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

Contents

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

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 payo 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 di erent 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 payo 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 at4(t.)-5i420(an)19-117(t)4T48rid 1rit Πiexp.eri0(e)-1(xp.)gldyt

This leads to what is known as the 'law of one price': if there exists two securities, both with the same payo, 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 payo 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-

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

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 $\tau = 0$.

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

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 payo 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 payo 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

$$D(t_i) = e^{-r}$$

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 payo 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

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

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

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

$$\mathbf{x} = \begin{bmatrix} a_1U_0^j + b_1U_1^j + c_1U_2^j - a_1U_0^{j+1} \\ a_2U_1^j + b_2U_2^j + c_2U_3^j \\ a_3U_2^j + b_3U_3^j + c_3U_4^j \\ \vdots \\ a_{N-2}U_{N-3}^j + b_{N-2}U_{N-2}^j + c_N{}^j \end{bmatrix}$$

By the definitions given in (32), it can be seen that $\alpha_i, \beta_i > 0$ 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

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

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

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

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

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

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

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

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

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

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

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

$$\alpha^k_i = \frac{1}{}$$

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

$\zeta_i^k \quad \zeta_i^k \quad \eta_i^k \quad \zeta_i^k \quad \zeta_i^k \zeta_i \quad \eta_i^k \quad \zeta_i$ $\beta\gamma\beta_{\beta\gamma}\ \beta\gamma$

b^0

$\underline{x} =$

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

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

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

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

for $k \quad \{1, 2, \dots M - 1\}$,

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

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

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

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

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

$\Phi^{j}_{i,k}$

4.5 Stability

Similarly to before, we shall use Fourier stability for this scheme. Let $U_{i,k}^{j} = \Lambda^{j} e^{-zni\delta S} e^{-zmk\delta v}$ where $z = \sqrt{-1}$ and n, m

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

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

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

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

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

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

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

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

The random

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 $U^j_{i,k}$ 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 nu-

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 di□erence 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

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

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:

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

$$R_{i,j} =$$

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

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

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 Di□erences 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;
```

```
 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;
```

```
   //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
```

```
}
```

```
//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);
```

```
     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
```

```
    //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;
```

```
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;
```

```
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;
```

```
}
}
}

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)+d
}
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.

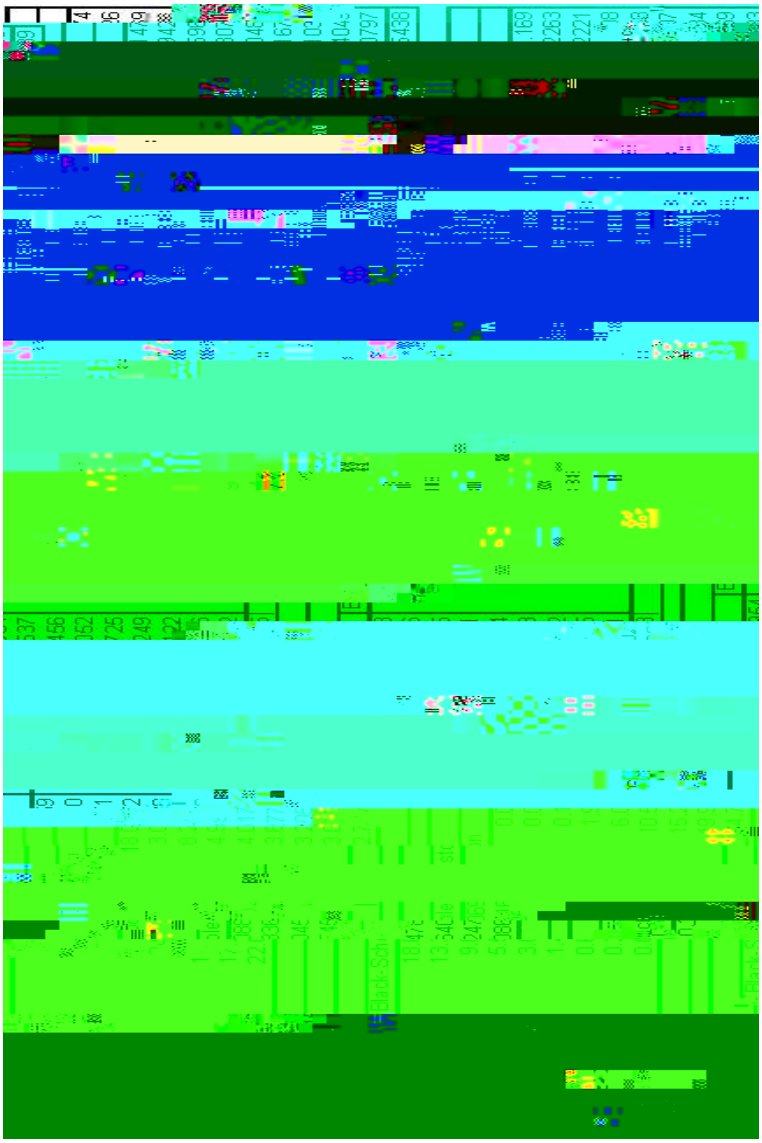
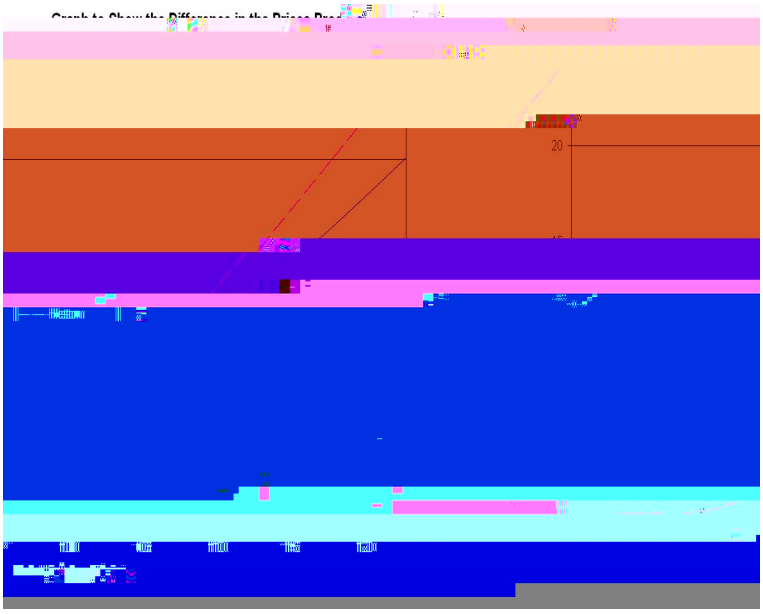


Figure 3: $\delta S = 0.5$, $\delta v = 0.005$, $\delta \tau = 0.001$


20

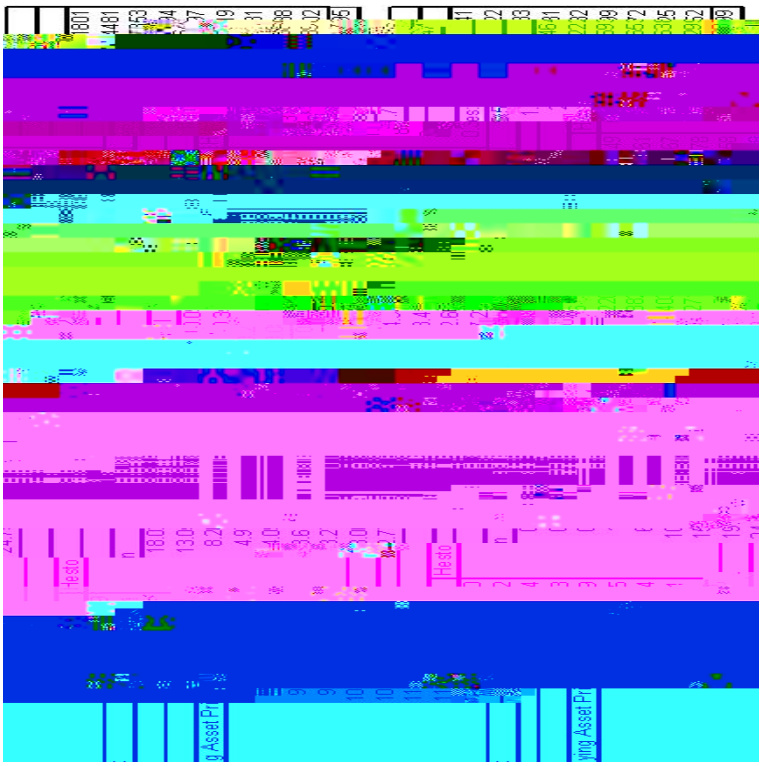


Figure 4: Left hand S boundary at $S_0 = 75$ and right hand S boundary at $S_0 = 125$; left hand v boundary at $v = 0.15$ and right hand v boundary at $v = 0.25$

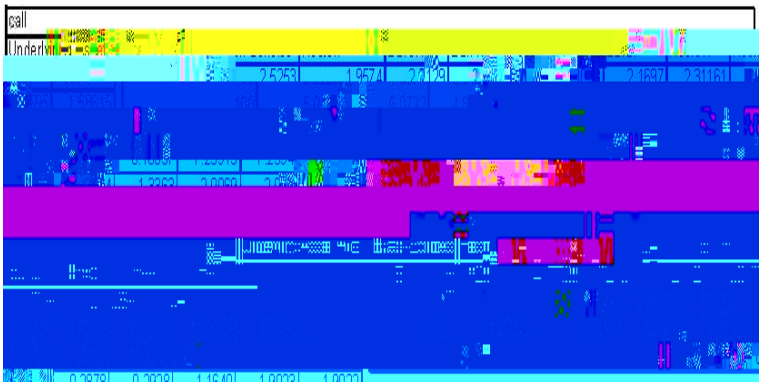


Figure 5: Left hand S boundary at $S_0 = 90$ and right hand S boundary at $S_0 = 110$; left hand v boundary at $v = 0.19$ and right hand v boundary at $v = 0.21$

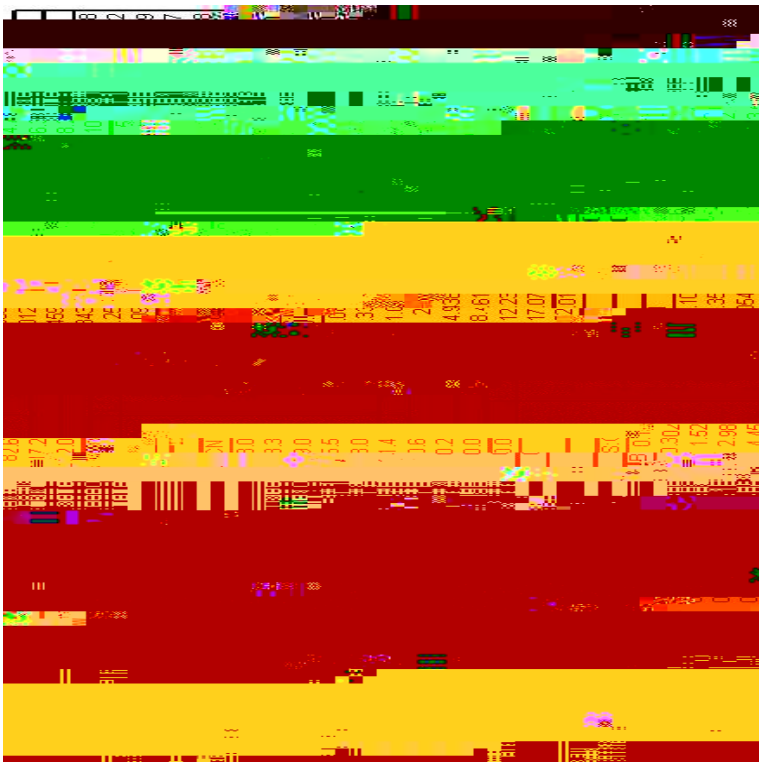


Figure 6: $\delta = 0.25$, $\delta v = 0.0025$, $\delta \tau = 0.01$

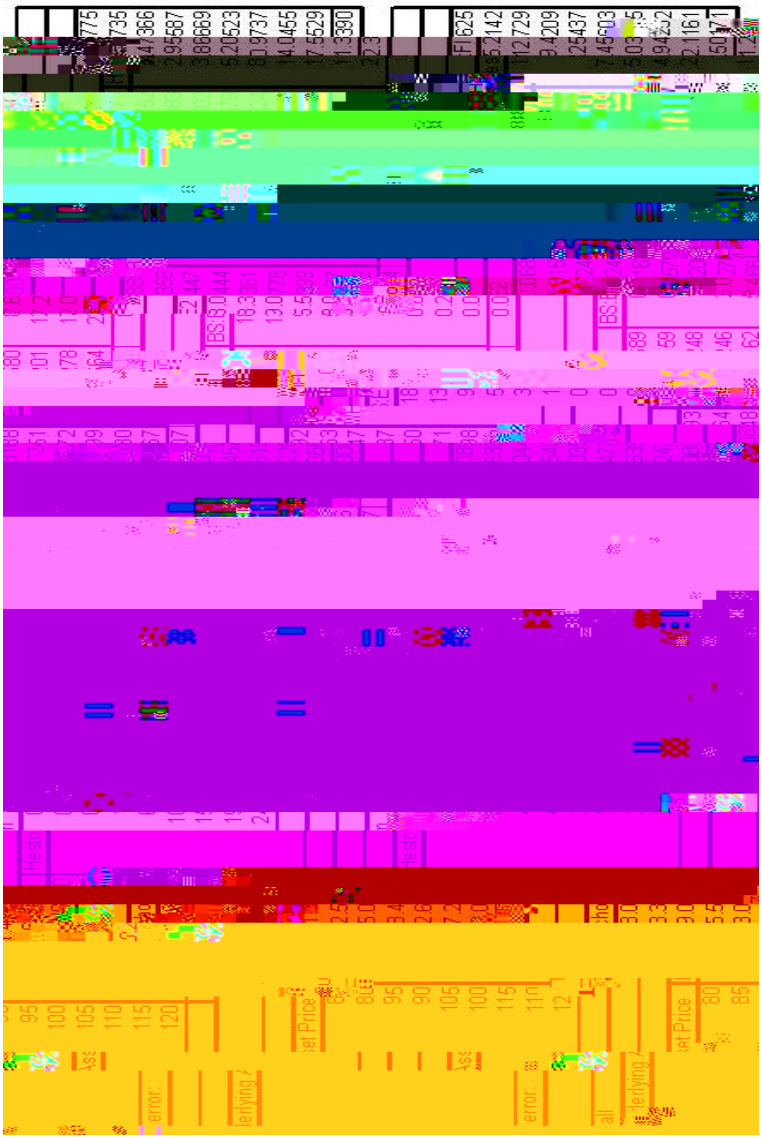
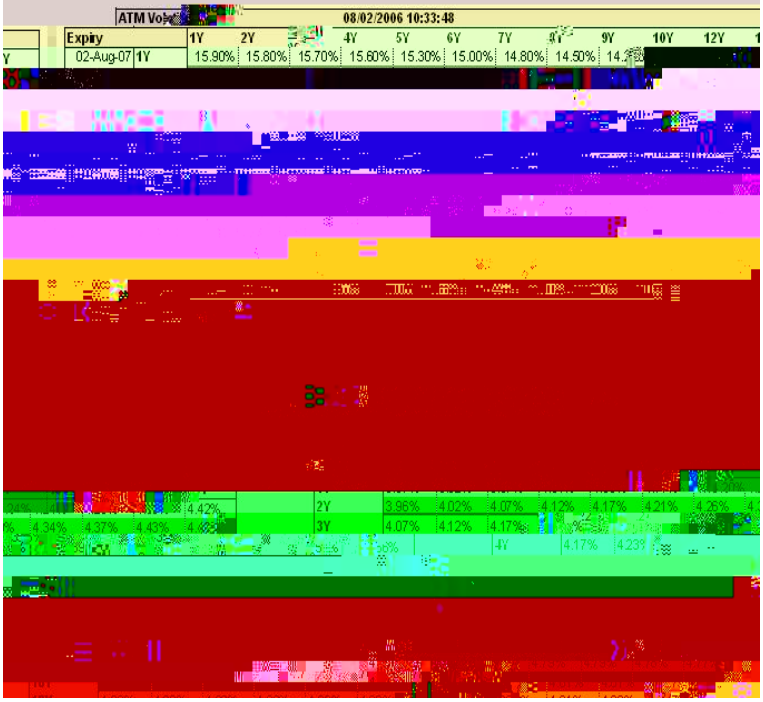


Figure 7: $\delta = 0.5$, $\delta v = 0.005$, $\delta \tau = 0.001$

Market Volatilities for European Swaptions [81]



Market Prices for European Straddle Swaptions [82]