_{0,M-1}^j + c_1^{M-1\prime}U_{0,M}^j - \big(a_1^{M-1}U_{0,M-2}^{j+1} + b_1^{M-1}U_{0,M-1}^{j+1} + c_1^{M-1}U_{0,M}^{j+1}\big) \\
0 \\
\vdots \\
0 \\
g_{N-1}^{M-1\prime}U_{N,M-2}^j + h_{N-1}^{M-1\prime}U_{N,M-1}^j + l_{N-1}^{M-1\prime}U_{N,M}^j - \big(g_{N-1}^{M-1}U_{N,M-2}^{j+1} + h_{N-1}^{M-1}U_{N,M-1}^{j+1} + l_{N-1}^{M-1}U_{N,M}^{j+1}\big) \\
(a_1^{M\prime} + c_1^{M\prime})U_{0,M-1}^j b_1^{M\prime}U_{0,M}^j - \big((a_1^M + c_1^M)U_{0,M-1}^{j+1} + b_1^M U_{0,M}^{j+1} + \big) \\
0 \\
\vdots \\
0 \\
(g_{N-1}^{M\prime} + l_{N-1}^{M\prime})U_{N,M-1}^j + h_{N-1}^{M\prime}U_{N,M}^j - \big((g_{N-1}^M + l_{N-1}^M)U_{N,M-1}^{j+1} + h_{N-1}^M U_{N,M}^{j+1}\big)
\end{pmatrix}
$$

and

$$
C_k = \begin{pmatrix}
d_1^k & g_1^k & 0 & \dots & 0 \\
a_2^k & d_2^k & g_2^k & & \vdots \\
0 & \ddots & \ddots & \ddots & 0 \\
\vdots & & a_{N-2}^k & d_{N-2}^k & g_{N-2}^k \\
0 & \dots & 0 & a_{N-1}^k & d_{N-1}^k
\end{pmatrix}
$$

$$
D_k = \begin{pmatrix}
e_1^k & h_1^k & 0 & \dots & 0 \\
b_2^k & e_2^k & h_2^k & & \vdots \\
0 & \ddots & \ddots & \ddots & 0 \\
\vdots & & b_{N-2}^k & e_{N-2}^k & h_{N-2}^k \\
0 & \dots & 0 & b_{N-1}^k & e_{N-1}^k
\end{pmatrix}
$$

$$E_k = \begin{pmatrix} f_1^k & l_1^k & 0 & \dots & 0 \\ c_2^k & f_2^k & l_2^k & & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & & c_{N-2}^k & f_{N-2}^k & l_{N-2}^k \\ 0 & \dots & 0 & c_{N-1}^k & f_{N-1}^k \end{pmatrix}$$

for $k \in \{1, 2, ...M-1\}$,

$$D_0 = \begin{pmatrix} e_1^0 & h_1^0 & 0 & \dots & 0 \\ b_2^0 & e_2^0 & h_2^0 & & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & & b_{N-2}^0 & e_{N-2}^0 & h_{N-2}^0 \\ 0 & \dots & 0 & b_{N-1}^0 & e_{N-1}^0 \end{pmatrix}$$

$$E_0 = \begin{pmatrix} f_1^0 + d_1^0 & l_1^0 + g_1^0 & 0 & \dots & 0 \\ c_2^0 + a_2^0 & f_2^0 + d_2^0 & l_2^0 + g_2^0 & & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & & c_{N-2}^0 + a_{N-2}^0 & f_{N-2}^0 + d_{N-2}^0 & l_{N-2}^0 + g_{N-2}^0 \\ 0 & \dots & 0 & c_{N-1}^0 + a_{N-1}^0 & f_{N-1}^0 + d_{N-1}^0, \end{pmatrix}$$

$$C_M = \begin{pmatrix} d_1^M + f_1^M & g_1^M + l_1^M & 0 & \dots & 0 \\ a_2^M + c_2^M & d_2^M + f_2^M & g_2^M + l_2^M & & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & & a_{N-2}^M + c_{N-2}^M & d_{N-2}^M + f_{N-2}^M & g_{N-2}^M + l_{N-2}^M \\ 0 & \dots & 0 & a_{N-1}^M + c_{N-1}^M & d_{N-1}^M + f_{N-1}^M \end{pmatrix}$$

$$D_M = \begin{pmatrix} e_1^M & h_1^M & 0 & \dots & 0 \\ b_2^M & e_2^M & h_2^M & & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & & b_{N-2}^M & e_{N-2}^M & h_{N-2}^M \\ 0 & \dots & 0 & b_{N-1}^M & e_{N-1}^M; \end{pmatrix}$$

and

$$C'_k = \begin{pmatrix} d_1^{k\prime} & g_1^{k\prime} & 0 & \dots & 0 \\ a_2^{k\prime} & d_2^{k\prime} & g_2^{k\prime} & & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & & a_{N-2}^{k\prime} & d_{N-2}^{k\prime} & g_{N-2}^{k\prime} \\ 0 & \dots & 0 & a_{N-1}^{k\prime} & d_{N-1}^{k\prime} \end{pmatrix}$$

$$D'_k = \begin{pmatrix} e_1^{k\prime} & h_1^{k\prime} & 0 & \dots & 0 \\ b_2^{k\prime} & e_2^{k\prime} & h_2^{k\prime} & & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & & b_{N-2}^{k\prime} & e_{N-2}^{k\prime} & h_{N-2}^{k\prime} \\ 0 & \dots & 0 & b_{N-1}^{k\prime} & e_{N-1}^{k\prime} \end{pmatrix}$$

$$E'_k = \begin{pmatrix} f_1^{k\prime} & l_1^{k\prime} & 0 & \dots & 0 \\ c_2^{k\prime} & f_2^{k\prime} & l_2^{k\prime} & & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & & c_{N-2}^{k\prime} & f_{N-2}^{k\prime} & l_{N-2}^{k\prime} \\ 0 & \dots & 0 & c_{N-1}^{k\prime} & f_{N-1}^{k\prime} \end{pmatrix}$$

for $k \in \{1, 2, ...M-1\}$,

$$D'_0 = \begin{pmatrix} e_1^{0\prime} & h_1^{0\prime} & 0 & \dots & 0 \\ b_2^{0\prime} & e_2^{0\prime} & h_2^{0\prime} & & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & & b_{N-2}^{0\prime} & e_{N-2}^{0\prime} & h_{N-2}^{0\prime} \\ 0 & \dots & 0 & b_{N-1}^{0\prime} & e_{N-1}^{0\prime} \end{pmatrix}$$

$$E'_0 = \begin{pmatrix} f_1^{0\prime} + d_1^{0\prime} & l_1^{0\prime} + g_1^{0\prime} & 0 & \dots & 0 \\ c_2^{0\prime} + a_2^{0\prime} & f_2^{0\prime} + d_2^{0\prime} & l_2^{0\prime} + g_2^{0\prime} & & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & & c_{N-2}^{0\prime} + a_{N-2}^{0\prime} & f_{N-2}^{0\prime} + d_{N-2}^{0\prime} & l_{N-2}^{0\prime} + g_{N-2}^{0\prime} \\ 0 & \dots & 0 & c_{N-1}^{0\prime} + a_{N-1}^{0\prime} & f_{N-1}^{0\prime} + d_{N-1}^{0\prime}, \end{pmatrix}$$

$$C'_M = \begin{pmatrix} d_1^{M\prime}+f_1^{M\prime} & g_1^{M\prime}+l_1^{M\prime} & 0 & \dots & 0 \\ a_2^{M\prime}+c_2^{M\prime} & d_2^{M\prime}+f_2^{M\prime} & g_2^{M\prime}+l_2^{M\prime} & & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & & a_{N-2}^{M\prime}+c_{N-2}^{M\prime} & d_{N-2}^{M\prime}+f_{N-2}^{M\prime} & g_{N-2}^{M\prime}+l_{N-2}^{M\prime} \\ 0 & \dots & 0 & a_{N-1}^{M\prime}+c_{N-1}^{M\prime} & d_{N-1}^{M\prime}+f_{N-1}^{M\prime} \end{pmatrix}$$

$$D'_M = \begin{pmatrix} e_1^{M\prime} & h_1^{M\prime} & 0 & \dots & 0 \\ b_2^{M\prime} & e_2^{M\prime} & h_2^{M\prime} & & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & & b_{N-2}^{M\prime} & e_{N-2}^{M\prime} & h_{N-2}^{M\prime} \\ 0 & \dots & 0 & b_{N-1}^{M\prime} & e_{N-1}^{M\prime}, \end{pmatrix}$$

N is the number of steps on the S axis, M the number of steps on the v axis, and a_i^k, b_i^k, etc. are defined as above.

4.3 Invertibility of the Scheme

If the matrix A is singular, then the finite difference equation cannot be solved. Using the same theorem as before, we can guarantee the invertibility of A by ensuring that $|e_i^k| > |a_i^k| + |b_i^k| + |c_i^k| + |d_i^k| + |f_i^k| + |g_i^k| + |h_i^k| + l_i^k$.

i.e.

$$\begin{aligned} |1+2\alpha_i^k\theta_1+2\gamma_i^k\theta_5-\mu_i^k\theta_{11}| \quad > \quad & |-\beta_i^k\theta_3|+|-\alpha_i^k\theta_1+\zeta_i\theta_7|+|\beta_i^k\theta_3|+|-\gamma_i^k\theta_5+\eta_i^k\theta_9| \\ & +|-\gamma_i^k\theta_5-\eta_i^k\theta_9|+|\beta_i^k\theta_3|+|-\alpha_i^k\theta_1-\zeta_i^k\theta_7|+|\beta_i^k\theta_3|. \end{aligned}$$

By the definitions given in (52), it can be seen that α_i, β_i, γ_i^k, $\zeta_i^k > 0$ and $\mu_i^k < 0$. It is also assumed that $\eta_i^k > 0$ as it is not possible to have a negative volatility. Hence, this equation is true if and only if

$$1+2\alpha_i^k\theta_1+2\gamma_i^k\theta_5-\mu_i^k\theta_{11}| > 4\beta_i^k\theta_3+|-\alpha_i^k\theta_1+\zeta_i^k theta_7+\alpha_i^k\theta_1+\zeta_i^k\theta_7+|-\gamma_i^k\theta_5+\eta_i^k\theta_9|+\gamma_i^k\theta_5+\eta_i^k\theta_9.$$

Consider the following four cases.

1. Suppose $\alpha_i^k\theta_1 < \zeta_i^k\theta_7$, $\gamma_i^k\theta_5 < \eta_i^k\theta_7$.

   Then we require that

$$1+2\alpha_i^k\theta_1+2\gamma_i^k\theta_5-\mu_i^k\theta_{11}| > 4\beta_i^k\theta_3+2\zeta_i^k\theta_7+2\eta_i^k\theta_9. \tag{59}$$

2. Suppose $\alpha_i^k\theta_1 < \zeta_i^k\theta_7$, $\gamma_i^k\theta_5 >= \eta_i^k\theta_7$.

   Then we require that
$$1 + 2\alpha_i^k\theta_1 - \mu_i^k\theta_{11}| > 4\beta_i^k\theta_3 + 2\zeta_i^k theta_7. \tag{60}$$

3. Suppose $\alpha_i^k\theta_1 >= \zeta_i^k\theta_7$, $\gamma_i^k\theta_5 < \eta_i^k\theta_7$.

   Then we require that
$$1 + 2\gamma_i^k\theta_5 - \mu_i^k\theta_{11}| > 4\beta_i^k\theta_3 + 2\eta_i^k\theta_9. \tag{61}$$

4. Suppose $\alpha_i^k\theta_1 >= \zeta_i^k\theta_7$, $\gamma_i^k\theta_5 >= \eta_i^k\theta_7$.

   Then we require that
$$1 - \mu_i^k\theta_{11}| > 4\beta_i^k\theta_3. \tag{62}$$

4.4 Accuracy

Similarly to before, the truncation error Φ_i^j can be measured using the equation $\Phi_i^j = L_{i,k}^j(U - U_{i,k}^j)L_{i,k}^j(U)$ where $L_{i,k}^j(U)$ represents the application of the numerical scheme to U.

In this case, this becomes

$$
\begin{aligned}
\Phi_{i,k}^{j} = & -\tfrac{1}{\delta\tau}\big(U(S_i,v_k,\tau_j+\delta\tau)-U(S_i,\tau_j)\big)\\
&+\tfrac{V_kS_i^2}{2\delta S^2}\Bigg[\theta_1\big(U(S_i-\delta S,v_k,\tau_j+\delta\tau)-2U(S_i,v_k,\tau_j+\delta\tau)+U(S_i+\delta S,v_k,\tau_j+\delta\tau)\big)\\
&\quad+\theta_2\big(U(S_i-\delta S,v_k,\tau_j)-2U(S_i,v_k,\tau_j)+U(S_i+\delta S,v_k,\tau_j)\big)\Bigg]\\
&+\tfrac{\rho v_kS_i}{4\delta S\delta v}\Bigg[\theta_3\big(U(S_i+\delta S,v_k+\delta_v,\tau_j+\delta\tau)-U(S_i-\delta S,v_k+\delta v,\tau_j+\delta\tau)\\
&\quad-U(S_i+\delta S,v_k-\delta v,\tau_j+\delta\tau)+U(S_i-\delta S,v_k-\delta v,t_j+\delta t)\big)\\
&\quad+\theta_4\big(U(S_i+\delta S,v_k+\delta_v,\tau_j)-U(S_i-\delta S,v_k+\delta v,\tau_j)\\
&\quad-U(S_i+\delta S,v_k-\delta v,\tau_j)+U(S_i-\delta S,v_k-\delta v,t_j)\big)\Bigg]\\
&+\tfrac{\sigma^2v_k}{2\delta v^2}\Bigg[\theta_5\big(U(S_i,v_k-\delta v,\tau_j+\delta\tau)-2U(S_i,v_k,\tau_j+\delta\tau)+U(S_i,v_k+\delta v,\tau_j+\delta t)\big)\\
&\quad+\theta_6\big(U(S_i,v_k-\delta v,\tau_j)-2U(S_i,v_k,\tau_j)+U(S_i,v_k+\delta v,\tau_j)\big)\Bigg]\\
&+\tfrac{rS_i}{2\delta S}\Bigg[\theta_7\big(U(S_i+\delta S,v_k,\tau_j+\delta\tau)-U(S_i-\delta S,v_k,\tau_j+\delta\tau)\big)\\
&\quad+\theta_8\big(U(S_i+\delta S,v_k,\tau_j)-U(S_i-\delta S,v_k,\tau_j)\big)\Bigg]\\
&+\tfrac{K[\theta-v_k]}{2\delta v}\Bigg[\theta_9\big(U(S_i,v_k+\delta v,\tau_j+\delta\tau)-U(S_i,v_k-\delta v,\tau_j+\delta t)\big)\\
&\quad+\theta_{10}\big(U(S_i,v_k+\delta v,\tau_j)-U(S_i,v_k-\delta v,\tau_j)\big)\Bigg]\\
&-r\Bigg[\theta_{11}U(S_i,v_k,\tau_j+\delta\tau)+\theta_{12}U(S_i,v_k,\tau_j)\Bigg]
\end{aligned}
$$

and using Taylor's theorem, we can expand around $U(S_i,\tau_j)$ to obtain

$$
\begin{aligned}
\Phi^j_{i,k} \quad = \quad & -\tfrac{1}{\delta\tau}\big(U+\delta\tau U_\tau+\tfrac{\delta\tau^2}{2}U_{\tau\tau}+\ldots-U\big)\\
&+\tfrac{V_kS_i^2}{2\delta S^2}\Big[\theta_1\big(U(S,v,\tau+\delta\tau)-\delta SU_s(S,v,\tau+\delta\tau)+\tfrac{\delta S^2}{2}U_S(S,v,\tau+\delta\tau)-\tfrac{\delta S^3}{6}u_{SSS}(S,v,\tau+\delta\tau)\\
&+\tfrac{\delta S^4}{24}U_{SSSS}(S,v,\tau+\delta\tau)+\ldots-2U(S,v,\tau+\delta\tau)+U(S,v,\tau+\delta\tau)+\delta SU_S(S,v,\tau+\delta\tau)\\
&+\tfrac{\delta S^2}{2}U_{SS}(S,v,\tau+\delta\tau)+\tfrac{\delta S^3}{6}U_{SSS}(S,v,\tau+\delta\tau)+\tfrac{\delta S^4}{24}U_{SSSS}(S,v,\tau+\delta\tau)+\ldots\big)\\
&+\theta_2\big(U-\delta SU_S+\tfrac{\delta S^2}{2}U_{SS}-\tfrac{\delta S^3}{6}U_{SSS}+\tfrac{\delta S^4}{24}+\ldots-2U\\
&+U+\delta SU_S+\tfrac{\delta S^2}{2}U_{SS}+\tfrac{\delta S^3}{6}U_{SSS}+\tfrac{\delta S^4}{24}U_{SSSS}+\ldots\big)\Big]\\
&+\tfrac{\rho v_kS_i}{4\delta S\delta v}\Big[\theta_3\big(4\delta S\delta vU_{Sv}(S,v,\tau+\delta\tau)+\tfrac{2\delta s\delta v^3}{3}U_{Svvv}(S,v,\tau+\delta\tau)+\tfrac{2\delta sav}{3}U_{SSSv}(S,v,\tau+\delta\tau)+\ldots\big)\\
&+\theta_4\big(4\delta S\delta vU_{Sv}+\tfrac{2\delta S\delta v^3}{3}U_{Svvv}+\tfrac{2\delta S^3\delta v}{3}U_{SSSv}+\ldots\big)\Big]\\
&+\tfrac{\sigma^2v_k}{2\delta v^2}\Big[\theta_5\big(U(S,v,\tau+\delta\tau)-\delta vU_v(S,v,\tau+\delta\tau)+\tfrac{\delta v^2}{2}U_{vv}(S,v,\tau+\delta\tau)-\tfrac{\delta v^3}{6}U_{vvv}(S,v,\tau+\delta\tau)\\
&+\tfrac{\delta v^4}{24}U_{vvvv}(S,v,\tau+\delta\tau)+\ldots-2U(S,v,\tau+\delta\tau)+U(S,v,\tau+\delta t)+\delta vU_v(S,v,\tau+\delta\tau)\\
&+\tfrac{\delta\ v^2}{2}U_{vv}(S,v,\tau+\delta\tau)+\tfrac{\delta v^3}{6}U_{vvv}(S,v,\tau+\delta\tau)+\tfrac{\delta v^4}{24}U_{vvvv}(S,v,\tau+\delta\tau)+\ldots\big)\\
&+\theta_6\big(U-\delta vU_v+\tfrac{\delta v^2}{2}U_{vv}-\tfrac{\delta v^3}{6}U_{vvv}+\tfrac{\delta v^4}{24}U_{vvvv}+\ldots\\
&-2U+U+\delta vU_{vv}+\tfrac{\delta v^2}{2}U_{vv}+\tfrac{\delta v^3}{6}U_{vvv}+\tfrac{\delta v^4}{24}U_{vvvv}+\ldots\big)\Big]\\
&+\tfrac{rS_i}{2\delta S}\Big[\theta_7\big(U(S,v,\tau+\delta\tau)+\delta SU_S(S,v,\tau+\delta\tau)+\tfrac{\delta S^2}{2}U_{SS}(S,v,\tau+\delta\tau)+\tfrac{\delta S^3}{6}U_{SSS}(S,v,\tau+\delta\tau)+\ldots\\
&-U(S,v,\tau+\delta\tau)+\delta SU_SS,v,\tau+\delta\tau)-\tfrac{\delta S^2}{2}U_{SS}(S,v,\tau+\delta\tau)+\tfrac{\delta S^3}{6}U_{SSS}(S,v,\tau+\delta\tau)+\ldots\big)\\
&+\theta_8\big(U+\delta SU_S+\tfrac{\delta S^2}{2}U_{SS}+\tfrac{\delta S^3}{6}U_{SSS}+\ldots-U+\delta SU_S-\tfrac{\delta S^2}{2}U_{SS}+\tfrac{\delta S^3}{6}U_{SSS}+\ldots\big)\Big]\\
&+\tfrac{K[\theta-v_k]}{2\delta v}\Big[\theta_9\big(U(S,v,\tau+\delta\tau)+\delta vU_v(S,v,\tau+\delta\tau)+\tfrac{\delta v^2}{2}U_{vv}(S,v,\tau+\delta\tau)+\tfrac{\delta v^3}{6}U_{vvv}(s,v,\tau+\delta\tau)+\ldots\\
&-U(S,v,\tau+\delta t)+\delta vU_v(S,v,\tau+\delta\tau)-\tfrac{\delta v^2}{2}U_{vv}(S,v,\tau+\delta\tau)+\tfrac{\delta v^3}{6}U_{vvv}(S,v,\tau+\delta\tau)+\big)\\
&+\theta_{10}\big(U+\delta vU_v+\tfrac{\delta v^2}{2}U_{vv}+\tfrac{\delta v^3}{6}U_{vvv}+\ldots-U+\delta vU_v-\tfrac{\delta v^2}{2}U_{vv}+\tfrac{\delta v^3}{6}U_{vvv}+\ldots\big)\Big]\\
&-r\Big[\theta_{11}\big(U+\delta\tau U_\tau+\ldots\big)+\theta_{12}U\Big]
\end{aligned}
$$

Remembering that $\theta_1+\theta_2=\theta_3+\theta_4=\theta_5+\theta_6=\theta_7+\theta_8=\theta_9+\theta_{10}=\theta_{11}+\theta_{12}=1$ and using equation (49), this can be simplified to

$$
\begin{aligned}
\Phi^j_i \quad = \quad & -\tfrac{\delta\tau}{2}U_{tt}+\tfrac{vS^2}{2}(\tfrac{\delta S^2}{12}U_{SSSS}+\theta_1\delta\tau U_{SSt})+\rho\sigma vS(\tfrac{\delta v^2}{6}U_{Svvv}+\tfrac{\delta v^2}{6}U_{SSSv}+\theta_3\delta\tau U_{Sv\tau})\\
&+frac\sigma^2v2(\tfrac{\delta v^2}{12}U_{vvvv}+\theta_5\delta\tau U_{vv\tau})+rS(\tfrac{\delta S^2}{6}U_{SSSS}+\theta_7\delta\tau U_{S\tau}+K[\theta-v](\tfrac{\delta v^2}{2}U_{vvv}\\
&+\theta_9\delta\tau U_{v\tau})-\theta_{11}r\delta\tau U_\tau)+\text{higher order terms.}
\end{aligned}
\tag{63}
$$

Hence, this numerical approximation is accurate to first order time, second order asset price and volatility.

4.5 Stability

Similarly to before, we shall use Fourier stability for this scheme. Let $U^j_{i,k} = \Lambda^j e^{-zni\delta S} e^{-zmk\delta v}$ where $z = \sqrt{-1}$ and n, m are arbitrary constants, and substitute this expression into the numerical scheme. In order for the scheme to be stable, $|\Lambda|$ must be less than 1.

Therefore in this case, it is necessary to choose values of θ_i and δS, $\delta\tau$ such that

$$\begin{array}{c} |a^{k\prime}_i e^{-z(n\delta S+m\delta v)} + b^{k\prime}_i e^{-zn\delta S} + c^{k\prime}_i e^{z(m\delta v-n\delta S)} + d^{k\prime}_i e^{-zm\delta v} + e^{k\prime}_i \\ +f^{k\prime}_i e^{zm\delta v} + g^{k\prime}_i e^{z(n\delta S-\delta vm)} + h^{k\prime}_i e^{zn\delta S} + l^{k\prime}_i e^{z(n\delta S+m\delta v)} \\ < \\ |a^{k}_i e^{-z(n\delta S+m\delta v)} + b^{k}_i e^{-zn\delta S} + c^{k}_i e^{z(m\delta v-n\delta S)} + d^{k}_i e^{zm\delta v} + e^{k}_i \\ +f^{k}_i e^{zm\delta v} + g^{k}_i e^{z(n\delta S-\delta vm)} + h^{k}_i e^{zn\delta S} + l^{k}_i e^{z(n\delta S+m\delta v)}| \end{array} \tag{64}$$

for all i.

4.6 Examples of Schemes

Scheme	θ_1	θ_2	θ_3	θ_4	θ_5	θ_6	θ_7	θ_8	θ_9	θ_{10}	θ_{11}	θ_{12}
Crank-Nicolson	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$
Kenneth-Vetzal	1	0	1	0	1	0	1	0	1	0	0	1
Fully Implicit	1	0	1	0	1	0	1	0	1	0	1	0

Scheme	S.D.D. Interval	Stability Interval
Crank-Nicolson	$v_k S_i > r, \sigma^2 v_k > K[\theta - v_k] - \lambda^j_{i,k}, \frac{2-r}{v_k S_i} > \rho\sigma$	Unconditional
Kenneth-Vetzal	$v_k S_i > r, \sigma^2 v_k > K[\theta - v_k] - \lambda^j_{i,k}, \frac{1-r}{v_k S_i} > \rho\sigma$	$(\sigma^2 v_k + K[\theta - v - k])(r + v_k S_i) > \frac{\rho\sigma v_k}{\delta\tau}$
Fully Implicit	$v_k S_i > r, \sigma^2 v_k > K[\theta - v_k] - \lambda^j_{i,k}, \frac{1-r}{v_k S_i} > \rho\sigma$	$(\sigma^2 v_k + K[\theta - v - k])(r + v_k S_i) > \frac{\rho\sigma v_k}{\delta\tau}$

5 Further Work: Inclusion of Bond Price

Bond prices are very sensitive to interest rates, and are often used to measure them. We shall therefore try and use them in our model so as to incorporate stochastic interest rates into the pricing problem[4]. Possible methods of solving this are left for future work.

5.1 Derivation of Partial Differential Equation

We begin assuming that the prices of the underlying asset and the bond are governed by the equations

$$\begin{gathered} dS = \mu_S S dt + \sigma_S dt + \sigma_S(t)\sqrt{v(t)} S dW_1 \\ dP = \mu_P P dt + \sigma_P(t)\sqrt{v(t)} P dW_2 \end{gathered} \tag{65}$$

where both values depend upon the same variable $v(t)$, and W_1 and W_2 represent Wiener processes as before. The bond used must be chosen carefully to ensure that this is a valid assumption.

We use the same model as before for the volatility v; i.e.

$$dv = K[\theta - v]dt + 2\sigma\sqrt{v} dW_3. \tag{66}$$

Let $U = U(S, P, v, t)$ represent the value of the derivative at time t according to the price of the underlying asset. Then by Taylor's theorem

$$\begin{aligned} dU &= U(S + dS, P + dP, v + dv, t + dt) - U(S, v, t) \\ &= \tfrac{\partial U}{\partial t}(S, P, v, t)dt + \tfrac{\partial U}{\partial S}(S, P, v, t)dS + \tfrac{\partial U}{\partial P}(S, P, v, t)dP + \tfrac{\partial U}{\partial v}(S, v, t)dv \\ &\quad + \tfrac{1}{2}\tfrac{\partial^2 U}{\partial S^2}(S, v, t)(dS)^2 + \tfrac{1}{2}\tfrac{partial^2 U}{\partial s \partial P}(S, P, v, t)dP^2 + \tfrac{1}{2}\tfrac{\partial^2 v}{\partial v^2}(S, P, v, t)(dv)^2 + \tfrac{\partial^2 U}{\partial S \partial v}(S, v, t)dSdv \\ &\quad + \tfrac{\partial^2 U}{\partial P \partial v}dPdv + \tfrac{\partial^2 U}{\partial S \partial P}dSdP + O(dt^{\frac{3}{2}}). \end{aligned} \tag{67}$$

By equations (65) and (66), and discarding terms of order $dt^{\frac{3}{2}}$ as before, we calculate

$$\begin{gathered} dS^2 = \sigma_S^2 v S^2 dt \\ dP^2 = \sigma_P^2 v P^2 dt \\ dv^2 = \sigma^2 v dt \\ dSdv = \sigma_S \sigma v S \rho_{S,v} dt \\ dPdv = \sigma_P \sigma v P \rho_{P,v} dt \\ dSdP = \sigma_S \sigma_P v S P \rho_{S,P} dt \end{gathered} \tag{68}$$

where $\rho_{S,v}$ represents the correlation between the processes W_1 and W_3, $\rho_{P,v}$ represents the correlation between the processes W_2 and W_3, and $\rho_{S,P}$ represents the correlation between the processes W_1 and W_2.

Equations (65), (66), and (68) can then be substituted back into (67), which can then be rearranged to get

$$\begin{aligned} dU \;=\; & \sigma_S\sqrt{v}S\frac{\partial U}{\partial S}dW_1+\sigma_P\sqrt{v}P\frac{\partial U}{\partial P}dW_2+\sigma\sqrt{v}\frac{\partial U}{\partial v}dW_3+(\frac{\partial U}{\partial t}+\mu S\frac{\partial U}{\partial S}+\mu P\frac{\partial U}{\partial P}+k[\theta-v]\frac{\partial U}{\partial v} \\ & +\frac{1}{2}\sigma_S^2S^2v\frac{\partial^2 U}{\partial S^2}+\frac{1}{2}\sigma^2v\frac{\partial^2 U}{\partial V^2}+\frac{1}{2}\sigma_P^2vP^2\frac{\partial^2 U}{\partial P^2}+\sigma_S\sigma Sv\rho_{S,v}\frac{\partial^(2)U}{\partial S\partial v}+\sigma_P\sigma vP\rho_{P,v}\frac{\partial^2 U}{\partial P\partial v}+\sigma_S\sigma_P vSP\rho_{S,P}\frac{\partial^2 U}{\partial S\partial P})dt. \end{aligned} \tag{69}$$

We can now construct a portfolio by buying one derivative and selling Δ_1 lots of the underlying asset and Δ_2 lots of the bond. Then the value of the portfolio Π is given by

$$\Pi = U-\Delta_1 S-\Delta_2 P. \tag{70}$$

The rate of change of the value of the portfolio can therefore be given by

$$d\Pi = dU-\Delta_1 dS-\Delta_2 dP$$

and substituting (65) into this equation and setting $\Delta_1=\frac{\partial U}{\partial S}$ and $\Delta_2=\frac{\partial U}{\partial P}$gives

$$\begin{aligned} d\Pi \;=\; & \sigma\sqrt{v}\frac{\partial U}{\partial v}dW_3+(\frac{\partial U}{\partial t}+k[\theta-v]\frac{\partial U}{\partial v}+\frac{1}{2}\sigma_S^2S^2v\frac{\partial^2 U}{\partial S^2}+\frac{1}{2}\sigma^2v\frac{\partial^2 U}{\partial V^2}+\frac{1}{2}\sigma_P^2vP^2\frac{\partial^2 U}{\partial P^2} \\ & +\sigma_S\sigma Sv\rho_{S,v}\frac{\partial^(2)U}{\partial S\partial v}+\sigma_P\sigma vP\rho_{P,v}\frac{\partial^2 U}{\partial P\partial v}+\sigma_S\sigma_P vSP\rho_{S,P}\frac{\partial^2 U}{\partial S\partial P})dt. \end{aligned} \tag{71}$$

The random term in W_2 can be replaced as before by $-\lambda(t,S,v)\frac{\partial U}{\partial v}dt$ where lambda is the price of volatility risk. As the random term has now vanished, this portfolio is risk free, and by the law of one price must therefore grow at the risk free rate. Hence

$$\begin{aligned} d\Pi &= r\Pi dt \\ &= r(U-S\frac{\partial U}{\partial S}-P\frac{\partial U}{\partial P})dt. \end{aligned} \tag{72}$$

The two equations (70) and (72) can then be set equal, rearranged, and divided by dt to get

$$\begin{aligned} & \frac{1}{2}\sigma_S^2S^2v\frac{\partial^2 U}{\partial S^2}+\frac{1}{2}\sigma_P^2vP^2\frac{\partial^2 U}{\partial P^2}+\frac{1}{2}\sigma^2v\frac{\partial^2 U}{\partial v^2}+\sigma_S\sigma_P vSP\rho_{S,P}\frac{\partial^2 U}{\partial S\partial P}+\sigma_S\sigma Sv\rho_{S,v}\frac{\partial^(2)U}{\partial S\partial v} \\ & +\sigma_P\sigma vP\rho_{P,v}\frac{\partial^2 U}{\partial P\partial v}+rS\frac{\partial U}{\partial S}+rP\frac{\partial U}{\partial P}+k[\theta-v]\frac{\partial U}{\partial v}-rU+\frac{\partial U}{\partial t} \\ & =0. \end{aligned} \tag{73}$$

5.2 Theta Method

The obvious route to take when solving the bond model would be to use the same sort of θ method as before. However, as the new model is in three dimensions this would necessitate the solving of a series of finite difference equations across the entire (S,P,v) plane for each time step. A program written to perform this task would be extremely slow and therefore of little practical use to derivatives traders, who need values quickly when dealing in the financial market. This could perhaps be solved by using explicit methods only, although great care would have to be taken to ensure stability, or by using larger step sizes in all directions, which may lead to a loss in accuracy such that the model beomes less correct than the application of the previous two models.

5.3 Alternating Direction Implicit Method

Another possible way of numerically solving this model might be to employ the alternating direction implicit method (or *A.D.I.*)[7]. In two dimensions, this method involves taking an explicit finite difference in one spatial dimension and an implicit finite difference in the other, at a time level $j + \frac{1}{2}$. The difference methods are then alternated between each dimension for each half time level, and rearranged to eliminate the solution at $j + \frac{1}{2}$. It might be possible to extend this method to three dimensions, either by alternating in three different directions, or by alternating between two of the three dimensions and a third and hence solving over a two dimensional grid for each step as before.

We also note here that in order to employ the A.D.I. method, it is necessary to first eliminate the terms which contain derivatives with respect to variables in more than one direction, e.g. $\frac{\partial^2 U}{\partial S \partial P}$. To do this the equation must be transformed into new co-ordinates. The new frame of reference may also require a non-uniform grid to be applied when using the finite difference methods.

5.4 Finite Element Method

An alternative approach may be to solve the new model using finite element instead of finite difference methods. The idea behind these methods is to find a piecewise linear solution to a differential equation, which can be made to approximate the solution in an integrational sense by integrating the terms in the equation until only U or $\frac{\partial U}{\partial x}$ remain. The piecewise linear solution can then be substituted into the resultant equation and solved to obtain the answer. For Dirichlet conditions, the solutions at the nodes are equal to the analytical answer. In the case of the model derived in this section, three dimensional elements are required when solving the equation.

5.5 Finite Volume Method

An alternative approach may be to solve the new model using finite volume instead of finite difference methods. Finite volume methods involve breaking down the domain into control volumes, for example volumes centred on each $U_{i,k}^{j}$ whose boundaries stretch for half a step in all directions, and then discretising the integral form of the differential equation. This is suggested as a possible extension because most of the initial conditions for financial derivatives involve a discontinuous derivative; such features are better approximated by finite volume methods than finite difference methods.

5.6 Comparison with Monte Carlo Simulations

Monte Carlo methods[9] are widely used in the fields of banking and insurance, as well as numerous other fields containing non-deterministic models. The idea of a Monte Carlo method is to specify an initial condition along with a series of stochastic equation with a given distribution. The algorithm then generates random numbers according to the distribution and evolves the model accordingly; repeating this process thousands of times should give a reasonable distribution of the expected final solution.

Monte Carlo methods are computationally very expensive to use, although the idea is a simple one and easy to program, and the execution may be speeded up application of techniques such as variance reduction. We propose that future work could include setting up a Monte Carlo method to mimic empirical data and then comparing the results to the deterministic pricing models derived in this project.

6 Algorithms for solving the equations

In this section we consider how to write and structure the programs required for solvng the models using the finite difference schemes outlined earlier.

6.1 Matrix Inversion

In order to solve the equations at each time step, it is necessary to calculate the value of A^{-1} x right hand side vector (rhs).

For the Black-Scholes numerical approximations, A is tri-diagonal. It can therefore be solved using the Thomas Algorithm, which works as follows:

Let $A = LR$ where

$$L = \begin{pmatrix} 1 & & & \\ l_2 & 1 & & \\ & \ddots & \ddots & \\ & & l_{N-1} & 1 \end{pmatrix}$$

$$R = \begin{pmatrix} r_1 & c_1 & & \\ & \ddots & \ddots & \\ & & r_{m-2} & c_{N-21} \\ & & & r_{N-1} \end{pmatrix}$$

and

$$r_1 = b_1$$

$$r_{i+1} = (b_{i+1} - a_{i+1}c_i)/r_i$$
$$l_{i+1} = a_{i+1}/r_i.$$

Firstly, let $y = RA^{-1}$

Thus:

$$Ly = rhs$$

This can be solved for y, giving:

$y(1) = rhs(1)$
and for i = 2,3, ... m-1

$y(i) = rhs(i) - l_{i-1}y_{i-1}$

Finally, the equation $RA^{-1} = y$ can be solved to give:

$solution(m-1) = y(m-1)/r_{N-1}$
and for i = N-2, N-3, ... 1
$solution(i) = (y(i) - c_i solution(i+1))/r_i$.

However, for the Heston solver, the matrix is block-tridiagonal, and cannot be solved directly. Instead, we shall solve the equation by using the Gauss-Seidel method, which works as follows:

Firstly, split the matrix A into $L - R$ where:

$$L_{i,j} = \begin{cases} A_{i,j} & \text{if } j \leq i \\ 0 & \text{otherwise} \end{cases}$$

$$R_{i,j} = \begin{cases} -A_{i,j} & \text{if } j > i \\ 0 & \text{otherwise} \end{cases}$$

Suppose $Ax = rhs$, where rhs is known and x is not. Thus $Lx = Rx + b$, and an iteration can be set up such that given some initial guess x_0, $Lx_{i+1} = Rx_i + b$ until the iteration homes in on some value of x.

This algorithm unfortunately is not very efficient, and if the (S, v, τ) grid becomes too large, will take a long time to run. This is unacceptable when pricing derivatives, as pricing may have to be done very quickly. It is therefore necessary to sacrifice some of the accuracy, both in the step sizes and in the error tolerance level of the iteration, in order to produce a result within a reasonable time.

For both of these algorithms, only the values in the diagonals shall be stored rather than the entire matrix, as the program will therefore take up much less memory.

6.2 Black-Scholes Solver

The program written for the numerical solution of the Black-Scholes equation works as follows:

1. Define variables and ask user for input
2. If the user has chosen a Swaption, ask for spot rates and calculate value with which to multiply the final call solution
3. Resize matrices and vectors appropriately
4. Fill in τ and S vectors with step values
5. Fill in the boundary and initial values
6. Define the diagonal entries for A

7. Loop through all τ steps and:
   I. Define entries in right hand side vector
   II. Use the Thomas algorithm to calculate solution
   III. Insert solution back into matrix for U
8. Output final row of solution to file to get prices for U at time $t = 0$.

A copy is included in the Appendix.

6.3 Heston Model Solver

The program written for the numerical solution of the Heston equation works as follows:

1. Define variables and ask user for input
2. If the user has chosen a Swaption, ask for spot rates and calculate value with which to multiply the final call solution
3. Resize matrices and vectors appropriately
4. Fill in τ, v, and S vectors with step values
5. Fill in the initial values
6. Loop through all τ steps and:
   I. Enter in boundary values
   II. Define entries of A
   III. Define entries in right hand side vector
   IV. Use the Gauss-Seidel iteration to calculate solution
   V. Insert solution back into for U for that time step
   VI. Set previous solution to be new solution for use in the next time step
7. Output final row of solution to file to get prices for U at time $t = 0$.

A copy is included in the Appendix.

7 Results

The first section of results contains a comparison with some of Stella Christodoulou's[6] results for standard options, and includes the results from the numerical Heston model for comparison.

As an examnination of the numerical approximation, we shall next consider the change in results as the boundaries are moved closer together, and the effect this has on the correctness of the solution. We shall also vary the step sizes in order to determine what sizes are required for accurate enough values.

Next, we include relevant market data for the pricing of the European Swaptions, along with market prices for comparison with the results. This is followed by the prices as calculated by each of the two numerical programs.

Finally, the numerical models shall be used to price several exotic options, and the solution compared to the market values. In order to perform this task correctly the correct parameters must be used; and although it is possible to find market data for the values of the current volatility (σ for the Black-Scholes model, θ for the Heston model) and the risk-free interest rate (r) for a given underlying share price, values are not given for K, ρ, or σ in the Heston model. The value of σ is a measurement of the volatility of the volatilities, and an appropriate estimate can be calculated from historical data. In order to price an exotic derivative, we have therefore run the model for a standard vanilla option at the strike rate and found values for which the solution is most accurate. The values obtained in this way shall then be used for pricing the exotic derivative.

7.1 Comparison with Stella Christodoulous Results

7.2 Variation of Boundaries and Step Sizes

7.3 Market Information for Volatilities and Forward Rates, and Prices for European Swaptions

7.4 Pricing of Knock-in Caps and Comparison with Market Data

7.5 Discussion of Results

1. Comparison with previous results

   The parameters used to produce the results in this table are: $S^- = 50$, $S^+ = 150$, $v^- = 0.1$, $v^+ = 0.3$, $T = 0.25$, $\delta S = 0.5$, $\delta v = 0.005$, $\delta\tau = 0.01$, $E = 100$, $r = 0/08$. In the Black-Scholes model $\sigma = 0.2$; in the Heston model $sigma = 4$, $\theta = 0.2$, $K = 0.5$, $\rho = -0.95$.

   The results for the Black-Scholes numerical model are comparable with previous work [6], and are generally accurate enough for the practical purpose of pricing. The error for the Crank-Nicolson scheme is much smaller than the rest, as it is second order in time instead of first order in time like the other schemes. The explicit schemes however are not stable for these values. To correct this, the size of $\delta\tau$ must be decreased.

   The results for the Heston model however are less accurate. We note that in this case not only the numerial approximation itself but also the Gauss-Seidel iteration are sources of error, and speculate that it is the iteration which is the main source of error here. Reducing the step size may help with the error generated by the numerical approximation; however, decreasing the step sizes in the S and v plane will increase the size of the matrix to be inverted and so slow down the program, and decreasing the step size in the τ plane will also slow down the program as the inversion must be repeated more times, which will also increase the effect of the error in the iteration. The entries in the matrix A will also be much smaller in comparison to the error tolerance, which will therefore not be small enough to demand the required accuracy. A possible way of dealing with this would be to vary the error tolerance along with the step sizes, although the smaller the tolerance the longer each iteration will take to complete. We also note that the results given as the analytical solution also contain some error, as the closed form for vanilla options contains an integration whose solution has been estimated using numerical methods.

   We consider the graphs showing the call and put values for the two models. As the value of the correlation ρ is negative, the expected volatility as calculated by the Heston model will be greater for smaller values of S and less for larger values of S than in the Black-Scholes model. For the call graph, the volatility is therefore less than expected by the Black-Scholes model for the out-of-the-money prices and the Heston model will thus evaluate these prices to be less than the Black-Scholes prices, whereas the volatility is less for the in-the-money options, and hence the Heston model assumes that the option is more likely to finish in-the-money than the Black-Scholes results. Similarly, the in-the-money prices for the put option is modelled as being more likely to finish out-of-the-money and hence are priced less by the Heston model than in the Black-Scholes, whereas the out-of-the-money prices are modelled as being more likely to result in a positive payoff, and hence are evaluated at a greater price than that calculated by the Black-Scholes model.

2. Variation of boundaries

The parameters here are the same as those used to produce the results detailed in the previous section, except that the boundaries have been moved as detailed in the tables.

Considering these results, we note that varying the boundaries makes a difference to the accuracy of the results, particularly for those prices nearer to the boundaries. However, the difference is very slight. For the boundaries in the S plane, accurate enough results can be produced for the Black-Scholes equation when the boundaries are placed $\frac{1}{4}$ of the value of the strike price away from this value, and for the Heston model at $\frac{1}{2}$ of the value away from the srike price. We shall also set the boundaries for the v plane at $\frac{1}{4}$ value spead in each direction away from the current rate.

3. Variation of step sizes

   The parameters here are the same as those used to produce the first set of results, except that the step sizes have been changed as detailed in the tables.

   We note from the results given here that although halving the size of the S step does approximately half the error in the Black-Scholes results, the difference is not necessary for accurate enough results. When the S and v step sizes are halved for the Heston model, not only does the program take an unacceptably long time to run, but the errors seem to have increased rather than decreased. This may be due to the error produced by using the Gauss-Seidel iteration. The values of δS and δv shall therefore remain at the same values for the production of the rest of the results.

   Decreasing the value of $\delta\tau$ not only gives more accurate resuls for the Black-Scholes model, it also produces stability for the explicit schemes. The value of $\delta\tau$ shall therefore be decreased to 0.001 for this model. The explicit schemes shall not be used for the remainder of the results, as they are too unstable for use in a commercial environment. Decreasing the value of $\delta\tau$ for the Heston model does not appear to improve the accuracy, and makes the program far too slow. The value of $\delta\tau = 0.01$ shall therefore be maintained for this model.

4. Pricing of swaptions

   For this section, we shall use the parameters as determined in the previous two sections.

   The results given here by the Black-Scholes numerical solution are comparable to the analytical results; however while the prices for swaptions with early maturities are reasonably accurate for the Heston numerical solution, the error becomes very large for those swaptions with a long term maturity. We also notice that for an underlying swap with a higher maturity (e.g. 4y-7y), any error is multiplied by the inclusion of the $A = (\sum_{i=1}^{m} D(t_i))$ value.

It is clear from the comparison with the market data that the Black-Scholes results are closer to the accepted market price. This is because it is the Black-Scholes model which traders currently use as a basis for the price. The model suggested here is believed to be a more accurate assessment of the true price, and is therefore more useful when brokers are trying to neutralise the risk associated with the derivative which has already been bought or sold by a trader.

5. Pricing of knock-in caps

A knock-in cap constructed of E_1 digitals combined with a call option, both with a strike E_2. The payoff is therefore $max(S - E_2, 0) + E_1 \frac{max(S-E_2,0)}{S-E_2}$. The model is used here to price a knock-in cap with a maturity of 1 year of varying strikes for a specific stock. In this case, the underlying asset price is currently 110.

The volatility curve shown in this section of results shows the market values[8], and parameters should be used which mimick this graph. Trial and error shows that the requisite parameters for the Heston model are $\sigma = 4$, $K = 1$, $\theta = 0.4$, and $\rho = -0.2$. The original volatility is 0.43; this shall be the point at which we take our result, and the value of σ in the Black-Scholes solution. The current value of the risk-free rate is $r = 0.045$.

Similarly to before, we note that the Black-Scholes solution produces values closer to the traded prices than the Heston solution. The solutions given by the Heston models also vary considerably between the Crank-Nicolson scheme and the other two. Based upon past performance, we believe the Crank-Nicolson solution to be more accurate.

As the Heston solution is more useful to a broker than a trader, more time may be taken to produce the results. We therefore suggest that in the future a much smaller error tolerance is used for the Gauss-Seidel iteration, as this is the greatest source of error but also of inefficiency.

8 Summary and Conclusion

During this rsearch we have considered the numerical solution of two models, the Black-Scholes model and the Heston model, which are used for pricing financial derivatives. We have used the θ method to find finite difference equations which numerically solve these two models. In order to solve these equations, we have written them in the form of a single matrix equation. For the Black-Scholes numerical solution, this matrix equation may be solved directly using the Thomas algorithm; for the Heston numerical solution, the equation is solved using an iterative process, in this case the Gauss-Seidel method. The results produced in this way are very accurate for the Black-Scholes solution, but less accurate for the Heston solution, with the Gauss-Seidel iteration being the largest source of error.

As this model would be used within a commercial envronment, it is important to produce the results within a reasonable period of time. When the maturity is long term, or the error tolerance is increased, the time taken to produce results is greatly increased. The number of step sizes in the S and v planes can only be decreased so far, as the numerical boundaries must be far enough apart to mimic the boundary conditions, and increasing the step size decreases efficiency and stability.

We propose that this model is used for brokers assessing the true value of derivatives, and that the error tolerance for the iteration is decreased in this case as more time may be taken to produce results.

For future work, it is possible to include the Bond price in the partial differential equation used for pricing derivatives. However, using the same θ method in this case is not commercially viable. Instead, we propose the use of the alternating direction implicit method, or a differet type of numerical solution such as a finite element or finite volume method. We also suggest the use of Monte-Carlo simulations in order to compare the accuracy of the models in terms of price risk.

References

[1] John C. Hull, 2000, "Options, Futures, & Other Derivatives", fourth edition., Prentice-Halll International

[2] F. Black & M. Scholes, 1973, "The Valuations of Options and Corporate Liabilities", Journal of Political Economy, 81, 637-654

[3] M. Rubenstein, 1985, "Nonparametric Tests of Alternative Option Pricing Models Using All Reported Trades and Quotes on the 30 Most Active CBOE Option Classes from August 23, 1976 through August 31, 1978", Journal of Financial and Quantitative Analysis, 22, 419-438

[4] Steven L. Heston, 1993, "A Closed-Form Solution for Options with Stochastic Volatility with Applications to Bond and Currency Options", Review of Financial Studies, 6,327-343

[5] E.M. Stein & J.C. Stein, 1991, "Stock Price Distributions with Stochastic Volatility: An Analytic Approach", Review of Financial Studies, 4, 727-752

[6] Stella Christodoulou, 2000, "Finite Differences Applied to Stochastic Problems in Pricing Derivatives", MSc Thesis, University of Reading

[7] D. Gnandi, 1998, "Alternating Direction Implicit Method Applied to a Stochastic Problem in Derivative Finance", MSc Thesis, University of Reading

[8] Bloomberg market data, Bloomberg

[9] Contingency Analysis, 1996, http://www.riskglossary.com/link/monte_carlo_method.htm,

[10] B. Hambly, 2005, http://www.maths.ox.ac.uk/ hambly/B10b.html, lecture course "Elementary Financial Derivatives", University of Oxford

[11] Paul Wilmott, 1995, "Mathematics of Financial Derivatives", Cambridge University Press

[12] Richard Riley 2003, "Teach Yourself C++", Teach Yourself

9 Appendix

```
Coding for Black-Scholes Program

Main.cpp

//Black-Scholes Numerical solution: Rachael England

//include necessary headers and namespace
#include <iostream>
#include "NSDE_maths.h"
#include "NSDE_output.h"
#include "dissertation.h"

using namespace std;

int main (void)
{
   bool straddle;
   int i;
   int j;
   int m;

   int deriv_choice;
   int scheme_choice;

   int num_t;
   int num_s;

   double delta_t;
   double delta_s;

   double min_s;
   double max_s;
   double final_t;
   double r;
   double sigma;
   double price;
   double bond_t;
   double bond_f;
   double A;
```

```
  double spot_rate;
  double inter_above;
  double inter_below;

  TmatlabMatrix u;
  TmatlabVector t;
  TmatlabVector s;
  Tvector a;
  Tvector b;
  Tvector c;
  Tvector known_entries;
  Tvector next_step;
  TmatlabVector solution;

  string file_name;


  //ask user for input
  cout<<"This program uses a numerical method to solve the ";
  cout<<"Black-Scholes equation."<<endl<<endl;

  deriv_choice=choose_deriv();

  if(deriv_choice==1 || deriv_choice==2 || deriv_choice==6 || deriv_choice==7)
  {
cout<<"In order to solve numerically, a suitably small and a suitably large value of the ";
cout<<"asset price must be used to mimick boundary conditions as the price approaches 0 or infinity
cout<<"Please enter a suitably small value:"<<endl<<endl;
cin>>min_s;

cout<<endl<<endl<<"Thank you. Now please enter a suitably large value:"<<endl<<endl;
cin>>max_s;

cout<<endl<<endl<<"Thank you. Next, please give the maturity time of the derivative (assuming the t
cin>>final_t;

cout<<endl<<endl<<"Please specify the number of asset price steps you wish to use"<<endl<<endl;
cin>>num_s;
num_s=num_s-1;

cout<<endl<<endl<<"Please specify the number of time steps you wish to use"<<endl<<endl;
cin>>num_t;
num_t=num_t-1;

cout<<endl<<endl<<"The option you have chosen requires a strike price. Please enter this now:"<<end
cin>>price;
```

```
 cout<<endl<<endl<<"Now give the rate of increase r of the riskless asset:"<<endl<<endl;
 cin>>r;

 //name matlab matrix and vectors
 u.matrix_name="u";
 s.vector_name="s";
 t.vector_name="t";
 solution.vector_name="u at t_0";

   }

   if(deriv_choice==3||deriv_choice==4 || deriv_choice==5)
   {

 cout<<endl<<endl<<"Thank you. Next, please give the maturity time of the derivative (assuming the t
 cin>>final_t;

 cout<<endl<<endl<<"Thank you. Now please give the final maturity time of the interest rate swap, as
 cin>>bond_t;

 cout<<endl<<endl<<"Please enter the forward "<<final_t<<","<<bond_t-final_t<<" rate for the swap:"<
 cin>>bond_f;
 bond_f=bond_f*10000;

 min_s=bond_f/2.0;
 max_s=bond_f*3/2.0;
 num_s=ceil((max_s-min_s)/0.25 - 1);

 cout<<endl<<endl<<"Thank you. Now please specify the number of time steps you wish to use"<<endl<<e
 cin>>num_t;
 num_t=num_t-1;

 cout<<endl<<endl<<"The option you have chosen requires a strike rate. Please enter this now:"<<endl
 cin>>price;
 price=price*10000;

 cout<<endl<<endl<<"Thank you. In order to calculate the present value of the interest swaps, please
 cin>>m;

 cout<<endl<<endl<<"Now please enter the spot rates for all times up to the end of the swap."<<endl<
 A=0;

 for(i=0;i<(floor(m*(bond_t-final_t)));i++)
 {
cout<<"Time t = "<<final_t+(i+1.0)/(m+0.0)<<":"<<endl<<endl;
cin>>spot_rate;
A=A+exp(-spot_rate*(final_t+(i+1.0)/(m+0.0)));
 }

       r=0;

//name matlab matrix and vectors
u.matrix_name="swaption price in points";
s.vector_name="forward rate in points";
t.vector_name="time";
solution.vector_name="swaption price in points";

   }

   scheme_choice=choose_scheme();

   cout<<endl<<endl<<"The volatility is also required. Please enter this now:"<<endl<<endl;
   cin>>sigma;

   cout<<endl<<endl<<"Thank you. Finally, please enter the name of the file you wish to write:"<<end
   cin>>file_name;
   cout<<endl<<endl;

   //calculate delta s and delta t from number of steps given by user
   delta_s=(max_s-min_s)/(num_s+1.0);
   delta_t=final_t/(num_t+1.0);

   //set vectors and matrices to correct size
   u.matrix.resize(num_t+2,num_s+2);
   s.vector.resize(num_s+2);
   t.vector.resize(num_t+2);
   a.resize(num_s);
   b.resize(num_s-1);
   c.resize(num_s-1);
   next_step.resize(num_s);
   known_entries.resize(num_s);
   if(deriv_choice==1 || deriv_choice==2 || deriv_choice==6 || deriv_choice==7)
   {
solution.vector.resize(num_s+2);
   }

   if(deriv_choice==3 || deriv_choice==4 || deriv_choice==5)
   {
solution.vector.resize(1);
solution.vector(0)=0;
   }
   //loop for all s steps to fill in s values
   for (i=0;i<s.vector.rows();i++)
      {
         s.vector(i)=min_s+delta_s*i;
      }

   //loop for all t steps to fill in t values
    for (i=0;i<t.vector.rows();i++)
      {
         t.vector(i)=delta_t*i;
      }

    do
{

//reset u matrix to 0
u.matrix=0;

//insert boundary values using functions
u.matrix=boundary_s(r,t.vector,s.vector,deriv_choice,u.matrix,price);
u.matrix=boundary_t(r,t.vector,s.vector,deriv_choice,u.matrix,price);

//fill in tridiagonal values for A matrix using a function
fill_matrix_A(a,b,c,sigma,s.vector,delta_t,delta_s,r,scheme_choice);

//loop through all time steps
for (i=1;i<t.vector.rows();i++)
{
//fill in entries for right hand side of equation
known_entries=fill_known(known_entries,u.matrix,i,sigma,s.vector,delta_t,delta_s,r,scheme_choice);

//solve tidiagonal equation
next_step=tridiag_solve(c,a,b,known_entries);

//indert solution back into u matrix
for (j=1;j<s.vector.rows()-1;j++)
{
u.matrix(i,j)=next_step(j-1);
}
}

if(deriv_choice==3 || deriv_choice==4 || deriv_choice==5)
{
for(i=0;i<s.vector.rows();i++)
{
```

```
if(s.vector(i)>bond_f && s.vector(i-1)<=bond_f)
{
inter_above=u.matrix(t.vector.rows()-1,i);
inter_below=u.matrix(t.vector.rows()-1,i-1);
solution.vector(0)=solution.vector(0)+((inter_above-inter_below)/(s.vector(i)-s.vector(i-1))*(bond_f
break;
}
}
}
if(deriv_choice==5)
{
straddle=true;
deriv_choice=4;
}
else
{
straddle=false;
}
}while(straddle==true);

if(open_m(file_name)==true)
{
//if file opens, write vectors to it and close file
   if(deriv_choice==1 || deriv_choice==2 || deriv_choice==6 || deriv_choice==7)
   {
writevector_m(s);
for(i=0;i<s.vector.rows();i++)
{
solution.vector(i)=u.matrix(t.vector.rows()-1,i);
}
   }

   writevector_m(solution);

       close_m();

      //inform user
      cout<<"File has been written"<<endl;
   }
   else
   {
       //otherwise, inform user file opening has failed
       cout<<"Sorry; there was an error opening this file"<<endl;
   }

   //pause file to allow user to view, before returning an integer
   system("PAUSE");

   return 1;
}
```

dissertation.h

```
//Header for outputting to file - Rachael England

//prevent looping error
#ifndef DISSERTATION2_H
#define DISSERTATION2_H

//include necessary libraries and header files
#include "NSDE_maths.h"
#include <string>

//define functions
int choose_deriv();
int choose_scheme();
Tmatrix boundary_s(double r,Tvector t,Tvector s,int choice, Tmatrix u,double E);
Tmatrix boundary_t(double r,Tvector t,Tvector s,int choice, Tmatrix u, double E);
void fill_matrix_A(Tvector& a,Tvector& b, Tvector& c,double sigma,Tvector s,double delta_t,double de
Tvector fill_known(Tvector d,Tmatrix u,int i,double sigma,Tvector s,double delta_t,double delta_s,do

#endif
```

dissertation.cpp

```
//include headers
#include <iostream>
#include "dissertation2.h"
#include "NSDE_maths.h"

using namespace std;

int choose_deriv()
{
   int choice=0;

   //list derivative choices and loop until user makes a valid choice
   do
   {
      cout<<endl<<endl<<"Now please choose one of the following derivatives to solve for:"<<endl<<en
      cout<<"1. European Call Option"<<endl;
      cout<<"2. European Put Option"<<endl;
  cout<<"3. European Call Swaption"<<endl;
  cout<<"4. European Put Swaption"<<endl;
  cout<<"5. European Straddle Swaption"<<endl;
  cout<<"6. Digital Call Option"<<endl;
  cout<<"7. Digital Put Option"<<endl;
      cin>>choice;
   }while (choice!=1 && choice!=2 && choice!=3 && choice!=4 && choice!=5 && choice!=6 && choice!=7);

   //pass user's choice back to main program
   return choice;
}

int choose_scheme()
{
   int choice=0;

   //list scheme choices and loop until user makes a valid choice
   do
   {
      cout<<endl<<endl<<"Thank you. Now please choose which of the following schemes you wish to use
      cout<<"1. Crank-Nicolson"<<endl;
      cout<<"2. Kenneth-Vetzal"<<endl;
      cout<<"3. Fully Implicit"<<endl;
      cout<<"4. Semi Implicit"<<endl;
      cout<<"5. Explicit 1"<<endl;
      cout<<"6. Explicit 2"<<endl;
      cin>>choice;
   }while (choice!=1 && choice!=2 && choice!=3 && choice!=4 && choice!=5 && choice!=6);

   //pass user's choice back to main program
   return choice;
}

Tmatrix boundary_s( double r,Tvector t,Tvector s,int choice, Tmatrix u,double E)
{
   int i;
   int end_s=s.rows()-1;

   //loop through u for all time steps
   for (i=0;i<t.rows();i++)
   {
      //insert boundary conditions for minimum and maximum S values
      //depending on the user's choice of derivative
      if (choice==1 || choice==3 || choice==5)
      {
         u(i,0)=0;
         u(i,end_s)=s(end_s)-E*exp(-r*t(i));
      }
      if (choice==2 || choice==4)
      {
         u(i,0)=E*exp(-r*t(i))-s(0);
         u(i,end_s)=0;
      }
      if(choice==6)
      {
         u(i,0)=0;
         u(i,end_s)=exp(-r*t(i));
      }
      if(choice==7)
      {
         u(i,0)=exp(-r*t(i));
         u(i,end_s)=0;
      }
   }

   //pass resultant matrix back to main program
   return u;
}

Tmatrix boundary_t(double r,Tvector t,Tvector s,int choice, Tmatrix u, double E)
{
   int i;

   //loop through for all s steps
   for (i=0;i<s.rows();i++)
   {
      //insert initial condition at t=0 depending on
      //user's choice of derivative
      if(choice==1 || choice==3 || choice==5)
      {
         if (s(i)-E>0)
         {
            u(0,i)=s(i)-E;
         }
         else
         {
            u(0,i)=0;
         }
      }
      if(choice==2 || choice==4)
      {
         if (E-s(i)>0)
         {
            u(0,i)=E-s(i);
         }
         else
         {
            u(0,i)=0;
         }
      }
      if(choice==6)
      {
         if(s(i)-E>0)
         {
            u(0,i)=1;
         }
         else
         {
             u(0,i)=0;
         }
      }
      if(choice==7)
      {
         if(s(i)-E>0)
         {
            u(0,i)=0;
         }
         else
         {
            u(0,i)=1;
            }
      }
   }

   //pass resultant matrix back to main program
   return u;
}

void fill_matrix_A(Tvector& a,Tvector& b,Tvector& c,double sigma,Tvector s,double delta_t,double del
{
    int i;
    double alpha;
    double beta;
    double gamma;
    double theta_1;
    double theta_2;
    double theta_3;
    double theta_4;
    double theta_5;
    double theta_6;

    //set theta values based on user's choice of numerical scheme
    if (choice==1)
    {
       theta_1=0.5;
       theta_2=0.5;
       theta_3=0.5;
       theta_4=0.5;
       theta_5=0.5;
       theta_6=0.5;
    }
    if (choice==2)
    {
       theta_1=1;
       theta_2=0;
       theta_3=1;
       theta_4=0;
       theta_5=0;
       theta_6=1;
    }
    if (choice==3)
    {
       theta_1=1;
       theta_2=0;
       theta_3=1;
       theta_4=0;
       theta_5=1;
       theta_6=0;
    }
    if (choice==4)
    {
       theta_1=1;
       theta_2=0;
       theta_3=0;
       theta_4=1;
       theta_5=1;
       theta_6=0;
    }
    if (choice==5)
    {
       theta_1=0;
       theta_2=1;
       theta_3=0;
       theta_4=1;
       theta_5=0;
       theta_6=1;
    }
    if (choice==6)
    {
       theta_1=0;
       theta_2=1;
       theta_3=0;
       theta_4=1;
       theta_5=1;
       theta_6=0;
    }

    //loop through all rows in the left hand side matrix
    for (i=0;i<a.rows();i++)
    {
       //caluclate alpha, beta, and gamma for use in calculations
       alpha=0.5*sigma*sigma*s(i+1)*s(i+1)*delta_t/(delta_s*delta_s);
       beta=0.5*r*s(i+1)*delta_t/delta_s;
       gamma=-r*delta_t;

       //set a, b, and c values for the ith row of the matrix there
       //is no value for c in the first row, and no value for b in th
       //final row. Hence the c and b vectors are 1 unit smaller that a.
       if (i!=0 && i!=(a.rows()-1) )
       {
          c(i-1)=-alpha*theta_1+beta*theta_3;
          a(i)=1+2*alpha*theta_1-gamma*theta_5;
          b(i)=-alpha*theta_1-beta*theta_3;
       }
       else if(i==0)
       {
          a(i)=1+2*alpha*theta_1-gamma*theta_5;
          b(i)=-alpha*theta_1-beta*theta_3;
       }
       else
       {
          c(i-1)=-alpha*theta_1+beta*theta_3;
          a(i)=1+2*alpha*theta_1-gamma*theta_5;
       }
    }
    //no need for return since a, b, and c were passed via pointers
```

```
}

Tvector fill_known(Tvector d,Tmatrix u,int i,double sigma,Tvector s,double delta_t,double delta_s,do
{
    int j;
    double alpha;
    double beta;
    double gamma;
    double a;
    double b;
    double c;
    double theta_1;
    double theta_2;
    double theta_3;
    double theta_4;
    double theta_5;
    double theta_6;

    //set theta values based on user's choice of numerical scheme
    if (choice==1)
    {
       theta_1=0.5;
       theta_2=0.5;
       theta_3=0.5;
       theta_4=0.5;
       theta_5=0.5;
       theta_6=0.5;
    }
    if (choice==2)
    {
       theta_1=1;
       theta_2=0;
       theta_3=1;
       theta_4=0;
       theta_5=0;
       theta_6=1;
    }
    if (choice==3)
    {
       theta_1=1;
       theta_2=0;
       theta_3=1;
       theta_4=0;
       theta_5=1;
       theta_6=0;
    }
```

```
if (choice==4)
{
   theta_1=1;
   theta_2=0;
   theta_3=0;
   theta_4=1;
   theta_5=1;
   theta_6=0;
}
if (choice==5)
{
   theta_1=0;
   theta_2=1;
   theta_3=0;
   theta_4=1;
   theta_5=0;
   theta_6=1;
}
if (choice==6)
{
   theta_1=0;
   theta_2=1;
   theta_3=0;
   theta_4=1;
   theta_5=1;
   theta_6=0;
}

//loop through all rows in the right hand side vector
for (j=0;j<d.rows();j++)
{
   //calculate alpha beta and gamma for use in calculating vector values
   alpha=0.5*sigma*sigma*s(j+1)*s(j+1)*delta_t/(delta_s*delta_s);
   beta=0.5*r*s(j+1)*delta_t/delta_s;
   gamma=-r*delta_t;

   //use alpha beta and gamma to calculate the entries for B
   c=alpha*theta_2-beta*theta_4;
   a=1-2*alpha*theta_2+gamma*theta_6;
   b=alpha*theta_2+beta*theta_4;

   //use a b and c to work out right hand side through equation
   //B u_j + boundary conditions for first and final rows
   if (j!=0 && j!=(d.rows()-1) )
   {
      d(j)=c*u(i-1,j)+a*u(i-1,j+1)+b*u(i-1,j+2);
```

```
       }
       else if(j==0)
       {
          d(j)=c*u(i-1,j)+a*u(i-1,j+1)+b*u(i-1,j+2)+(alpha*theta_1-beta*theta_3)*u(i,j);
       }
       else
       {
          d(j)=c*u(i-1,j)+a*u(i-1,j+1)+b*u(i-1,j+2)+(alpha*theta_1+beta*theta_3)*u(i,j+2);
       }
    }
    return d;
}
```

NSDE_maths.h

```
//Maths header file - Rachael England

//Prevent looping error
#ifndef NSDE_MATHS_H
#define NSDE_MATHS_H

//include all libraries needed
#include <cmath>
#include <complex>
#include <blitz\array.h>

//Add in namespace blitz
using namespace blitz;

//define matrix and vector types
typedef Array <double,1> Tvector;
typedef Array <double,2> Tmatrix;

Tvector matrixXvector(Tmatrix A, Tvector b);
Tmatrix matrixXmatrix(Tmatrix A, Tmatrix B);
Tvector tridiag_solve(Tvector a,Tvector b,Tvector c,Tvector r);
Tmatrix tridiag_solve_matrix(Tvector a,Tvector b,Tvector c,Tmatrix r);
Tvector gauss_seidel(Tmatrix A, Tvector b);
double max_d(double a, double b);

#endif
```

NSDE_maths.cpp

```
//Mathematical Functions - Rachael England

//include headers
#include <iostream>
#include "NSDE_maths.h"

using namespace std;

Tvector matrixXvector(Tmatrix A, Tvector b)
{
   int i;
   int j;

   Tvector answer;

   answer.resize(A.rows());
   //multiply the matrix by the vector if possible
   if (A.cols()==b.rows())
   {
      //loop for all rows of the resultant vector
      for (i=0; i<A.rows(); i++)
      {
         j=0;
         answer(i)=0;

         for (j=0; j<A.cols(); j++)
         {
            //using the formula for multiplying a matrix with a vector
            answer(i)=answer(i)+A(i,j)*b(j);
         }
      }
   }
   else
   {
      cout<<"Sorry, it is not possible to multiply the matrix by the vector"<<endl<<endl;
  system("PAUSE");
   }

   return answer;
}

Tmatrix matrixXmatrix(Tmatrix A, Tmatrix B)
{
   int i;
   int j;
   Tmatrix solution;
   Tvector x;

   solution.resize(B.rows(),B.cols());
   x.resize(B.rows());

   for(i=0;i<B.cols();i++)
   {
      for(j=0;j<B.rows();j++)
  {
         x(j)=B(j,i);
  }
  x=matrixXvector(A,x);
  for(j=0;j<B.rows();j++)
  {
     solution(j,i)=x(j);
  }
   }
   return solution;
}

Tvector tridiag_solve(Tvector a,Tvector b,Tvector c,Tvector r)
{
   int i;
   int n;
   Tvector w;
   Tvector alpha;
   Tvector beta;
   Tvector gamma;
   Tvector u;

   n=r.rows();

   w.resize(n);
   alpha.resize(n);
   beta.resize(n);
   gamma.resize(n);
   u.resize(n);

   //test correct number of rows for solver to work
   if(b.rows()==a.rows()+1 && b.rows()==c.rows()+1 && b.rows()==r.rows())
   {

      //use new vectors to store diagonals for lower and upper triangular factorisation
      //amd use w vector to store forward substitution stage
      beta(0) = b(0);
      gamma(0) = c(0);
      w(0) = r(0);
      for(i=1;i<n;i++)
      {
         alpha(i) = a(i-1)/beta(i-1);
         beta(i) = b(i) - alpha(i)*gamma(i-1);
         gamma(i) = c(i);
         w(i) = r(i) - alpha(i)*w(i-1);
      }

      //backwards substitution calculated and stored in y
      u(n-1)=w(n-1)/beta(n-1);
      for(i=n-2;i>=0;i--)
      {
         u(i)= (w(i)-gamma(i)*u(i+1))/beta(i);
      }
   }
   else
   {
      cout<<"Sorry, it is not possible to solve this as the number of  diagonal entries do not match
  system("PAUSE");
   }

   //solution in y passed back to main function
   return u;
}

Tmatrix tridiag_solve_matrix(Tvector a,Tvector b,Tvector c,Tmatrix r)
{
   int i;
   int j;
   Tmatrix solution;
   Tvector x;

   solution.resize(r.rows(),r.cols());
   x.resize(r.rows());

   for(i=0;i<r.cols();i++)
   {
      for(j=0;j<r.rows();j++)
  {
         x(j)=r(j,i);
  }
  x=tridiag_solve(a,b,c,x);
  for(j=0;j<r.rows();j++)
  {
     solution(j,i)=x(j);
  }
   }
   return solution;
}

Tvector gauss_seidel(Tmatrix A, Tvector b)
{
   int j;
   int k;
   Tvector x_new;
   Tvector x_old;
   double sum1;
   double sum2;
   double check;

   //set x to correct size
   x_old.resize(b.rows());
   x_new.resize(b.rows());
   x_old=0;

   //check columns of A matches with rows of b
   if(A.cols()==x_new.rows())
   {
      //loopm until within tolerance level
   do
      {
     //loop through all rows
         for(j=0;j<b.rows();j++)
         {
            sum1=0;
        sum2=0;

//loop through rows to get first sum, i.e. sum from k=0 to j-1 of A(j,k)x_new(k)
for(k=0;k<j;k++)
        {
           sum1=sum1+(A(j,k)*x_new(k));
        }

//loop through for second sum; i.e. sum from k=j+1 to final row of A(j,k)x_old(k)
        for(k=j+1;k<b.rows();k++)
        {
           sum2=sum2+(A(j,k)*x_old(k));
        }

//perform calculation: x(j)=1/A(j,j)[b(j) - sum1 - sum2]
```

```
            x_new(j)=(1.0/A(j,j))*(b(j) - sum1-sum2);
         }

 //check error
 //reset check to 0
     check=0;
     for(j=0;j<b.rows();j++)
     {
//add |new x - old x| for row j onto check
    check=check+fabs(x_new(j)-x_old(j));
     }

 //set old x for next loop
 x_old=x_new;

  //loop until within tolerance level
      }while (check>0.00001*b.rows());
   }
   else
   {
      cout<<"Sorry; it is not possible to solve this matrix equation";
  system("PAUSE");
   }

   return x_new;
}

double max_d(double a, double b)
{
double solution;

solution=0.5*(fabs(a-b) + a + b);

return solution;
}



NSDE_output.h

//Header for outputting to file - Rachael England

//prevent looping error
#ifndef NSDE_OUTPUT_H
#define NSDE_OUTPUT_H
```

```
//include necessary libraries and header files
#include "NSDE_maths.h"
#include <fstream>
#include <string>

//define matlab matrix and vector structures
typedef struct
{
 string matrix_name;
 Tmatrix matrix;

} TmatlabMatrix;

typedef struct
{
 string vector_name;
 Tvector vector;
} TmatlabVector;


//define output functions for NSDE_output.cpp
bool open_m (string file_name);
void close_m (void);
void writevector_m (TmatlabVector mat_vector);
void writematrix_m (TmatlabMatrix mat_matrix);
void plot_m (TmatlabVector x1, TmatlabVector x2);
void plot2_m (TmatlabVector x1, TmatlabVector x2, TmatlabVector x3);
void plot3D_m (TmatlabVector x1, TmatlabVector x2, TmatlabMatrix u);
void plot_function(string function_str, double a, double b);
void output_string(string string_value);

#endif
```

NSDE_output.cpp

```
//Outputting to file - Rachael England

//include headers
#include <iostream>
#include "NSDE_output.h"
#include "NSDE_maths.h"

using namespace std;

//define global variable
```

```
ofstream fout;

bool open_m (string file_name)
{
   //open file and return boolean depending on whether operation succeeded
   fout.open (file_name.c_str ());
   return fout.good ();
}

void close_m (void)
{
   //if file opened,close file
   if(fout.good())
   {
    fout.close ();
   }
}

void writevector_m (TmatlabVector mat_vector)
{
   //define variables
   int i;

   //write to file as 'name = ['
   fout<<mat_vector.vector_name;
   fout<<" = [";

   for (i=0; i<mat_vector.vector.rows(); i++)
   {
      //for each value in the vector, add onto the file 'value'
      fout<<mat_vector.vector(i);

      //if it's the last value, put '];' on the file, else add ', '
      if (i!=mat_vector.vector.rows()-1)
      {
         fout<<", ";
      }
      else
      {
         fout<<"];";
      }
   }

   //final output looks like: 'name = [ value, value, value, ... ];'; write this to the file
   //add a new line for matlab to seperate out commands
   fout<<endl;
}

void writematrix_m (TmatlabMatrix mat_matrix)
{
   //define variables
   int i,j;

   //write to file as 'name = ['
   fout<<mat_matrix.matrix_name;
   fout<<" = [";

   for (i=0; i<mat_matrix.matrix.rows();i++)
   {
      //add onto ';' for the start of each new row in the matrix
      if (i!=0)
      {
         fout<<";"<<endl;
      }

      for (j=0; j<mat_matrix.matrix.cols();j++)
      {
         //for each value in the row, add onto the file 'value'
         fout<<mat_matrix.matrix(i,j);

         //if it's not the last value in the row, add ', '
         if (j!=mat_matrix.matrix.cols()-1)
         {
            fout<<", ";
         }
      }
   }

   //finish off the string with '];'
   fout<<"];";

   //the final output looks like:
   //'name = [ value, value, value, ... ;value, value, value, ... ; ... ];'
   //add line to inform matlab of new command
   fout<<endl;
}

void plot_m (TmatlabVector x1, TmatlabVector x2)
{
   //output Matlab instructions for plotting graph
   fout<<"figure; plot ("<<x1.vector_name<<","<<x2.vector_name<<");"<<endl;
}
```

```
void plot2_m (TmatlabVector x1, TmatlabVector x2, TmatlabVector x3)
{
    //output Matlab instructions for plotting graph with extra variable
    fout<<"figure; plot ("<<x1.vector_name<<","<<x2.vector_name<<","<<x1.vector_name<<","<<x3.vector
}

void plot3D_m (TmatlabVector x1, TmatlabVector x2, TmatlabMatrix u)
{
    //output Matlab instructions for plotting 3D graph of a matrix
    fout<<"figure; surf("<<x1.vector_name<<", "<<x2.vector_name<<","<<u.matrix_name<<");"<<endl;
}

void plot_function(string function_str, double a, double b)
{

   //output Matlab instructions for plotting specified function between x=a and x=b
   fout<<"y=linspace("<<a<<","<<b<<");"<<endl;
   fout<<"plot(y,"<<function_str<<");"<<endl;
}

void output_string(string string_value)
{
   //output string to file
   fout<<string_value<<endl;
}
```

Coding for Heston Program

Hston_main.cpp

```
//Rachael England: Program for numerical solution of the Heston equation

//include necessary headers and namespace
#include <iostream>
#include "NSDE_maths.h"
#include "NSDE_output.h"
#include "heston.h"

using namespace std;

int main (void)
{
//define variables
```

```
int j;
int k;
int num_s;
int num_v;
int num_t;
double delta_s;
double delta_v;
double delta_t;
double t_end;
double s_min;
double s_max;
double v_min;
double v_max;
double bond_f;
double bond_t;
double rho;
double sigma;
double r;
double K;
double theta;
double E;
double E2;
double A;
double inter_above;
double inter_below;
int scheme;
int deriv;
TmatlabVector s;
TmatlabVector v;
Tvector t;
TmatlabMatrix u;
Tmatrix u_old;
TmatlabVector solution;
Tmatrix rhs;
Tmatrix a;
Tmatrix b;
Tmatrix c;
Tmatrix d;
Tmatrix e;
Tmatrix f;
Tmatrix g;
Tmatrix h;
Tmatrix l;
string file_name;

//ask user for input
```

```
cout<<"This program uses a numerical method to solve the ";
cout<<"Heston equation."<<endl<<endl;

deriv=choose_deriv();

//required input varies depending on choice of derivative
if(deriv==1 || deriv==2 || deriv==6)
{
cout<<"In order to solve numerically, a suitably small and a suitably large value of the ";
cout<<"asset price must be used to mimick boundary conditions as the asset price approaches 0 or inf
cout<<"Please enter a suitably small value:"<<endl<<endl;
cin>>s_min;

cout<<endl<<endl<<"Thank you. Now please enter a suitably large value:"<<endl<<endl;
cin>>s_max;

cout<<endl<<endl<<"Similar conditions are required for the volatility."<<endl;
cout<<"Please enter a suitably small value:"<<endl<<endl;
     cin>>v_min;

cout<<endl<<endl<<"Thank you. Now please enter a suitably large value:"<<endl<<endl;
cin>>v_max;

cout<<endl<<endl<<"Thank you. Next, please give the final time (assuming the time at the start is 0)
cin>>t_end;

cout<<endl<<endl<<"Please specify the number of asset price steps you wish to use"<<endl<<endl;
cin>>num_s;
num_s=num_s-1;

cout<<endl<<endl<<"Thank you. Please enter a value for theta (the average volatility):"<<endl<<endl;
cin>>theta;

cout<<endl<<endl<<"Please specify the number of volatility steps you wish to use"<<endl<<endl;
cin>>num_v;
num_v=num_v-1;

cout<<endl<<endl<<"Please specify the number of time steps you wish to use"<<endl<<endl;
cin>>num_t;
num_t=num_t-1;

if(deriv==6)
{
cout<<endl<<endl<<"The derivative you have chosen requies two strike prices. Firstly, please enter t
cin>>E2;
```

```
cout<<endl<<endl<<"Thank you. Now please enter the srike price E1, used to calculate the payoff (i.e
cin>>E;
}
else
{
cout<<endl<<endl<<"The option you have chosen requires a strike price. Please enter this now:"<<endl
cin>>E;
}

cout<<endl<<endl<<"Now give the rate of increase r of the riskless asset:"<<endl<<endl;
cin>>r;

u.matrix_name="Price of Derivative U at Initial Time";
s.vector_name="Asset Price S";
v.vector_name="Volatility v";

}

if(deriv==3||deriv==4 || deriv==5)
{

cout<<endl<<endl<<"Thank you. Next, please give the maturity time of the derivative (assuming the ti
cin>>t_end;

cout<<endl<<endl<<"Thank you. Now please give the final maturity time of the interest rate swap, ass
cin>>bond_t;

cout<<endl<<endl<<"Please enter the forward "<<t_end<<","<<bond_t-t_end<<" rate for the swap:"<<endl
cin>>bond_f;
bond_f=bond_f*10000;

s_min=bond_f*3.0/4.0;
s_max=bond_f*5.0/4.0;

num_s=ceil((s_max-s_min)/0.5 - 1);

cout<<endl<<endl<<"Thank you. Now please specify the number of time steps you wish to use"<<endl<<en
cin>>num_t;
num_t=num_t-1;

cout<<endl<<endl<<"The option you have chosen requires a strike rate. Please enter this now:"<<endl<
cin>>E;
E=E*10000;

cout<<endl<<endl<<"Thank you. Please enter a value for theta (the average volatility):"<<endl<<endl;
cin>>theta;
v_min=theta*1.0/2.0;
v_max=theta*3.0/2.0;

num_v=ceil((v_max-v_min)/0.005 - 1);

cout<<endl<<endl<<"Thank you. In order to calculate the present value of the interest swaps, please
cin>>m;

cout<<endl<<endl<<"Now please enter the spot rates for all times up to the end of the swap."<<endl<<
A=0;

for(i=0;i<(floor(m*(bond_t-final_t)));i++)
{
cout<<"Time t = "<<final_t+(i+1.0)/(m+0.0)<<":"<<endl<<endl;
cin>>spot_rate;
A=A+exp(-spot_rate*(final_t+(i+1.0)/(m+0.0)));
}
A=A/(m+0.0);

r=0;

//name matlab matrix and vectors

u.matrix_name="Price of Swaption at Initial Time";
s.vector_name="Forward Rate";
v.vector_name="Volatility";
solution.vector_name="solution";

solution.vector.resize(num_v+2);
   }

scheme=choose_scheme();

cout<<endl<<endl<<"The value of sigma is also required. Please enter this now:"<<endl<<endl;
cin>>sigma;

cout<<endl<<endl<<"Please enter a value for rho:"<<endl<<endl;
cin>>rho;

cout<<endl<<endl<<"Please enter a value for k:"<<endl<<endl;
cin>>K;

cout<<endl<<endl<<"Thank you. Finally, please enter the name of the file you wish to write:"<<endl<<
cin>>file_name;
cout<<endl<<endl;
```

```
    //calculate step sizes and resize matrices and ectors
delta_s=(s_max-s_min)/(num_s+1.0);
delta_v=(v_max-v_min)/(num_v+1.0);
delta_t=t_end/(num_t+1.0);

s.vector.resize(num_s+2);
v.vector.resize(num_v+2);
t.resize(num_t+2);
u.matrix.resize(num_s+2,num_v+2);
u_old.resize(num_s+2,num_v+2);
a.resize(num_s,num_v+2);
b.resize(num_s,num_v+2);
c.resize(num_s,num_v+2);
d.resize(num_s,num_v+2);
e.resize(num_s,num_v+2);
f.resize(num_s,num_v+2);
g.resize(num_s,num_v+2);
h.resize(num_s,num_v+2);
l.resize(num_s,num_v+2);
rhs.resize(num_s,num_v+2);

//fill in discrete values for S, v, and tau
fill_grid(s.vector,s_min,delta_s);
fill_grid(v.vector,v_min,delta_v);
fill_grid(t,0,delta_t);

//loop through all time steps
for(j=0;j<t.rows();j++)
{
if(j==0)
{
//fil in initial values if j=0
t_boundary(r,s.vector,deriv,u.matrix,E,E2);
}
else
{
//fill in boundary values
s_boundary(r,t,s.vector,deriv,u.matrix,E,j);

//enter values into diagonals of A
define_A(a,b,c,d,e,f,g,h,l,s.vector,v.vector,delta_s,delta_v,delta_t,rho,sigma,r,K,theta,j,scheme);

//enter vlues into right hand side vector
define_rhs(u_old,u.matrix,a,b,c,d,e,f,g,h,l,rhs,s.vector,v.vector,delta_s,delta_v,delta_t,rho,sigma,
//solve matrix equation using Gauss-Seidel iteration
solve_gs(a,b,c,d,e,f,g,h,l,u.matrix,rhs);
}
//put values into u_old matrix for use in next time step
u_old=u.matrix;
}

   if(open_m(file_name)==true)
   {
       //if file opens, write vectors to it and close file
       writevector_m(v);
   if(deriv==1 || deriv==2 || deriv==6)
   {
writevector_m(s);
writematrix_m(u);
   }
   if(deriv==3 || deriv==4 || deriv==5)
   {
for(j=1;j<s.vector.rows();j++)
{
if(s.vector(j)>bond_f && s.vector(j-1)<=bond_f)
{
for(k=0;k<v.vector.rows();k++)
{
inter_above=u.matrix(j,k);
inter_below=u.matrix(j-1,k);
solution.vector(k)=((inter_above-inter_below)/(s.vector(j)-s.vector(j-1))*(bond_f-s.vector(j-1))+int
}
}
}
writevector_m(solution);
   }

       close_m();

      //inform user
      cout<<"File has been written"<<endl;
   }
   else
   {
       //otherwise, inform user file opening has failed
       cout<<"Sorry; there was an error opening this file"<<endl;
   }

   //pause file to allow user to view, before returning an integer
   system("PAUSE");
   return 1;
}
```

Heston.h

```
//prevent looping error
#ifndef HESTON2_H
#define HESTON2_H

//include necessary libraries and header files
#include "NSDE_maths.h"
#include <fstream>
#include <string>

//define functions
int choose_scheme(void);
int choose_deriv(void);
void fill_grid(Tvector& x,double min,double delta_x);
void s_boundary(double r,Tvector& t,Tvector& s,int choice, Tmatrix& u,double E,int j);
void t_boundary(double r,Tvector& s,int choice, Tmatrix& u, double E,double E2);
void define_A(Tmatrix& a, Tmatrix& b, Tmatrix& c, Tmatrix& d, Tmatrix& e, Tmatrix& f, Tmatrix& g, Tm
void define_rhs(Tmatrix& u_old, Tmatrix& u, Tmatrix& a, Tmatrix& b, Tmatrix& c, Tmatrix& d, Tmatrix&
void solve_gs(Tmatrix& a,Tmatrix& b,Tmatrix& c,Tmatrix& d,Tmatrix& e,Tmatrix& f,Tmatrix& g,Tmatrix&
double lambda(double s, double v, double t);

#endif
```

Heston.cpp

```
//include headers
#include <iostream>
#include "heston2.h"
#include "NSDE_maths.h"

using namespace std;

int choose_deriv()
{
int choice=0;

//list derivative choices and loop until user makes a valid choice
do
{
```

```
cout<<endl<<endl<<"Now please choose one of the following derivatives to solve for:"<<endl<<endl;
cout<<"1. Call Option"<<endl;
cout<<"2. Put Option"<<endl;
cout<<"3. European Call Swaption"<<endl;
cout<<"4. European Put Swaption"<<endl;
cout<<"5. European Straddle Swaption"<<endl;
cout<<"6. Knock-in Cap"<<endl;
cin>>choice;
}while (choice!=1 && choice!=2 && choice!=3 && choice!=4 && choice!=5 && choice!=6);

//pass user's choice back to main program
return choice;
}

int choose_scheme()
{
   int choice=0;

   //list scheme choices and loop until user makes a valid choice
   do
   {
      cout<<endl<<endl<<"Thank you. Now please choose which of the following schemes you wish to use
      cout<<"1. Crank-Nicolson"<<endl;
      cout<<"2. Kenneth-Vetzal"<<endl;
      cout<<"3. Fully Implicit"<<endl;
      cout<<"4. Semi Implicit"<<endl;
      cout<<"5. Explicit 1"<<endl;
      cout<<"6. Explicit 2"<<endl;
      cin>>choice;
   }while (choice!=1 && choice!=2 && choice!=3 && choice!=4 && choice!=5 && choice!=6);

   //pass user's choice back to main program
   return choice;
}

void fill_grid(Tvector& x,double min,double delta_x)
{
int i;

//loop through vector and fill in discret value for each step
for(i=0;i<x.rows();i++)
{
   x(i)=min+i*delta_x;
}
}
```

```
void s_boundary( double r,Tvector& t,Tvector& s,int choice, Tmatrix& u,double E,int j)
{
   int k;
   int end_s=s.rows()-1;

   //loop through u for all v steps
   for (k=0;k<u.cols();k++)
   {
      //insert boundary conditions for minimum and maximum s values
      //depending on the user's choice of derivative
      if (choice==1 || choice==3 || choice==6)
      {
         u(0,k)=0;
         u(end_s,k)=s(end_s)-E*exp(-r*t(j));
      }
      if (choice==2 || choice==4)
      {
         u(0,k)=E*exp(-r*t(j))-s(0);
         u(end_s,k)=0;
      }
      if (choice==5)
      {
         u(0,k)=E*exp(-r*t(j))-s(0);
         u(end_s,k)=s(end_s)-E*exp(-r*t(j));
      }
   }
}

void t_boundary(double r,Tvector& s,int choice, Tmatrix& u, double E, double E2)
{
   int i;
   int k;

   //loop through for all s steps
   for (i=0;i<s.rows();i++)
   {
      //loop through for all v steps
      for(k=0;k<u.cols();k++)
  {
     //insert initial condition at t=0 depending on
         //user's choice of derivative
 if(choice==1 || choice==3)
 {
            if (s(i)-E>0)
            {
               u(i,k)=s(i)-E;
            }
            else
            {
               u(i,k)=0;
            }
         }
         if(choice==2 || choice==4)
         {
            if (E-s(i)>0)
            {
               u(i,k)=E-s(i);
            }
            else
            {
               u(i,k)=0;
            }
 }
         if(choice==5)
         {
            if (E-s(i)>0)
            {
               u(i,k)=E-s(i);
            }
            else
            {
               u(i,k)=s(i)-E;
            }
 }
 if(choice==6)
 {
   if(s(i)>E2)
{
u(i,k)=s(i)-E;
}
else
{
u(i,k)=0;
}
 }
      }
   }
}

void define_A(Tmatrix& a, Tmatrix& b, Tmatrix& c, Tmatrix& d, Tmatrix& e, Tmatrix& f, Tmatrix& g, Tm
{
int i;
int k;
double alpha;
double beta;
double gamma;
double zeta;
double eta;
double mu=-r*delta_t;
double theta_1;
double theta_2;
double theta_3;
double theta_4;
double theta_5;
double theta_6;
double theta_7;
double theta_8;
double theta_9;
double theta_10;
double theta_11;
double theta_12;

    //set theta values based on user's choice of numerical scheme
    if (choice==1)
    {
       theta_1=0.5;
       theta_2=0.5;
       theta_3=0.5;
       theta_4=0.5;
       theta_5=0.5;
       theta_6=0.5;
   theta_7=0.5;
   theta_8=0.5;
   theta_9=0.5;
   theta_10=0.5;
   theta_11=0.5;
   theta_12=0.5;
    }
    if (choice==2)
    {
       theta_1=1;
       theta_2=0;
       theta_3=1;
       theta_4=0;
   theta_5=1;
   theta_6=0;
   theta_7=1;
   theta_8=0;
```

```
theta_9=1;
theta_10=0;
    theta_11=0;
    theta_12=1;
 }
 if (choice==3)
 {
    theta_1=1;
    theta_2=0;
    theta_3=1;
    theta_4=0;
    theta_5=1;
    theta_6=0;
theta_7=1;
theta_8=0;
theta_9=1;
theta_10=0;
theta_11=1;
theta_12=0;
 }
 if (choice==4)
 {
    theta_1=1;
    theta_2=0;
    theta_3=1;
theta_4=0;
theta_5=1;
theta_6=0;
theta_7=0;
theta_8=1;
theta_9=0;
    theta_10=1;
    theta_11=1;
    theta_12=0;
 }
 if (choice==5)
 {
    theta_1=0;
    theta_2=1;
    theta_3=0;
    theta_4=1;
    theta_5=0;
    theta_6=1;
theta_7=0;
theta_8=1;
theta_9=0;
```

```
   theta_10=1;
   theta_11=0;
   theta_12=1;
    }
    if (choice==6)
    {
       theta_1=0;
       theta_2=1;
       theta_3=0;
       theta_4=1;
   theta_5=0;
   theta_6=1;
   theta_7=0;
   theta_8=1;
   theta_9=0;
   theta_10=1;
       theta_11=1;
       theta_12=0;
    }

//loop through all S and v values
for(i=0;i<a.rows();i++)
{
for(k=0;k<a.cols();k++)
{
//calculate alpha, etc.
alpha=0.5*v(k)*s(i+1)*s(i+1)*delta_t/(delta_s*delta_s);
beta=0.25*rho*sigma*v(k)*s(i+1)*delta_t/(delta_v*delta_s);
gamma=0.5*sigma*sigma*v(k)*delta_t/(delta_v*delta_v);
zeta=0.5*r*s(i+1)*delta_t/delta_s;
eta=0.5*(K*(theta-v(k))-lambda(s(i+1),v(k),delta_t*j))*delta_t/delta_v;

//calculate a, b, etc. using alpha, etc.
a(i,k)=-beta*theta_3;
b(i,k)=-alpha*theta_1+zeta*theta_7;
c(i,k)=beta*theta_3;
d(i,k)=-gamma*theta_5+eta*theta_9;
e(i,k)=1+2*alpha*theta_1+2*gamma*theta_5-mu*theta_11;
f(i,k)=-gamma*theta_5-eta*theta_9;
g(i,k)=beta*theta_3;
h(i,k)=-alpha*theta_1-zeta*theta_7;
l(i,k)=-beta*theta_3;
}
}
}
```

```
void define_rhs(Tmatrix& u_old, Tmatrix& u, Tmatrix& a, Tmatrix& b, Tmatrix& c, Tmatrix& d, Tmatrix&
{
int i;
int k;
double alpha;
double beta;
double gamma;
double zeta;
double eta;
double mu=-r*delta_t;
double a_;
double b_;
double c_;
double d_;
double e_;
double f_;
double g_;
double h_;
double l_;
double theta_1;
double theta_2;
double theta_3;
double theta_4;
double theta_5;
double theta_6;
double theta_7;
double theta_8;
double theta_9;
double theta_10;
double theta_11;
double theta_12;

    //set theta values based on user's choice of numerical scheme
    if (choice==1)
    {
       theta_1=0.5;
       theta_2=0.5;
       theta_3=0.5;
       theta_4=0.5;
       theta_5=0.5;
       theta_6=0.5;
   theta_7=0.5;
   theta_8=0.5;
   theta_9=0.5;
   theta_10=0.5;
   theta_11=0.5;
```

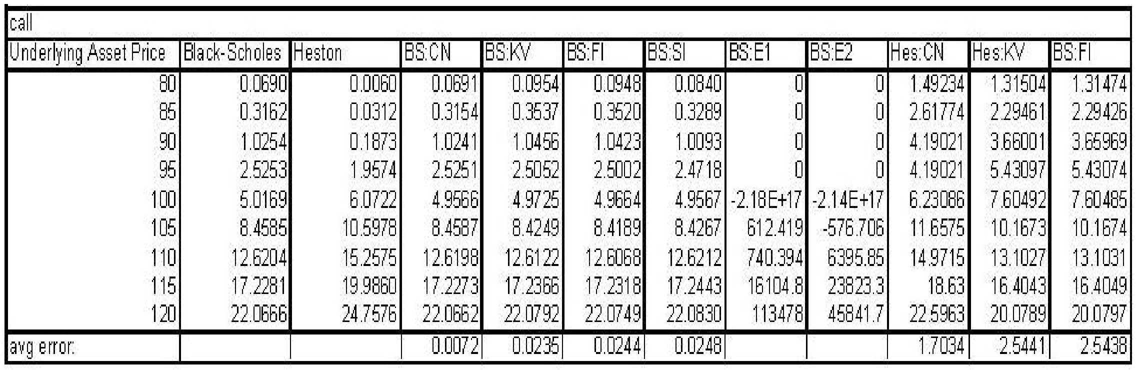
```
theta_12=0.5;
 }
 if (choice==2)
 {
    theta_1=1;
    theta_2=0;
    theta_3=1;
    theta_4=0;
theta_5=1;
theta_6=0;
theta_7=1;
theta_8=0;
theta_9=1;
theta_10=0;
    theta_11=0;
    theta_12=1;
 }
 if (choice==3)
 {
    theta_1=1;
    theta_2=0;
    theta_3=1;
    theta_4=0;
    theta_5=1;
    theta_6=0;
theta_7=1;
theta_8=0;
theta_9=1;
theta_10=0;
theta_11=1;
theta_12=0;
 }
 if (choice==4)
 {
    theta_1=1;
    theta_2=0;
    theta_3=1;
theta_4=0;
theta_5=1;
theta_6=0;
theta_7=0;
theta_8=1;
theta_9=0;
    theta_10=1;
    theta_11=1;
    theta_12=0;
    }
    if (choice==5)
    {
       theta_1=0;
       theta_2=1;
       theta_3=0;
       theta_4=1;
       theta_5=0;
       theta_6=1;
   theta_7=0;
   theta_8=1;
   theta_9=0;
   theta_10=1;
   theta_11=0;
   theta_12=1;
    }
    if (choice==6)
    {
       theta_1=0;
       theta_2=1;
       theta_3=0;
       theta_4=1;
   theta_5=0;
   theta_6=1;
   theta_7=0;
   theta_8=1;
   theta_9=0;
   theta_10=1;
       theta_11=1;
       theta_12=0;
    }

//loop through all S and v values
for(i=0;i<a.rows();i++)
{
for(k=0;k<a.cols();k++)
{
//calculate alpha, beta, etc.
alpha=0.5*v(k)*s(i+1)*s(i+1)*delta_t/(delta_s*delta_s);
beta=0.25*rho*sigma*v(k)*s(i+1)*delta_t/(delta_s*delta_v);
gamma=0.5*sigma*sigma*v(k)*delta_t/(delta_v*delta_v);
zeta=0.5*r*s(i+1)*delta_t/delta_s;
eta=0.5*(K*(theta-v(k))-lambda(s(i+1),v(k),delta_t*j))*delta_t/delta_v;

//use alpha, etc. to calculate a_, b_, etc.
a_=beta*theta_4;
```

```
b_=alpha*theta_2-zeta*theta_8;
c_=-beta*theta_4;
d_=gamma*theta_6-eta*theta_10;
e_=1-2*alpha*theta_2-2*gamma*theta_6+mu*theta_12;
f_=gamma*theta_6+eta*theta_10;
g_=-beta*theta_4;
h_=alpha*theta_2+zeta*theta_8;
l_=beta*theta_4;

//calculate entry for right hand side vector
if(k==0)
{
rhs(i,k)=a_*u_old(i,k+1)+b_*u_old(i,k)+c_*u_old(i,k+1)+d_*u_old(i+1,k+1)+e_*u_old(i+1,k)+f_*u_old(i+
if(i==0)
{
rhs(i,k)=rhs(i,k)-a(i,k)*u(i,k+1)-b(i,k)*u(i,k)-c(i,k)*u(i,k+1);
}
else if(i==e.rows()-1)
{
rhs(i,k)=rhs(i,k)-g(i,k)*u(i+2,k+1)-h(i,k)*u(i+2,k)-l(i,k)*u(i+2,k+1);
}
}
else if(k==e.cols()-1)
{
rhs(i,k)=a_*u_old(i,k-1)+b_*u_old(i,k)+c_*u_old(i,k-1)+d_*u_old(i+1,k-1)+e_*u_old(i+1,k)+f_*u_old(i+
if(i==0)
{
rhs(i,k)=rhs(i,k)-a(i,k)*u(i,k-1)-b(i,k)*u(i,k)-c(i,k)*u(i,k-1);
}
else if(i==e.rows()-1)
{
rhs(i,k)=rhs(i,k)-g(i,k)*u(i+2,k-1)-h(i,k)*u(i+2,k)-l(i,k)*u(i+2,k-1);
}
}
else
{
rhs(i,k)=a_*u_old(i,k-1)+b_*u_old(i,k)+c_*u_old(i,k+1)+d_*u_old(i+1,k-1)+e_*u_old(i+1,k)+f_*u_old(i+
if(i==0)
{
rhs(i,k)=rhs(i,k)-a(i,k)*u(i,k-1)-b(i,k)*u(i,k)-c(i,k)*u(i,k+1);
}
else if(i==e.rows()-1)
{
rhs(i,k)=rhs(i,k)-g(i,k)*u(i+2,k-1)-h(i,k)*u(i+2,k)-l(i,k)*u(i+2,k+1);
}
}
```

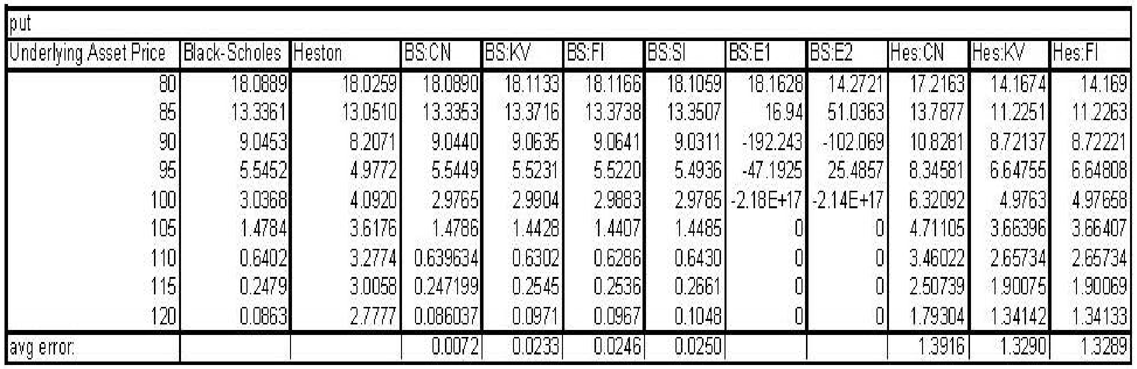
```
}
}
}

void solve_gs(Tmatrix& a,Tmatrix& b,Tmatrix& c,Tmatrix& d,Tmatrix& e,Tmatrix& f,Tmatrix& g,Tmatrix&
{
int i;
int k;
Tmatrix x;
double x_old;
double check;

x.resize(rhs.rows(),rhs.cols());
x=0;

//loop through iterations until error is within tolerance
do
{
//reset error check variable to 0
check=0;

    for(i=0;i<x.rows();i++)
{
for(k=0;k<x.cols();k++)
{
x_old=x(i,k);

//calculate next iteration based on Gauss-Seidel formula
if(k==0)
{
if(i==0)
{
x(i,k)=(1.0/e(i,k))*(rhs(i,k)-((f(i,k)+d(i,k))*x(i,k+1) + h(i,k)*x(i+1,k) + (l(i,k)+g(i,k))*x(i+1,k+
}
else if(i==x.rows()-1)
{
x(i,k)=(1.0/e(i,k))*(rhs(i,k)-(b(i,k)*x(i-1,k)+(c(i,k)+a(i,k))*x(i-1,k+1)+(f(i,k)+d(i,k))*x(i,k+1)))
}
else
{
x(i,k)=(1.0/e(i,k))*(rhs(i,k)-(b(i,k)*x(i-1,k)+h(i,k)*x(i+1,k)+(c(i,k)+a(i,k))*x(i-1,k+1)+(f(i,k)+d(
}
}
else if (k==x.cols()-1)
{
if(i==0)
```

```
{
x(i,k)=(1.0/e(i,k))*(rhs(i,k)-((f(i,k)+d(i,k))*x(i,k-1) + h(i,k)*x(i+1,k) + (l(i,k)+g(i,k))*x(i+1,k-
}
else if(i==x.rows()-1)
{
x(i,k)=(1.0/e(i,k))*(rhs(i,k)-(b(i,k)*x(i-1,k)+(c(i,k)+a(i,k))*x(i-1,k-1)+(f(i,k)+d(i,k))*x(i,k-1)))
}
else
{
x(i,k)=(1.0/e(i,k))*(rhs(i,k)-(b(i,k)*x(i-1,k)+h(i,k)*x(i+1,k)+(c(i,k)+a(i,k))*x(i-1,k-1)+(f(i,k)+d(
}
}
else
{
if(i==0)
{
x(i,k)=(1.0/e(i,k))*(rhs(i,k)-(f(i,k)*x(i,k+1)+d(i,k)*x(i,k-1) + h(i,k)*x(i+1,k) + l(i,k)*x(i+1,k+1)
}
else if(i==x.rows()-1)
{
x(i,k)=(1.0/e(i,k))*(rhs(i,k)-(b(i,k)*x(i-1,k)+c(i,k)*x(i-1,k+1)+a(i,k)*x(i-1,k-1)+f(i,k)*x(i,k+1)+c
}
else
{
x(i,k)=(1.0/e(i,k))*(rhs(i,k)-(b(i,k)*x(i-1,k)+h(i,k)*x(i+1,k)+c(i,k)*x(i-1,k+1)+a(i,k)*x(i-1,k-1)+f
}
}

check=max_d(check,fabs(x(i,k)-x_old));
}
}
}while(check>0.001);

    for(i=0;i<x.rows();i++)
{
for(k=0;k<x.cols();k++)
{
u(i+1,k)=x(i,k);
}
}
}

double lambda (double s, double v, double t)
{
//possible to channge value of lamba (price of volatility risk) for later use of program
return 0.0;
}
```

`NSDE_maths.h, NSDE_maths.cpp, NSDE_output.cpp, and NSDE_output.cpp same as before.`

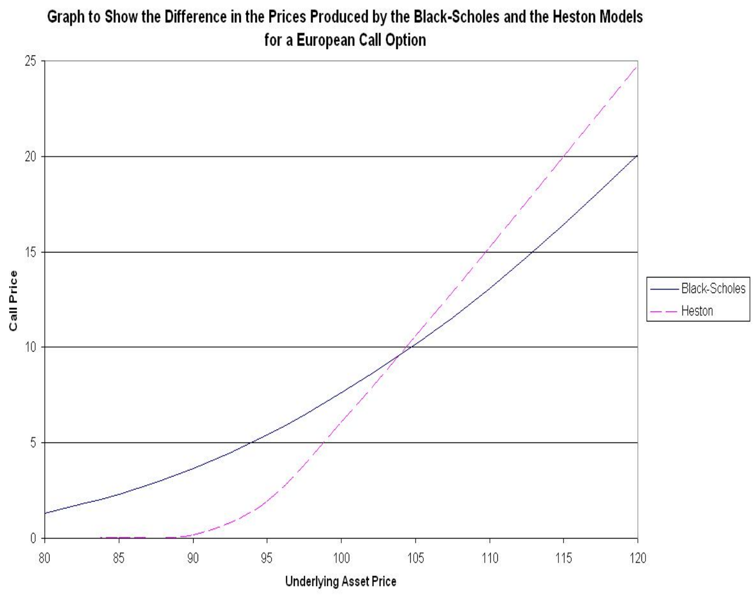


call											
Underlying Asset Price	Black-Scholes	Heston	BS:CN	BS:KV	BS:FI	BS:SI	BS:E1	BS:E2	Hes:CN	Hes:KV	BS:FI
80	0.0690	0.0060	0.0691	0.0954	0.0948	0.0840	0	0	1.49234	1.31504	1.31474
85	0.3162	0.0312	0.3154	0.3537	0.3520	0.3289	0	0	2.61774	2.29461	2.29426
90	1.0254	0.1873	1.0241	1.0456	1.0423	1.0093	0	0	4.19021	3.66001	3.65969
95	2.5253	1.9574	2.5251	2.5052	2.5002	2.4718	0	0	4.19021	5.43097	5.43074
100	5.0169	6.0722	4.9566	4.9725	4.9664	4.9567	-2.18E+17	-2.14E+17	6.23086	7.60492	7.60485
105	8.4585	10.5978	8.4587	8.4249	8.4189	8.4267	612.419	-576.706	11.6575	10.1673	10.1674
110	12.6204	15.2575	12.6198	12.6122	12.6068	12.6212	740.394	6395.85	14.9715	13.1027	13.1031
115	17.2281	19.9860	17.2273	17.2366	17.2318	17.2443	16104.8	23823.3	18.63	16.4043	16.4049
120	22.0666	24.7576	22.0662	22.0792	22.0749	22.0830	113478	45841.7	22.5963	20.0789	20.0797
avg error:			0.0072	0.0235	0.0244	0.0248			1.7034	2.5441	2.5438



put											
Underlying Asset Price	Black-Scholes	Heston	BS:CN	BS:KV	BS:FI	BS:SI	BS:E1	BS:E2	Hes:CN	Hes:KV	Hes:FI
80	18.0889	18.0259	18.0890	18.1133	18.1166	18.1059	18.1628	14.2721	17.2163	14.1674	14.169
85	13.3361	13.0510	13.3353	13.3716	13.3738	13.3507	16.94	51.0363	13.7877	11.2251	11.2263
90	9.0453	8.2071	9.0440	9.0635	9.0641	9.0311	-192.243	-102.069	10.8281	8.72137	8.72221
95	5.5452	4.9772	5.5449	5.5231	5.5220	5.4936	-47.1925	25.4857	8.34581	6.64755	6.64808
100	3.0368	4.0920	2.9765	2.9904	2.9883	2.9785	-2.18E+17	-2.14E+17	6.32092	4.9763	4.97658
105	1.4784	3.6176	1.4786	1.4428	1.4407	1.4485	0	0	4.71105	3.66396	3.66407
110	0.6402	3.2774	0.639634	0.6302	0.6286	0.6430	0	0	3.46022	2.65734	2.65734
115	0.2479	3.0058	0.247199	0.2545	0.2536	0.2661	0	0	2.50739	1.90075	1.90069
120	0.0863	2.7777	0.086037	0.0971	0.0967	0.1048	0	0	1.79304	1.34142	1.34133
avg error:			0.0072	0.0233	0.0246	0.0250			1.3916	1.3290	1.3289

Figure 3: $\delta S = 0.5$, $\delta v = 0.005$, $\delta \tau = 0.001$


Graph to Show the Difference in the Prices Produced by the Black-Scholes and the Heston Models for a European Call Option
Call Price
25
20
15
10
5
0
80
85
90
95
100
105
110
115
120
Underlying Asset Price
Black-Scholes
Heston

call									
Underlying Asset Price	Black-Scholes	Heston	BS:CN	BS:KV	BS:FI	BS:SI	Hes:CN	Hes:KV	BS:FI
80	0.0690	0.0060	0.0682	0.0923	0.0917	0.0814	0.951249	0.561847	0.561801
85	0.3162	0.0312	0.3154	0.3532	0.3515	0.3285	2.05261	1.24488	1.24481
90	1.0254	0.1873	1.0241	1.0455	1.0422	1.0092	3.44467	2.1736	2.17353
95	2.5253	1.9574	2.5251	2.5051	2.5001	2.4718	5.22378	3.46497	3.46494
100	5.0169	6.0722	4.9566	4.9723	4.9664	4.9566	7.458	5.24494	5.24497
105	8.4585	10.5978	8.4586	8.4244	8.4185	8.4263	10.1998	7.66175	7.6619
110	12.6204	15.2575	12.6193	12.6105	12.6055	12.6198	13.4957	10.8979	10.8981
115	17.2281	19.9860	17.2251	17.2313	17.2275	17.2395	17.394	15.1734	15.1738
120	22.0666	24.7576	22.0576	22.0644	22.0622	22.0686	21.9331	20.64	20.6402
avg error:			0.0085	0.0216	0.0234	0.0223	2.0503	2.4796	2.4795

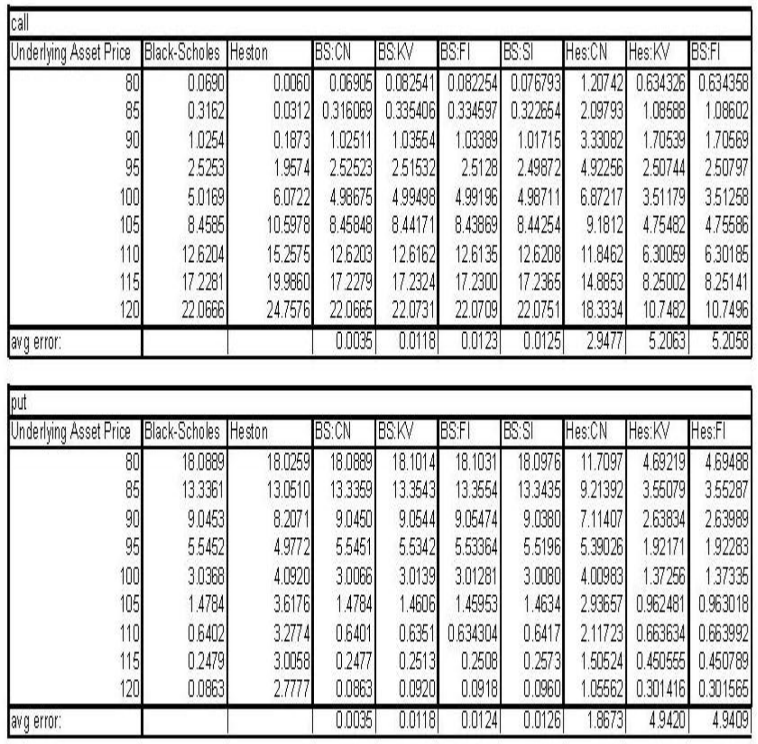
put									
Underlying Asset Price	Black-Scholes	Heston	BS:CN	BS:KV	BS:FI	BS:SI	Hes:CN	Hes:KV	Hes:FI
80	18.0889	18.0259	18.0881	18.1108	18.1130	18.1027	17.4167	15.4648	15.4641
85	13.3361	13.0510	13.3352	13.3712	13.3732	13.3501	13.0359	10.2231	10.2222
90	9.0453	8.2071	9.0440	9.0635	9.0640	9.0310	9.71492	6.86031	6.85933
95	5.5452	4.9772	5.5449	5.5231	5.5220	5.4936	7.17914	4.65398	4.6531
100	3.0368	4.0920	2.9765	2.9903	2.9882	2.9784	5.23039	3.16456	3.16382
105	1.4784	3.6176	1.4785	1.4424	1.4403	1.4481	3.71959	2.13057	2.12999
110	0.6402	3.2774	0.6392	0.6286	0.6271	0.6414	2.52911	1.38914	1.38872
115	0.2479	3.0058	0.2450	0.2497	0.2488	0.2609	1.56311	0.833294	0.833025
120	0.0863	2.7777	0.0775	0.0833	0.0829	0.0894	0.742203	0.388651	0.38852
avg error:			0.0085	0.0218	0.0230	0.0222	1.0890	1.7693	1.7699

Figure 4: Left hand S boundary at $S_0 = 75$ and right hand S boundary at $S_0 = 125$; left hand v boundary at $v = 0.15$ and right hand v boundary at $v = 0.25$

call									
Underlying Asset Price	Black-Scholes	Heston	BS:CN	BS:KV	BS:FI	BS:SI	Hes:CN	Hes:KV	Hes:FI
95	2.5253	1.9574	2.2129	2.1819	2.1803	2.1697	2.31161	1.58595	1.58615
100	5.0169	6.0722	4.8470	4.8133	4.8112	4.8060	4.90687	3.78149	3.78182
105	8.4585	10.5978	8.1717	8.1456	8.1441	8.1449	8.10857	7.23915	7.23944
avg error:			0.2564	0.2866	0.2884	0.2934	1.3363	2.0069	2.0067

put									
Underlying Asset Price	Black-Scholes	Heston	BS:CN	BS:KV	BS:FI	BS:SI	Hes:CN	Hes:KV	Hes:FI
95	5.5452	4.9772	5.2328	5.2012	5.2007	5.1901	5.11289	4.19523	4.19537
100	3.0368	4.0920	2.8669	2.8325	2.8317	2.8265	2.98174	1.98625	1.98635
105	1.4784	3.6176	1.1916	1.1650	1.1645	1.1653	1.37148	0.798538	0.798586
			0.2564	0.2873	0.2878	0.2928	1.1640	1.9023	1.9022

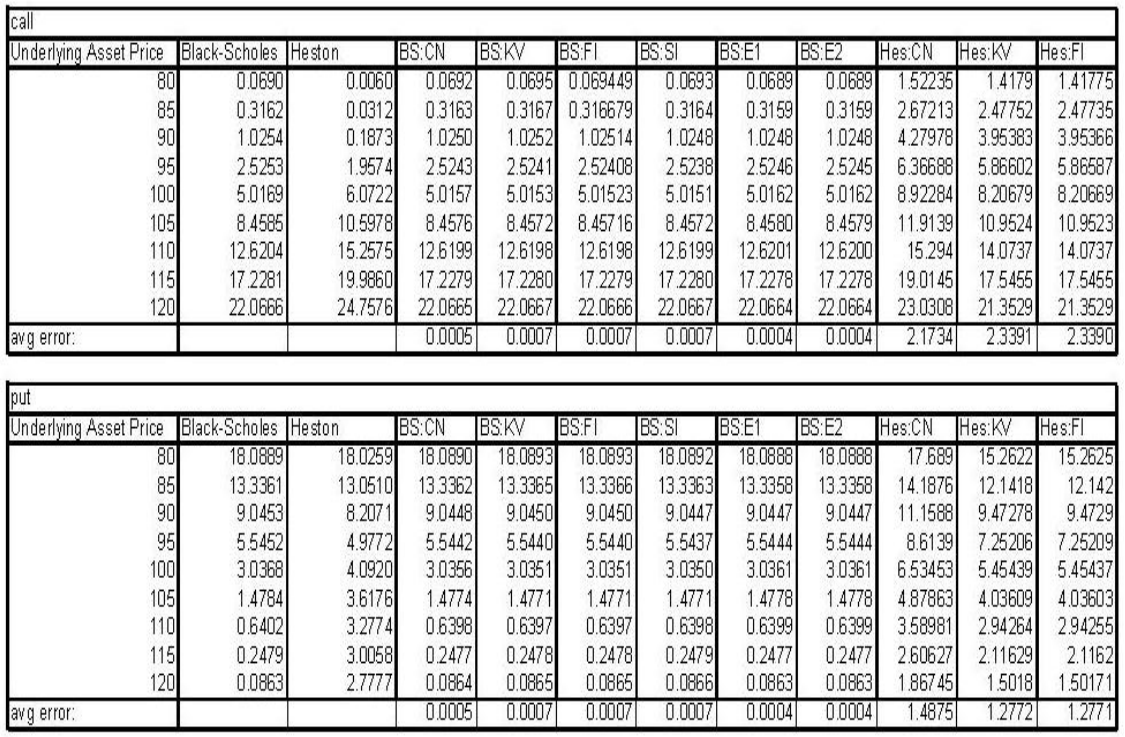
Figure 5: Left hand S boundary at $S_0 = 90$ and right hand S boundary at $S_0 = 110$; left hand v boundary at $v = 0.19$ and right hand v boundary at $v = 0.21$



call									
Underlying Asset Price	Black-Scholes	Heston	BS:CN	BS:KV	BS:FI	BS:SI	Hes:CN	Hes:KV	BS:FI
80	0.0690	0.0060	0.06905	0.082541	0.082254	0.076793	1.20742	0.634326	0.634358
85	0.3162	0.0312	0.316069	0.335406	0.334597	0.322654	2.09793	1.08588	1.08602
90	1.0254	0.1873	1.02511	1.03554	1.03389	1.01715	3.33082	1.70539	1.70569
95	2.5253	1.9574	2.52523	2.51532	2.5128	2.49872	4.92256	2.50744	2.50797
100	5.0169	6.0722	4.98675	4.99498	4.99196	4.98711	6.87217	3.51179	3.51258
105	8.4585	10.5978	8.45848	8.44171	8.43869	8.44254	9.1812	4.75482	4.75586
110	12.6204	15.2575	12.6203	12.6162	12.6135	12.6208	11.8462	6.30059	6.30185
115	17.2281	19.9860	17.2279	17.2324	17.2300	17.2365	14.8853	8.25002	8.25141
120	22.0666	24.7576	22.0665	22.0731	22.0709	22.0751	18.3334	10.7482	10.7496
avg error:			0.0035	0.0118	0.0123	0.0125	2.9477	5.2063	5.2058

put									
Underlying Asset Price	Black-Scholes	Heston	BS:CN	BS:KV	BS:FI	BS:SI	Hes:CN	Hes:KV	Hes:FI
80	18.0889	18.0259	18.0889	18.1014	18.1031	18.0976	11.7097	4.69219	4.69488
85	13.3361	13.0510	13.3359	13.3543	13.3554	13.3435	9.21392	3.55079	3.55287
90	9.0453	8.2071	9.0450	9.0544	9.05474	9.0380	7.11407	2.63834	2.63989
95	5.5452	4.9772	5.5451	5.5342	5.53364	5.5196	5.39026	1.92171	1.92283
100	3.0368	4.0920	3.0066	3.0139	3.01281	3.0080	4.00983	1.37256	1.37335
105	1.4784	3.6176	1.4784	1.4606	1.45953	1.4634	2.93657	0.962481	0.963018
110	0.6402	3.2774	0.6401	0.6351	0.634304	0.6417	2.11723	0.663634	0.663992
115	0.2479	3.0058	0.2477	0.2513	0.2508	0.2573	1.50524	0.450555	0.450789
120	0.0863	2.7777	0.0863	0.0920	0.0918	0.0960	1.05562	0.301416	0.301565
avg error:			0.0035	0.0118	0.0124	0.0126	1.8673	4.9420	4.9409

Figure 6: $\delta = 0.25$, $\delta v = 0.0025$, $\delta \tau = 0.01$

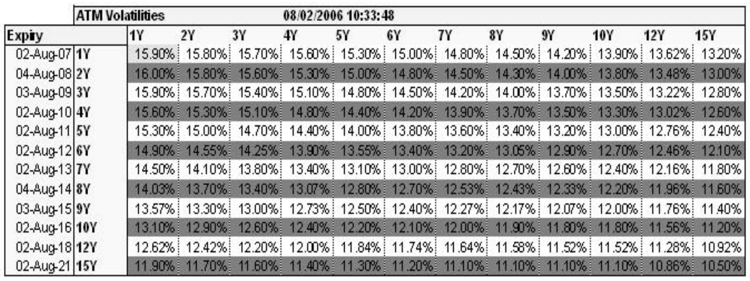


call											
Underlying Asset Price	Black-Scholes	Heston	BS:CN	BS:KV	BS:FI	BS:SI	BS:E1	BS:E2	Hes:CN	Hes:KV	Hes:FI
80	0.0690	0.0060	0.0692	0.0695	0.069449	0.0693	0.0689	0.0689	1.52235	1.4179	1.41775
85	0.3162	0.0312	0.3163	0.3167	0.316679	0.3164	0.3159	0.3159	2.67213	2.47752	2.47735
90	1.0254	0.1873	1.0250	1.0252	1.02514	1.0248	1.0248	1.0248	4.27978	3.95383	3.95366
95	2.5253	1.9574	2.5243	2.5241	2.52408	2.5238	2.5246	2.5245	6.36688	5.86602	5.86587
100	5.0169	6.0722	5.0157	5.0153	5.01523	5.0151	5.0162	5.0162	8.92284	8.20679	8.20669
105	8.4585	10.5978	8.4576	8.4572	8.45716	8.4572	8.4580	8.4579	11.9139	10.9524	10.9523
110	12.6204	15.2575	12.6199	12.6198	12.6198	12.6199	12.6201	12.6200	15.294	14.0737	14.0737
115	17.2281	19.9860	17.2279	17.2280	17.2279	17.2280	17.2278	17.2278	19.0145	17.5455	17.5455
120	22.0666	24.7576	22.0665	22.0667	22.0666	22.0667	22.0664	22.0664	23.0308	21.3529	21.3529
avg error:			0.0005	0.0007	0.0007	0.0007	0.0004	0.0004	2.1734	2.3391	2.3390

put											
Underlying Asset Price	Black-Scholes	Heston	BS:CN	BS:KV	BS:FI	BS:SI	BS:E1	BS:E2	Hes:CN	Hes:KV	Hes:FI
80	18.0889	18.0259	18.0890	18.0893	18.0893	18.0892	18.0888	18.0888	17.689	15.2622	15.2625
85	13.3361	13.0510	13.3362	13.3365	13.3366	13.3363	13.3358	13.3358	14.1876	12.1418	12.142
90	9.0453	8.2071	9.0448	9.0450	9.0450	9.0447	9.0447	9.0447	11.1588	9.47278	9.4729
95	5.5452	4.9772	5.5442	5.5440	5.5440	5.5437	5.5444	5.5444	8.6139	7.25206	7.25209
100	3.0368	4.0920	3.0356	3.0351	3.0351	3.0350	3.0361	3.0361	6.53453	5.45439	5.45437
105	1.4784	3.6176	1.4774	1.4771	1.4771	1.4771	1.4778	1.4778	4.87863	4.03609	4.03603
110	0.6402	3.2774	0.6398	0.6397	0.6397	0.6398	0.6399	0.6399	3.58981	2.94264	2.94255
115	0.2479	3.0058	0.2477	0.2478	0.2478	0.2479	0.2477	0.2477	2.60627	2.11629	2.1162
120	0.0863	2.7777	0.0864	0.0865	0.0865	0.0866	0.0863	0.0863	1.86745	1.5018	1.50171
avg error:			0.0005	0.0007	0.0007	0.0007	0.0004	0.0004	1.4875	1.2772	1.2771

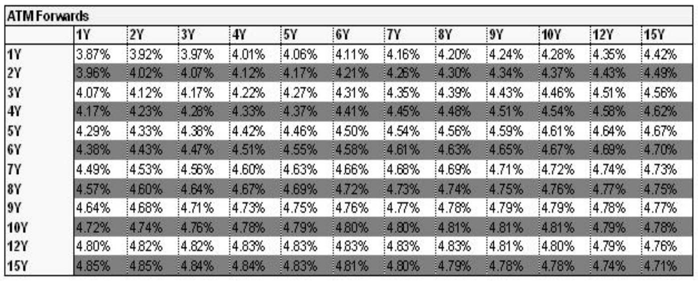
Figure 7: $\delta = 0.5$, $\delta v = 0.005$, $\delta\tau = 0.001$

Market Volatilities for European Swaptions [8]



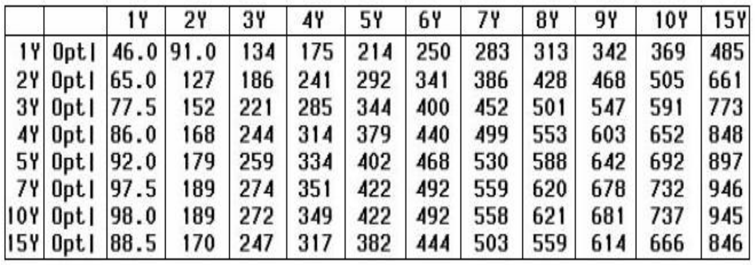
	ATM Volatilities				08/02/2006 10:33:48								
Expiry		1Y	2Y	3Y	4Y	5Y	6Y	7Y	8Y	9Y	10Y	12Y	15Y
02-Aug-07	1Y	15.90%	15.80%	15.70%	15.60%	15.30%	15.00%	14.80%	14.50%	14.20%	13.90%	13.62%	13.20%
04-Aug-08	2Y	16.00%	15.80%	15.60%	15.30%	15.00%	14.80%	14.50%	14.30%	14.00%	13.80%	13.48%	13.00%
03-Aug-09	3Y	15.90%	15.70%	15.40%	15.10%	14.80%	14.50%	14.20%	14.00%	13.70%	13.50%	13.22%	12.80%
02-Aug-10	4Y	15.60%	15.30%	15.10%	14.80%	14.40%	14.20%	13.90%	13.70%	13.50%	13.30%	13.02%	12.60%
02-Aug-11	5Y	15.30%	15.00%	14.70%	14.40%	14.00%	13.80%	13.60%	13.40%	13.20%	13.00%	12.76%	12.40%
02-Aug-12	6Y	14.90%	14.55%	14.25%	13.90%	13.55%	13.40%	13.20%	13.05%	12.90%	12.70%	12.46%	12.10%
02-Aug-13	7Y	14.50%	14.10%	13.80%	13.40%	13.10%	13.00%	12.80%	12.70%	12.60%	12.40%	12.16%	11.80%
04-Aug-14	8Y	14.03%	13.70%	13.40%	13.07%	12.80%	12.70%	12.53%	12.43%	12.33%	12.20%	11.96%	11.60%
03-Aug-15	9Y	13.57%	13.30%	13.00%	12.73%	12.50%	12.40%	12.27%	12.17%	12.07%	12.00%	11.76%	11.40%
02-Aug-16	10Y	13.10%	12.90%	12.60%	12.40%	12.20%	12.10%	12.00%	11.90%	11.80%	11.80%	11.56%	11.20%
02-Aug-18	12Y	12.62%	12.42%	12.20%	12.00%	11.84%	11.74%	11.64%	11.58%	11.52%	11.52%	11.28%	10.92%
02-Aug-21	15Y	11.90%	11.70%	11.60%	11.40%	11.30%	11.20%	11.10%	11.10%	11.10%	11.10%	10.86%	10.50%

Market Forward Rates [8]

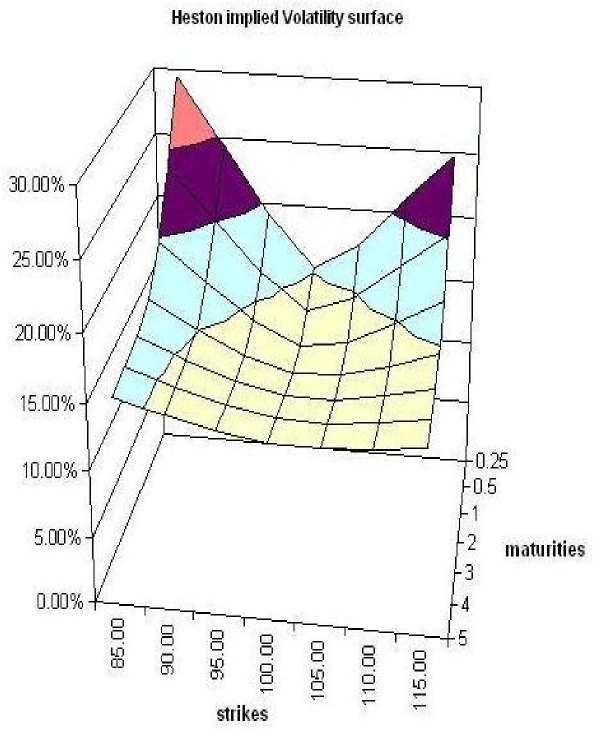


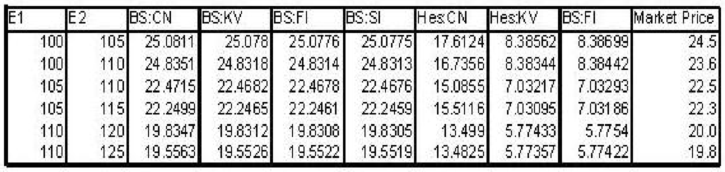
ATM Forwards												
	1Y	2Y	3Y	4Y	5Y	6Y	7Y	8Y	9Y	10Y	12Y	15Y
1Y	3.87%	3.92%	3.97%	4.01%	4.06%	4.11%	4.16%	4.20%	4.24%	4.28%	4.35%	4.42%
2Y	3.96%	4.02%	4.07%	4.12%	4.17%	4.21%	4.26%	4.30%	4.34%	4.37%	4.43%	4.49%
3Y	4.07%	4.12%	4.17%	4.22%	4.27%	4.31%	4.35%	4.39%	4.43%	4.46%	4.51%	4.56%
4Y	4.17%	4.23%	4.28%	4.33%	4.37%	4.41%	4.45%	4.48%	4.51%	4.54%	4.58%	4.62%
5Y	4.29%	4.33%	4.38%	4.42%	4.46%	4.50%	4.54%	4.56%	4.59%	4.61%	4.64%	4.67%
6Y	4.38%	4.43%	4.47%	4.51%	4.55%	4.58%	4.61%	4.63%	4.65%	4.67%	4.69%	4.70%
7Y	4.49%	4.53%	4.56%	4.60%	4.63%	4.66%	4.68%	4.69%	4.71%	4.72%	4.74%	4.73%
8Y	4.57%	4.60%	4.64%	4.67%	4.69%	4.72%	4.73%	4.74%	4.75%	4.76%	4.77%	4.75%
9Y	4.64%	4.68%	4.71%	4.73%	4.75%	4.76%	4.77%	4.78%	4.79%	4.79%	4.78%	4.77%
10Y	4.72%	4.74%	4.76%	4.78%	4.79%	4.80%	4.80%	4.81%	4.81%	4.81%	4.79%	4.78%
12Y	4.80%	4.82%	4.82%	4.83%	4.83%	4.83%	4.83%	4.83%	4.81%	4.80%	4.79%	4.76%
15Y	4.85%	4.85%	4.84%	4.84%	4.83%	4.81%	4.80%	4.79%	4.78%	4.78%	4.74%	4.71%

Market Prices for European Straddle Swaptions [8]



		1Y	2Y	3Y	4Y	5Y	6Y	7Y	8Y	9Y	10Y	15Y
1Y	Opt	46.0	91.0	134	175	214	250	283	313	342	369	485
2Y	Opt	65.0	127	186	241	292	341	386	428	468	505	661
3Y	Opt	77.5	152	221	285	344	400	452	501	547	591	773
4Y	Opt	86.0	168	244	314	379	440	499	553	603	652	848
5Y	Opt	92.0	179	259	334	402	468	530	588	642	692	897
7Y	Opt	97.5	189	274	351	422	492	559	620	678	732	946
10Y	Opt	98.0	189	272	349	422	492	558	621	681	737	945
15Y	Opt	88.5	170	247	317	382	444	503	559	614	666	846





E1	E2	BS:CN	BS:KV	BS:FI	BS:SI	Hes:CN	Hes:KV	BS:FI	Market Price
100	105	25.0811	25.078	25.0776	25.0775	17.6124	8.38562	8.38699	24.5
100	110	24.8351	24.8318	24.8314	24.8313	16.7356	8.38344	8.38442	23.6
105	110	22.4715	22.4682	22.4678	22.4676	15.0855	7.03217	7.03293	22.5
105	115	22.2499	22.2465	22.2461	22.2459	15.5116	7.03095	7.03186	22.3
110	120	19.8347	19.8312	19.8308	19.8305	13.499	5.77433	5.7754	20.0
110	125	19.5563	19.5526	19.5522	19.5519	13.4825	5.77357	5.77422	19.8

Prices for knock-in Caps