ABSTRACT

Heston stochastic volatility model has been widely used in financial derivative pricing and risk management. One of the reasons for that is that vanilla options in Heston model have close form solutions. This makes the calibration of the model computationally much efficient and accurate. To understand these closed form formulas, this essay in the first half will introduce the Fourier transform method for option pricing and its application to the Heston model. Additionally, a characteristic function based method is also discussed, which extends the Heston model to have piecewise time-dependent parameters.

Finite element method (FEM) has been developed for decades to solve partial differential equations arise from science and engineering problems. It is well known for requiring a low order storage and for its capability to handle complicated irregular computational domains comparing to finite difference method (FDM). This characteristic advantage makes FEM an ideal numerical method for valuation of exotic options. This essay in the second half will present a detailed implementation of FEM and its application to pricing a double barrier knock-out option with an underlying asset modeled by Heston stochastic volatility process.
# TABLE OF CONTENTS

Abstract ..........................................................................................................................................................2  
Table of Contents .........................................................................................................................................3  
1. Kolmogorov Forward and Backward Equations ..................................................................................5  
   1.1. Kolmogorov Forward Equation ....................................................................................................5  
   1.2. Kolmogorov Backward Equation ..................................................................................................8  
2. Heston Model .....................................................................................................................................10  
   2.1. Heston Stochastic Volatility Model .............................................................................................10  
      2.1.1. Market Price of Risk ....................................................................................................... 10  
      2.1.2. Radon-Nikodym Derivative ............................................................................................ 15  
      2.1.3. Feller Condition............................................................................................................... 16  
   2.2. Probability Distribution of Spot Returns .....................................................................................16  
      2.2.1. Derivation of the Transition Probability .......................................................................... 17  
      2.2.2. Moment Generating Function (in progress…) ................................................................ 20  
   2.3. Analytical Solution of Vanilla Options .......................................................................................21  
      2.3.1. Fourier Transform and Characteristic Function .............................................................. 21  
      2.3.2. Characteristic Function ................................................................................................... 25  
      2.3.3. Vanilla Option Prices....................................................................................................... 29  
         2.3.3.1. Analogy to Cumulative Density Function ....................................................................... 29  
         2.3.3.2. Analogy to Probability Density Function ........................................................................ 31  
         2.3.3.3. Heston’s Original Solution .............................................................................................. 32  
   2.4. Piecewise Time Dependent Heston Model ..................................................................................35  
3. Heston Model: PDE by Finite Element Method ................................................................................38  
   3.1. The Partial Differential Equation ................................................................................................39  
   3.2. Numerical Solution of the PDE ...................................................................................................42  
      3.2.1. Temporal Discretization ..................................................................................................... 42  
      3.2.2. Two-Dimensional Finite Element Method ........................................................................ 43  
         3.2.2.1. Weak Formulation ........................................................................................................ 43  
         3.2.2.2. Boundary Conditions ................................................................................................ 44  
         3.2.2.3. Mesh and Basis Functions ........................................................................................... 46  
         3.2.2.4. Affine Transformation .................................................................................................. 48  
         3.2.2.5. Stiffness and Mass Matrix ............................................................................................ 48  
         3.2.2.6. Computation of Integrals ............................................................................................ 51  
   3.2.3. Iterative Linear Solvers ................................................................................................... 53  
      3.2.3.1. Implementation of Sparse Matrix .................................................................................... 53  
      3.2.3.2. Iterative Methods for Asymmetric Sparse Matrix........................................................... 54
3.2.4. Interpolation of Numerical Solution ................................................................. 54
3.3. Solutions and Discussions .................................................................................. 55
  3.3.1. Vanilla Call ................................................................................................. 55
  3.3.2. Vanilla Put ................................................................................................. 58
  3.3.3. Double Barrier Knock-out Call ................................................................. 59
  3.3.4. Double Barrier Knock-out Put ................................................................. 61
  3.3.5. Numerical Solutions of Double Barrier Knock-out Options ................... 63
3.4. Conclusions ....................................................................................................... 64
References .............................................................................................................. 65
This note is to summarize Heston stochastic volatility model. It was started as my degree essay for the M.S. in Mathematical Finance, which only covered the barrier option pricing by solving PDE using finite element method, i.e. the Chapter 3 in this note. I have been continuously expanding it with more mathematical background, such as the derivation of market price of spot/volatility risk, the Fourier transform method for option pricing, the derivation of characteristic function of the joint spot-variance process, the probability distribution of spot return, the piecewise time dependent Heston parameters, etc.

At first, let’s start from the Kolmogorov forward and backward equations, which fundamentally govern the transition probability density function of a diffusion process.

1. Kolmogorov Forward and Backward Equations

The time evolution of the transition probability density function is governed by Kolmogorov forward and backward equations, which will be introduced as follows, without loss of generality, in multi-dimension.

1.1. Kolmogorov Forward Equation

Let’s consider the following \( m \)-dimensional stochastic spot process \( S_t \in \mathbb{R}^m \) driven by an \( n \)-dimensional Brownian motion \( W_t \) whose correlation matrix \( \rho \) is given by

\[
\rho \rho \rho \rho = \rho W_t \rho \rho \rho \rho = A(t, S_t) dS_t + B(t, S_t) dW_t
\]

where

\[
\begin{align*}
\mathbb{J} h &= \frac{\partial h}{\partial S_i} \\
\mathbb{H} h &= \frac{\partial^2 h}{\partial S_i \partial S_j}
\end{align*}
\]

Expanding the expression in (2), we have
\[
dh = \sum_{i=1}^{m} \frac{\partial h}{\partial S_i} A_i dt + \sum_{i=1}^{m} \frac{\partial h}{\partial S_i} \sum_{k=1}^{n} B_{ik} dW_k + \frac{1}{2} \sum_{i,j=1}^{m} \frac{\partial^2 h}{\partial S_i \partial S_j} \sum_{k=1}^{n} B_{ik} \rho_{ij} B_{jk} dt
\]

\[
= \left( \sum_{i=1}^{m} \frac{\partial h}{\partial S_i} A_i + \frac{1}{2} \sum_{i,j=1}^{m} \frac{\partial^2 h}{\partial S_i \partial S_j} \Sigma_{ij} \right) dt + \sum_{i=1}^{m} \frac{\partial h}{\partial S_i} \sum_{k=1}^{n} B_{ik} dW_k
\]

where \( \Sigma = B \rho B' \) is the \( m \times m \) instantaneous variance-covariance matrix of \( dS \). Integrating on both sides of (4) from \( t \) to \( T \), we have

\[
h(S_T) - h(S_t) = \int_t^T \left( \sum_{i=1}^{m} \frac{\partial h}{\partial S_i} A_i + \frac{1}{2} \sum_{i,j=1}^{m} \frac{\partial^2 h}{\partial S_i \partial S_j} \Sigma_{ij} \right) du + \int_t^T \sum_{i=1}^{m} \frac{\partial h}{\partial S_i} \sum_{k=1}^{n} B_{ik} dW_k
\]

Taking expectation on both sides of (5), we get

\[
\text{LHS} = \mathbb{E}_t [h(S_T)] - h(S_t) = \int_{\Omega} h_y p_{T,y|t,x} dy - h_x
\]

\[
\text{RHS} = \mathbb{E}_t \left[ \int_t^T \left( \sum_{i=1}^{m} \frac{\partial h}{\partial S_i} A_i + \frac{1}{2} \sum_{i,j=1}^{m} \frac{\partial^2 h}{\partial S_i \partial S_j} \Sigma_{ij} \right) du \right] + \mathbb{E}_t \left[ \int_t^T \sum_{i=1}^{m} \frac{\partial h}{\partial S_i} \sum_{k=1}^{n} B_{ik} dW_k \right]
\]

\[
= \int_t^T \sum_{i=1}^{m} \mathbb{E}_t \left[ \frac{\partial h}{\partial S_i} A_i \right] du + \frac{1}{2} \int_t^T \sum_{i,j=1}^{m} \mathbb{E}_t \left[ \frac{\partial^2 h}{\partial S_i \partial S_j} \Sigma_{ij} \right] du
\]

where \( p_{T,y|t,x} \) is the transition probability density having \( S_T = y \) at \( T \) given \( S_t = x \) at \( t \) (i.e. if we solve the equation (1) with the initial condition \( S_t = x \in \mathbb{R}^m \), then the random variable \( S_T = y \in \Omega \) has a density \( p_{T,y|t,x} \) in the \( y \) variable at time \( T \)). Differentiating (6) with respect to \( T \) on both sides, we have

\[
\int_{\Omega} h_y \frac{\partial p_{T,y|t,x}}{\partial T} dy = \sum_{i=1}^{m} \mathbb{E}_t \left[ \frac{\partial h}{\partial S_i} A_i \right] + \frac{1}{2} \sum_{i,j=1}^{m} \mathbb{E}_t \left[ \frac{\partial^2 h}{\partial S_i \partial S_j} \Sigma_{ij} \right]
\]

\[
= \sum_{i=1}^{m} \int_{\Omega} \frac{\partial h_y}{\partial y_i} A_i p_{T,y|t,x} dy + \frac{1}{2} \sum_{i,j=1}^{m} \int_{\Omega} \frac{\partial^2 h_y}{\partial y_i \partial y_j} \Sigma_{ij} p_{T,y|t,x} dy
\]
If we assume $\Omega \equiv \mathbb{R}^m$ and also assume the probability density $p$ and its first derivatives $\partial p/\partial y_i$ vanish at a higher order of rate than $h$ and $\partial h/\partial y_i$ as $y_i \to \pm \infty \forall i = 1, \ldots, m$, then we can integrate by parts for the right hand side of (7), once for the first integral and twice for the second

$$
\int_{\Omega} \frac{h_y}{\partial y_i} A_i p dy = \int_{\Omega} \frac{h_y A_i p}{y_i = -\infty} d y_i - \int_{\Omega} \frac{h_y}{\partial y_i} (A_i p) dy \quad \text{and}
$$

$$
\int_{\Omega} \frac{\partial^2 h_y}{\partial y_i \partial y_j} \Sigma_{ij} p dy = \int_{\Omega} \frac{\partial h_y}{\partial y_j} (\Sigma_{ij} p) \Bigg|_{y_i = -\infty} d y_i - \int_{\Omega} \frac{\partial h_y}{\partial y_j} (\Sigma_{ij} p) d y_j
$$

$$
= -\int_{\Omega} \frac{h_y}{\partial y_i} (\Sigma_{ij} p) \Bigg|_{y_j = -\infty} d y_j + \int_{\Omega} \frac{h_y}{\partial y_i \partial y_j} (\Sigma_{ij} p) d y_j
$$

where $\int_{\Omega} (\cdot) d y_i = \int_{\mathbb{R}} \cdots \int_{\mathbb{R}} \cdots \int_{\mathbb{R}} (\cdot) dy_1 \cdots dy_{i-1} dy_{i+1} \cdots dy_m$

Plugging the results of (8) into (7), we have

$$
\int_{\Omega} \frac{h_y}{\partial T} d y = -\sum_{i=1}^m \int_{\Omega} \frac{h_y}{\partial y_i} (A_i p) d y + \frac{1}{2} \sum_{i=1}^m \int_{\Omega} \frac{h_y}{\partial y_i} (\Sigma_{ij} p) d y_j
$$

$$
\Rightarrow \int_{\Omega} h_y \left( \frac{\partial p}{\partial T} + \sum_{i=1}^m \frac{\partial (A_i p)}{\partial y_i} - \frac{1}{2} \sum_{i=1}^m \frac{\partial^2 (\Sigma_{ij} p)}{\partial y_i \partial y_j} \right) d y = 0
$$

(9)

By the arbitrariness of $h$, we conclude that for any $y \in \Omega$

$$
\frac{\partial p}{\partial T} + \sum_{i=1}^m \frac{\partial (A_i p)}{\partial y_i} - \frac{1}{2} \sum_{i=1}^m \frac{\partial^2 (\Sigma_{ij} p)}{\partial y_i \partial y_j} = 0, \quad \Sigma = B \rho B' \quad \text{(10)}
$$

This is the Multi-dimensional Fokker-Planck Equation (a.k.a. Kolmogorov Forward Equation) [1]. In this equation, the $t$ and $x$ are held constant, while the $T$ and $y$ are variables (called “forward variables”). In the one-dimensional case, it reduces to

$$
\frac{\partial p}{\partial T} + \frac{\partial (Ap)}{\partial y} - \frac{1}{2} \frac{\partial^2 (B^2 p)}{\partial y^2} = 0
$$

(11)

where $A = A(T, y)$ and $B = B(T, y)$ are then scalar functions.
1.2. Kolmogorov Backward Equation

Let’s write conditional expectation of $h(S_t)$ by

$$g(t, S_t) = \mathbb{E}[h(S_T) | \mathcal{F}_t] = \mathbb{E}_t[h(S_T)]$$

(12)

Because for $s \leq t \leq T$ we have

$$g(s, S_s) = \mathbb{E}_s[h(S_T)] = \mathbb{E}_s[\mathbb{E}_t[h(S_T)]] = \mathbb{E}_s[g(t, S_t)]$$

(13)

the $g(t, S_t)$ is a martingale by the tower rule (i.e. If $\mathcal{H}$ holds less information than $\mathcal{G}$, then $\mathbb{E} \mathbb{E}[X|\mathcal{G}]|\mathcal{H}] = \mathbb{E}[X|\mathcal{H}]$). The dynamics of the $g(t, S_t)$ is given by

$$\frac{dg}{dt} = \frac{\partial g}{\partial t} dt + \int_{1 \times m \times 1} J_g dS_t + \frac{1}{2} \int_{1 \times m \times m \times 1} H_g dS_t$$

(14)

$$= \frac{dg}{dt} dt + J_g Adt + J_g BdW_t + \frac{1}{2} dW_t B' H_g B dW_t$$

where $J_g$ is the Jacobian (i.e. the same as gradient if $g$ is a scalar-valued function) and $H_g$ the Hessian of $g$ with respect to $S$ (with subscripts denoting the indices of vector components)

$$J_g = \begin{pmatrix} \frac{\partial g}{\partial S_1} & \cdots & \frac{\partial g}{\partial S_m} \end{pmatrix}, \quad H_g = \begin{pmatrix} \frac{\partial^2 g}{\partial S_1^2} & \cdots & \frac{\partial^2 g}{\partial S_1 \partial S_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 g}{\partial S_m \partial S_1} & \cdots & \frac{\partial^2 g}{\partial S_m^2} \end{pmatrix}$$

(15)

Expanding (14), we have

$$dg = \left( \frac{\partial g}{\partial t} + \sum_{i=1}^{m} \frac{\partial g}{\partial S_i} A_i + \frac{1}{2} \sum_{i,j=1}^{m} \frac{\partial^2 g}{\partial S_i \partial S_j} \Sigma_{ij} \right) dt + \sum_{i=1}^{m} \frac{\partial g}{\partial S_i} \sum_{k=1}^{n} B_{ik} dW_k$$

(16)

Since $g(t, S_t)$ is a martingale, the $dt$-term must vanish, which gives

$$\frac{\partial g}{\partial t} + \sum_{i=1}^{m} \frac{\partial g}{\partial S_i} A_i + \frac{1}{2} \sum_{i,j=1}^{m} \frac{\partial^2 g}{\partial S_i \partial S_j} \Sigma_{ij} = 0$$

(17)

This is the multi-dimensional Feynman-Kac formula.

Using the transition probability density $p_{T,y|t,x}$, we can write the expectation as
\[ g_{t,x} = \mathbb{E}_t[h(S_T)] = \int_\Omega h_y p_{T,y|t,x} dy \]  

(18)

The formula (17) defines that

\[
\frac{\partial}{\partial \rho} \int_\Omega h_y p dy + \sum_{i=1}^{m} A_i \frac{\partial}{\partial x_i} \int_\Omega h_y p dy + \frac{1}{2} \sum_{i,j=1}^{m} \Sigma_{ij} \frac{\partial^2}{\partial x_i \partial x_j} \int_\Omega h_y p dy = 0
\]

\[
\Rightarrow \int_\Omega h_y \left( \frac{\partial p}{\partial \rho} + \sum_{i=1}^{m} A_i \frac{\partial p}{\partial x_i} + \frac{1}{2} \sum_{i,j=1}^{m} \Sigma_{ij} \frac{\partial^2 p}{\partial x_i \partial x_j} \right) dy = 0
\]

(19)

By the arbitrariness of \( h \), we have

\[
\frac{\partial p}{\partial \rho} + \sum_{i=1}^{m} A_i \frac{\partial p}{\partial x_i} + \frac{1}{2} \sum_{i,j=1}^{m} \Sigma_{ij} \frac{\partial^2 p}{\partial x_i \partial x_j} = 0, \quad \Sigma = B \rho B'
\]

(20)

This is the multi-dimensional Kolmogorov Backward Equation. In this equation, the \( T \) and \( y \) are held constant, while the \( t \) and \( x \) are variables (called “backward variables”). In the 1-D case, it reduces to

\[
\frac{\partial p}{\partial t} + A \frac{\partial p}{\partial x} + \frac{1}{2} B^2 \frac{\partial^2 p}{\partial x^2} = 0
\]

(21)

where \( A = A(t,x) \) and \( B = B(t,x) \) are then scalar functions.
2. HESTON MODEL

In this chapter, we will briefly introduce the Heston stochastic volatility model, which has become quite popular in industry to model volatility smiles. One of the reasons for that is that vanilla options in Heston model have close form solutions. This makes the calibration of the model computationally much efficient and accurate. To understand these closed form formulas, we will introduce the Fourier transform method for option pricing and its application to the Heston model. Additionally, a characteristic function based method is also discussed, which extends the Heston model to have piecewise time-dependent parameters.

2.1. Heston Stochastic Volatility Model

The stochastic volatility in Heston’s model is a mean-reverting square-root process defined by the following stochastic differential equations (SDE)

\[
\frac{dS_t}{S_t} = (\mu - q) dt + \sqrt{v_t} dW_{1,t}
\]

\[
dv_t = \epsilon(\theta - v_t)dt + \xi \sqrt{v_t} dW_{2,t}
\]

\[dW_{1,t} dW_{2,t} = \rho dt\]

where \(t\) denotes the time, \(S_t\) the spot process, \(\mu\) and \(q\) the drift and dividend of the spot, \(v_t\) the variance process, \(\epsilon\) the mean reversion rate, \(\theta\) the mean variance, \(\xi\) the volatility of the variance and \(dW_{1,t}, dW_{2,t}\) the two Brownian motions correlated by \(\rho\) under physical measure \(\mathbb{P}\). All the parameters \(\mu, q, \epsilon, \theta\) and \(\rho\) are time and state homogenous (invariant).

2.1.1. Market Price of Risk

In the Black-Scholes model, a contingent claim is dependent on one or more tradable assets \(S_t\). The randomness in the option value is solely due to the randomness of these tradable assets. Therefore, the option can be hedged by continuously trading the underlying. This makes the market complete (i.e. every contingent claim can be replicated). In a stochastic volatility model, a contingent claim is dependent on both the randomness of the asset \(S_t\) and the randomness associated with the instantaneous volatility \(v_t\).
of the asset. Since the volatility is not a traded asset, this renders the market incomplete and has many implications to the pricing of options.

Firstly let’s consider two arbitrary derivative securities (presume they are available in the traded markets) whose prices can be written as functions \( U(t, S_t, \nu_t) \) and \( V(t, S_t, \nu_t) \) of variables \( t, S_t \) and \( \nu_t \) respectively. We construct a self-financing portfolio with a price process \( X_t \) by having long one share of \( U(t, S_t, \nu_t) \), short \( \Gamma_t \) shares of \( V(t, S_t, \nu_t) \) and short \( \Delta_t \) shares of \( S_t \), that is

\[
X_t = U_t - \Gamma_t V_t - \Delta_t S_t \tag{23}
\]

Following the Ito’s lemma, we can derive the price dynamics of the derivative as

\[
dU = \frac{\partial U}{\partial t} dt + \frac{\partial U}{\partial S} dS + \frac{1}{2} \frac{\partial^2 U}{\partial S^2} dS^2 + \frac{\partial U}{\partial \nu} d\nu + \frac{1}{2} \frac{\partial^2 U}{\partial \nu^2} d\nu^2 + \frac{\partial U}{\partial S} d\nu + \frac{\partial U}{\partial \nu} dS
\]

\[
= \frac{\partial U}{\partial t} dt + \frac{\partial U}{\partial S} (\mu - q) S dt + \frac{\nu S^2}{2} \frac{\partial^2 U}{\partial S^2} dS^2 + \frac{\partial U}{\partial \nu} \epsilon(\theta - \nu) dt + \frac{\partial U}{\partial \nu} \xi \sqrt{\nu} dW_2
\]

\[
+ \frac{\nu \xi^2}{2} \frac{\partial^2 U}{\partial \nu^2} dt + S \nu \xi \rho \frac{\partial^2 U}{\partial \nu \partial S} dt
\]

and derive the price dynamics of the self-financing portfolio as

\[
dX = dU - \Gamma dV - \Delta dS - \Delta d\nu dt
\]

\[
= \left( \frac{\partial U}{\partial t} + \frac{\partial U}{\partial S} (\mu - q) S + \frac{\nu S^2}{2} \frac{\partial^2 U}{\partial S^2} + \frac{\partial U}{\partial \nu} \epsilon(\theta - \nu) + \frac{\nu \xi^2}{2} \frac{\partial^2 U}{\partial \nu^2} + S \nu \xi \rho \frac{\partial^2 U}{\partial \nu \partial S} \right) dt
\]

\[
- \Gamma \left( \frac{\partial V}{\partial t} + \frac{\partial V}{\partial S} (\mu - q) S + \frac{\nu S^2}{2} \frac{\partial^2 V}{\partial S^2} + \frac{\partial V}{\partial \nu} \epsilon(\theta - \nu) + \frac{\nu \xi^2}{2} \frac{\partial^2 V}{\partial \nu^2} + S \nu \xi \rho \frac{\partial^2 V}{\partial \nu \partial S} \right) dt
\]

\[
+ \left( \frac{\partial U}{\partial S} - \Delta - \Gamma \frac{\partial V}{\partial S} \right) S \sqrt{\nu} dW_1 + \left( \frac{\partial U}{\partial \nu} - \Gamma \frac{\partial V}{\partial \nu} \right) \xi \sqrt{\nu} dW_2 - \Delta (\mu - q) S dt - \Delta d\nu dt
\]

In order to eliminate both spot and volatility risk, we must have

\[
\frac{\partial U}{\partial S} - \Delta - \Gamma \frac{\partial V}{\partial S} = 0 \quad \text{and} \quad \frac{\partial U}{\partial \nu} - \Gamma \frac{\partial V}{\partial \nu} = 0 \tag{26}
\]

and therefore
\[ dX = \left( \frac{\partial U}{\partial t} + \frac{vS^2}{2} \frac{\partial^2 U}{\partial S^2} + \frac{v\xi^2}{2} \frac{\partial^2 U}{\partial v^2} + S\nu\xi\rho \frac{\partial^2 U}{\partial v \partial S} - \frac{\partial U}{\partial S} S \right) dt \]

\[-\Gamma \left( \frac{\partial V}{\partial t} + \frac{vS^2}{2} \frac{\partial^2 V}{\partial S^2} + \frac{v\xi^2}{2} \frac{\partial^2 V}{\partial v^2} + S\nu\xi\rho \frac{\partial^2 V}{\partial v \partial S} - \frac{\partial V}{\partial S} S \right) dt \]

\[ \text{(27)} \]

In this case, the portfolio is riskless and must have a return at risk free rate in order to avoid arbitrage

\[ dX = rXdt = r(U - \Gamma V - \Delta S)dt = r Ud\tau - r\Gamma Vd\tau - r \frac{\partial U}{\partial S} Sd\tau + r \frac{\partial V}{\partial S} Sd\tau \]

\[ \text{(28)} \]

which in turn gives

\[ \frac{\partial U}{\partial t} + \frac{vS^2}{2} \frac{\partial^2 U}{\partial S^2} + \frac{v\xi^2}{2} \frac{\partial^2 U}{\partial v^2} + S\nu\xi\rho \frac{\partial^2 U}{\partial v \partial S} + \frac{\partial U}{\partial S} S(r - q) - rU \]

\[ \frac{\partial V}{\partial t} + \frac{vS^2}{2} \frac{\partial^2 V}{\partial S^2} + \frac{v\xi^2}{2} \frac{\partial^2 V}{\partial v^2} + S\nu\xi\rho \frac{\partial^2 V}{\partial v \partial S} + \frac{\partial V}{\partial S} S(r - q) - rV \]

\[ \equiv \eta \]

\[ \text{(29)} \]

In the above equation, the left-hand side is a function of \( U \) only and the right-hand side is a function of \( V \) only. The only way for the equality to hold is for both sides to equal a common function \( \eta \) of the independent variables \( t, S_t \) and \( v_t \).

Now let’s consider a delta-neutral portfolio \( Y_t \) by having long one share of \( U(t, S_t, v_t) \) and short \( \Delta_t \) shares of \( S_t \)

\[ Y_t = U_t - \Delta_t S_t \]

\[ \text{(30)} \]

The price dynamics of the portfolio is

\[ dY = dU - \Delta dS - \Delta S q dt \]

\[ \left( \frac{\partial U}{\partial t} + \frac{\partial U}{\partial S} (\mu - q) S + \frac{vS^2}{2} \frac{\partial^2 U}{\partial S^2} + \frac{v\xi^2}{2} \frac{\partial^2 U}{\partial v^2} + S\nu\xi\rho \frac{\partial^2 U}{\partial v \partial S} \right) dt \]

\[ + \left( \frac{\partial U}{\partial S} - \Delta \right) S \sqrt{\nu dW_1} + \frac{\partial U}{\partial v} \xi \sqrt{\nu dW_2} - \Delta (\mu - q) S dt - \Delta S q dt \]

\[ \text{(31)} \]

Since delta-neutral implies \( \frac{\partial U}{\partial S} - \Delta = 0 \), we are able to derive the dynamics of the discounted portfolio, a martingale under risk neutral measure, as
\[
\frac{d(D_tY_t)}{D_t} = dY - rYdt = dU - \Delta dS - \Delta Sqdt - rUdt + r\Delta dt \\
= \left(\frac{\partial U}{\partial v} \epsilon(\theta - v) + \frac{\partial U}{\partial t} + \frac{vS^2}{2} \frac{\partial^2 U}{\partial S^2} + \frac{v\xi^2}{2} \frac{\partial^2 U}{\partial v^2} + Sv\xi \rho \frac{\partial^2 U}{\partial v \partial S} + \frac{\partial U}{\partial S} (r - q) - rU\right) dt \\
+ \frac{\partial U}{\partial v} \xi \sqrt{v} dW_2 \\
= \frac{\partial U}{\partial v} (\epsilon(\theta - v) + \eta) dt + \frac{\partial U}{\partial v} \xi \sqrt{v} dW_2 = \frac{\partial U}{\partial v} \xi \sqrt{v} d\tilde{W}_2
\]

by defining
\[
d\tilde{W}_2 = dW_2 + \phi_2 dt \quad \text{and} \quad \phi_2 = \frac{\epsilon(\theta - v) + \eta}{\xi \sqrt{v}} \quad (33)
\]

In the above, the \(\tilde{W}_2\) is a Brownian motion under risk neutral measure \(Q\) and \(\phi_2\) is the market price of volatility risk.

We next consider a vega-neutral portfolio \(Z_t\) by having long a share of \(U(t, S_t, v_t)\) and short \(\Gamma_t\) shares of \(V(t, S_t, v_t)\)
\[
Z_t = U_t - \Gamma_t V_t \quad (34)
\]

The dynamics of the portfolio reads
\[
dZ = dU - \Gamma dV \\
= \left(\frac{\partial U}{\partial t} + \frac{\partial U}{\partial S} (\mu - q) S + \frac{vS^2}{2} \frac{\partial^2 U}{\partial S^2} + \frac{v\xi^2}{2} \frac{\partial^2 U}{\partial v^2} + Sv\xi \rho \frac{\partial^2 U}{\partial v \partial S} \right) dt \\
- \Gamma \left(\frac{\partial V}{\partial t} + \frac{\partial V}{\partial S} (\mu - q) S + \frac{vS^2}{2} \frac{\partial^2 V}{\partial S^2} + \frac{v\xi^2}{2} \frac{\partial^2 V}{\partial v^2} + Sv\xi \rho \frac{\partial^2 V}{\partial v \partial S} \right) dt \\
+ \left(\frac{\partial U}{\partial S} - \Gamma \frac{\partial V}{\partial S}\right) S\sqrt{v} dW_1 + \left(\frac{\partial U}{\partial v} - \Gamma \frac{\partial V}{\partial v}\right) \xi \sqrt{v} d\tilde{W}_2 \quad (35)
\]

Since vega-neutral implies \(\frac{\partial U}{\partial v} - \Gamma \frac{\partial V}{\partial v} = 0\), we can derived the dynamics of the discounted portfolio as
\[
\frac{d(D_tZ_t)}{D_t} = dZ - rZ dt = dU - \Gamma dV - rU dt + r\Gamma V dt \quad (36)
\]
\[
\left( \frac{\partial U}{\partial t} + \frac{\partial U}{\partial S}(\mu - q)S + \nu S^2 \frac{\partial^2 U}{\partial S^2} + \frac{\nu \xi^2}{2} \frac{\partial^2 U}{\partial v^2} + Sv\xi \rho \frac{\partial^2 U}{\partial v \partial S} - rU \right) dt \\
- \Gamma \left( \frac{\partial V}{\partial t} + \frac{\partial V}{\partial S}(\mu - q)S + \nu S^2 \frac{\partial^2 V}{\partial S^2} + \frac{\nu \xi^2}{2} \frac{\partial^2 V}{\partial v^2} + Sv\xi \rho \frac{\partial^2 V}{\partial v \partial S} - rV \right) dt \\
+ \left( \frac{\partial U}{\partial S} - \Gamma \frac{\partial V}{\partial S} \right) S\sqrt{v} dW_1 \\
\left( \frac{\partial U}{\partial t} + \frac{\nu S^2}{2} \frac{\partial^2 U}{\partial S^2} + \frac{\nu \xi^2}{2} \frac{\partial^2 U}{\partial v^2} + Sv\xi \rho \frac{\partial^2 U}{\partial v \partial S} - rU \right) dt \\
- \Gamma \left( \frac{\partial V}{\partial t} + \frac{\nu S^2}{2} \frac{\partial^2 V}{\partial S^2} + \frac{\nu \xi^2}{2} \frac{\partial^2 V}{\partial v^2} + Sv\xi \rho \frac{\partial^2 V}{\partial v \partial S} - rV \right) dt + \left( \frac{\partial U}{\partial S} - \Gamma \frac{\partial V}{\partial S} \right) dS \\
= \left( \frac{\eta}{\partial v} - \frac{\partial U}{\partial S} S(r - q) - \Gamma \frac{\partial U}{\partial S} S(r - q) \right) dt + \left( \frac{\partial U}{\partial S} - \Gamma \frac{\partial V}{\partial S} \right) dS \\
= \left( \frac{\partial U}{\partial S} - \Gamma \frac{\partial V}{\partial S} \right) (dS - S(r - q) dt) = \left( \frac{\partial U}{\partial S} - \Gamma \frac{\partial V}{\partial S} \right) S \left( \left( \mu - r \right) dt + \sqrt{v} dW_1 \right) \\
= \left( \frac{\partial U}{\partial S} - \Gamma \frac{\partial V}{\partial S} \right) S\sqrt{v} d\tilde{W}_1
\]

by defining

\[
d\tilde{W}_1 = dW_1 + \phi_1 dt \quad \text{and} \quad \phi_1 = \frac{\mu - r}{\sqrt{v}} \quad (37)
\]

where \(d\tilde{W}_1\) is a Brownian motion under risk neutral measure \(\mathbb{Q}\) and \(\phi_1\) is the market price of spot risk.

According to (33) and (37), the Heston SDE (22) can be written under risk neutral measure as

\[
\frac{dS_t}{S_t} = (\mu - q) dt + \sqrt{v_t} (d\tilde{W}_{1,t} - \phi_1 dt) = (r - q) dt + \sqrt{v_t} d\tilde{W}_{1,t} \\
\frac{dv_t}{v_t} = \epsilon(\theta - v_t) dt + \xi \sqrt{v_t} (d\tilde{W}_{2,t} - \phi_2 dt) = (\epsilon(\theta - v_t) - \phi_2 \xi \sqrt{v_t}) dt + \xi \sqrt{v_t} d\tilde{W}_{2,t} \quad (38)
\]

Based on the Consumption-based Capital Asset Pricing Model, Heston [22] assumes that the market price of volatility risk is proportional to volatility, that is

\[
\phi_2 = c\sqrt{v} \quad \text{for a constant} \quad c \quad \Rightarrow \quad \phi_2 \xi \sqrt{v} = c\xi v = \lambda v \quad \text{where} \quad \lambda = c\xi \quad (39)
\]

If we define
then the Heston SDE under risk neutral measure $\mathbb{Q}$ becomes

\[
\frac{dS_t}{S_t} = (r - q)dt + \sqrt{\nu_t}d\tilde{W}_t, \quad d\nu_t = \kappa(\theta - \nu_t)dt + \xi\sqrt{\nu_t}d\tilde{W}_2, \quad d\tilde{W}_{1,t}d\tilde{W}_{2,t} = \rho dt
\] (41)

which retains the form of the equation under the transformation from the physical measure $\mathbb{P}$ to the risk neutral measure $\mathbb{Q}$.

Since the volatility is not a traded asset, the incompleteness of the market implies the risk neutral measure is not unique and depends on the value of the market price of volatility risk $\phi_2$. To estimate the model parameters, one may calibrate the Heston’s model using historical spot data, however the historical calibration does not allow for the estimation of $\phi_2$. Instead of using the spot data, one may also calibrate the model to the volatility smile (i.e. prices of vanilla options). In this case, the market price of volatility risk has already been implied in the market smile, and consequently embedded into the calibrated model parameters $\kappa$ and $\theta$ through (40).

2.1.2. Radon-Nikodym Derivative

The change of measure from $\mathbb{P}$ to $\mathbb{Q}$ is achieved through Radon-Nikodym derivative via multi-dimensional Girsanov’s theorem [2]. To derive this derivative, we may write correlated $n$-dimensional Brownian motions as $dW_t$ and $d\tilde{W}_t$ under physical measure $\mathbb{P}$ and risk neutral measure $\mathbb{Q}$ respectively. The matrix $\rho$ denotes the instantaneous correlation, e.g. $dW_t dW_t' = \rho dt$. It should be noted that $dW_t$ and $d\tilde{W}_t$ possess the same correlation structure only if each is under its own probability measure, $\mathbb{P}$ or $\mathbb{Q}$, otherwise this property does not hold. From (33) and (37), we represent the market price of risk by correlation matrix an $n$-dimensional vector $\phi$ such that

\[
d\tilde{W}_t = dW_t + \rho\phi_t dt
\] (42)

The Radon-Nikodym derivative is then given for $t > s$ by
\[
\frac{dQ}{dP} = \exp \left( -\frac{1}{2} \int_s^t \phi_u' \rho \phi_u du - \int_s^t \phi_u' dW_u \right)
\]  
(43)

To check this, let’s assume under \( Q \) we have a martingale process for \( t > s \)

\[
X_t = X_s \exp \left( -\frac{1}{2} \int_s^t \sigma_u' \rho \sigma_u du + \int_s^t \sigma_u' d\tilde{W}_u \right)
\]  
(44)

where \( \sigma_u \) is a vector representing an adapted volatility process. According to (42) we have

\[
X_t \frac{dQ}{dP} = X_s \exp \left( -\frac{1}{2} \int_s^t \sigma_u' \rho \sigma_u du + \int_s^t \sigma_u' dW_u + \int_s^t \sigma_u' \rho \phi_u du \right)
\]  
and

\[
X_t \frac{dQ}{dP} = X_s \exp \left( -\frac{1}{2} \int_s^t \sigma_u' \rho \sigma_u du + \int_s^t \sigma_u' \rho \phi_u du - \frac{1}{2} \int_s^t \phi_u' \rho \phi_u du + \int_s^t (\sigma_u - \phi_u)' dW_u \right)
\]  
(45)

\[
= X_s \exp \left( -\frac{1}{2} \int_s^t (\sigma_u - \phi_u)' \rho (\sigma_u - \phi_u) du + \int_s^t (\sigma_u - \phi_u)' dW_u \right)
\]

The \( X_t \frac{dQ}{dP} \) is a martingale under \( P \). Hence we have the following equation

\[
\mathbb{E}_s [X_t] = \mathbb{E}_s \left[ X_t \frac{dQ}{dP} \right] = X_s
\]  
(46)

as expected.

2.1.3. Feller Condition

Feller observed that the variance process \( \nu_t \) in (41) remains strictly positive with probability one for all times \( t > s \), if \( \nu_s > 0 \) and the Feller condition [3] [4] is satisfied

\[
2 \kappa \theta \geq \xi^2
\]  
(47)

If the condition is not satisfied, i.e. \( 0 < 2 \kappa \theta < \xi^2 \), the variance will visit zero recurrently but will not stay at zero, i.e. the zero boundary is strongly reflecting. In typical applications, the Feller condition is often violated due to the convexities of volatility smiles typically encountered in practice. Indeed the process \( \nu_t \) often has a strong affinity for the area around the origin. However, this is not a complete disaster, as the process \( \nu_t \) can only hit zero for an infinitesimally small amount of time.

2.2. Probability Distribution of Spot Returns
In this section, we present a derivation of the distribution of the spot returns in Heston’s model [5]. Let’s firstly make a change of variable from (22) to have a centered log-spot for $t > s$

$$x_t = \ln \frac{S_t}{F_{s,t}} = \ln \frac{S_t}{S_s} \exp\left(\frac{(r - q)(t - s)}{s}\right) = \ln S_t S_s^{-1} - (r - q)(t - s) \quad (48)$$

The Heston’s model (41) under $\mathbb{Q}$ can then be converted by Ito’s lemma to the following form

$$dx_t = -\frac{v_t}{2} dt + \sqrt{v_t} d\bar{W}_{1,t}, \quad dv_t = \kappa(\theta - v_t)dt + \xi \sqrt{v_t} d\bar{W}_{2,t}, \quad d\bar{W}_{1,t} d\bar{W}_{2,t} = \rho dt \quad (49)$$

This defines a 2-D stochastic process characterized by a joint transition probability density function $p_{t,x,v|\nu_s}$, which is the probability having log-spot $x_t$ and instantaneous variance $v_t$ at time $t$, conditional on $v_s$ at time $t = s$ (independent on $x_s$ because $x_s = 0$ almost surely).

2.2.1. Derivation of the Transition Probability

We may rewrite (49) in terms of a 2-D Brownian motion $d\bar{W}_t$

$$dZ_t = A_t dt + C_t d\bar{W}_t \quad \text{with}$$

$$Z_t = (x_t, v_t), \quad A_t = \left(\begin{array}{c} -\frac{v_t}{2} \\ \kappa(\theta - v_t) \end{array} \right), \quad C_t = \left(\begin{array}{cc} \sqrt{v_t} & 0 \\ 0 & \xi \sqrt{v_t} \end{array} \right), \quad d\bar{W}_t d\bar{W}_t' = \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix} dt \quad (50)$$

The instantaneous covariance matrix of $dZ_t$ becomes

$$\Sigma_t = C_t \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix} C_t = \begin{pmatrix} v_t & \rho \xi v_t \\ \rho \xi v_t & \xi^2 v_t \end{pmatrix} \quad (51)$$

The 2-D Markov process is characterized by the transition probability $p_{t,x,v|\nu_s}$. The Fokker-Planck equation that governs the time evolution of the transition probability is given by (10)

$$\frac{\partial p}{\partial t} - \frac{1}{2} \frac{\partial(v p)}{\partial x} + \frac{\partial(\kappa(\theta - v)p)}{\partial v} - \frac{1}{2} \frac{\partial^2(v p)}{\partial x^2} - \rho \xi \frac{\partial^2(v p)}{\partial x \partial v} - \frac{\xi^2}{2} \frac{\partial^2(v p)}{\partial v^2} = 0 \quad (52)$$

with initial condition $p_{t=x,v|\nu_s} = \delta_{x-x_s} \delta_{v-v_s} = \delta_{\nu-x_s} \delta_{\nu-v_s}$, where $\delta$ is the Dirac delta function. The marginal probability density of the variance alone

$$\zeta_{\nu,v_s} = \int_{\mathbb{R}} p_{t,x,v|\nu_s} dx \quad (53)$$

satisfies the following Fokker-Planck equation obtained from (52) by integration over $x$. 

\[ \zeta_{\nu,v_s} \]
\[
\frac{\partial \zeta}{\partial t} = \frac{\partial (\kappa (v - \theta) \zeta)}{\partial v} + \frac{\xi^2 \partial^2 (v \zeta)}{2 \partial v^2}
\]

(54)

Feller has shown that this equation is well defined on the interval \( v \in [0, +\infty) \) as long as \( \theta > 0 \). Equation (54) has a stationary solution, which is a Gamma distribution

\[
\zeta_v^* = \frac{\alpha^\alpha}{\Gamma(\alpha)} \frac{v^{\alpha - 1}}{\theta^\alpha} \exp\left(-\frac{\alpha v}{\theta}\right) \quad \text{and} \quad \alpha = \frac{2 \kappa \theta}{\xi^2}
\]

(55)

Since \( x \) appears in (52) only in the derivative operator, it is convenient to take the Fourier transform, such that

\[
\hat{p}_{t, x, v|v_s} = \int_{\mathbb{R}} e^{-i\omega x} p_{t, x, v|v_s} \, dx \quad \text{and} \quad p_{t, x, v|v_s} = \frac{1}{2\pi} \int_{\mathbb{R}} e^{i\omega x} \hat{p}_{t, x, v|v_s} \, d\omega
\]

(56)

Inserting (56) into (52), we have

\[
\frac{\partial \hat{p}}{\partial t} = \frac{\partial (\kappa (v - \theta) \hat{p})}{\partial v} + \frac{i \omega - \omega^2}{2} v \hat{p} + i \rho \xi \partial (v \hat{p}) + \frac{\xi^2 \partial^2 (v \hat{p})}{2 \partial v^2}
\]

(57)

Since (57) is linear in \( v \) and quadratic in \( \frac{\partial}{\partial v} \), it can be simplified by taking the Laplace transform over \( v \)

\[
\tilde{p}_{t, x, \lambda|v_s} = \int_{\mathbb{C}} e^{-\lambda v} \hat{p}_{t, x, v|v_s} \, dv
\]

(58)

The PDE satisfied by \( \tilde{p}_{t, x, \lambda|v_s} \) is of the first order

\[
\frac{\partial \tilde{p}}{\partial t} = \left(\frac{\omega^2 - \xi^2 \lambda^2 - i \omega}{2} - \gamma \lambda\right) \frac{\partial \tilde{p}}{\partial \lambda} - \kappa \theta \lambda \tilde{p}
\]

(59)

with initial condition \( \tilde{p}_{t=s, x, \lambda|v_s} = e^{-\lambda v_s} \), where \( \gamma = \kappa + i \rho \xi \omega \). The solution of this PDE is given by the method of characteristics

\[
\tilde{p}_{t, x, \lambda|v_s} = \exp\left(-\tilde{\lambda}_s v_s - \kappa \theta \int_s^t \tilde{\lambda}_u \, du\right)
\]

(60)

where the function \( \tilde{\lambda}_t \) is the solution of the characteristic (ordinary) differential equation

\[
\frac{d \tilde{\lambda}_u}{du} = \gamma \tilde{\lambda}_u + \frac{\xi^2}{2} \tilde{\lambda}_u^2 - \frac{\omega^2 - i \omega}{2}
\]

(61)
With a boundary condition \( \tilde{\lambda}_t = \lambda \) specified at time \( t \), the \((61)\) is a Riccati equation with constant coefficients and its solution is

\[
\tilde{\lambda}_u = \frac{2\Omega}{\xi^2} \frac{1}{\psi e^{\Omega(t-u)} - 1} \frac{\nu - \Omega}{\xi^2} \quad \text{with} \quad \Omega = \sqrt{\nu^2 + \xi^2(\omega^2 - i\omega)}, \quad \Psi = 1 + \frac{2\Omega}{\xi^2\lambda + \nu - \Omega} \quad (62)
\]

Plugging \((62)\) into \((60)\), we have

\[
\hat{p}_{t,\omega,\lambda|v_s} = \exp \left( -\tilde{\lambda}_s v_s + \frac{\nu}{\xi^2} \frac{\nu^2 + \xi^2(\omega^2 - i\omega)}{\xi^2} \frac{\nu}{\xi^2} \frac{\nu^2 + \xi^2(\omega^2 - i\omega)}{\xi^2} \right) \quad (63)
\]

Normally we are interested only in distribution of log-spot \( x_t \) and do not care about variance \( v_t \). Therefore we derive the marginal probability density for \( x_t \) with \( \lambda = 0 \)

\[
p_{t|x|v_s} = \int_{\mathbb{R}^+} p_{t,x,v|v_s} dv
\]

\[
= \frac{1}{2\pi} \int_{\mathbb{R}^+} \int_{\mathbb{R}} e^{i\omega x} \hat{p}_{t,\omega,v|v_s} d\omega dv = \frac{1}{2\pi} \int_{\mathbb{R}} e^{i\omega x} \int_{\mathbb{R}^+} \hat{p}_{t,\omega,v|v_s} dv d\omega = \frac{1}{2\pi} \int_{\mathbb{R}} e^{i\omega x} \hat{p}_{t,\omega,0|v_s} d\omega
\]

\[
= \frac{1}{2\pi} \int_{\mathbb{R}} \exp \left( i\omega x - \frac{\nu^2 + \xi^2(\omega^2 - i\omega)}{\nu} \frac{\nu}{\xi^2} \frac{\nu^2 + \xi^2(\omega^2 - i\omega)}{\xi^2} \ln \left( \cosh \frac{\Omega t}{2} + \frac{\gamma}{\Omega} \sinh \frac{\Omega t}{2} \right) \right) d\omega
\]

where the last step comes from substitution of \( \hat{p}_{t,\omega,0|v_s} \) in \((60)\) into \((56)\). The derived density function \( p_{t,x|v_s} \) in \((64)\) is still dependent on the unknown initial variance \( v_s \). To remove the dependence, the \( v_s \) is assumed to have the stationary distribution density as in \((55)\). Thus the unconditional transition density function \( p_t(x) \), is derived by averaging \((64)\) over \( v_s \) with the weight \( \zeta^* \)

\[
p_{t,x} = \int_{\mathbb{R}^+} p_{t,x|v_s} \zeta^*_s dz \quad (65)
\]

The integral over \( v_s \) is similar to the one of the Gamma function and can be taken explicitly. The final result is the Fourier integral

\[
p_{t,x} = \frac{1}{2\pi} \int_{\mathbb{R}} e^{i\omega x + F_{t,\omega}} d\omega, \quad F_{t,\omega} = \frac{\kappa\theta t}{\xi^2} - \frac{2\kappa\theta}{\xi^2} \ln \left( \cosh \frac{\Omega t}{2} + \frac{\gamma^2 + 2\kappa\gamma}{2\kappa\Omega} \sinh \frac{\Omega t}{2} \right) \quad (66)
\]
It is easy to check that \( p_{t,x} \) is real, because \( \Re[F_{t,\omega}] \) is an even function of \( \omega \) and \( \Im[F_{t,\omega}] \) is an odd one.

One can also check that \( F_{t,\omega=0} = 0 \), which implies that \( p_{t,x} \) is correctly normalized at all times.

2.2.2. Moment Generating Function (in progress…)

We can integrate (49) over time from \( s \) to \( t \) to get

\[
x_t = x_s - \frac{1}{2} \int_s^t v_u \, du + \int_s^t \sqrt{v_u} \, d\tilde{W}_{1,u}, \quad v_t = v_s + \kappa \theta t - \kappa \int_s^t v_u \, du + \xi \int_s^t \sqrt{v_u} \, d\tilde{W}_{2,u} \tag{67}
\]

where \( \tau = t - s \) and \( d\tilde{W}_{1,t} = \rho d\tilde{W}_{2,t} + \eta dB_t \) for \( \eta = \sqrt{1 - \rho^2} \) and the Brownian motion \( dB_t \) is independent of \( d\tilde{W}_{2,t} \). Defining a function \( h_{t;\alpha, \beta} \) such that

\[
h_{t;\alpha, \beta} = \mathbb{E}_s[\exp(\alpha x_t + \beta v_t)] = \mathbb{E}_s \left[ \exp \left( \alpha x_t + \left( \beta + \frac{\alpha \rho}{\xi} \right) v_t - \frac{\alpha \rho}{\xi} v_t \right) \right]
\]

\[
= \mathbb{E}_s \left[ \exp \left( \alpha x_s - \alpha \rho \frac{v_s + \kappa \theta t}{\xi} + \left( \beta + \frac{\alpha \rho}{\xi} \right) v_t + \left( \frac{\alpha \rho \kappa}{\xi} - \frac{\alpha}{2} \right) \int_s^t v_u \, du + \alpha \eta \int_s^t \sqrt{v_u} \, dB_u \right) \right]
\]

\[
= \exp \left( \alpha x_s - \alpha \rho \frac{v_s + \kappa \theta t}{\xi} \right) \mathbb{E}_s \left[ \exp \left( \left( \beta + \frac{\alpha \rho}{\xi} \right) v_t + \left( \frac{\alpha \rho \kappa}{\xi} - \frac{\alpha}{2} \right) \int_s^t v_u \, du + \alpha \eta \int_s^t \sqrt{v_u} \, dB_u \right) \right]
\]

\[
e^{\alpha x_s - \alpha \rho \frac{v_s + \kappa \theta t}{\xi}} \mathbb{E}_s \left[ e^{\left( \beta + \frac{\alpha \rho}{\xi} \right) v_t + \left( \frac{\alpha \rho \kappa}{\xi} - \frac{\alpha}{2} \right) \int_s^t v_u \, du + \alpha \eta \int_s^t \sqrt{v_u} \, dB_u} \right] \quad (v_t \text{ is independent of } B_t)
\]

\[
= \mathbb{E}_s \left[ \exp \left( \left( \beta + \frac{\alpha \rho}{\xi} \right) v_t + \left( \frac{\alpha \rho \kappa}{\xi} - \frac{\alpha}{2} + \frac{\alpha^2 \eta^2}{2} \right) \int_s^t v_u \, du \right) \right]
\]

we can derive the differential of \( \exp(\alpha x_t + \beta v_t) \) as

\[
d(e^{\alpha x_t + \beta v_t}) = e^{\alpha x_t + \beta v_t} \left( d(\alpha x_t + \beta v_t) + \frac{1}{2} d(\alpha x_t + \beta v_t) d(\alpha x_t + \beta v_t) \right)
\]

\[
e^{\alpha x_t + \beta v_t} \left( \left( -\frac{\alpha}{2} + \frac{\alpha^2}{2} + \frac{\beta^2}{2} + \alpha \beta \xi \rho - \beta \kappa \right) v_t dt + \beta \kappa \rho \xi \xi dt + \alpha \sqrt{v_t} d\tilde{W}_{1,t} + \beta \sqrt{v_t} d\tilde{W}_{2,t} \right) \quad (69)
\]

Integrating both sides gives
\[ e^{ax_t + \beta v_t} = e^{ax_s + \beta v_s} + \left(\frac{\alpha}{2} + \frac{\beta^2 \xi^2}{2} + \alpha \beta \xi \rho - \beta \kappa\right) \int_s^t e^{ax_u + \beta v_u} v_u du \]

\[ + \beta \kappa \theta \int_s^t e^{ax_u + \beta v_u} du + \alpha \int_s^t e^{ax_u + \beta v_u} \sqrt{v_u} d\bar{W}_1 u + \beta \xi \int_s^t e^{ax_u + \beta v_u} \sqrt{v_u} d\bar{W}_2 u \]

Taking expectation, we find

\[ h_{t,\alpha,\beta} = e^{ax_s + \beta v_s} + \left(\frac{\alpha}{2} + \frac{\beta^2 \xi^2}{2} + \alpha \beta \xi \rho - \beta \kappa\right) \int_s^t \mathbb{E}_s \left[v_u e^{ax_u + \beta v_u}\right] du \]

\[ + \beta \kappa \theta \int_s^t h_{u,\alpha,\beta} du \]  

\[ \Rightarrow h_{t,\alpha,\beta} = h_{s,\alpha,\beta} + \left(\frac{\alpha}{2} + \frac{\beta^2 \xi^2}{2} + \alpha \beta \xi \rho - \beta \kappa\right) \int_s^t \frac{\partial h_{u,\alpha,\beta}}{\partial \beta} du + \beta \kappa \theta \int_s^t h_{u,\alpha,\beta} du \]  

Differentiating with respect to \( t \), we have the PDE

\[ \frac{\partial h_{t,\alpha,\beta}}{\partial t} - p_{\alpha,\beta} \frac{\partial h_{t,\alpha,\beta}}{\partial \beta} = \beta \kappa \theta h_{t,\alpha,\beta} \]  

(To be continued, reference [6])

2.3. Analytical Solution of Vanilla Options

Vanilla option price in Heston model can be computed semi-analytically. Shortly we will see that the spot process in Heston model admits a closed-form characteristic function, which allows us to express the option prices in terms of Fourier-inversion integrals that can be evaluated numerically (e.g. using Gaussian quadratures).

2.3.1. Fourier Transform and Characteristic Function

There is no standard definition of a Fourier transform and its inverse. The one that we use for characteristic functions follows the angular frequency definition, which is (as in \( n \)-dimensional case)

Forward: \( f_\omega = \int_{\mathbb{R}^n} e^{i\omega'x} f_x dx \quad \forall \ \omega \in \mathbb{R}^n \)

Inverse: \( f_x = \frac{1}{(2\pi)^n} \int_{\mathbb{R}^n} e^{-i\omega'x} f_\omega d\omega \quad \forall \ x \in \mathbb{R}^n \)
where the $\omega$ is the angular frequency and the $(\prime)$ denotes matrix transpose (the dot product of two column vectors $\omega$ and $x$). In fact the normalization factors multiplying the forward and inverse transforms (here $1$ and $1/(2\pi)^n$, respectively) and the signs of the exponents are merely conventions and differ in some treatments. The only requirements of these conventions are: 1. the forward and inverse transforms have opposite-sign exponents, and 2. the product of their normalization factors is $1/(2\pi)^n$. In one-dimension, the (73) reduces to

$$
\text{Forward: } \hat{f}_\omega = \int_{\mathbb{R}} e^{i\omega x} f_x \, dx \quad \forall \omega \in \mathbb{R}
$$

$$
\text{Inverse: } f_x = \frac{1}{2\pi} \int_{\mathbb{R}} e^{-i\omega x} \hat{f}_\omega \, d\omega \quad \forall x \in \mathbb{R}
$$

The transform is equivalent to the case we write a periodic function $f_x$ with period $2\pi/h$ in terms of Fourier series expansion

$$
f_x = \frac{h}{2\pi} \sum_{k=-\infty}^{\infty} c_{hk} e^{-ihkx} \quad \text{where} \quad c_{hk} = \int_{-\frac{\pi}{h}}^{\frac{\pi}{h}} f_x e^{ihkx} \, dx \quad \forall h > 0
$$

Taking the limit $h \to 0^+$, we have

$$
\lim_{h \to 0^+} c_{hk} = \lim_{h \to 0^+} \int_{-\frac{\pi}{h}}^{\frac{\pi}{h}} f_x e^{ihkx} \, dx = \int_{\mathbb{R}} f_x e^{i\omega x} \, dx = \hat{f}_\omega \quad \text{and}
$$

$$
f_x = \lim_{h \to 0^+} \frac{h}{2\pi} \sum_{k=-\infty}^{\infty} c_{hk} e^{-ihkx} = \frac{1}{2\pi} \int_{\mathbb{R}} \hat{f}_\omega e^{-i\omega x} \, d\omega \quad \text{where} \quad \omega = hk
$$

The sum can be regarded as an approximating Riemann sum for the integral [7].

Characteristic function $\phi_\omega$ of any random variable $X$ complete defines its probability distribution.

On the real line the characteristic function is given by the following formula

$$
\phi_\omega \equiv \mathbb{E}\left[e^{i\omega X}\right] = \int_{\mathbb{R}} e^{i\omega x} p_x \, dx = \int_{\Omega} e^{i\omega x} dP_x \quad \forall \omega \in \mathbb{R}
$$
where the \( p_x \) denotes the probability density function (PDF) and the \( P_x = \int_{-\infty}^{x} p_y dy \) is the cumulative density function (CDF). The characteristic function is merely a Fourier transform of the \( p_x \), likewise the \( p_x \) can be recovered from \( \phi_\omega \) through the inverse Fourier transform [8]

\[
p_x = \frac{1}{2\pi} \int_{\mathbb{R}} e^{-i\omega x} \phi_\omega d\omega \quad \forall \omega \in \mathbb{R}
\]

(78)

The \( P_x \) can be computed from \( \phi_\omega \) through Levy’s Inversion Formula [9] [10] [11]

\[
P_x = \frac{\phi_0}{2} + \frac{1}{2\pi} \int_{\mathbb{R}^+} \frac{e^{i\omega x} \phi_{-\omega} - e^{-i\omega x} \phi_{\omega}}{i\omega} d\omega
\]

(79)

where \( \phi_0 = 1 \) if \( \phi_\omega \) is a characteristic function of a random variable. Before proving the formula, we need to find Fourier transform of signum function \( S_x \), which is defined as

\[
S_x = \begin{cases} 
-1 & \text{if } x < 0 \\
1 & \text{if } x > 0 
\end{cases}
\]

(80)

Its transform cannot be obtained via direct integration. However we can consider an odd two-sided exponential function \( S_x^h \) with \( h > 0 \)

\[
S_x^h = \begin{cases} 
-e^{hx} & \text{if } x < 0 \\
e^{-hx} & \text{if } x > 0 
\end{cases}
\]

(81)

The \( \hat{S}_\omega^h \), i.e. the Fourier transform of \( S_x^h \), can then be derived as

\[
\hat{S}_\omega^h = \int_{\mathbb{R}} e^{i\omega x} S_x^h dx = -\int_{\mathbb{R}^-} e^{(i\omega + h)x} dx + \int_{\mathbb{R}^+} e^{(i\omega - h)x} dx
\]

\[
= -\frac{e^{(i\omega + h)x}}{i\omega + h} \bigg|_{x = -\infty}^{0} + \frac{e^{(i\omega - h)x}}{i\omega - h} \bigg|_{x = 0}^{\infty} = \frac{1}{i\omega + h} - \frac{1}{i\omega - h} = \frac{2i\omega}{\omega^2 + h^2}
\]

(82)

The parameter \( h \) controls how rapidly the exponential function decays. As we let \( h \to 0 \), the exponential function resembles more and more closely the signum function. This suggests that

\[
\hat{S}_\omega = \lim_{h \to 0} \hat{S}_\omega^h = \lim_{h \to 0} \frac{2i\omega}{\omega^2 + h^2} = \frac{2}{i\omega}
\]

(83)

Hence the inverse transform of the \( \hat{S}_\omega \) gives
\[
\mathcal{S}_x = \frac{1}{2\pi} \int_{\mathbb{R}} e^{-i\omega x} \mathcal{S}_\omega d\omega = -\frac{1}{\pi} \int_{\mathbb{R}} \frac{e^{-i\alpha x}}{i\alpha} d\alpha = -\frac{1}{\pi} \int_{\mathbb{R}} \frac{\cos \alpha x - i \sin \alpha x}{i\omega} d\omega
\]

\[
= -\frac{1}{\pi} \int_{\mathbb{R}} \frac{\cos \alpha x}{\alpha} d\alpha + \frac{1}{\pi} \int_{\mathbb{R}} \frac{\sin \alpha x}{\alpha} d\alpha = \frac{2}{\pi} \int_{\mathbb{R}^+} \frac{\sin \alpha x}{\alpha} d\alpha
\]

With the help of the signum function \(\mathcal{S}_x\) in (84), the proof of (79) is given as follows

\[
\int_{\mathbb{R}^+} e^{i\omega x} \phi_{-\omega} - e^{-i\omega x} \phi_{\omega} d\omega = \int_{\mathbb{R}^+} e^{i\omega x} \int_{\mathbb{R}} e^{-i\omega y} p_y dy - e^{-i\omega x} \int_{\mathbb{R}} e^{i\omega y} p_y dy d\omega
\]

\[
= \int_{\mathbb{R}} \int_{\mathbb{R}^+} \frac{e^{i\omega (x-y)} - e^{-i\omega (x-y)}}{i\omega} p_y dy d\omega \quad \text{(by Fubini's theorem)}
\]

\[
= 2 \int_{\mathbb{R}} \int_{\mathbb{R}^+} \frac{\sin \omega (x-y)}{\omega} d\omega p_y dy \quad \text{(by } e^{ix} = \cos x + i \sin x\text{)}
\]

\[
= \pi \int_{\mathbb{R}} \mathcal{S}_{x-y} p_y dy = \pi \left(- \int_{\mathbb{R}} p_y dy + \int_{\mathbb{R}}^x p_y dy\right) = \pi \left(- \int_{\mathbb{R}} p_y dy + 2 \int_{-\infty}^x p_y dy\right)
\]

\[
= \pi (-\phi_0 + 2P_x)
\]

The Levy Inversion Formula can also be written in the following form

\[
P_x = \frac{\phi_0}{2} - \frac{1}{2\pi} \int_{\mathbb{R}} \frac{e^{-i\omega x} \phi_{\omega}}{i\omega} d\omega
\]

which can be proved as

\[
\int_{\mathbb{R}} e^{i\omega x} \phi_{\omega} d\omega = \int_{\mathbb{R}} e^{i\omega x} \int_{\mathbb{R}} e^{-i\omega y} p_y dy d\omega = \int_{\mathbb{R}} \int_{\mathbb{R}^+} \frac{e^{i\omega (y-x)}}{i\omega} p_y dy d\omega
\]

\[
= \int_{\mathbb{R}} \int_{\mathbb{R}^+} \frac{e^{i\omega (y-x)}}{i\omega} p_y dy = \pi \int_{\mathbb{R}} \mathcal{S}_{x-y} p_y dy = \pi (-\phi_0 + 2P_x)
\]

where

\[
\int_{\mathbb{R}} \frac{e^{i\omega x}}{i\omega} d\omega = \int_{\mathbb{R}^-} \frac{e^{i\omega x}}{i\omega} d\omega + \int_{\mathbb{R}^+} \frac{e^{i\omega x}}{i\omega} d\omega = \int_{\mathbb{R}^+} \frac{e^{i\omega x} - e^{-i\omega x}}{i\omega} d\omega = 2 \int_{\mathbb{R}^+} \frac{\sin \omega x}{\omega} d\omega = \pi \mathcal{S}_x
\]
Note that if $p_x$ is real-valued function (i.e. probability density function), its Fourier transform $\phi_\omega$ is then even in its real part and odd in its imaginary part \[12\], we therefore have $\phi_\omega = \overline{\phi_{-\omega}}$, the inversion formula (86) becomes identical to (79), which can be further reduced to

$$P_x = \frac{\phi_0}{2} + \frac{1}{2\pi} \int_{\mathbb{R}^+} e^{i\omega x} \phi_{-\omega} - e^{-i\omega x} \phi_\omega d\omega = \frac{\phi_0}{2} + \frac{1}{2\pi} \int_{\mathbb{R}^+} \overline{e^{-i\omega x} \phi_\omega} - e^{-i\omega x} \phi_\omega d\omega$$

(88)

For $P_x^C = \int_x^\infty p_y dy$, the complementary of $P_x$, we can use the simple relation $P_x^C = \phi_0 - P_x$

$$P_x^C = \frac{\phi_0}{2} - \frac{1}{2\pi} \int_{\mathbb{R}^+} e^{i\omega x} \phi_{-\omega} - e^{-i\omega x} \phi_\omega d\omega = \frac{\phi_0}{2} + \frac{1}{2\pi} \int_{\mathbb{R}^+} \Re \left[ \frac{e^{-i\omega x} \phi_\omega}{i\omega} \right] d\omega$$

(89)

### 2.3.2. Characteristic Function

In this section, we present a derivation of the closed form characteristic function of the spot in Heston model \[13\]. The definition of the Heston joint process in (49) will be used. Suppose there exists a payoff function $g_{x_T, v_T}$ on $x_T$ and $v_T$, we may calculate risk neutral expectation of the payoff function as

$$h_t = \mathbb{E}_t[g_{x_T, v_T}]$$

(90)

For example, the characteristic function of the joint distribution of $x_T$ and $v_T$ would be given by

$$\phi_{t, \alpha, \beta} = \mathbb{E}_t[g_{x_T, v_T; \alpha, \beta}] \quad \text{where} \quad g_{x_T, v_T; \alpha, \beta} = \exp(i\alpha x_T + i\beta v_T)$$

(91)

The risk neutral expectation is actually a martingale because

$$h_t = \mathbb{E}_t[g_{x_T, v_T}] = \mathbb{E}_t\left[\mathbb{E}_s[g_{x_T, v_T}]\right] = \mathbb{E}_t[h_s]$$

(92)

Applying Ito’s lemma to $h$ and forcing the drift to be zero (martingale property), we end up with a PDE

$$\frac{\partial h}{\partial t} - \frac{\nu}{2} \frac{\partial h}{\partial x} + \kappa(\theta - v) \frac{\partial h}{\partial v} + \frac{\nu}{2} \frac{\partial^2 h}{\partial x^2} + \frac{\xi^2 v}{2} \frac{\partial^2 h}{\partial v^2} + \xi \rho \frac{\partial^2 h}{\partial x \partial v} = 0$$

(93)
To determine the solution of (93), the terminal condition $h_T = \mathbb{E}_T[g_{x_T,\nu_T}] = g_{x_T,\nu_T}$ at time $T$ must be specified. The terminal payoff function that we will consider has the form $g_{x_T,\nu_T} = e^{\gamma + \delta \nu_T + i\alpha x_T}$. If $\gamma = 0$ and $\delta = i\beta$, the resulting payoff function becomes $g_{x_T,\nu_T;\alpha,\beta}$ in (91), corresponding to the characteristic function of the joint distribution.

Heston [22] guessed a solution that has the form $h_t = \mathbb{E}_t[g_{x_T,\nu_T}] = e^{C + D\nu_T + i\alpha x_T}$ where $C = C_{T,\alpha,\gamma,\delta}$, $D = D_{T,\alpha,\gamma,\delta}$, $\tau = T - t$ (94)

Substituting the tentative solution (94) in (93) yields

$$\frac{\partial C}{\partial t} + v \frac{\partial D}{\partial t} - \frac{iav}{2} + \kappa(\theta - v)D - \frac{\alpha^2 v}{2} + \frac{\xi^2 v}{2}D^2 + i\alpha D \xi \rho v = 0$$

$$\Rightarrow \frac{\partial C}{\partial t} + \kappa \theta D + \left(\frac{\partial D}{\partial t} + \frac{\xi^2}{2}D^2 - mD - \frac{\alpha(i + \alpha)}{2}\right)v = 0 \quad \text{where} \quad m = \kappa - \alpha \xi \rho i$$

As the $v$ is an independent variable, (95) is zero only if

$$\frac{\partial C}{\partial t} + \kappa \theta D = 0 \quad \text{and} \quad \frac{\partial D}{\partial t} + \frac{\xi^2}{2}D^2 - mD - \frac{\alpha(i + \alpha)}{2} = 0$$

(96)

Changing the variable $t$ to $\tau = T - t$, we have

$$\frac{\partial C}{\partial \tau} - \kappa \theta D = 0 \quad \text{and} \quad \frac{\partial D}{\partial \tau} - \frac{\xi^2}{2}D^2 + mD + \frac{\alpha(i + \alpha)}{2} = 0$$

(97)

The terminal condition for this system of equations is given by $C_{\tau=0} = \gamma$ and $D_{\tau=0} = \delta$. The ODE for $D$ is a Riccati equation that only depends on $D$. This Riccati equation can be turned into an ODE through the change of variable, $Z = (D - \bar{D})^{-1}$, where $\bar{D}$ is a particular solution to the second equation in (97)

$$\frac{\partial Z}{\partial \tau} = -\frac{1}{(D - \bar{D})^2} \frac{\partial(D - \bar{D})}{\partial \tau} = -Z^2 \left(\frac{\xi^2}{2}D^2 - mD - \frac{\xi^2}{2} \bar{D}^2 + m\bar{D}\right)$$

$$= -Z \left(\frac{\xi^2}{2}(D + \bar{D}) - m\right) = -Z \left(\frac{\xi^2}{2}(D - \bar{D} + 2\bar{D})\right) + mZ = -(\xi^2 \bar{D} - m)Z - \frac{\xi^2}{2}$$

(98)

$$\Rightarrow \frac{\partial Z}{\partial \tau} + (\xi^2 \bar{D} - m)Z + \frac{\xi^2}{2} = 0 \quad \text{or}$$
\[
\frac{\partial Z}{\partial \tau} + BZ + A = 0 \quad \text{where} \quad A = \frac{\xi^2}{2}, \quad B = \xi^2 \hat{D} - m
\]

The solution to (98) is given by

\[
Z = -\frac{A}{B} + \left( Z_0 + \frac{A}{B} \right) e^{-B \tau} \quad \text{where} \quad Z_0 = \frac{1}{D_0 - \hat{D}} = \frac{1}{\delta - \hat{D}}
\]

\[
\Rightarrow D = \frac{1}{-\frac{A}{B} + \left( Z_0 + \frac{A}{B} \right) e^{-B \tau}} + \hat{D}
\]  
(99)

The particular solution \( \hat{D} \) can be as simple as a constant, which implies it would be the solution of the quadratic equation

\[
-\frac{\xi^2}{2} \hat{D}^2 + m \hat{D} + \frac{\alpha(i + \alpha)}{2} = 0 \Rightarrow \hat{D} = \frac{m + d}{\xi^2} \quad \text{where} \quad d = \pm \sqrt{m^2 + \xi^2 \alpha(i + \alpha)}
\]  
(100)

This particular solution for \( \hat{D} \) makes \( B = d \). The solution for \( D \) in (99) can then be derived

\[
D = \frac{1}{-\frac{\xi^2}{2d} + \left( \frac{-\frac{\xi^2}{2d} + \frac{\xi^2}{2d}}{m + d} \right) e^{-d \tau}} + \frac{m + d}{\xi^2}
\]

\[
\Rightarrow \xi^2 D = -\frac{2d(m + d - \delta \xi^2)}{2e^{-d \tau} d + (m + d - \delta \xi^2)(1 - e^{-d \tau})} + m + d
\]

\[
= -\frac{2d(m + d - \delta \xi^2)}{(m + d - \delta \xi^2)(1 - e^{-d \tau})} + m + d
\]

\[
= -\frac{2d}{1 - \hat{g} e^{-d \tau}} + m + d \quad \text{(by defining} \quad \hat{g} = \frac{m - d - \delta \xi^2}{m + d - \delta \xi^2})
\]

\[
= \frac{(m + d)(1 - \hat{g} e^{-d \tau}) - 2d}{1 - \hat{g} e^{-d \tau}} = \frac{m - d -(m + d) \hat{g} e^{-d \tau}}{1 - \hat{g} e^{-d \tau}}
\]

\[
= \frac{(m + d) g - \hat{g} e^{-d \tau}}{1 - \hat{g} e^{-d \tau}} \quad \text{(by defining} \quad g = \frac{m - d}{m + d})
\]

\[
\Rightarrow D = \frac{m + d}{\xi^2} \frac{g - \hat{g} e^{-d \tau}}{1 - \hat{g} e^{-d \tau}}
\]

The \( D \) can then be plugged into the ODE for \( C \) in (97)
\[
\frac{\partial C}{\partial \tau} - \kappa \theta \frac{m + d g - \hat{g} e^{-d \tau}}{\xi^2} = 0 \quad \Rightarrow \quad C = \kappa \theta \frac{m + d}{\xi^2} \int g - \hat{g} e^{-d \tau} d\tau + K_c
\]  

(102)

where \(K_c\) is a constant to be fixed by terminal condition. The indefinite integral in (102) can be solved through change of variable \(u = e^{-d \tau}\) where \(\frac{du}{d\tau} = -ud\)

\[
\int \frac{g - \hat{g} e^{-d \tau}}{1 - \hat{g} e^{-d \tau}} d\tau = - \int \frac{g - \hat{g} u}{1 - \hat{g} u} \frac{1}{u d} du = - \frac{1}{d} \int \left( \frac{g - \hat{g} u}{1 - \hat{g} u} \right) du
\]

\[
= - \frac{1}{d} \left( \frac{g - \hat{g} - (1 - \hat{g} u)}{1 - \hat{g} u} + \frac{g}{u} \right) du = - \frac{1}{d} \int \left( \frac{(g - 1) \hat{g} + g}{u} \right) du
\]

\[
= g - \frac{1}{d} \ln(1 - \hat{g} u) - g \frac{1}{d} \ln u = g - \frac{1}{d} \ln(1 - \hat{g} e^{-d \tau}) + g \tau
\]

which gives

\[
C = \kappa \theta \frac{m + d}{\xi^2} \left( g - \frac{1}{d} \ln(1 - \hat{g} e^{-d \tau}) + g \tau \right) + K_c
\]

(103)

\[
= \frac{\kappa \theta}{\xi^2} (-2 \ln(1 - \hat{g} e^{-d \tau}) + (m - d) \tau) + K_c
\]

We then fix \(K_c\) by \(C_0 = \gamma\)

\[
C_0 = -2 \frac{\kappa \theta}{\xi^2} \ln(1 - \hat{g}) + K_c = \gamma \Rightarrow K_c = 2 \frac{\kappa \theta}{\xi^2} \ln(1 - \hat{g}) + \gamma
\]

(104)

Therefore we have the solution for \(C\)

\[
C = \frac{\kappa \theta}{\xi^2} \left( 2 \ln \frac{1 - \hat{g}}{1 - \hat{g} e^{-d \tau}} + (m - d) \tau \right) + \gamma
\]

(105)

Combining the solutions in (101) and (106), the \(h_t\) is of the following form

\[
h_t = \mathbb{E}_t [g_{x_t, v_t}] = \exp(C + D v_t + i \alpha x_t)
\]

where

\[
m = \kappa - \alpha \xi \rho i, \quad d = \pm \sqrt{m^2 + \xi^2 \alpha (i + \alpha)}, \quad g = \frac{m - d}{m + d}, \quad \hat{g} = \frac{m - d - \delta \xi^2}{m + d - \delta \xi^2}
\]

\[
C = \frac{\kappa \theta}{\xi^2} \left( 2 \ln \frac{1 - \hat{g}}{1 - \hat{g} e^{-d \tau}} + (m - d) \tau \right) + \gamma, \quad D = \frac{m + d g - \hat{g} e^{-d \tau}}{\xi^2} \frac{1 - \hat{g} e^{-d \tau}}{1 - \hat{g} e^{-d \tau}}
\]

(106)
The marginal characteristic function of $x_T$ is given by $\gamma = 0$ and $\delta = 0$ where $\hat{g} = g$

$$\phi^x_{x_T} = \exp\left(\frac{\kappa \theta}{\xi^2} \left(2 \ln \frac{1 - g}{1 - ge^{-\delta \tau}} + (m - d) \tau \right) + \frac{m - d}{\xi^2} \frac{1 - e^{-\delta \tau}}{1 - ge^{-\delta \tau}} v_t + i \alpha x_t\right)$$

where $m = \kappa - \alpha \xi \rho i$, $d = \pm \sqrt{m^2 + \xi^2 \alpha (i + \alpha)}$, $g = \frac{m - d}{m + d}$

### 2.3.3. Vanilla Option Prices

Once we know the analytical form of the characteristic function $\phi^x_{x_T}$ of the centered log-spot $x_T$, we are able to compute the vanilla option prices using inversion methods. Here, we are going to discuss two methods, which treat the option price function analogous to the cumulative density function or the probability density function, respectively. In addition, we also summarize the original Heston’s method [22].

#### 2.3.3.1. Analogy to Cumulative Density Function

For $t \leq T$, using a European call option as an example we change the variable to $x_t$ as in (48)

$$C_{T,K} = \mathbb{E}_t \left[ \frac{M_t}{M_T} (S_T - K)^+ \right] = e^{-r \tau} F_{t,T} \mathbb{E}_t [(e^{x_T} - e^K)^+]$$

with $\tau = T - t$, $\mu = r - q$, $F_{t,T} = S_t e^{\mu \tau}$, $x_T = \ln \frac{S_T}{F_{t,T}}$, $\mathcal{K} = \ln \frac{K}{F_{t,T}}$

where the cash account $M_T = M_t \exp \left(\int_t^T r_u du\right)$ and the interest rate $r$ and dividend yield $q$ are assumed constant. We then define an option forward price in percentage of the underlying forward as a function of moneyness $\mathcal{K}$ defined as the log strike over forward

$$C_{\mathcal{K}} = \frac{C_{T,K}}{e^{-r \tau} F_{t,T}} = \mathbb{E}_t [(e^{x_T} - e^K)^+] = \mathbb{E}_t [(e^{x_T} - e^K) \mathbb{1}_{\{x_T > \mathcal{K}\}}]$$

where $\mathbb{1}_{\{x > X\}} = \begin{cases} 1 & \text{if } x > X \\ 0 & \text{otherwise} \end{cases}$

Given we know the characteristic function of the log-spot process $x_T$, we can derive the Fourier transform of the call option, then use numerical inversion to obtain option prices directly [14]. Since $C_{T,K} \in \mathbb{C}$.
[0, e^{-rT}F_{sT}], we have $C_K \in [0,1]$, which can be treated as a cumulative density function on $K$, the Fourier transform of the option price is then given by

$$
\chi^C = \int_{K \in \mathbb{R}} e^{i\omega K} dC_K = e^{i\omega K} C_K \bigg|_{K = -\infty}^\infty - \int_{\mathbb{R}} i\omega e^{i\omega K} C_K dK
$$

$$
= -e^{-i\omega \infty} - \int_{\mathbb{R}} i\omega e^{i\omega K} C_K dK \quad \text{(by } C_\infty = 0 \text{ and } C_{-\infty} = 1) 
$$

$$
= -e^{-i\omega \infty} - \int_{\Omega} i\omega \int_{\mathbb{R}} (e^x - e^K) \mathbb{1}\{x > K\} dP_x^{xT} dK
$$

$$
= -e^{-i\omega \infty} - \int_{\Omega} \int_{\mathbb{R}} i\omega (e^{i\omega K} + x - e^{i(\omega + 1)K}) \mathbb{1}\{x > K\} dK dP_x^{xT} \quad \text{(by Fubini’s theorem)}
$$

$$
= -e^{-i\omega \infty} - \int_{\Omega} \int_{-\infty}^\infty i\omega (e^{i\omega K} + x - e^{i(\omega + 1)K}) dK dP_x^{xT}
$$

$$
= -e^{-i\omega \infty} - \int_{\Omega} \left(e^{i(\omega + 1)K} - e^{-i\omega \infty + x} - \frac{i\omega e^{i(\omega + 1)K}}{i\omega + 1} + \frac{i\omega e^{-i(\omega + 1)\infty}}{i\omega + 1 = 0}\right) dP_x^{xT}
$$

$$
= -e^{-i\omega \infty} + e^{-i\omega \infty} \int_{\Omega} e^x dP_x^{xT} - \int_{\Omega} \left(e^{i(\omega + 1)K} - \frac{i\omega e^{i(\omega + 1)K}}{i\omega + 1}\right) dP_x
$$

$$
= -\frac{1}{i\omega + 1} \int_{\Omega} e^{i(\omega - i)x} dP_x^{xT} \quad \text{(by } \int_{\Omega} e^x dP_x^{xT} = \frac{1}{F_T} \mathbb{E}_0[S_T] = 1) 
$$

$$
= -\frac{\phi^{xT}_{\omega-i}}{i\omega + 1}
$$

The $\chi^C$ is the characteristic function of the option forward price $C_K$, which is treated just as a cumulative density function. The price is then given by the Levy’s Inversion Formula (79) [15]

$$
C_K = \frac{\chi_0}{2} + \frac{1}{2\pi} \int_{\mathbb{R}^+} e^{i\omega K} \frac{\chi^C - e^{-i\omega K} \chi_0}{i\omega} d\omega
$$

(112)
\[
\begin{align*}
&= \frac{1}{2} + \frac{1}{2\pi} \int_{\mathbb{R}^+} e^{i\omega \zeta} \frac{\phi^{x_T}_{-\omega-i} + e^{-i\omega \zeta} \phi^{x_T}_{\omega+i}}{i\omega} d\omega \\
&= \frac{1}{2} + \frac{1}{2\pi} \int_{\mathbb{R}^+} \left( e^{i\omega \zeta} \phi^{x_T}_{-\omega-i} + e^{-i\omega \zeta} \phi^{x_T}_{\omega+i} \right) d\omega \\
&= \frac{1}{2} - \frac{1}{2\pi} \int_{\mathbb{R}^+} \left( e^{i\omega \zeta} \phi^{x_T}_{-\omega-i} + e^{-i\omega \zeta} \phi^{x_T}_{\omega+i} \right) d\omega
\end{align*}
\] (by \( \chi_0 = -\phi_{-i} = 1 \))

In the integrand of (112), the \( \phi^{x_T}_{u\omega-i} \) for \( u = \pm 1 \) can be derived from the characteristic function \( \phi^{x_T}_{\alpha} \) in (108) and given as follows (note that \( x_t = 0 \))

\[
\phi^{x_T}_{u\omega-i} = \exp \left( \frac{\kappa \theta}{\xi^2} \left( 2 \ln \frac{1 - g}{1 - ge^{-d\tau}} + (m - d)\tau \right) + \frac{m - d}{\xi^2} \frac{1 - e^{-d\tau}}{1 - ge^{-d\tau}} v_t \right) 
\]

with \( \tau = T - t, \ m = \kappa - u\omega \xi i - \xi \rho, \ d = \pm \sqrt{m^2 + \xi^2 \omega(\omega - u)i}, \ g = \frac{m - d}{m + d} \) \hspace{1cm} (113)

The inversion formula in (112) involves evaluation of the \( \phi^{x_T}_{u\omega-i} \) function twice, which is less efficient.

Since \( C_\zeta \) is real-valued, we may use (88) to perform the inversion

\[
C_\zeta = \frac{X_0}{2} - \frac{1}{\pi} \int_{\mathbb{R}^+} \Im \left[ e^{-i\omega \zeta} \frac{\alpha^c}{\omega} \right] \frac{1}{\pi} \int_{\mathbb{R}^+} \Im \left[ e^{-i\omega \zeta} \frac{\phi^{x_T}_{\omega-i}}{i\omega^2 + \omega} \right] d\omega
\]

where \( \phi^{x_T}_{\omega-i} \) is given in (113) with \( u = 1 \) and the integral can be estimated numerically by Gauss-Laguerre quadrature. Once we have \( C_\zeta \) for log-moneyness, the call option price can be computed by (110)

\[
C_{T,K} = C_\zeta e^{-rt} S_t e^{\mu t} = C_\zeta S_t e^{-qt} \tag{115}
\]

and the put option price by call-put parity

\[
C_{T,K} - P_{T,K} = e^{-rt} (F_{t,T} - K) \Rightarrow P_{T,K} = C_{T,K} - S_t e^{-qt} + e^{-rt} K \tag{116}
\]

2.3.3.2. Analogy to Probability Density Function

In this section, we treat \( C_\zeta \) analogous to a probability density [16] [17] and define in terms of generalized Fourier transform with \( z = z_r - iz_i, \ z_r \in \mathbb{R}, \ z_i \in \mathcal{D} \subseteq \mathbb{R}^+ \)

\[
\chi_p^p = \int_{\mathbb{R}} e^{iz \zeta} C_\zeta d\zeta = \int_{\mathbb{R}} e^{iz \zeta} \mathbb{E}_t [(e^{x_T} - e^\zeta) \mathbb{1}\{x_T > \zeta\}] d\zeta \tag{117}
\]
For some return distributions, the return transform $\phi_{x-i}^{x_T}$ is well-defined only when $z_i$ is in a subset of the real line. We use $\mathcal{D} \subseteq \mathbb{R}^+$ to denote the subset that both guarantees the convergence of $e^{izK}$ and $e^{iz(z-i)K}$ as $K \to -\infty$, and assures the finiteness of the transform $\phi_{z-i}^{x_T}$.

The option forward price $C_K$ is then given by the inverse Fourier transform

\[
C_K = \frac{1}{2\pi} \int_{-\infty}^{\infty} e^{-izK} \chi_z^p dz = \frac{1}{2\pi} \int_{-\infty}^{\infty} e^{-izK} \chi_z^p d(z_r - iz_i) = \frac{1}{2\pi} \int_{\mathbb{R}} e^{-izK} \chi_z^p dz_r
\]

\[
= \frac{1}{2\pi} \int_{\mathbb{R}} e^{-izK} \frac{\phi_{z-i}^{x_T}}{iz - z^2} dz_r = \frac{1}{\pi} \int_{\mathbb{R}^+} \Re \left[ e^{-izK} \frac{\phi_{z-i}^{x_T}}{iz - z^2} \right] dz_r
\]

The last equality holds because $C_K$ is real, which implies that the function $\chi_z^p$ is odd in its imaginary part and even in its real part. The functional form of $\phi_{z-i}^{x_T}$ can be derived again from (108) in a similar way

\[
\phi_{z-i}^{x_T} = \exp \left( \frac{k\theta}{\xi^2} \left( 2 \ln \frac{1 - g}{1 - ge^{-d\tau}} + (m - d) \tau \right) + \frac{m - d}{\xi^2} \frac{1 - e^{-d\tau}}{1 - ge^{-d\tau} v_t} \right)
\]

\[
m = \kappa - (z_i + 1)\xi \rho - z_r\xi \rho i, \quad d = \pm \sqrt{m^2 + \xi^2(z_r - i(z_i + 1))(z_r - iz_i)}, \quad g = \frac{m - d}{m + d}
\]

2.3.3.3. Heston’s Original Solution

At time $t$, the price of a European call on a stock (assuming no dividend payment to qualify as a numeraire) with spot $S_t$ and strike $K$ is given by the no-arbitrage formula

\[
C_{t,K,S_t} = \mathbb{E}_t \left[ \frac{M_t}{M_T} (S_T - K)^+ \right] = \mathbb{E}_t \left[ \frac{M_t}{M_T} S_T \mathbb{1}_{\{S_T > K\}} \right] - K \mathbb{E}_t \left[ \frac{M_t}{M_T} \mathbb{1}_{\{S_T > K\}} \right]
\]

\[
= \mathbb{E}_t^S \left[ \frac{S_t}{S_T} S_T \mathbb{1}_{\{S_T > K\}} \right] - K \mathbb{E}_t^T \left[ \frac{B_{t,T}}{B_{T,T}} \mathbb{1}_{\{S_T > K\}} \right] \quad \text{(numeraire $M_t \to S_t$ and $M_t \to B_{t,T}$)}
\]
\[ S_t \mathbb{P}_t^S [S_T > K] - KB_t, T \mathbb{P}_t^T [S_T > K] \]
\[ = S_t \mathbb{P}_t^S [S_T > K] - KB_t, T \mathbb{P}_t^T [S_T > K] \]

where \( \mathbb{P}_t^S [S_T > K] \) and \( \mathbb{P}_t^T [S_T > K] \) are both conditional probabilities of spot finishing in-the-money at maturity. The \( \mathbb{P}_t^S [S_T > K] \) is computed under the measure associated with the stock as numeraire, whereas the \( \mathbb{P}_t^T [S_T > K] \) is computed under \( T \)-forward measure associated with a zero coupon bond \( B_{t, T} \) as underlying numeraire [18]. In Black-Scholes model

\[ C_{T, K}^{BS} = S_t N(d_+) - KB_{t, T} N(d_-) \quad (121) \]

the \( \mathbb{P}_t^S [S_T > K] \) and \( \mathbb{P}_t^T [S_T > K] \) are computed as \( N(d_+) \) and \( N(d_-) \) respectively. Since the drift adjustment due to change of numeraire is

\[ d\mathbb{W} \text{ Under } N = d\mathbb{W} \text{ Under } Q - \sigma_N dt \quad (122) \]

where \( N \) denotes the measure associated with numeraire \( N \) and \( Q \) the risk neutral measure. The stock process under the measure associated with itself as the numeraire is given by

\[ \frac{dS_t}{S_t} = rt + \sigma d\mathbb{W}_t = (r + \sigma^2)dt + \sigma d\mathbb{W}_t^S \quad (123) \]

The total drift adjustment \( \sigma^2 \tau \) for period \( \tau = T - t \) is then normalized by the total volatility \( \sigma \sqrt{\tau} \) of the stock to give a shift term \( \sigma \sqrt{\tau} \) as the difference between \( d_+ \) and \( d_- \) in the classic Black-Scholes formula.

Suppose we use the definition in (109) for a call option, the two conditional probabilities can be expressed as

\[ C_{T, K} = e^{-rt} F_{t, T} \mathbb{E}_t [(e^{x_T} - e^K)^+] = e^{-rt} F_{t, T} \left( \mathbb{E}_t [e^{x_T} 1\{x_T > K}\} - e^K \mathbb{E}_t [1\{x_T > K}\}] \right) \]
\[ = e^{-rt} F_{t, T} P_+^K - e^{-rt} K P_-^K \quad \text{with} \quad P_+^K = \mathbb{E}_t [e^{x_T} 1\{x_T > K}\}, \quad P_-^K = \mathbb{E}_t [1\{x_T > K}\}] \quad (124) \]

Because in Heston model the \( P_+^K \) and \( P_-^K \) are not available in closed form, Heston [22] firstly derived the characteristic functions (i.e. the Fourier transforms) of \( P_+^K \) and \( P_-^K \) by solving the PDE (93) for each of them, and then found inverse of the two characteristic functions to obtain the option price. Since we have
already derived the characteristic function of $x_T$ as in (108), we can easily derive those for $P^+_\mathcal{K}$ and $P^-_{\mathcal{K}}$.

For example, the $P^+_\mathcal{K}$ can be treated as a CDF and then the characteristic function of $P^+_\mathcal{K}$ becomes

$$
\chi^+_\omega = \int_{\mathcal{K} \in \mathbb{R}} e^{i\omega \mathcal{K}} dP^+_\mathcal{K} = e^{i\omega \mathcal{K}} P^+_\mathcal{K}|_{\mathcal{K}=\infty} - \int_{\mathbb{R}} i\omega e^{i\omega \mathcal{K}} P^+_\mathcal{K} d\mathcal{K}
$$

$$
= -e^{-i\omega \infty} - \int_{\mathbb{R}} i\omega e^{i\omega \mathcal{K}} P^+_\mathcal{K} d\mathcal{K} \quad \text{(by } P^+_\infty = 0 \text{ and } P^+_{\infty} = \mathbb{E}_t[e^{x_T}] = 1) \)
$$

$$
= -e^{-i\omega \infty} - \int_{\mathbb{R}} i\omega e^{i\omega \mathcal{K}} \int_{\Omega} e^x \mathbb{1}\{x > \mathcal{K}\} dP^x_T d\mathcal{K}
$$

$$
= -e^{-i\omega \infty} - \int_{\mathbb{R}} \int_{\Omega} i\omega e^{i\omega \mathcal{K}+x} \mathbb{1}\{x > \mathcal{K}\} d\mathcal{K} dP^x_T \quad \text{(by Fubini’s theorem)}
$$

$$
= -e^{-i\omega \infty} - \int_{\Omega} \int_{-\infty}^{\infty} i\omega e^{i\omega \mathcal{K}+x} d\mathcal{K} dP^x_T = -e^{-i\omega \infty} - \int_{\Omega} e^{i\omega \mathcal{K}+x} \mathbb{1}_{\mathcal{K}=\infty} dP^x_T
$$

$$
= -e^{-i\omega \infty} - \int_{\Omega} [e^{i(\omega-1)x} - e^{-i\omega \infty + x}] dP^x_T
$$

$$
= -e^{-i\omega \infty} + e^{-i\omega \infty} \int_{\Omega} e^x dP^x_T - \int_{\Omega} e^{i(\omega-1)x} dP_x
$$

$$
= - \int_{\Omega} e^{i(\omega-1)x} dP^x_T \quad \text{(by } \int_{\Omega} e^x dP^x_T = \mathbb{E}_t[e^{x_T}] = 1) \)
$$

$$
= -\phi^{x_T}_{\omega-1}
$$

Similarly, the characteristic function of $P^-_{\mathcal{K}}$ is given by

$$
\chi^-_{\omega} = \int_{\mathcal{K} \in \mathbb{R}} e^{i\omega \mathcal{K}} dP^-_{\mathcal{K}} = e^{i\omega \mathcal{K}} P^-_{\mathcal{K}}|_{\mathcal{K}=\infty} - \int_{\mathbb{R}} i\omega e^{i\omega \mathcal{K}} P^-_{\mathcal{K}} d\mathcal{K}
$$

$$
= -e^{-i\omega \infty} - \int_{\mathbb{R}} i\omega e^{i\omega \mathcal{K}} P^-_{\mathcal{K}} d\mathcal{K} \quad \text{(by } P^-_\infty = 0 \text{ and } P^-_{\infty} = 1) \)
$$

$$
= -e^{-i\omega \infty} - \int_{\mathbb{R}} i\omega e^{i\omega \mathcal{K}} \int_{\Omega} \mathbb{1}\{x > \mathcal{K}\} dP^x_T d\mathcal{K} = -e^{-i\omega \infty} - \int_{\Omega} \int_{\mathbb{R}} i\omega e^{i\omega \mathcal{K}} \mathbb{1}\{x > \mathcal{K}\} d\mathcal{K} dP^x_T
$$

$$
= -e^{-i\omega \infty} - \int_{\Omega} \int_{-\infty}^{\infty} i\omega e^{i\omega \mathcal{K}} d\mathcal{K} dP^x_T = -e^{-i\omega \infty} - \int_{\Omega} e^{i\omega \mathcal{K}} \mathbb{1}_{\mathcal{K}=\infty} dP^x_T
$$

(126)
\[ = -e^{-i\omega \infty} - \int_{\Omega} (e^{i\omega x} - e^{-i\omega \infty}) dP_x^{x_T} = -e^{-i\omega \infty} + e^{-i\omega \infty} \int_{\Omega} dP_x^{x_T} - \int_{\Omega} e^{i\omega x} dP_x \]

\[ = -\phi^{x_T}_x \]

The \( P^+ \) and \( P^- \) are then obtained through the inverse of \( \chi_+^x \) and \( \chi^-_x \) respectively. Given that \( \chi^+_0 = \chi^-_0 = 1 \), the Heston’s vanilla option price formula can be derived from \( \phi^{x_T}_x \) in (108) and the inverse formula in (88) directly. Here we summarize the solution as follows

\[
\text{HestonVanilla}(\kappa, \theta, \xi, \rho, q, \nu_t, S_t, K, \tau, \eta) = e^{-q\tau} S_t (P_+ - \eta) - e^{-\tau r} K (P_- - \eta)
\]

with \( \eta = \begin{cases} 0 & \text{for call} \\ 1 & \text{for put} \end{cases} \), \( m = \begin{cases} \kappa - \rho \xi \omega i - \xi \rho & \text{for } P_+ \\ \kappa - \rho \xi i & \text{for } P_- \end{cases} \), \( u = \begin{cases} 1 & \text{for } P_+ \\ -1 & \text{for } P_- \end{cases} \)

\( g = \frac{m - d}{m + d} \), \( d = \pm \sqrt{m^2 + \xi^2 \omega (\omega - ui)} \), \( C = \frac{\kappa \theta}{\xi^2} \left( 2 \ln \frac{1 - g}{1 - ge^{-d\tau}} + (m - d) \tau \right) \)

\( D = \frac{m - d}{\xi^2} \frac{1 - e^{-d\tau}}{1 - ge^{-d\tau}} \), \( \chi_\omega = \exp \left( C + D \nu_t + i\omega \ln S_t e^{(r-q)\tau} \right) \)

\[ P = \frac{1}{2} + \frac{1}{\pi} \int_{\mathbb{R}^+} \Im \left[ \frac{e^{-i\omega \ln K} \chi_\omega}{\omega} \right] d\omega \]

where the \( \eta \) denotes the option type with \( \eta = 1 \) for calls and \( \eta = -1 \) for puts. In Heston’s original formula [22], the negative solution of \( d \) is used, which makes the calculation of the complex logarithm prone to numerical instabilities. This is because taking the principal value of the logarithm causes \( C \) to jump discontinuously each time the imaginary part of the argument of the logarithm crosses the negative real axis (i.e. discontinuity due to branch cut of complex numbers), especially for long maturities. Albrecher et al. [19] presents an extensive study proving that both solutions are completely equivalent from a theoretical point of view, it is also mentioned that rather than using the negative solution of \( d \), the positive \( d \) guarantees numerical stability of the resulting formula under a full dimensional and unrestricted parameter space.

2.4. Piecewise Time Dependent Heston Model
The time-dependent Heston model we present here was proposed by Elices [20] in 2008. The model relies on the characteristic function of the two dimensional Markov process, which we have derived in (107). It bootstraps a series of piecewise constant Heston parameters starting from the earliest maturity, each parameter set for a period of time, which allows the model to fit to a term structure of the implied volatility surfaces.

Consider a Markov $n$-dimensional stochastic process $Y$ (i.e. $Y = \left( x_t \atop \nu_t \right)$ in Heston model), we define $p_{Y_t}^{s,t}$ the transition probability density function for having $Y = y_t$ at time $t$ conditional on $Y = y_s$ at time $s$ where $o \leq s \leq t \leq T$. Its characteristic function $\phi^{s,t}_{\omega}$ is the multi-dimensional Fourier transform of the transition density $p_{Y_t}^{s,t}$ such that

$$
\phi^{o,t}_{\omega} = \int_{\mathbb{R}^n} e^{i \omega' y_t} p_{Y_t}^{o,t} dy_t = \int_{\mathbb{R}^n} e^{i \omega' y_t} \int_{\mathbb{R}^n} p_{Y_t}^{s,t} p_{y_s}^{o,s} dy_s dy_t = \int_{\mathbb{R}^n} \int_{\mathbb{R}^n} e^{i \omega' y_t} p_{Y_t}^{s,t} p_{y_s}^{o,s} dy_s dy_t
$$

$$
= \int_{\mathbb{R}^n} \int_{\mathbb{R}^n} e^{i \omega' y_t} p_{Y_t}^{s,t} dy_t p_{y_s}^{o,s} dy_s = \int_{\mathbb{R}^n} \phi^{s,t}_{\omega} p_{y_s}^{o,s} dy_s
$$

Consider a family of exponential characteristic functions with exponent linear in the state $y_s$ at time $s$

$$
\phi^{s,t}_{\omega} = \exp(C^{s,t}(\omega) + D^{s,t}(\omega)' y_s)
$$

we have

$$
\phi^{o,t}_{\omega} = \int_{\mathbb{R}^n} \exp(C^{s,t}(\omega) + D^{s,t}(\omega)' y_s) p_{y_s}^{o,s} dy_s = \exp(C^{s,t}(\omega)) \int_{\mathbb{R}^n} \exp(D^{s,t}(\omega)' y_s) p_{y_s}^{o,s} dy_s
$$

$$
= \exp(C^{s,t}(\omega)) \int_{\mathbb{R}^n} \exp \left[ i \left( -i D^{s,t}(\omega) \right)' y_s \right] p_{y_s}^{o,s} dy_s = \exp(C^{s,t}(\omega)) \phi^{o,s}(-i D^{s,t}(\omega))
$$

$$
= \exp \left( C^{s,t}(\omega) + C^{o,s}(-i D^{s,t}(\omega)) + D^{o,s}(-i D^{s,t}(\omega))' y_o \right)
$$

Identifying terms between the (129) for $s = o$ and the (130), we find that

$$
C^{o,t}(\omega) = C^{s,t}(\omega) + C^{o,s}(-i D^{s,t}(\omega)) \quad \text{and} \quad D^{o,t}(\omega) = D^{o,s}(-i D^{s,t}(\omega))
$$

(131)
Formula (131) gives a recursive definition of $C^{o,t}(\omega)$ and $D^{o,t}(\omega)$, which can be used to bootstrap the Heston model parameters for each time period starting from the earliest maturity. For illustrative purpose, the first 3 periods are presented below

<table>
<thead>
<tr>
<th>Period</th>
<th>$C$</th>
<th>$D$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_0 \rightarrow t_1$</td>
<td>$C^{0,1}(\omega)$</td>
<td>$D^{0,1}(\omega)$</td>
</tr>
<tr>
<td>$t_1 \rightarrow t_2$</td>
<td>$C^{1,2}(\omega) + C^{0,1}(-iD^{1,2}(\omega))$</td>
<td>$D^{0,1}(-iD^{1,2}(\omega))$</td>
</tr>
<tr>
<td>$t_2 \rightarrow t_3$</td>
<td>$C^{2,3}(\omega) + C^{1,2}(-iD^{2,3}(\omega)) + C^{0,1}(-iD^{1,2}(-iD^{2,3}(\omega)))$</td>
<td>$D^{0,1}(-iD^{1,2}(-iD^{2,3}(\omega)))$</td>
</tr>
</tbody>
</table>
3. **Heston Model: PDE by Finite Element Method**

Valuation of financial derivatives has become a very important subject in modern financial theory and practice. In 1973, Black and Scholes introduced a simple formula [21] to price European-style options under a few strong assumptions. It is the first successful attempt to provide an arbitrage free valuation of financial derivatives. However due to limitations, it fails to capture some critical features observed in financial markets, such as heavy tails of return, skewness and smile in implied volatility, clustering and autocorrelation in volatility, etc. Many approaches have been proposed to address these issues. One of them is to employ stochastic volatility models to describe the price dynamics in financial markets, in which the volatility of the underlying is assumed to be stochastic. Heston [22] proposed a stochastic volatility model in 1993. It extends the Black-Scholes model and includes it as a special case. One major advantage of the Heston model is that it can be solved in closed-form for vanilla options. For exotic options, closed-form solutions are generally not available, numerical methods must be considered for option pricing. Monte Carlo simulations, though easy to implement, are not very suitable for computing risk sensitivities. It turns out that solving partial differential equation (PDE) is not only more numerically efficient but also more stable in risk sensitivity estimation. Traditionally, the PDE’s that arise from option pricing are solved dominantly by finite difference method (FDM). FDM is straightforward to implement, but meanwhile it also imposes many constraints, such as, sufficiently smooth terminal and boundary conditions, rectangular domains, etc. To relax these constraints, it becomes the primary motivation of this essay to develop an efficient finite element methods (FEM) for solving the PDE’s. Many studies have been focused on this topic. Topper [23] provides an excellent introduction to the FEM in the context of financial engineering applications. Previous work by Winkler et al. [24] illustrates an application of FEM to valuation of vanilla option in Heston stochastic volatility model. Achdou and Tchou [25] Propose a finite element analysis for Black-Scholes equation with stochastic volatility process. A recent publication by Miglio and Sgarra [26] discusses an application of finite element method for option pricing in a stochastic volatility model with jumps, known as the Bates model.
In this chapter, we will present an application of finite element method in the Heston model to price exotic options (specifically, the double barrier knock-out options). It presents a detailed implementation of the finite element method and its application in option pricing. The numerical results and conclusions are outlined in the end.

3.1. The Partial Differential Equation

Let $U(t,S_t,v_t) = \frac{1}{D_t} \mathbb{E}_t[D_T U(T,S_T,v_T)]$ be the price of a contingent payoff $U(T,S_T,v_T)$ that occur at maturity $T$. Assuming deterministic interest rate, we have the dynamics of $D_t U(t,S_t,v_t)$ in Heston model (22) under risk neutral measure $\mathbb{Q}$, that is

$$
\frac{1}{D_t} d(D_t U(t,S_t,v_t)) = dU - rU dt
$$

$$
= \frac{\partial U}{\partial t} dt + \frac{\partial U}{\partial S} dS + \frac{1}{2} \frac{\partial^2 U}{\partial S^2} dS dS + \frac{\partial U}{\partial v} dv + \frac{1}{2} \frac{\partial^2 U}{\partial v^2} dv dv + \frac{\partial^2 U}{\partial v \partial S} dv dS - rU dt
$$

$$
= \frac{\partial U}{\partial t} dt + \frac{\partial U}{\partial S} (r - q) S dt + \frac{\partial U}{\partial S} S \sqrt{v} d\tilde{W}_1 + \frac{\nu S^2}{2} \frac{\partial^2 U}{\partial S^2} dt + \frac{\partial U}{\partial v} \kappa (\theta - v) dt + \frac{\partial U}{\partial v} \xi \sqrt{v} d\tilde{W}_2
$$

$$
+ \frac{\nu \xi^2}{2} \frac{\partial^2 U}{\partial v^2} dt + S \nu \xi \rho \frac{\partial^2 U}{\partial v \partial S} dt - rU dt
$$

Since the $D_t U(t,S_t,v_t)$ is a martingale under $\mathbb{Q}$, the $dt$-term in (132) must vanish, which defines a PDE that the derivative price $U$ must follow

$$
\frac{\partial U}{\partial t} + \nu \xi^2 \frac{\partial^2 U}{\partial v^2} + S \nu \xi \rho \frac{\partial^2 U}{\partial v \partial S} + \frac{\nu S^2}{2} \frac{\partial^2 U}{\partial S^2} + \kappa (\theta - v) \frac{\partial U}{\partial v} + (r - q) S \frac{\partial U}{\partial S} - rU = 0
$$

(133)

Prices of different types of derivative securities can be obtained by solving the PDE (133) under different terminal and boundary conditions. We firstly define a rectangle domain

$$
\Omega := \{(S,v) : S \in (S_{\min},S_{\max}), \ v \in (v_{\min},v_{\max})\}
$$

(134)

The domain boundaries are then taken as

$$
\Gamma_1 := \{(S,v) : S \in (S_{\min},S_{\max}), \ v = v_{\min}\}
$$

(135)

$$
\Gamma_2 := \{(S,v) : S \in (S_{\min},S_{\max}), \ v = v_{\max}\}
$$
\[ \Gamma_3 := \{(S, v) : S = S_{\text{min}}, \ v \in [v_{\text{min}}, v_{\text{max}}]\} \]

\[ \Gamma_4 := \{(S, v) : S = S_{\text{max}}, \ v \in [v_{\text{min}}, v_{\text{max}}]\} \]

The terminal condition is defined at option maturity. A vanilla call \((\eta = 1)\) or put \((\eta = -1)\) that matures in \(\tau = T - t\) would have a terminal condition defined by the maturity payoff function

\[ U(T, S_T, v_T) = \left(\eta(S_T - K)\right)^+ \] (136)

Given a strike \(K\) and a properly defined min/max range for \(S\) and \(v\), the Dirichlet boundary conditions for a vanilla option can be defined as follows (here we choose Dirichlet boundary condition because it is easier to handle. Ideally a zero gamma boundary condition should be used, however the choice of boundary condition becomes insignificant when the domain for \(S\) and \(v\) gets large enough)

\[ U(t, S_t, v_{\text{min}}) = \left(\eta(S_t e^{-q\tau} - Ke^{-r\tau})\right)^+ \forall (S, v) \in \Gamma_1 \]

\[ U(t, S_t, v_{\text{max}}) = \frac{1 + \eta}{2} S_t e^{-q\tau} + \frac{1 - \eta}{2} Ke^{-r\tau} \forall (S, v) \in \Gamma_2 \] (137)

\[ U(t, S_{\text{min}}, v_t) = \frac{1 - \eta}{2} \left(\eta(S_{\text{min}} e^{-q\tau} - Ke^{-r\tau})\right)^+ \forall (S, v) \in \Gamma_3 \]

\[ U(t, S_{\text{max}}, v_t) = \frac{1 + \eta}{2} \left(\eta(S_{\text{max}} e^{-q\tau} - Ke^{-r\tau})\right)^+ \forall (S, v) \in \Gamma_4 \]

For a European up-and-out-down-and-out double barrier call/put option, the terminal condition is of the same form. The Dirichlet boundary conditions change a bit as the min/max value for \(S\) is limited to the up and down barriers

\[ U(t, S_t, v_{\text{min}}) = \left(\eta(S_t e^{-q\tau} - Ke^{-r\tau})\right)^+ \forall (S, v) \in \Gamma_1 \]

\[ U(t, S_t, v_{\text{max}}) = 0 \ \forall (S, v) \in \Gamma_2 \] (138)

\[ U(t, S_{\text{min}}, v_t) = 0 \ \forall (S, v) \in \Gamma_3 \]

\[ U(t, S_{\text{max}}, v_t) = 0 \ \forall (S, v) \in \Gamma_4 \]

Equation (133) can be further simplified by change of variable, which yields
\[
\frac{\partial w}{\partial t} + \frac{\xi^2 v \partial^2 w}{2 \partial v^2} + \rho \xi v \frac{\partial^2 w}{\partial v \partial y} + \frac{v \partial^2 w}{2 \partial y^2} + \kappa(\theta - v) \frac{\partial w}{\partial v} + \left( r - q - \frac{v}{2} \right) \frac{\partial w}{\partial y} - rw = 0 \tag{139}
\]

where we define \( y_t = \ln \frac{S_t}{k} \) and \( w(t, y_t, v_t) = U(t, S_t, v_t) \). The PDE (139) is actually identical to our previously derived (93) with variables

\[
w(t, y_t, v_t) = e^{rt}h(t, x_t, v_t), \quad x_t = \ln \frac{S_t}{S_0} - (r - q)t \tag{140}
\]

This can be shown by using the chain rule to derive partial derivatives after change of variables

\[
\frac{\partial w}{\partial t} = \left( \frac{\partial}{\partial t} + \frac{\partial x}{\partial t} \frac{\partial}{\partial x} \right) (e^{rt}h) = e^{rt} \left( r h + \frac{\partial h}{\partial t} + \frac{\partial x}{\partial t} \frac{\partial h}{\partial x} \right) = e^{rt} \left( r h + \frac{\partial h}{\partial t} - (r - q) \frac{\partial h}{\partial x} \right)
\]

\[
\kappa(\theta - v) \frac{\partial w}{\partial v} = e^{rt} \kappa(\theta - v) \frac{\partial h}{\partial v}, \quad \frac{\xi^2 v \partial^2 w}{2 \partial v^2} = e^{rt} \frac{\xi^2 v \partial^2 h}{2 \partial v^2}
\]

\[
\left( r - q - \frac{v}{2} \right) \frac{\partial w}{\partial y} = \left( r - q - \frac{v}{2} \right) \frac{\partial (e^{rt}h) \partial x}{\partial y} = e^{rt} \left( r - q - \frac{v}{2} \right) \frac{\partial h}{\partial x} \tag{141}
\]

\[
\frac{v \partial^2 w}{2 \partial y^2} = e^{rt} \frac{v \partial}{2 \partial y} \left( \frac{\partial h}{\partial t} + \frac{\partial h}{\partial x} \frac{\partial x}{\partial y} \right) = e^{rt} \frac{v \partial^2 h}{2 \partial x^2}
\]

\[
\rho \xi v \frac{\partial^2 w}{\partial v \partial y} = \rho \xi v \frac{\partial}{\partial v} \left( \frac{\partial (e^{rt}h)}{\partial x} \frac{\partial x}{\partial y} \right) = e^{rt} \rho \xi v \frac{\partial^2 h}{\partial v \partial x}
\]

For simplicity, the PDE (139) can be written in terms of the gradient and divergence operator

\[
(\partial_t + \nabla \cdot \mathbf{A} \mathbf{v} - \mathbf{b} \cdot \nabla - r)w = 0, \quad \mathbf{A} = \frac{v}{2} \begin{pmatrix} \xi^2 & \rho \xi \\ \rho \xi & 1 \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} \frac{\xi^2}{2} - \kappa(\theta - v) \\ v + \rho \xi - (r - q) \end{pmatrix} \tag{142}
\]

where \( \partial_t = \partial / \partial t \), the gradient operator \( \nabla = \left( \frac{\partial v}{\partial y} \right) \), and the divergence operator \( \nabla \cdot = \left( \frac{\partial v}{\partial y} \right) \). Note that we have the following identities
\[ \nabla \cdot \mathbf{A} \nabla = \left( \frac{\partial}{\partial y} \right)^T \frac{v}{2} \begin{pmatrix} \xi^2 & \rho \xi \\ \rho \xi & 1 \end{pmatrix} \left( \frac{\partial}{\partial y} \right) = \left( \frac{\partial}{\partial y} \right)^T \frac{v}{2} \left( \xi^2 \frac{\partial}{\partial y} + \rho \xi \frac{\partial}{\partial y} \right) \]

\[ = \frac{\xi^2}{2} \frac{\partial}{\partial y} + \frac{\rho \xi}{2} \frac{\partial}{\partial y} + \frac{\rho \xi v}{2} \frac{\partial}{\partial y} + \frac{\rho \xi v}{2} \frac{\partial}{\partial y} \]

\[ = \frac{\xi^2}{2} \frac{\partial}{\partial y} + \frac{\rho \xi}{2} \frac{\partial}{\partial y} + \frac{\rho \xi v}{2} \frac{\partial}{\partial y} + \frac{\rho \xi v}{2} \frac{\partial}{\partial y} \]

\[ = \frac{\xi^2}{2} \frac{\partial}{\partial y} + \frac{\rho \xi}{2} \frac{\partial}{\partial y} + \frac{\rho \xi v}{2} \frac{\partial}{\partial y} + \frac{\rho \xi v}{2} \frac{\partial}{\partial y} \]

Henceforth, we will use \( (\cdot) \) to denote dot product. If we think of column vectors as the standard vectors, the divergence operator can be regarded as transpose of the gradient operator (a bit abuse of notation).

3.2. Numerical Solution of the PDE

The PDE (142) shows a 2-D dynamic problem. To solve this problem, a semi-discretization in time is applied, which yields a series of 2-D static boundary value problems. These 2-D static boundary value problems are then solved numerically by a 2-D finite element method as time advances.

3.2.1. Temporal Discretization

It should be noted that the time advances backwards, such that the initial value is given at the terminal \( t^N = T \) and the solution is sought at \( t^0 = 0 \). Note that the superscript is used to denote the index of time-step. The partial derivative with respect to time is approximated by finite difference method. According to (142), the \( \sigma \)-weighted finite difference scheme for time is given as

\[ \frac{w^{k+1} - w^k}{t^{k+1} - t^k} + (1 - \sigma)(\nabla \cdot \mathbf{A} \nabla - \mathbf{b} \cdot \nabla - r)w^{k+1} + \sigma(\nabla \cdot \mathbf{A} \nabla - \mathbf{b} \cdot \nabla - r)w^k = 0 \]  

In the above equation, the scheme becomes purely explicit when \( \sigma = 0 \), purely implicit when \( \sigma = 1 \), and becomes Crank-Nicolson scheme when \( \sigma = 0.5 \). The computation starts backwards from \( t^N = T \) when \( w^N \) is known which is given by initial condition. At each time-step, provided that the \( w^{k+1} \) is computed
from previous step, the \( w^k \) can be solved from (144) recursively. Rearrangement of (144) gives a new equation below, in which all the \( w^{k+1} \) terms are collected on the right hand side (RHS) and all the \( w^k \) terms on the left hand side (LHS)

\[
\mathcal{L}w^k = \mathcal{R}w^{k+1} \quad \text{where}
\]

\[
\mathcal{L} = \sigma(\nabla \cdot A\nabla - b \cdot \nabla) - c \quad \text{and} \quad \mathcal{R} = (\sigma - 1)(\nabla \cdot A\nabla - b \cdot \nabla) - (c - r)
\]

\[
c = \sigma r + \frac{1}{t^{k+1} - t^k}
\]

3.2.2. Two-Dimensional Finite Element Method

Finite element method is based on weak formulation. It is ideal for solving PDE when the solution lacks smoothness and when the domain is irregular, dynamically changing, and/or unevenly spaced. Valuation of complex exotic options can often exhibit these features, which makes FEM an ideal tool for this type of applications.

3.2.2.1. Weak Formulation

Equation (145) can be transformed to a weak formulation by multiplying both sides with a scalar-valued test function \( \psi \) (in the subsequent sections, we will see that the \( \psi \) can be constructed as a linear combination of basis functions on the 2-D domain)

\[
\int_\Omega \psi \mathcal{L}w^k d\Omega = \int_\Omega \psi \mathcal{R}w^{k+1} d\Omega \quad \text{where}
\]

\[
\text{LHS} = \sigma \int_\Omega \psi \nabla \cdot A\nabla w^k d\Omega - \sigma \int_\Omega \psi \nabla w^k d\Omega - c \int_\Omega \psi w^k d\Omega
\]

\[
\text{RHS} = (\sigma - 1) \int_\Omega \psi \nabla \cdot A\nabla w^{k+1} d\Omega - (\sigma - 1) \int_\Omega \psi \nabla w^{k+1} d\Omega - (c - r) \int_\Omega \psi w^{k+1} d\Omega
\]

where \( \Omega \) is the open domain given in (134).

Assuming that \( u \) is a scalar function and \( \mathbf{v} \) a vector-valued function, both are continuously differentiable, then the integration by parts in multi-dimension is given by
\[
\int_{\Omega} u \nabla \cdot v d\Omega = \int_{\Gamma} u v \cdot n d\Gamma - \int_{\Omega} \nabla u \cdot v d\Omega
\]  
(147)

where \( \Gamma \) denotes the piecewise smooth boundary, and \( n \) denotes the outward unit surface normal to \( \Gamma \).

Applying the transformation (147) to (146) yields (here, it becomes the first Green’s identity)

\[
\int_{\Omega} \psi \mathcal{L} w^k d\Omega = \int_{\Omega} \psi \mathcal{R} w^{k+1} d\Omega \quad \text{where}
\]

\[
\text{LHS} = \sigma \int_{\Gamma} \psi A \nabla w^k \cdot n d\Gamma - \sigma \int_{\Omega} \nabla \psi \cdot A \nabla w^k d\Omega - \sigma \int_{\Omega} \psi b \cdot \nabla w^k d\Omega - c \int_{\Omega} \psi w^k d\Omega
\]

\[
\text{RHS} = (\sigma - 1) \int_{\Gamma} \psi A \nabla w^{k+1} \cdot n d\Gamma - (\sigma - 1) \int_{\Omega} \nabla \psi \cdot A \nabla w^{k+1} d\Omega
\]

\[
- (\sigma - 1) \int_{\Omega} \psi b \cdot \nabla w^{k+1} d\Omega - (c - r) \int_{\Omega} \psi w^{k+1} d\Omega
\]  
(148)

Now let’s define a few functional spaces for the trial solution \( w \) and the test function \( \psi \)

\[
L^2(\Omega) := \{ f : \| f \|_{L^2} < \infty \} \quad \text{where} \quad \| f \|_{L^p} = \left( \int_{\Omega} |f|^p d\Omega \right)^{1/p}
\]

\[
H^1(\Omega) := \{ f \in L^2(\Omega) : \mathcal{D}^1 f \in L^2(\Omega) \}
\]

\[
H^1_0(\Omega) := \{ f \in H^1(\Omega) : f = 0 \text{ on } \Gamma \}
\]

\[
H^1_g(\Omega) := \{ f \in H^1(\Omega) : f = g \text{ on } \Gamma \}
\]  
(149)

where \( \mathcal{D}^1 f \) denotes the first order weak partial derivatives of function \( f \). The \( L^2(\Omega) \) is the Lebesgue space with Euclidean norm which coincides with Hilbert space in this case. The \( H^1(\Omega) \), \( H^1_0(\Omega) \) and \( H^1_g(\Omega) \) are all Sobolev space. The variational form (148) is also called the weak form. The requirements of the solution \( w \) in (148) have been considerably weakened over the strong form in (142). In the weak formulation of the problem, the trial solution \( w \) and the test function \( \psi \) must belong to \( H^1(\Omega) \), it is not necessary that all functions and derivatives be continuous.

\[3.2.2.2. \quad \text{Boundary Conditions}\]

Given different types of boundary conditions, (148) can be further transformed as follows
3.2.2.2.1. Homogeneous Dirichlet Boundary Condition \((w = 0 \text{ on } \Gamma)\)

In this case, \(w \in H^1_0(\Omega)\) and \(\psi \in H^1_0(\Omega)\). The boundary integral \(\int_{\Gamma} \psi \mathbf{A} \nabla w \cdot \mathbf{n} d\Gamma\) in (148) vanishes because \(\psi\) is zero on \(\Gamma\). We gather the known terms on the right hand side, and the unknowns on the left hand side, which yields the equation below

\[
\sigma \int_{\Omega} \nabla \psi \cdot \nabla w^k d\Omega + \sigma \int_{\Omega} \psi \mathbf{b} \cdot \nabla w^k d\Omega + c \int_{\Omega} \psi w^k d\Omega = \]

\[
= (\sigma - 1) \int_{\Omega} \nabla \psi \cdot \nabla w^{k+1} d\Omega + (\sigma - 1) \int_{\Omega} \psi \mathbf{b} \cdot \nabla w^{k+1} d\Omega + (c - r) \int_{\Omega} \psi w^{k+1} d\Omega
\]

(150)

3.2.2.2.2. Inhomogeneous Dirichlet Boundary Condition \((w = g \text{ on } \Gamma)\)

Since \(g\) is a function defined on \(\Gamma\), we assume that there is a function \(G \in H^1(\Omega)\) such that \(G = g\) on \(\Gamma\). It turns out that \(\psi \in H^1_0(\Omega)\) still holds in this case. The solution has the form \(w = u + G\) and \(w \in H^1_0(\Omega)\), where \(G\) is assumed to be known and \(u \in H^1_0(\Omega)\) is unknown. The boundary integral \(\int_{\Gamma} \psi \mathbf{A} \nabla w \cdot \mathbf{n} d\Gamma\) in (148) vanishes because \(\psi\) is zero on \(\Gamma\), which yields

\[
\sigma \int_{\Omega} \nabla \psi \cdot \nabla w^k d\Omega + \sigma \int_{\Omega} \psi \mathbf{b} \cdot \nabla w^k d\Omega + c \int_{\Omega} \psi w^k d\Omega = \]

\[
= (\sigma - 1) \int_{\Omega} \nabla \psi \cdot \nabla w^{k+1} d\Omega + (\sigma - 1) \int_{\Omega} \psi \mathbf{b} \cdot \nabla w^{k+1} d\Omega + (c - r) \int_{\Omega} \psi w^{k+1} d\Omega
\]

Writing \(w^k\) in terms of sum of function \(u^k\) and \(G^k\), we have

\[
\sigma \int_{\Omega} \nabla \psi \cdot \nabla u^k d\Omega + \sigma \int_{\Omega} \psi \mathbf{b} \cdot \nabla u^k d\Omega + c \int_{\Omega} \psi u^k d\Omega = \]

\[
= (\sigma - 1) \int_{\Omega} \nabla \psi \cdot \nabla u^{k+1} d\Omega + (\sigma - 1) \int_{\Omega} \psi \mathbf{b} \cdot \nabla u^{k+1} d\Omega + (c - r) \int_{\Omega} \psi u^{k+1} d\Omega
\]

(151)

\[
- \sigma \int_{\Omega} \nabla \psi \cdot \nabla G^k d\Omega - \sigma \int_{\Omega} \psi \mathbf{b} \cdot \nabla G^k d\Omega - c \int_{\Omega} \psi G^k d\Omega
\]

(152)
3.2.2.2.3. Homogeneous Neumann Boundary Condition ($\nabla w = 0$ on $\Gamma$)

In this case, $w \in H^1(\Omega)$ and $\psi \in H^1(\Omega)$. The boundary integral $\oint_{\Gamma} \psi A \nabla w \cdot n d\Gamma$ in (148) vanishes because $\nabla w \cdot n$ is zero on $\Gamma$ (Note that this is different from that of Dirichlet boundary condition, in which it was the test function $\psi = 0$ on $\Gamma$ that causes the boundary integral to vanish), which yields

$$\int_{\Omega} \nabla \psi \cdot \nabla w^k d\Omega + \int_{\Omega} \psi b \cdot \nabla w^k d\Omega + c \int_{\Omega} \psi w^k d\Omega$$

(153)

It should be noted that the weak form is the same as that of the homogeneous Dirichlet boundary condition; however the trial and test function spaces are different.

3.2.2.2.4. Inhomogeneous Neumann Boundary Condition ($\nabla w = h$ on $\Gamma$)

In this case, the $w \in H^1(\Omega)$ and $\psi \in H^1(\Omega)$. The boundary integral $\oint_{\Gamma} \psi A \nabla w \cdot n d\Gamma$ is known, because $\nabla w \cdot n$ on $\Gamma$ is given. Thus we have

$$\int_{\Omega} \nabla \psi \cdot \nabla w^k d\Omega + \int_{\Omega} \psi b \cdot \nabla w^k d\Omega + c \int_{\Omega} \psi w^k d\Omega$$

(154)

$$= (\sigma - 1) \int_{\Omega} \nabla \psi \cdot A \nabla w^{k+1} d\Omega + (\sigma - 1) \int_{\Omega} \psi b \cdot \nabla w^{k+1} d\Omega + (c - r) \int_{\Omega} \psi w^{k+1} d\Omega$$

$$- (\sigma - 1) \oint_{\Gamma} \psi A h^{k+1} \cdot n d\Gamma + \sigma \oint_{\Gamma} \psi A h^{k} \cdot n d\Gamma$$

Boundaries involve a combination of these boundary conditions will be treated accordingly in a similar manner [27]. The problem studied in this essay is a 2D dynamic problem with inhomogeneous Dirichlet boundary condition. Hence we will primarily focus on (152).

3.2.2.3. Mesh and Basis Functions

A triangular mesh (Figure 1) is used for the computation. The mesh can be either structured or unstructured. We denote the closed domain $\tilde{\Omega} = \Omega \cup \Gamma = [v_{\min}, v_{\max}] \times [y_{\min}, y_{\max}]$. The $\tilde{\Omega}$ can be decomposed into a mesh consisting of triangles $T_i, i = 1, \cdots, N^T$ and their nodes $D_i, i = 1, \cdots, N^D$. A mesh generator provides the mesh information by two arrays. The first array indexes all the nodes with
their corresponding coordinates and boundary ID’s. The boundary ID indicates the boundary information of a node (for example, whether an interior (free) node or a boundary (constrained) node, which boundary it belongs to, etc.). The second array indexes all the triangular elements with their three vortices. Figure 1 illustrates a simple evenly spaced structured mesh used in the computation. It is obvious that the mesh consists of total $I \times J$ squares, $N_T = 2IJ$ triangles and $N_D = (I + 1)(J + 1)$ nodes, where $I$ and $J$ are the spatial resolution for variable $v$ and $y$, respectively. The basis functions $\phi_i$, $i = 1, \ldots, N_D$ are defined as piecewise linear functions that take the value 1 at node $D_i$ and 0 at all other nodes. This basis is usually called Lagrange basis or nodal basis.

To simplify the computation of integrals over the triangles, each triangle is mapped by affine transformation to a reference triangle $\hat{T} = \{ (\hat{\nu}, \hat{y}) : \hat{\nu} \in [0, 1], \hat{y} \in [0, 1 - \hat{\nu}] \}$, where a “hat” is used to denote the variables defined on the reference triangle. The reference triangle is shown in Figure 2. The three vortices $\hat{D}_i, i = 1, 2, 3$ are $(0, 0)$, $(1, 0)$ and $(0, 1)$, respectively. The open circle denotes $\hat{\phi}_i = 0$ at node $\hat{D}_i$ if $i \neq j$ while the full circle denotes $\hat{\phi}_i = 1$ at node $\hat{D}_j$ if $i = j$. Hence the three basis functions on the reference triangle are

$$\hat{\phi}_0(\hat{\nu}, \hat{y}) = 1 - \hat{\nu} - \hat{y}, \quad \hat{\phi}_1(\hat{\nu}, \hat{y}) = \hat{\nu} \quad \text{and} \quad \hat{\phi}_2(\hat{\nu}, \hat{y}) = \hat{y}$$

(155)
3.2.2.4. **Affine Transformation**

The mapping is achieved by affine transformation. Considering an arbitrary triangle $T$ with vortices $D_0 = (v_0, y_0)$, $D_1 = (v_1, y_1)$ and $D_2 = (v_2, y_2)$, the transformation maps the reference triangle $\hat{T}$ onto the triangle $T$ by

$$(v, y) = F(\hat{v}, \hat{y}) = \begin{pmatrix} \hat{v} \\ \hat{y} \end{pmatrix} + \begin{pmatrix} v_0 \\ y_0 \end{pmatrix} \quad \text{and} \quad \mathbf{J} = \begin{pmatrix} v_1 - v_0 & v_2 - v_0 \\ y_1 - y_0 & y_2 - y_0 \end{pmatrix}$$  \hspace{1cm} (156)

3.2.2.5. **Stiffness and Mass Matrix**

Our problem involves inhomogeneous Dirichlet boundary condition, e.g. $w = g$ on $\Gamma$. To incorporate such boundaries, we need to extend the function $g$ to any function $G$ on the closed domain $\bar{\Omega}$ such that $G \in H^1(\Omega)$ and $G = g$ on $\Gamma$. It is not easy to find such $G$ function exactly, but it is easy to define a function $G$ that approximately satisfies the boundary condition. Indeed, following the same definition of the basis function $\phi$, the continuous piecewise linear function $G$ defined by

$$G(D_i) = \begin{cases} 0, & \text{if } D_i \notin \Gamma \\ g_i \phi_i, & \text{if } D_i \in \Gamma \end{cases} \quad \text{where} \quad g_i = g(D_i)$$  \hspace{1cm} (157)

agrees with $g$ at the endpoints of every constrained edge. The linear combination

$$G = \sum_{D_i \in \Gamma} g_i \phi_i$$  \hspace{1cm} (158)

is a sufficiently good approximation to $g$ for the purpose of the finite element method. As discussed previously, the trial solution has the form $w = u + G$ where $G \in H^1(\Omega)$ is known, $u \in H_0^1(\Omega)$ is unknown and $w \in H_G^1(\Omega)$. In the discrete case, at time $t^k$, the trial solution $w^k$ can be expressed as
\[ w^k = G^k + u^k \quad \text{where} \quad G^k = \sum_{D_i \in E} g_i^k \phi_i \quad \text{and} \quad u^k = \sum_{D_i \in \Omega} u_i^k \phi_i \quad (159) \]

The test function \( \psi \in H^1_0(\Omega) \) can be chosen as a linear combination of the basis functions \( \psi = \sum_{D_i \in \Omega} b_i \phi_i \). Although \( b_i \) can be arbitrarily chosen, it has no impact because eventually it will be cancelled out when we multiply the test function to both side of the PDE. As such, it allows us to simplify the test function to \( \psi = \sum_{D_i \in \Omega} \phi_i \).

According to (148), we have

\[
\int_{\Omega} \left( \sum_{D_i \in \Omega} \phi_i \right) L \left( \sum_{D_i \in \Omega} g_i^k \phi_i + \sum_{D_i \in \Omega} u_i^k \phi_i \right) d\Omega = \int_{\Omega} \left( \sum_{D_i \in \Omega} \phi_i \right) R \left( \sum_{D_i \in \Omega} w_i^{k+1} \phi_i \right) d\Omega \quad (160)
\]

and we will try to solve the following linear system

\[
\int_{\Omega} \left( \sum_{D_i \in \Omega} \phi_i \right) L \left( \sum_{D_i \in \Omega} u_i^k \phi_i \right) d\Omega = \int_{\Omega} \left( \sum_{D_i \in \Omega} \phi_i \right) R \left( \sum_{D_i \in \Omega} w_i^{k+1} \phi_i \right) d\Omega - \int_{\Omega} \left( \sum_{D_i \in \Omega} \phi_i \right) L \left( \sum_{D_i \in \Omega} g_i^k \phi_i \right) d\Omega \quad (161)
\]

This is consistent with (152), and can be subsequently translated into the following linear system

\[
\begin{align*}
S \begin{bmatrix} u^k \end{bmatrix} & = M \begin{bmatrix} w^{k+1} \end{bmatrix} - P \begin{bmatrix} g^k \end{bmatrix} \quad \text{where} \\
S_{ij} & = \int_{\Omega} \phi_i L \phi_j d\Omega \quad \forall \ D_i, D_j \in \Omega, \quad M_{ij} = \int_{\Omega} \phi_i R \phi_j d\Omega \quad \forall \ D_i \in \Omega, \ D_j \in \partial \Omega \quad (162) \\
P_{ij} & = \int_{\Omega} \phi_i L \phi_j d\Omega \quad \forall \ D_i \in \Omega, \ D_j \in \Gamma
\end{align*}
\]

\( S, M \) and \( P \) are matrices of size \( f \times f, f \times n \) and \( f \times c \) respectively, where the \( f, n \) and \( c \) denote the number of free (interior) nodes, total nodes and constrained (Dirichlet boundary) nodes, respectively. Note that these matrices can be evaluated from the basis functions, and because the \( w^{k+1} \) and \( G^k \) are known, the \( u^k \) can be computed using a linear system solver.
Because constructing the matrix $S$ and $P$ needs a 1-to-1 mapping of the matrix entries onto the node indices, the process is complicated and error prone. Alternatively, we consider another and yet simpler approach for the implementation. This approach can be easily adapted to other boundary conditions with minor modifications. The steps are described as follows:

1. We first construct a matrix $S$ such that $S_{ij} = \int_{\Omega} \phi_i \mathcal{L} \phi_j d\Omega \quad \forall \ D_i, D_j \in \Omega$ (rather than $D_i, D_j \in \Omega$ originally for $S_{ij}$) and a matrix $M$ such that $M_{ij} = \int_{\Omega} \phi_i \mathcal{R} \phi_j d\Omega \quad \forall \ D_i, D_j \in \Omega$ (rather than $D_i \in \Omega$ and $D_j \in \Omega$ originally for $M_{ij}$). Both $S$ and $M$ are $(n \times n)$ matrix, where $n = N^D$ is the total number of nodes in the closed domain $\Omega$. The matrix $S$ can be subdivided into four sub-matrices $S = (S_{ff} \ S_{fc} \ S_{cf} \ S_{cc})$, where the subscript $f$ and $c$ represent a sub-matrix $S_{ij}$ corresponding to integrals between the free and constrained nodes. The sub-matrix $S_{fc}$, for example, is defined as $S_{ij} = \int_{\Omega} \phi_i \mathcal{L} \phi_j d\Omega$ with $D_i \in \Omega$ (free nodes) and $D_j \in \Gamma'$ (constrained nodes). Similarly, we have $M = (M_{ff} \ M_{fc} \ M_{cf} \ M_{cc})$.

2. If $D_i \in \Gamma'$, the value of $w_i^k = G^k_i = g^k(D_i)$ is known. Hence we create an $(n \times 1)$ vector $\omega^k$ such that $\omega_i^k = \begin{cases} 0, & \text{if } D_i \in \Omega, \\ G^k_i, & \text{if } D_i \in \Gamma'. \end{cases}$ In other words, $\omega^k = \begin{pmatrix} 0 \\ w^k_c \end{pmatrix}$ if we subdivide the vector $w^k = \begin{pmatrix} w^k_f \\ w^k_c \end{pmatrix}$ by following the same definition as in step 1.

3. We create a vector $h$ as a right hand side term such that

$$h_n = \frac{M}{n}w_{n+1}^k - \frac{S}{n} \omega^k = \begin{pmatrix} M_{ff} \ M_{fc} \\ M_{cf} \ M_{cc} \end{pmatrix} \begin{pmatrix} w^k_{f+1} \\ w^k_{c+1} \end{pmatrix} - \begin{pmatrix} S_{ff} \ S_{fc} \\ S_{cf} \ S_{cc} \end{pmatrix} \begin{pmatrix} 0 \\ w^k_c \end{pmatrix},$$

then manually assign $h_i = G^k_i \quad \forall \ D_i \in \Gamma$, that is
\[
    h = \left( \frac{\mathcal{M}_{ff} w_{f}^{k+1} + \mathcal{M}_{fc} w_{c}^{k+1} - S_{fc} w_{c}^{k}}{w_{c}^{k}} \right) \tag{164}
\]

4. Because \( w_{c}^{k} \) is known, computation for this part is not necessary. We further process matrix \( S \) by replacing its sub-matrix \( S_{cc} \) with an identity matrix and replacing \( S_{fc}, S_{cf} \) with zero matrices. We end up with \( S = \begin{pmatrix} S_{ff} & 0 \\ 0 & I \end{pmatrix} \).

5. Finally, we have an augmented linear system.

\[
    S w^{k}_{n \times n \times 1} = h_{n \times 1} \Rightarrow \begin{pmatrix} S_{ff} & 0 \\ 0 & I \end{pmatrix} \begin{pmatrix} w_{f}^{k} \\ w_{c}^{k} \end{pmatrix} = h \Rightarrow \begin{pmatrix} S_{ff} & 0 \\ 0 & I \end{pmatrix} \begin{pmatrix} w_{f}^{k} \\ w_{c}^{k} \end{pmatrix} = \begin{pmatrix} \mathcal{M}_{ff} w_{f}^{k+1} + \mathcal{M}_{fc} w_{c}^{k+1} - S_{fc} w_{c}^{k} \\ w_{c}^{k} \end{pmatrix} \tag{165}
\]

\[
    \Rightarrow S_{ff} w_{f}^{k} = \mathcal{M}_{ff} w_{f}^{k+1} + \mathcal{M}_{fc} w_{c}^{k+1} - S_{fc} w_{c}^{k}
\]

The last equation is equivalent to (162). The solution \( w^{k} = S^{-1} h \) can be solved by an iterative linear solver.

3.2.2.6. Computation of Integrals

Based on previous discussion, we know that the construction of the matrix \( S \) and \( \mathcal{M} \) is directly related to the computation of the integral \( \int_{\Omega} \phi_{i} \mathcal{L} \phi_{j} \ d\Omega \) and \( \int_{\Omega} \phi_{i} \mathcal{R} \phi_{j} \ d\Omega \)

\[
    S_{ij} = \int_{\Omega} \phi_{i} \mathcal{L} \phi_{j} \ d\Omega \quad \forall \ D_{i}, D_{j} \in \tilde{\Omega} \quad \text{and} \quad \mathcal{M}_{ij} = \int_{\Omega} \phi_{i} \mathcal{R} \phi_{j} \ d\Omega \quad \forall \ D_{i}, D_{j} \in \tilde{\Omega} \tag{166}
\]

The computation of the two integrals shares a great similarity. Each can be decompose into three parts

\[
    \int_{\Omega} \phi_{i} \mathcal{L} \phi_{j} \ d\Omega = \sigma \int_{\Omega} \nabla \phi_{i} \cdot \mathbf{A} \nabla \phi_{j} \ d\Omega + \sigma \int_{\Omega} \phi_{i} \mathbf{b} \cdot \nabla \phi_{j} \ d\Omega + c \int_{\Omega} \phi_{i} \phi_{j} \ d\Omega \quad \text{and}
\]

\[
    \int_{\Omega} \phi_{i} \mathcal{R} \phi_{j} \ d\Omega = (\sigma - 1) \int_{\Omega} \nabla \phi_{i} \cdot \mathbf{A} \nabla \phi_{j} \ d\Omega + (\sigma - 1) \int_{\Omega} \phi_{i} \mathbf{b} \cdot \nabla \phi_{j} \ d\Omega + (c - r) \int_{\Omega} \phi_{i} \phi_{j} \ d\Omega \tag{167}
\]

Instead of directly computing the integral over the whole domain \( \Omega \), we can compute the integral element by element and then sum all the pieces together. Given a triangular element \( T \), it is obvious that there are locally only three of the basis functions (corresponding to the three nodes) are nonzero. The reference triangle \( \tilde{T} \) is mapped onto the triangular element \( T \) and the integral evaluated over \( \tilde{T} \) is then transformed back to \( T \).
The first integral appearing in the right hand side of (167) is denoted by \( P_1 \). Over a given triangular element \( T \), the \( P_1 \) is a 3 \( \times \) 3 matrix and is evaluated as

\[
P_1 = \left[ \int_T \nabla \phi_i \cdot \mathbf{A} \mathbf{v} \phi_j dT \right]_{i,j=0,1,2} = \left[ \int_T \left( \mathbf{J}^{-T} \nabla \hat{\phi}_i \right)^T \left( \mathbf{A} \mathbf{J}^{-T} \nabla \hat{\phi}_j \right) \left[ \frac{\partial F(\hat{v},\hat{y})}{\partial (\nu,\mu)} \right] d\hat{T} \right]_{i,j=0,1,2} = |\text{det} \mathbf{J}| \left[ \int_T \left( \mathbf{J}^{-T} \mathbf{G} \right)^T \left( \int_T \mathbf{A} d\hat{T} \right) \right]^{-T} \mathbf{G}
\]

\[
\approx \frac{1}{2} |\text{det} \mathbf{J}| \mathbf{G}^T \mathbf{J}^{-1} \mathbf{A} \mathbf{J}^{-T} \mathbf{G}
\]

(168)

where \( \mathbf{G} = [\nabla \hat{\phi}_j]_{j=0,1,2} = \begin{pmatrix} -1 & 1 & 0 \\ -1 & 0 & 1 \end{pmatrix} \) and

\[
\int_T \mathbf{A} d\hat{T} \approx \bar{\mathbf{A}} \int_T d\hat{T} = \left( \text{average of } \mathbf{A} \text{ over triangular element} \right) \times \left( \text{area over reference triangle} \right) = \frac{\sum_{n=0}^{2} \mathbf{A}_n}{3} \frac{1}{2}
\]

In (168), the subscript \( i \) and \( j \) denote the matrix row and column index respectively, the \( |\text{det} \mathbf{J}| \) is the determinant absolute of matrix \( \mathbf{J} \), the \( \mathbf{G} \) is the gradient matrix in \( \hat{T} \) for the three local basis functions defined in (155). Since \( \mathbf{G} \) is constant, it can be moved out of the integral. The \( \mathbf{A}_n \), \( n = 0,1,2 \) are the matrix \( \mathbf{A} \) evaluated at the three vortices of the elemental triangle \( T \). To compute the integral \( \int_T \mathbf{A} d\hat{T} \), matrix \( \mathbf{A} \) is assumed to be constant within \( T \) and takes an average over the three vortices.

The second integral denoted by \( P_2 \) is computed as

\[
P_2 = \left[ \int_T \phi_i \mathbf{b} \cdot \nabla \phi_j dT \right]_{i,j=0,1,2} \approx |\text{det} \mathbf{J}| \left[ \int_T \hat{\phi}_i d\hat{T} \right]_{i=0,1,2} \mathbf{b}^T \mathbf{J}^{-T} \mathbf{G}
\]

\[
= |\text{det} \mathbf{J}| \frac{1}{6} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \mathbf{b}^T \mathbf{J}^{-T} \mathbf{G} \quad \text{where} \quad \mathbf{b} = \frac{1}{3} \sum_{n=0}^{2} \mathbf{b}_n
\]

(169)

In (169), the \( \left[ \int_T \hat{\phi}_i d\hat{T} \right]_{i=0,1,2} \) is simply the volume below the basis functions integrated over the reference triangle \( \hat{T} \). Similarly as with matrix \( \mathbf{A} \), The vector \( \mathbf{b} \) is treated constant within the element triangle and takes an average over the three vortices. Hence it can be moved out of the integral along with the \( \mathbf{G} \) matrix.

The computation of the third integral \( P_3 \) is even simpler
\[ P_3 = \left[ \int_T \phi_i \phi_j dT \right]_{i,j=0,1,2} = |\det J| \left[ \int_T \hat{\phi}_i \hat{\phi}_j d\hat{T} \right]_{i,j=0,1,2} = |\det J| \frac{1}{24} \begin{pmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{pmatrix} \] (170)

It is obvious that \( P_1, P_2 \) and \( P_3 \) are all \( 3 \times 3 \) matrix and just \( P_2 \) appears asymmetric. Within each element triangle, we have

\[
\left[ \int_T \phi_i \psi_j dT \right]_{i,j=0,1,2} = \sigma (P_1 + P_2) + cP_3 \quad \text{and} \quad \left[ \int_T \phi_i \chi_j dT \right]_{i,j=0,1,2} = (\sigma - 1)(P_1 + P_2) + (c - r)P_3
\] (171)

Aggregating integrals over all the triangular elements in the domain, the global stiffness matrix \( \mathcal{S} \) and mass matrix \( \mathcal{M} \) can then be assembled.

3.2.3. Iterative Linear Solvers

3.2.3.1. Implementation of Sparse Matrix

As one can see, due to the use of localized basis functions, each node can interact with at most, including itself, seven nodes. The stiffness and mass matrix can therefore be sparse. Since this is a 2-D problem, as the resolution increases, the storage for the matrix increases quadratically. To avoid the unnecessary memory storage for the zeros, a sparse matrix implementation is developed. The rows of a sparse matrix are implemented as an array. Each entry of the array consists of a map, which maps the column index to the matrix entry. Each map stores all the nonzero elements of a row vector of the sparse matrix. Given a matrix index \( i \) and \( j \), it first finds the \( i \)-th row through the map array, and then locate the \( j \)-th column entry through the \( i \)-th map. Using this implementation one can easily and efficiently access the matrix members. This turns out to be very handy for constructing the sparse matrices.

Since the matrix is generally very large, for example, a \( 100 \times 100 \) domain will generate (roughly) a \( 10000 \times 10000 \) matrix. In order to apply the iterative methods for the linear system, the matrix-vector multiplication operation must be extremely efficient. To address this issue, the fully constructed sparse matrix is then converted to a compressed row storage format, which consists of only three arrays: two
arrays store the matrix indices \(i\) and \(j\) respectively, and one array stores the corresponding matrix entries. Although adding or removing the matrix entries is no longer as straightforward as before, the matrix multiplications can be done very efficiently.

### 3.2.3.2. Iterative Methods for Asymmetric Sparse Matrix

The linear systems resulted from this problem are usually very large. Traditional direct method and simple iterative methods are no longer appropriate for this type of applications. Furthermore, as one can see that the \(P_2\) in (169) is asymmetric, and so is the global stiffness matrix \(\mathbf{S}\). In this case, the widely used Conjugate Gradient method is no longer applicable. Instead, the Conjugate Gradient Squared (CGS) and/or Generalized Minimal Residual (GMRES) method must be used to solve the linear systems. Since the two methods have been widely documented, the details of the algorithm will not be described here. The implementation of the two methods is based on Kelley’s book [28]. In general, CGS method performs faster than GMRES method as it has much fewer floating point operations in each iteration, however in some cases CGS method does not guarantee a convergence. In our application, the tolerance for convergence is set to \(1.0 \times 10^{-8}\).

### 3.2.4. Interpolation of Numerical Solution

Numerical solution to the PDE is provided as a vector \(\mathbf{w}\) with values at each of the prescribed nodes. To obtain a solution for an arbitrary position in the domain, interpolation must be performed. Since the mesh can be either structured or unstructured, bilinear interpolation is not always appropriate. In this case, the point coordinate is converted to a barycentric coordinate in an element triangle to perform the linear interpolation. Firstly, it is desired to determine the triangle where the point locates. Given a point coordinate \((v, y)\), the barycentric coordinates \(\lambda_0, \lambda_1, \text{and } \lambda_2\) at this point can be computed based upon the three triangle vortices \((v_0, y_0), (v_1, y_1), (v_2, y_2)\)

\[
\begin{align*}
\lambda_0 &= \frac{(v_0 - v_2)(y_1 - y_2) - (v_1 - v_2)(y_0 - y_2)}{(v_0 - v_2)(y_1 - y_2) - (v_1 - v_2)(y_0 - y_2)}, \\
\lambda_1 &= \frac{(v_0 - v_2)(y_1 - y_2) - (v_0 - v_1)(y_0 - y_2)}{(v_0 - v_2)(y_1 - y_2) - (v_1 - v_2)(y_0 - y_2)}, \\
\lambda_2 &= 1 - \lambda_0 - \lambda_1
\end{align*}
\]

If \(\lambda_i \in [0,1]\ \forall\ i = 0, 1, 2\), then the point is inside the triangle and the interpolated solution is
\[ w(v, y) = \lambda_0 w(v_0, y_0) + \lambda_1 w(v_1, y_1) + \lambda_2 w(v_2, y_2) \]  
(173)

3.3. Solutions and Discussions

The correctness of the algorithm is examined against the solutions obtained from closed form formulas for special cases. All the tests are based on the default model parameters listed in Table 1 unless otherwise stated explicitly. The CGS is used as the default linear solver. Crank-Nicolson scheme is assumed for the temporal discretization.

<table>
<thead>
<tr>
<th>Table 1. default model parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial Spot Price ( S_0 )</td>
</tr>
<tr>
<td>Initial Variance ( v_0 )</td>
</tr>
<tr>
<td>Risk Free Interest Rate ( r )</td>
</tr>
<tr>
<td>Dividend Yield Rate ( q )</td>
</tr>
<tr>
<td>Mean Variance ( \theta )</td>
</tr>
<tr>
<td>Call Strike ( K_c )</td>
</tr>
<tr>
<td>Mean Reversion Rate ( \kappa )</td>
</tr>
<tr>
<td>Put Strike ( K_p )</td>
</tr>
<tr>
<td>Volatility of Variance ( \xi )</td>
</tr>
<tr>
<td>Up-and-Out Barrier ( B_u )</td>
</tr>
<tr>
<td>Correlation ( \rho )</td>
</tr>
<tr>
<td>Down-and-Out Barrier ( B_d )</td>
</tr>
<tr>
<td>Spatial Resolution</td>
</tr>
<tr>
<td>Temporal Resolution</td>
</tr>
</tbody>
</table>

3.3.1. Vanilla Call

<p>| Table 2. vanilla call with different temporal and spatial resolutions |
|------------------------|------------------------|------------------------|------------------------|</p>
<table>
<thead>
<tr>
<th>Resolution in ( v )</th>
<th>Resolution in ( S )</th>
<th>Resolution in ( t )</th>
<th>Computational Time (seconds)</th>
<th>Closed-Form</th>
<th>Crank-Nicolson FEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>25</td>
<td>25</td>
<td>0.211</td>
<td>14.391785</td>
<td>13.922379</td>
</tr>
<tr>
<td>50</td>
<td>50</td>
<td>50</td>
<td>0.778</td>
<td>13.866197</td>
<td>13.856007</td>
</tr>
<tr>
<td>100</td>
<td>100</td>
<td>100</td>
<td>4.167</td>
<td>13.853335</td>
<td>13.854737</td>
</tr>
<tr>
<td>150</td>
<td>150</td>
<td>150</td>
<td>13.873</td>
<td>13.854719</td>
<td>13.854719</td>
</tr>
<tr>
<td>200</td>
<td>200</td>
<td>200</td>
<td>35.824</td>
<td></td>
<td></td>
</tr>
<tr>
<td>400</td>
<td>400</td>
<td>400</td>
<td>330.796</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The first test is to price a vanilla call option. Winkler’s implementation [29] of closed form formula for call options is used as a benchmark. In this test, the domain is set to \( v \in [0, 3] \) and \( S \in [25, 400] \). The computational time is estimated for a fully optimized program running on an Intel Core 2 Duo 1.86 GHz processor. Varying the resolution in \( v, S \) and \( t \), a series of prices can be computed and the results are shown in Table 2. It shows that the solutions are quite accurate when the resolution is at or above 100. As
one can see, the CGS solver performs very well. It takes only 5.5 minutes to solve a problem with resolution as large as $400 \times 400 \times 400$.

To examine the convergence property with respect to resolution in time, the spatial resolution is fixed at $150 \times 150$, the temporal resolution varies from 5 to 200. The results are listed in Table 3. It shows that the result is still acceptable even with a temporal resolution down to 5. The accuracy is improved rapidly as the temporal resolution increases.

<table>
<thead>
<tr>
<th>temporal resolution</th>
<th>Computational Time (seconds)</th>
<th>Closed-Form</th>
<th>Crank-Nicolson FEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>4.217</td>
<td></td>
<td>13.761879</td>
</tr>
<tr>
<td>10</td>
<td>4.883</td>
<td>13.853335</td>
<td>13.814776</td>
</tr>
<tr>
<td>20</td>
<td>5.879</td>
<td></td>
<td>13.835519</td>
</tr>
<tr>
<td>50</td>
<td>8.220</td>
<td></td>
<td>13.849429</td>
</tr>
<tr>
<td>100</td>
<td>11.414</td>
<td></td>
<td>13.854338</td>
</tr>
<tr>
<td>200</td>
<td>16.155</td>
<td></td>
<td>13.856847</td>
</tr>
</tbody>
</table>

The vanilla call prices are also evaluated at different strikes and shown in Table 4. The computed prices are in excellent agreement with the closed form solutions. It should be noted that the computational domain must be adjusted accordingly for the strike of 200 due to its large value. Since the call option with a strike of 200 is far out of the money, it is expected that there is a relative large error in the computed option price. This can be seen from the last row of the Table 4.

<table>
<thead>
<tr>
<th>Strike $K$</th>
<th>Closed-Form</th>
<th>Crank-Nicolson FEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>105</td>
<td>15.935692</td>
<td>15.939498</td>
</tr>
<tr>
<td>110</td>
<td>13.853335</td>
<td>13.856007</td>
</tr>
<tr>
<td>115</td>
<td>11.975267</td>
<td>11.980188</td>
</tr>
<tr>
<td>130</td>
<td>7.4758044</td>
<td>7.4790939</td>
</tr>
<tr>
<td>150</td>
<td>3.6874922</td>
<td>3.6916604</td>
</tr>
<tr>
<td>200</td>
<td>0.42266949</td>
<td>0.43570695</td>
</tr>
</tbody>
</table>
Table 5. vanilla call with different maturities

<table>
<thead>
<tr>
<th>Maturity Time T</th>
<th>Closed-Form</th>
<th>Crank-Nicolson FEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/12</td>
<td>2.180542</td>
<td>2.1765933</td>
</tr>
<tr>
<td>1/4</td>
<td>5.7929266</td>
<td>5.7890237</td>
</tr>
<tr>
<td>1/2</td>
<td>9.3177983</td>
<td>9.3154635</td>
</tr>
<tr>
<td>1</td>
<td>13.853335</td>
<td>13.856007</td>
</tr>
<tr>
<td>2</td>
<td>19.56252</td>
<td>19.44822</td>
</tr>
<tr>
<td>5</td>
<td>29.918653</td>
<td>28.500867</td>
</tr>
<tr>
<td>10</td>
<td>N/A</td>
<td>36.782293</td>
</tr>
</tbody>
</table>

Resolutions increase from 150 to 300:

| 5               | 29.918653   | 28.938853          |

Figure 3. A typical solution surface for a vanilla call

Table 5 shows the computed option price with different maturity times. As compared with the closed form solutions, the agreement is excellent except for those long term maturities (5 years and up). Larger domain and higher resolutions can improve the accuracy. This has been shown for the case of 5
years maturity time. A typical solution surface for the vanilla call options is shown in Figure 3. Note that the axis for sport process has been log transformed to $y = \ln \frac{s}{K}$.

3.3.2. Vanilla Put

The second test is to price a vanilla put option. In order to compare the results with closed form solutions, the dividend rate $q = 0$ is assumed. Rouah-Vainbergh’s Excel spreadsheet [30] is used as a benchmark, which analytically solves for the put option price. In this test, the domain is also set as $v \in [0, 3]$ and $S \in [25, 400]$. The put price is evaluated at different strikes. The results are shown in the Table 6. The computed prices are in excellent agreement with the closed form solutions, except for the strike of 20 due to the extreme out-of-the-money. It should be noted that the computational domain must be adjusted accordingly for small strikes.

<table>
<thead>
<tr>
<th>Strike $K$</th>
<th>Rouah-Vainbergh’s Spreadsheet</th>
<th>Crank-Nicolson FEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>95</td>
<td>N/A</td>
<td>11.816262</td>
</tr>
<tr>
<td>90</td>
<td>9.8286519</td>
<td>9.8259365</td>
</tr>
<tr>
<td>85</td>
<td>8.0667635</td>
<td>8.0587285</td>
</tr>
<tr>
<td>70</td>
<td>3.9702575</td>
<td>3.9744305</td>
</tr>
<tr>
<td>50</td>
<td>1.0635452</td>
<td>1.0610897</td>
</tr>
<tr>
<td>20</td>
<td>0.0118353</td>
<td>0.018820474</td>
</tr>
</tbody>
</table>

Table 7. vanilla put with different maturities

<table>
<thead>
<tr>
<th>Maturity Time $T$</th>
<th>Rouah-Vainbergh’s Spreadsheet</th>
<th>Crank-Nicolson FEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/12</td>
<td>1.8136404</td>
<td>1.8153803</td>
</tr>
<tr>
<td>1/4</td>
<td>4.6993203</td>
<td>4.7001984</td>
</tr>
<tr>
<td>1/2</td>
<td>7.2118815</td>
<td>7.2117123</td>
</tr>
<tr>
<td>1</td>
<td>9.8286519</td>
<td>9.8259365</td>
</tr>
<tr>
<td>2</td>
<td>N/A</td>
<td>11.796426</td>
</tr>
</tbody>
</table>

Table 7 shows the computed put prices with different maturities. As compared with the benchmark values, the computed prices are very close to the closed form solutions. A typical solution surface for the vanilla put options is shown in Figure 4.
3.3.3. **Double Barrier Knock-out Call**

![Figure 4. A typical solution surface for a vanilla put](image)

<table>
<thead>
<tr>
<th>Down Barrier</th>
<th>Up Barrier</th>
<th>Strike K</th>
<th>Faulhaber’s Spreadsheet</th>
<th>Crank-Nicolson FEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>200</td>
<td>110</td>
<td>5.99628132</td>
<td>5.986791</td>
</tr>
<tr>
<td>50</td>
<td>150</td>
<td>110</td>
<td>1.296335529</td>
<td>1.2971659</td>
</tr>
<tr>
<td>80</td>
<td>120</td>
<td>110</td>
<td>0.006384768</td>
<td>0.0066368712</td>
</tr>
<tr>
<td>20</td>
<td>200</td>
<td>100</td>
<td>8.812262221</td>
<td>8.8034578</td>
</tr>
<tr>
<td>50</td>
<td>150</td>
<td>100</td>
<td>2.655277021</td>
<td>2.6582231</td>
</tr>
<tr>
<td>80</td>
<td>120</td>
<td>100</td>
<td>0.051849246</td>
<td>0.053989178</td>
</tr>
<tr>
<td>20</td>
<td>200</td>
<td>90</td>
<td>12.51875619</td>
<td>12.510153</td>
</tr>
<tr>
<td>50</td>
<td>150</td>
<td>90</td>
<td>4.766805909</td>
<td>4.7734182</td>
</tr>
<tr>
<td>80</td>
<td>120</td>
<td>90</td>
<td>0.166447999</td>
<td>0.17333944</td>
</tr>
</tbody>
</table>

Resolutions increase from 150 to 300:

<table>
<thead>
<tr>
<th>Down Barrier</th>
<th>Up Barrier</th>
<th>Strike K</th>
<th>Faulhaber’s Spreadsheet</th>
<th>Crank-Nicolson FEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>80</td>
<td>120</td>
<td>90</td>
<td>0.166447999</td>
<td>0.16993712</td>
</tr>
</tbody>
</table>
The third test is to price double barrier knock-out call options. To allow the existence of closed form solutions, the correlation \( \rho = 0 \) and the dividend rate, \( q = r \), the risk free interest rate are assumed. Faulhaber's Excel spreadsheet [31] is used as a benchmark, which analytically solve for the double barrier option prices. In this test, the domain is assumed as \( v \in [0, 3] \) and \( S \in [B_d, B_u] \). The double barrier call price is evaluated with different strikes and barriers. The results are shown in Table 8. The differences between the computed prices and the closed form solutions are quite small, except for those close-to-zero prices, in which the problems are ill-conditioned and are very difficult for the solver to compute a solution accurately. As shown in the table, higher resolution can improve the accuracy.

Table 9 shows the computed double barrier knock-out call prices with different maturity times. To avoid the close-to-zero prices, the down barrier \( B_d = 50 \) and the up barrier \( B_u = 150 \) are assumed. As compared with the benchmark values, the numerical solutions are accurate, especially for the short term options. A typical solution surface for the double barrier knock-out call options is shown in Figure 5. To avoid the out-of-the-money, the barriers have been assume to be \( B_d = 50 \) and \( B_u = 150 \).

<table>
<thead>
<tr>
<th>Maturity Time</th>
<th>Rouah-Vainbergh Spreadsheet</th>
<th>Crank-Nicolson FEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/12</td>
<td>2.073151427</td>
<td>2.056223</td>
</tr>
<tr>
<td>1/4</td>
<td>2.748504846</td>
<td>2.7439708</td>
</tr>
<tr>
<td>1/2</td>
<td>2.048575635</td>
<td>2.049062</td>
</tr>
<tr>
<td>1</td>
<td>1.296335529</td>
<td>1.2971659</td>
</tr>
<tr>
<td>2</td>
<td>0.744555858</td>
<td>0.77808438</td>
</tr>
</tbody>
</table>
3.3.4. Double Barrier Knock-out Put

Table 10. double barrier knock-out put with different barriers and strikes

<table>
<thead>
<tr>
<th>Down Barrier</th>
<th>Up Barrier</th>
<th>Strike $K$</th>
<th>Faulhaber’s Spreadsheet</th>
<th>Crank-Nicolson FEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>200</td>
<td>110</td>
<td>21.87927024</td>
<td>21.872058</td>
</tr>
<tr>
<td>50</td>
<td>150</td>
<td>110</td>
<td>12.46280153</td>
<td>12.452305</td>
</tr>
<tr>
<td>80</td>
<td>120</td>
<td>110</td>
<td>0.235744502</td>
<td>0.24543624</td>
</tr>
<tr>
<td>20</td>
<td>200</td>
<td>100</td>
<td>15.93365322</td>
<td>15.927314</td>
</tr>
<tr>
<td>50</td>
<td>150</td>
<td>100</td>
<td>8.307725123</td>
<td>8.2976528</td>
</tr>
<tr>
<td>80</td>
<td>120</td>
<td>100</td>
<td>0.090320836</td>
<td>0.094025866</td>
</tr>
<tr>
<td>20</td>
<td>200</td>
<td>90</td>
<td>10.87854926</td>
<td>10.872598</td>
</tr>
<tr>
<td>50</td>
<td>150</td>
<td>90</td>
<td>4.905236115</td>
<td>4.8971387</td>
</tr>
<tr>
<td>80</td>
<td>120</td>
<td>90</td>
<td>0.014031445</td>
<td>0.014613467</td>
</tr>
</tbody>
</table>

The fourth test is to price a double barrier knock-out put option. In this case, the correlation $\rho = 0$ and the dividend rate, $q = r$, the risk free interest rate are assumed. Faulhaber’s excel spreadsheet is once again used as a benchmark. In this test, the domain is $v \in [0, 3]$ and $S \in [B_d, B_u]$. The double barrier put
price is evaluated with different strikes and barriers. The results are shown in Table 10 and Table 11. The computed prices are in good agreement with the closed form solutions. As usual, higher resolution leads to a better accuracy.

<table>
<thead>
<tr>
<th>Down Barrier</th>
<th>Up Barrier</th>
<th>Strike $K$</th>
<th>Faulhaber’s Spreadsheet</th>
<th>Crank-Nicolson FEM Resolutions: 150</th>
<th>Crank-Nicolson FEM Resolutions: 300</th>
</tr>
</thead>
<tbody>
<tr>
<td>80</td>
<td>120</td>
<td>110</td>
<td>0.235744502</td>
<td>0.24543624</td>
<td>0.24058918</td>
</tr>
<tr>
<td>80</td>
<td>120</td>
<td>100</td>
<td>0.090320836</td>
<td>0.094025866</td>
<td>0.092165229</td>
</tr>
<tr>
<td>80</td>
<td>120</td>
<td>90</td>
<td>0.014031445</td>
<td>0.014613467</td>
<td>0.014317593</td>
</tr>
</tbody>
</table>

Table 12 shows the computed double barrier knock-out put prices with different maturity times. To avoid the close-to-zero prices, the down barrier $B_d = 50$ and up barrier $B_u = 150$ are assumed. A typical solution surface for the double barrier double barrier knock-out put options is shown in Figure 6. As compared with the benchmark values, the numerical solutions are accurate, especially for the short term options.

<table>
<thead>
<tr>
<th>Maturity Time $T$</th>
<th>Rouah-Vainbergh Spreadsheet</th>
<th>Crank-Nicolson FEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/12</td>
<td>1.790372434</td>
<td>1.7880031</td>
</tr>
<tr>
<td>1/4</td>
<td>4.564088509</td>
<td>4.5554437</td>
</tr>
<tr>
<td>1/2</td>
<td>5.606674031</td>
<td>5.5991206</td>
</tr>
<tr>
<td>1</td>
<td>4.905236115</td>
<td>4.8971387</td>
</tr>
<tr>
<td>2</td>
<td>3.368687428</td>
<td>3.3595535</td>
</tr>
</tbody>
</table>
3.3.5. Numerical Solutions of Double Barrier Knock-out Options

The Crank-Nicolson finite element solver is used to price the double barrier options for our problem. Specifically, the double barrier options are assumed to follow Heston stochastic volatility model and take the values for the parameters as listed in Table 1. The computation is performed on a domain defined as $v \in [0, 3]$ and $S \in [B_d, B_u]$. The spatial and temporal resolutions are set to various levels to gain an overview about the accuracy and convergence trend. The solutions are shown in Table 13. Since these two option prices are very close to zero, the results computed at lower resolutions may have large errors. As the resolution improves, we do see the convergence of the solution. Another observation is that at the same resolution the computational time for double barrier options is much longer than that of plain vanilla options. This is partially because that double barrier options usually have more dynamic features than plain vanillas and therefore they are more computationally challenging.
Table 13. Double Barrier Knock-out Options

<table>
<thead>
<tr>
<th>Type</th>
<th>Resolution (spatial and temporal)</th>
<th>Computational Time (seconds)</th>
<th>Computed Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Call</td>
<td>100</td>
<td>30.6</td>
<td>0.019140956</td>
</tr>
<tr>
<td></td>
<td>150</td>
<td>125.9</td>
<td>0.014483851</td>
</tr>
<tr>
<td></td>
<td>200</td>
<td>367.1</td>
<td>0.012706429</td>
</tr>
<tr>
<td></td>
<td>300</td>
<td>1328.3</td>
<td>0.011207521</td>
</tr>
<tr>
<td></td>
<td>400</td>
<td>3942.1</td>
<td>0.010623075</td>
</tr>
<tr>
<td></td>
<td>500</td>
<td>9174.5</td>
<td>0.010275859</td>
</tr>
<tr>
<td>Put</td>
<td>100</td>
<td>28.9</td>
<td>0.0047008941</td>
</tr>
<tr>
<td></td>
<td>150</td>
<td>138.2</td>
<td>0.0045130817</td>
</tr>
<tr>
<td></td>
<td>200</td>
<td>325.7</td>
<td>0.0044583508</td>
</tr>
<tr>
<td></td>
<td>300</td>
<td>1399.3</td>
<td>0.0044205309</td>
</tr>
<tr>
<td></td>
<td>400</td>
<td>3698.4</td>
<td>0.0044209021</td>
</tr>
<tr>
<td></td>
<td>500</td>
<td>9461.7</td>
<td>0.0044168854</td>
</tr>
</tbody>
</table>

3.4. Conclusions

It has been illustrated how to price a double barrier knock-out option (the knock-in type barrier options can be priced in a similar manner, except that the vanilla option prices are used for Dirichlet boundaries) in a stochastic volatility model using finite element method. The numerical solutions of the testing cases are in excellent agreement with the closed form values. This strongly suggests the accuracy and efficiency advantages of this method, as compared with finite difference method and Monte Carlo simulations. With appropriate boundary condition definitions, this method can be easily extended to price more complicated exotic options. It can also be further improved by applying adaptive mesh refinement and/or higher order basis functions.
REFERENCES


#include "Parameter.h"
#include "FEM.h"
#include "HestonClosedForm.h"

unsigned main (){
    Parameter P;
    P.S0 = 100.0;  //initial spot
    P.v0 = 0.25;   //initial variance
    P.th = 0.09;   //theta
    P.kp = 1.0;    //kappa
    P.xi = 0.4;    //xi, volatility of variance
    P.rho= -0.7;   //rho, covariance
    P.ir = 0.05;   //interest rate
    P.dr = 0.01;   //dividend rate
    P.Ko = 110.0;  //option strike
    P.Tm = 1.0;    //time to maturity

    /**************************************************************************
    * Euro Vanilla       : Put = -1, Call = 1;
    * Euro Double Barrier: Put = -2, Call = 2;
    **************************************************************************/
    P.type = 1;    //option type

    /**************************************************************************
    * The min and max for variance
    * may be set to any reasonable values.
    */
    P.vmin = 0.0;  //min variance
    P.vmax = 3.0;  //max variance

    /**************************************************************************
    * The min and max for spot may be
    * set to any reasonable values.
    * Note that, the spot min CANNOT be zero
    * due to the change of variable.
    *
    * In the case of double barrier options,
    * these are the DOWN and UP Barriers.
    */
    P.Smin = 25;   //max spot
    P.Smax = 400;  //min spot

    /**************************************************************************
    * spatial and temporal resolutions
    */
    P.N = 150;     //resolution in variance
    P.M = 150;     //resolution in spot process
    P.Z = 150*P.Tm;//resolution in time / year

    cout.precision(8);

    if(P.type == 1){
        cout<<"C-N FEM Heston European vanilla call price = "<<FEM(P).solve()<<endl;
        cout<<"Closed-form Heston European vanilla call price = "<<Heston_Call(P)<<endl;
    }

    if(P.type == -1){
        cout<<"C-N FEM Heston European vanilla put price = "<<FEM(P).solve()<<endl;
    }

    if(P.type == 2){
        cout<<"C-N FEM Heston European double barrier call price = "<<FEM(P).solve()<<endl;
    }

    if(P.type == -2){
        cout<<"C-N FEM Heston European double barrier put price = "<<FEM(P).solve()<<endl;
    }

    return 0;
}
#ifndef FEM_H_
#define FEM_H_

#include <ctime>
#include <fstream>
#include "Parameter.h"
#include "Basics.h"
#include "MeshGenerator.h"
#include "CGS.h"
#include "MGMRES.h"

class FEM {
 public:
  FEM(const Parameter& P_);
  double solve();

 private:

  // [i, j], i-th test function, j-th trial function
  void get_local_stiffness_and_mass(const TriVertex& tv, DMatrix& mLS, DMatrix& mLM);

  void update_Jacobian_parameters(const TriVertex& tv); // matrix M
  DMatrix get_mAI (const TriVertex& tv); // matrix A integral
  DMatrix get_mb (const TriVertex& tv);
  double s2S(double s); // s = log(S/K);
  double S2s(double S); // s = log(S/K);

  double payoff(double spot, double strike);
  void set_initial(Vector& iv);
  Vector & zero_boundary(Vector& vw);

  // discontinuous Dirichlet boundaries
  Vector& set_Dirichlet(Vector& vw, double tau);
  Vector& zero_interior(Vector& vw);

  void output_datafile(Vector& vw);
  double interpolate(Vector& w, double x, double y);

  const Parameter P;
  const double smin; // = log(Smin/Ko)
  const double smax; // = log(Smax/Ko)
  const double dt; // size of timestep
  const double sigma; // sigma = 0.5 for Crank-Nicolson method
  const double cc; // const c

  StructuredMeshGenerator mesh;

  // The below are identical for each element
  DMatrix cmG; // const, gradient matrix
  DMatrix cmL; // const, linear integral matrix, "unit double volume"
  DMatrix cmA; // const, constant part of A matrix
  DMatrix cmB; // const, "unit single volume" matrix

  // The below are different for each element
  DMatrix mJ; // Jacobian matrix of linear mapping, maps to unit triangle.
  DMatrix mMG;
  double detJ; // = area of element triangle * 2 = absolute determinant of matrix J
};

#endif
#include "FEM.h"

FEM::FEM(const Parameter& P_)
  :P(P_),
    smin(log(P.Smin/P.Ko)),
    smax(log(P.Smax/P.Ko)),
    dt(P.Tm/P.Z), // delta T
    sigma(0.6),
    cc(sigma*P.ir + 1/dt),
    mesh(P.N, P.M, P.vmin, P.vmax, smin, smax),
    cmG (2,3,0.0),
    cml (3,3,1.0),
    cmA (2,2,1.0),
    cmB (3,1,1.0/6),
    mJ (2,2,0.0),
    mMG (3,3,0.0){
    /* cmG =
      | -1  1  0 |
      | -1  0  1 |
      */
    cmG[1][0] = cmG[0][0] = -1;
    cmG[1][2] = cmG[0][1] =  1;

    /* cmL =
      1    | 2  1  1 |
      --- * | 1  2  1 |
      24    | 1  1  2 |
      */
    cmL [2][2] = cmL [1][1] = cmL [0][0] = 2;
    cmL = cmL*(1.0/24);

    /* cmA =
      0.5 * |xi^2  xi*rho |
      |xi*rho  1   |
      */
    cmA [0][0] = P.xi * P.xi;
    cmA[1][0] = cmA[0][1] = P.xi * P.rho;
    cmA = cmA * 0.5;
    detJ = 0.0;
}

double FEM::solve(){
  int n_node = mesh.n_node;
  int n_elem = mesh.n_elem;
  SMatrix mGS (n_node, n_node);//Global Stiffness matrix
  SMatrix mGM (n_node, n_node);//Global Mass matrix
  DMatrix mLS (3,3,0.0);
  DMatrix mLM (3,3,0.0);
  double Time = clock();

  //construct global stiffness and mass matrix from local
  cout<<"Constructing Stiffness and Mass matrix ...\n";
  for(int elem = 0; elem<n_elem; elem++){
    TriVertex tv = mesh.e2tv(elem);
    update_Jacobian_parameters(tv);
    get_local_stiffness_and_mass(tv, mLS, mLM);
    int node[3] = {tv.a, tv.b, tv.c};//3 nodes of each element
    for(int i=0; i<3; i++){
      for(int j=0; j<3; j++){
        int I = node[i];
        int J = node[j];
        mGS.set(I, J, mGS.get(I, J)+mLS[i][j]);
        mGM.set(I, J, mGM.get(I, J)+mLM[i][j]);
      }
    }

    //convert to tri-array matrix
    TMatrix tmGS_full (mGS);
    TMatrix tmGM_full (mGM);
    //release memory for sparse matrix
    mGS.clear();
    mGM.clear();
//set Dirichlet boundary part of Stiffness to identity matrix
TMatrix tmGS_part = tmGS_full; // copy of full matrix
for(int n = 0; n < n_node; n++)
if(mesh.n2c(n).b < 0) // boundary node ID < 0
    tmGS_part.set_identity(n);

// initiate first vector with intial and boundary values
Vector vw(n_node);
set_initial(vw);
set_Dirichlet(vw, 0.0);

CGS LinearSolver;
// MGMRES LinearSolver;

Vector bound(n_node, 0.0);
for(unsigned k = 1; k <= P.Z; k++)
    cout << "Timestep " << k << " :\n";
    set_Dirichlet(bound, dt*k);
    Vector rhs = tmGM_full*vw - tmGS_full*bound;
    set_Dirichlet(rhs, dt*k);
    LinearSolver.solve(tmGS_part, vw, rhs, 1000, 1e-8);

    cout << "Time elapsed = " << (clock() - Time)/CLOCKS_PER_SEC << " seconds\n";

    output_datafile(vw); // output the last vector to file
    return interpolate(vw, P.v0, log(P.S0/P.Ko));

void FEM::get_local_stiffness_and_mass(const TriVertex& tv, DMatrix& mLS, DMatrix& mLM) {
    // 1st integral
    // A is treated as a constant in each element, hence simple quadrature
    mLS = mMG.trans() * get_mAI(tv) * mMG * detJ * sigma;

    // 2nd integral
    // b is treated as a constant in each element, hence simple quadrature
    mLS += cmB * get_mb(tv) * mMG * detJ * sigma;

    // Major part of the 1st and 2nd integral are the same for Stiffness and Mass
    mLM = mLS * ((sigma-1)/sigma);

    // 3rd integral
    mLS += cmL * detJ * cc;
    mLM += cmL * detJ * (cc - P.ir);
}

void FEM::update_Jacobian_parameters(const TriVertex& tv) {
    Coordinate c0 = mesh.n2c(tv.a);
    Coordinate c1 = mesh.n2c(tv.b);
    Coordinate c2 = mesh.n2c(tv.c);

    mJ[0][0] = c1.x - c0.x;
    mJ[0][1] = c2.x - c0.x;
    mJ[1][0] = c1.y - c0.y;
    mJ[1][1] = c2.y - c0.y;

    mMG = mJ.inv().trans() * cmG;
    detJ = fabs(mJ.det());
}

DMatrix FEM::get_mAI(const TriVertex& tv) {
    Coordinate c0 = mesh.n2c(tv.a);
    Coordinate c1 = mesh.n2c(tv.b);
    Coordinate c2 = mesh.n2c(tv.c);

    mJ[0][0] = c1.x - c0.x;
    mJ[0][1] = c2.x - c0.x;
    mJ[1][0] = c1.y - c0.y;
    mJ[1][1] = c2.y - c0.y;

    mMG = mJ.inv().trans() * cmG;
    detJ = fabs(mJ.det());
}

DMatrix FEM::get_mb(const TriVertex& tv) {
    Coordinate c0 = mesh.n2c(tv.a);
    Coordinate c1 = mesh.n2c(tv.b);
    Coordinate c2 = mesh.n2c(tv.c);
    double v = (c0.x + c1.x + c2.x) / 3.0;
    return v * 0.5; // 0.5 is the reference triangle area
}

DMatrix FEM::get_mA(const TriVertex& tv) {
    Coordinate c0 = mesh.n2c(tv.a);
    Coordinate c1 = mesh.n2c(tv.b);
    Coordinate c2 = mesh.n2c(tv.c);
    return v * 0.5; // 0.5 is the reference triangle area
}
DMatrix mb(1,2);
mb[0][0] = -P.kp * (P.th - v_bar) + 0.5*P.xi*P.xi;
mb[0][1] = -(P.ir-P.dr) + 0.5*(v_bar + P.rho*P.xi);
return mb;
}

double FEM::s2S(double s){//S = K * e^s;
return P.Ko * exp(s);
}

double FEM::S2s(double S){//s = log(S/K);
return log(S/P.Ko);
}

double FEM::payoff(double spot, double strike){
  double type = P.type > 0 ? 1.0 : -1.0;
  double diff = (spot - strike) * type;
  return diff > 0 ? diff : 0.0;
}

void FEM::set_initial(Vector& iv){
  int n_node = mesh.n_node;
  for(int n = 0; n<n_node; n++)
    if(mesh.n2c(n).b == 1){//interior nodes
      iv[n] = payoff(s2S(mesh.n2c(n).y), P.Ko);
    }
  return;
}

Vector& FEM::zero_boundary(Vector& vw){
  int n_node = mesh.n_node;
  for(int n = 0; n<n_node; n++)
    if(mesh.n2c(n).b < 0){
      vw[n] = 0.0;
    }
  return vw;
}

//discontinuous Dirichlet boundaries
Vector& FEM::set_Dirichlet(Vector& vw, double tau){
  int n_node = mesh.n_node;
  if(abs(P.type) == 1){//vanilla option
    for(int n = 0; n<n_node;n++)
      if(b_id < 0){
        if(b_id == -1){ //x_min boundary, ~v_min
          vw[n] = payoff(d_S, d_K);
        }
      }
  }else{//double barrier option
    for(int n = 0; n < n_node;n++)
      if(b_id < 0){
        if(b_id == -1){ //x_min boundary, ~v_min
          vw[n] = payoff(d_S, d_K);
        }
      }
  }
  return vw;
}
Vectors FEM::zero_interior(Vector& vw) {
    int n_node = mesh.n_node;
    for(int n = 0; n<n_node; n++) {
        if(mesh.n2c(n).b == 1) {
            vw[n] = 0.0;
        }
    }
    return vw;
}

void FEM::output_datafile(Vector& vw) {
    fstream pfs("param.txt", ios::in | ios::out | ios::trunc);
    if (!pfs.bad()) {
        pfs.precision(8);
        pfs << P.Smin << " " << P.Smax << " " << P.vmin << " " << P.vmax << " " << P.Ko << endl;
        pfs.close();
    }
    fstream fs("data.txt", ios::in | ios::out | ios::trunc);
    if (!fs.bad()) {
        fs.precision(8);
        for(unsigned n = 0; n < vw.size(); n++) {
            fs << vw[n] << "\t";
            if((n+1)%mesh.MM == 0)
                fs << endl;
        }
        fs.close();
    }
}

double FEM::interpolate(Vector& w, double x, double y) {
    // interpolation of the final result
    DMatrix mT(2,2);
    Vector r(2);
    for(int elem = 0; elem<mesh.n_elem; elem++) {
        TriVertex tv = mesh.e2tv(elem);
        mT[0][0] = mesh.n2c(tv.a).x - mesh.n2c(tv.c).x;
        mT[0][1] = mesh.n2c(tv.b).x - mesh.n2c(tv.c).x;
        mT[1][0] = mesh.n2c(tv.a).y - mesh.n2c(tv.c).y;
        mT[1][1] = mesh.n2c(tv.b).y - mesh.n2c(tv.c).y;
        r[0] = x - mesh.n2c(tv.c).x;
        r[1] = y - mesh.n2c(tv.c).y;
        r = mT.inv() * r;
        if (r[0] >= 0 && r[0] <= 1 &&
            r[1] >= 0 && r[1] <= 1) {
            return r[0]*w[tv.a] + r[1]*w[tv.b] + (1-r[0]-r[1])*w[tv.c];
        }
    }
    return -999;
}
/* Created on: Apr 10, 2009
 * Author: Changwei Xiong
 * Compiler: Visual Studio
 */

#ifndef MESHGENERATOR_H_
#define MESHGENERATOR_H_

#include <iostream>
using namespace std;
#include <valarray>

#define PRT(st) cout<< " " + string((# st)) + " = " << (st) <<endl

class Coordinate{
public:
  double x, y;
  int b;
};

class TriVertex{
public:
  int a, b, c;
};

class StructuredMeshGenerator {
public:
  StructuredMeshGenerator(
    int N_,
    int M_,
    double xmin_,
    double xmax_,
    double ymin_,
    double ymax_);

  void print_node();
  void print_element();

  Coordinate n2c(int n);

  TriVertex e2tv(int e);

  const double xmin, xmax, ymin, ymax;
  const int NN, MM; //N++ and M++

  const int n_node; // # of grid nodes (interior + boundary)
  const int n_elem; // # of elements

  const double dx, dy;

private:
  int idx(int i, int j);
  valarray<Coordinate> node;
  valarray<TriVertex> element;
};

#endif
/*
 *  Created on: Apr 10, 2009  
 *      Author: Changwei Xiong
 *    Compiler: Visual Studio
 */

#include "MeshGenerator.h"

StructuredMeshGenerator::StructuredMeshGenerator( 
 int N_,
 int M_,
 double xmin_,
 double xmax_,
 double ymin_,
 double ymax_)
 : xmin(xmin_),
 xmax(xmax_),
 ymin(ymin_),
 ymax(ymax_),
 NN(N_+1),
 MM(M_+1),
 n_node(NN*M_),
 n_element(NN*M_+2),
 dx((xmax-xmin)/N_),
 dy((ymax-ymin)/M_),
 node(n_node),
 element(n_element){
// i, NN for x; j, MM for y
 for(int i = 0; i < NN; i++){
   for(int j = 0; j < MM; j++){
     node[idx(i,j)].x = xmin + dx * i;
     node[idx(i,j)].y = ymin + dy * j;
     int b_id = 1; //interior node type = -1
     if(i == 0){   //x_min boundary = -1
       b_id = -1;
     }
     if(i == NN-1){//x_max boundary = -2
      b_id = -2;
     }
     if(j == 0){   //y_min boundary = -3
      b_id = -3;
     }
     if(j == MM-1){//y_max boundary = -4
      b_id = -4;
     }
     node[idx(i,j)].b = b_id;
   }
  }
  // record trangles' 3 vertices in counter clockwise order
  int elem = 0;
  for(int i = 0; i< NN-1; i++){ 
    for(int j = 0; j < MM-1 ; j++){
      element[elem].a = idx(i  , j  );
      element[elem].b = idx(i+1, j  );
      element[elem].c = idx(i  , j+1);
      elem ++;
      element[elem].a = idx(i+1, j  );
      element[elem].b = idx(i+1, j+1);
      element[elem].c = idx(i  , j+1);
      elem ++;
    }
  }

  void StructuredMeshGenerator::print_node(){
   PRT (node.size());
   for(unsigned n = 0; n< node.size(); n++){
     cout<<" index = "<<n<<"\ttx\txy" <<node[n].x<<"\t"<<node[n].y<<"\tb = "<<node[n].b<<endl;
   }
   cout<<endl;
   }

  void StructuredMeshGenerator::print_element(){
   PRT(element.size());
   for(unsigned e = 0; e< element.size(); e++){
     cout<<" index = "<<e<<"\ta, b, c = "  <<element[e].a<<"\t"<<element[e].b<<"\t"<<element[e].c<<endl;
   }
   cout<<endl;
  }
Coordinate StructuredMeshGenerator::n2c(int n) {
    return node[n];
}

TriVertex StructuredMeshGenerator::e2tv(int e) {
    return element[e];
}

int StructuredMeshGenerator::idx(int i, int j) {
    return i*MM + j;
}
/*
 *  Created on: Apr 10, 2009
 *      Author: Changwei Xiong
 *    Compiler: Visual Studio
 */

#ifndef BASICS_H_
#define BASICS_H_

#include <iostream>
#include <valarray>
#include <map>
using namespace std;

//PRINT Macro for debugging
#include <string>
#define PRT(st) cout<< " "+string((# st)) + " = " << (st) <<endl
#define Pause (cin.get())

typedef valarray<double> dval;

class Vector :public dval {
public:
  Vector (){}
  Vector (unsigned N_):dval(N_){};
  Vector (unsigned N_, double val):dval(val, N_){};
  Vector (dval & v):dval(v){};
  Vector (Vector & v):dval(v){};
  Vector & operator = (double val);
  Vector & operator = (dval & val);
  double* c_array();
  void print(unsigned adj = 20);
};

//************************************************************
//            dense matrix class by hash_map
//************************************************************
class DMatrix {
public:
  DMatrix (const unsigned N_, const unsigned M_, double val=0.0);
  dval & operator [] (int i);
  Vector operator * (const Vector& x);
  DMatrix operator * (const double val);
  DMatrix operator * (DMatrix& B);
  DMatrix operator + (DMatrix& B);
  DMatrix & operator += (DMatrix& B);
  DMatrix & operator -= (DMatrix& B);
  DMatrix inv ();
  DMatrix trans ();
  double det();
  void print_matrix();
private:
  unsigned N;
  unsigned M;
  valarray<dval> mat;
};

//************************************************************
//            sparse matrix class by map
//************************************************************
class SMatrix {
public:
  typedef map<unsigned, double> Row;
  SMatrix (const unsigned N_, const unsigned M_);
  ~SMatrix();
  double get(unsigned i, unsigned j);
  void set(unsigned i, unsigned j, double val);
  void convert(unsigned ia[], unsigned ja[], double a[]);
  // indices follows C convention
  unsigned n_rows ();//# of rows
  unsigned n_cols();//# of columns
  unsigned n_nz();//# of nonzeros
  void clear();
}
```cpp
82  void print_matrix();
83 private:
84  SMatrix& operator= (SMatrix&);
85
86  Row* mat;
87  unsigned N; // # of rows
88  unsigned M; // # of columns
89 );
90
91 //---------------------------------------------------------------
92 //            matrix class in triplet form
93 //---------------------------------------------------------------
94 class TMatrix{
95 public:
96  TMatrix(SMatrix& A);
97  TMatrix(TMatrix& A);
98  ~TMatrix();
99  Vector operator* (Vector& x);
100  double* mult(double x[], double b[]);
101  void set_identity(unsigned n);
102  void clear();
103  unsigned n_rows();
104  unsigned n_cols();
105  unsigned n_nz();
106  void print_matrix();
107 private:
108  unsigned* i_A;
109  unsigned* j_A;
110  double* val_A;
111  unsigned N;
112  unsigned M;
113  unsigned NZN;
114 #endif
```
/*
 *  Created on: Apr 10, 2009
 *      Author: Changwei Xiong
 *    Compiler: Visual Studio
 */

#include "Basics.h"

/** Vector member functions

 ************************************
 **  Vector member functions
 ************************************
*/

Vector& Vector::operator=(double val){
  dval::operator= (val);
  return *this;
}

Vector& Vector::operator=(dval& val){
  dval::operator= (val);
  return *this;
}

double* Vector::c_array(){
  return &(*this)[0];
}

void Vector::print(unsigned adj){
  cout << endl;
  for(unsigned i = 0; i< this->size(); i++){
    cout << (*this)[i] << "\n";
    if((i+1)%adj == 0)
      cout << endl;
  }
  cout <<endl<<endl;
}

//************************************************************
//            sparse matrix class by map
//************************************************************

#define BOUND_TEST(i,j) if( i >= N || j >= M){  \
  cout <<" Out of bound (SMatrix) ..."<<endl;  \
  exit(1);  \
}

SMatrix::SMatrix(const unsigned N_, const unsigned M_)
  :N(N_),
   M(M_){
  mat = new Row[N];
  }
SMatrix::~SMatrix(){
  clear ();
}

double SMatrix::get(unsigned i, unsigned j){
  BOUND_TEST (i,j);
  Row ::iterator iter = mat[i].find(j);
  return iter==mat[i].end() ? 0.0 : iter->second;
}

void SMatrix::set(unsigned i, unsigned j, double val){
  BOUND_TEST (i,j);
  if(0.0 == val){
    Row ::iterator iter = mat[i].find(j);
    if(iter != mat[i].end()){
      mat[i].erase(iter);
    }
  }else{
    mat[i][j] = val;
  }
  return;
}

void SMatrix::convert(unsigned ia[], unsigned ja[], double a[]){
  unsigned k = 0;
  Row::iterator it;
  for(unsigned i=0; i<N; i++){
    for( it = mat[i].begin(); it != mat[i].end() ; it++ ){
      a[k] = it->first;
      ja[k] = it->second;
      k++;
      ia[k] = i;
    }
  }
82   
83  
84  
85  // indices follows C convention
86 unsigned SMatrix::n_rows(){//# of rows
87  return N;
88 
89 
90  
91 unsigned SMatrix::n_cols(){//# of columns
92  return M;
93 
94 
95 unsigned SMatrix::n_nz(){//# of nonzeros
96  unsigned sum = 0;
97  for(unsigned i=0; i<N; i++){
98   sum += mat[i].size();
99  }
100  return sum;
101 
102 void SMatrix::clear(){
103  N = M = 0;
104  delete[] mat;
105  mat = NULL;
106 
107 
108 void SMatrix::print_matrix(){
109  cout <<"("<<N<<", "<<M<<")  
110  PRT (n_nz());
111  for(unsigned i = 0; i<N; i++){
112   for(unsigned j = 0; j<M; j++){
113    cout <<get(i,j)<<\\t";
114   }
115  
116  cout<<endl;
117  }
118  cout<<endl;
119 }
120 
121 #undef BOUND_TEST
122 
123  
124  */
125  ********************************************
126  ** Dense Matrix member functions
127  ********************************************
128  */
129 
130 DMatrix::DMatrix(const unsigned N_, const unsigned M_, double val):N(N_),M(M_){
131  mat .resize(5);
132  for(unsigned i = 0; i<N; i++){
133   mat [i].resize(M);
134   mat [i] = val;
135  }
136  }
137 
138 dval& DMatrix:: operator[](int i){
139  return mat[i];
140 
141 Vector DMatrix:: :operator* (const Vector& x){
142  if(M != x.size()){
143   cout<<"Dimensions not match (DMatrix) ...";
144   exit(1);
145 }  
146  
147 Vector b(M);
148  for(unsigned i = 0; i<N; i++){
149   double val = 0.0;
150  for(unsigned j = 0; j<M; j++){
151    val += mat[i][j] * x[j];
152  }
153  b[i] = val;
154  }
155  return b;
156 
157 DMatrix DMatrix:: :operator* (DMatrix& B){
158 if(N != B.N)|
159  cout<<"Dimensions not match (DMatrix) ...";
160  exit(1);
161  
162 
DMatrix C(N,B,M);
for(unsigned i = 0; i<C.N; i++){
  for(unsigned j = 0; j<C.M; j++){
    double val = 0.0;
    for(unsigned k = 0; k<M; k++){
      val += mat[i][k] * B[k][j];
    }
    C[i][j] = val;
  }
}
return C;

DMatrix DMatrix::operator* (const double val){
  DMatrix A(N,M);
  for(unsigned i = 0; i<N; i++){
    A[i] = mat[i] * val;
  }
  return A;
}

DMatrix DMatrix::operator+ (DMatrix& B){
  if(N != B.N || M != B.M){
    cout <<"Dimensions not match (DMatrix) ..."; exit(1);
  };
  DMatrix C(N,M);
  for(unsigned i = 0; i<N; i++){
    for(unsigned j = 0; j<M; j++){
      C[i][j] = mat[i][j] + B[i][j];
    }
  }
  return C;
}

DMatrix& DMatrix::operator+= (DMatrix& B){
  if(N != B.N || M != B.M){
    cout <<"Dimensions not match (DMatrix) ...";
    exit(1);
  };
  for(unsigned i = 0; i<N; i++){
    for(unsigned j = 0; j<M; j++){
      mat[i][j] = mat[i][j] + B[i][j];
    }
  }
  return (*this);
}

DMatrix DMatrix::operator- (DMatrix& B){
  if(N != B.N || M != B.M){
    cout <<"Dimensions not match (DMatrix) ...";
    exit(1);
  };
  DMatrix C(N,M);
  for(unsigned i = 0; i<N; i++){
    for(unsigned j = 0; j<M; j++){
      C[i][j] = mat[i][j] - B[i][j];
    }
  }
  return C;
}

DMatrix& DMatrix::operator-= (DMatrix& B){
  if(N != B.N || M != B.M){
    cout <<"Dimensions not match (DMatrix) ...";
    exit(1);
  };
  for(unsigned i = 0; i<N; i++){
    for(unsigned j = 0; j<M; j++){
      mat[i][j] = mat[i][j] - B[i][j];
    }
  }
  return (*this);
}

DMatrix DMatrix::inv(){
  if(N != 2 || M != 2){
    cout <<"2X2 matrix only (DMatrix) ...";
    exit(1);
  };
}
244 }
245 DMatrix R(2,2);
246 double a = mat[0][0];
247 double b = mat[0][1];
248 double c = mat[1][0];
249 double d = mat[1][1];
250 double f = 1.0/(a*d-b*c);
251 R[0][0] =  d * f;
252 R[0][1] = -b * f;
253 R[1][0] = -c * f;
254 R[1][1] =  a * f;
255 return R;
256 }
257 }
258 DMatrix DMatrix::trans(){
259 DMatrix T(M,N);
260 for(unsigned i = 0; i<N; i++){
261 for(unsigned j = 0; j<M; j++){
262 T[j][i] = mat[i][j];
263 }
264 } return T;
265 }
266 }
267 }
268 double DMatrix::det(){
269 if(N != 2 || M != 2){
270 cout <<"2X2 matrix only (DMatrix) ..."
271 exit(1);
272 return mat[0][0]*mat[1][1] - mat[0][1]*mat[1][0];
273 }
274 void DMatrix::print_matrix(){
275 cout <<"(<<N<"<<N<<","<<M<<")"endl;
276 for(unsigned i = 0; i<N; i++){
277 for(unsigned j = 0; j<M; j++){
278 cout <<mat[i][j]<<"t";
279 }
280 cout <<endl;
281 }
282 }
283 */
284 /* **********************************************/
285 ** Tri-Array Matrix member functions
286 **********************************************/
287 */
288 TMatrix::TMatrix(SMatrix& A){
289 if(A.n_nz() != 0){
290 N = A.n_rows();
291 M = A.n_cols();
292 NZN = A.n_nz();
293 i_A = new unsigned[NZN];
294 j_A = new unsigned[NZN];
295 val_A = new double[NZN];
296 for(int k=0; k<NZN; k++){
297 i_A[k] = A.i_A[k];
298 j_A[k] = A.j_A[k];
299 val_A[k] = A.val_A[k];
300 } else{
301 i_A = NULL;
302 j_A = NULL;
303 val_A = NULL;
304 N = M = NZN = 0;
305 }
306 }
307 TMatrix::TMatrix(TMatrix& A){
308 if(A.NZN != 0){
309 N = A.N;
310 M = A.M;
311 NZN = A.NZN;
312 i_A = new unsigned[NZN];
313 j_A = new unsigned[NZN];
314 val_A = new double[NZN];
315 for(int k=0; k<NZN; k++){
316 i_A[k] = A.i_A[k];
317 j_A[k] = A.j_A[k];
318 val_A[k] = A.val_A[k];
319 } else{
i_A = NULL;
j_A = NULL;
val_A = NULL;
N = M = NZN = 0;
}
}

TMatrix::~TMatrix(){
clear();
}

Vector TMatrix::operator* (Vector& x){
if(x.size() != M){
cout<<"Dimensions not match ...");
return x;
}
Vector b(M);
mult(x.c_array(), b.c_array());
return b;
}

double* TMatrix::mult(double x[], double b[]){
for (unsigned i = 0; i < M; i++) {
b[i] = 0.0;
}
for (unsigned k=0; k < NZN; k++) {
b[i_A[k]] += val_A[k] * x[j_A[k]];
}
return b;
}

void TMatrix::set_identity(unsigned n){
for (unsigned k=0; k < NZN; k++) {
if(i_A[k] == n || j_A[k] == n){
if(i_A[k] == n && j_A[k] == n){
val_A[k] = 1.0;
}else{
val_A[k] = 0.0;
}
}
}
}

void TMatrix::clear(){
delete[] i_A;
delete[] j_A;
delete[] val_A;
i_A = NULL;
j_A = NULL;
val_A = NULL;
N = M = NZN = 0;
}

unsigned TMatrix::n_rows(){
return N;
}
unsigned TMatrix::n_cols(){
return M;
}

unsigned TMatrix::n_nonzeros(){
return NZN;
}

void TMatrix::print_matrix(){}
#ifndef PARAMETERS_H_
#define PARAMETERS_H_

class Parameter{
public:
  double S0; //initial spot
  double v0; //initial variance
  double th; //theta
  double kp; //kappa
  double xi; //xi, volatility of variance
  double rho; //rho, covariance
  double ir; //interest rate
  double dr; //dividend rate
  double Ko; //option strike
  double Tm; //maturity
  double vmin; //min variance
  double vmax; //max variance
  double Smax; //max spot
  double Smin; //min spot
  int type; //option type

  /*
   * Vanilla       : Put = -1, Call = 1; 
   * Double Barrier: Put = -2, Call = 2; 
   */
  unsigned N; //resolution in variance
  unsigned M; //resolution in spot
  unsigned Z; //resolution in Time
};

#endif//PARAMETERS_H_
class CGS {
public:
    void solve(TMatrix& A,
               Vector& x_,
               Vector& rhs_,
               const unsigned iter_max,
               const double eps);
private:
    unsigned N;
    double dot(double x[], double y[]);
    double norm(double x[]);
};

#endif // CGS_H_
#include <iostream>
using namespace std;
#include <math.h>
#include "CGS.h"

// PRINT Macro for debuggig
#include <string>
define PRT(st) cout<< " " + string((# st)) + " = " << (st) <<endl

typedef double* const DArray;

void CGS::solve(TMatrix& A, Vector& x_, Vector& b_, const unsigned max_iter, const double tol){
  N = A.n_cols();
  if( N == 0 ||
    N != A.n_rows() ||
    N != x_.size() ||
    N != b_.size() ) {
    cout << "CGS - invalid parameters ...
    return;
  }

  DArray x  = x_.c_array();
  DArray b  = b_.c_array();

  DArray p  = new double[N*6];
  DArray q  = p + N*1;
  DArray v  = p + N*2;
  DArray u  = p + N*3;
  DArray r  = p + N*4;
  DArray r0 = p + N*5;

  A.mult(x, r); //tmp = A * x;
  for (unsigned i = 0; i < N; i++ ) {
    r[i] = b[i] - r[i];// r = b - A * x;
  }
  for (unsigned i = 0; i < N; i++ ) {
    r0[i] = r[i]; //r0 = r;
  }

  double rho1, rho2, alpha, beta, sigma, normb_tol;
  if(norm(b) == 0.0){
    normb_tol = tol;
  }else{
    normb_tol = tol * norm(b);
  }

  if (norm(r) > normb_tol){
    for (unsigned k = 1; ; k++) {
      if(k > max_iter){
        cout << " CGS: NOT converged after " << k << " iterations, rho = " << norm(r)/(normb_tol/tol) << endl;
        break;
      }
      rho1 = dot(r, r0);
      if (rho1 == 0){
        break;
      }
      rho2 = rho1 / rho2;
      if (k == 0){
        p[i] = u[i] = r[i]; //u = r; p = u;
      } else {
        beta = rho1 / rho2;
        for (unsigned i = 0; i < N; i++ ) {
          if (k > max_iter){
            cout << " CGS: NOT converged after " << k << " iterations, rho = " << norm(r)/(normb_tol/tol) << endl;
            break;
          }
          if((k == 1){
            p[i] = u[i] = r[i]; //u = r; p = u;
          } else {
            beta = rho1 / rho2;
            for (unsigned i = 0; i < N; i++ ) {
              u[i] = r[i] + beta * q[i]; //u = r + beta * q;
              p[i] = u[i] + beta * (q[i] + beta*p[i]); //p = u + beta*(q + beta*p);
            }
          }
        }
      }
    }
  }
}
for (unsigned i = 0; i < N; i++) {
    q[i] = u[i] - alpha * v[i]; // q = u - alpha * v;
    u[i] += q[i]; // Preconditing here if applicable, u = M.solve(u + q);
    x[i] += alpha * u[i]; // x = x + alpha * u;
}
A.mult(u, v); // v = A * u; v is a tmp vector;
for (unsigned i = 0; i < N; i++) {
    r[i] -= alpha * v[i]; // r = r - alpha * v;
}
rho2 = rho1;
if (norm(r) < normb_tol) {
    cout << "CGS: Converged after " << k << " iterations, rho = " << norm(r)/(normb_tol/tol) << endl;
    break;
}
delete[] p;
}
return;
}
//****************************************************************************
double CGS::dot (double x[], double y[]) {
    double sum = 0.0;
    for (unsigned i = 0; i < N; i++) {
        sum += x[i] * y[i];
    }
    return sum;
}
double CGS::norm (double x[]) {
    double sum = 0.0;
    for (unsigned i = 0; i < N; i++) {
        sum += x[i] * x[i];
    }
    return sqrt(sum);
/*
 *  Created on: Apr 10, 2009
 *      Author: Changwei Xiong
 *    Compiler: Visual Studio
 */

#ifndef MGMRES_H_
#define MGMRES_H_
#include "Basics.h"

class MGMRES {
 public:
  void solve(TMatrix& A, Vector& x_, Vector& rhs_, const unsigned mr, const double eps);

 private:
  unsigned N;
  void mult_Givens(double csin, double sin, double g[2]);
  double dot(double x[], double y[]);
  double norm(double x[]);
  void zeros(double x[], unsigned n);
};

#endif // MGMRES_H_
/*
 *  Created on: Apr 10, 2009
 *      Author: Changwei Xiong
 *    Compiler: Visual Studio
 */

#include <iostream>
#include "MGMRES.h"

//PRINT Macro for debugging
#define PRT(st) cout<< " " + string((# st)) + " = " << (st) <<endl

typedef double* const DArray;

void MGMRES::solve(TMATRIX A, Vector &x_, Vector &rhs_, unsigned mr, const double eps) {
  double iter_max = 100;
  N = A.n_cols();
  if(mr > N)
    mr = N;

  if( N != A.n_rows() ||
      N != x_.size() ||
      N != rhs_.size() ||
      N < mr ){
    cout << "MGMRES - invalid parameters ...
    return;
  }
  DArray x = x_.c_array();
  DArray rhs = rhs_.c_array();

  const unsigned ms = mr+1;
  DArray g = new double[ms];
  DArray y = new double[ms];
  DArray r = new double[N];
  DArray H = new double[mr * ms];
  DArray V = new double[N * ms];
  DArray csin = new double[mr];
  DArray sine = new double[mr];

  unsigned itr_total = 0;
  double rho, rho_tole;
  const double tole_rel = eps;
  const double tole_abs = eps;
  for (unsigned iter = 1; iter <= iter_max; iter++ ) {
    A.mult(x, r);
    for (unsigned i = 0; i < N; i++ ) {
      r[i] = rhs[i] - r[i];
    }
    rho = norm(r);
    if(iter == 1) {
      rho_tole = rho * tole_rel;
    }
    for (unsigned i = 0; i < N; i++) {
      V[i+0*N] = r[i] / rho;
    }
    zeros(H, ms*mr);
    g[0] = rho;
    unsigned k, k_old;
    for(k = 1; k <= mr; k++) { //inner iteration
      k_old = k;
      DArray v_k = &V[k*N];
      DArray h_f = &H[(k-1)*ms]; // index f = k - 1;
      A.mult(v_k-N, v_k);
      double av = norm(v_k);
      for (unsigned j = 0; j < k; j++ ){
        DArray v_j = &V[j*N];
        register double dp = h_f[j] = dot(v_k, v_j);
        for (unsigned i = 0; i < N; i++ ){
          v_k[i] -= dp * v_j[i];
        }
      }
      //heavy computation falls into this loop
      for (unsigned j = 0; j < k; j++){
        for (unsigned i = 0; i < N; i++){
          V[i+0*N] = r[i] / rho;
        }
      }
      for (unsigned i = 0; i < N; i++) {
        V[i+0*N] = r[i] / rho;
      }
      zeros(H, ms*mr);
      g[0] = rho;
      for(k = 1; k <= mr; k++) { //inner iteration
        k_old = k;
        DArray v_k = &V[k*N];
        DArray h_f = &H[(k-1)*ms]; // index f = k - 1;
        A.mult(v_k-N, v_k);
        double av = norm(v_k);
        for (unsigned j = 0; j < k; j++ ){
          DArray v_j = &V[j*N];
          register double dp = h_f[j] = dot(v_k, v_j);
          for (unsigned i = 0; i < N; i++ ){
            v_k[i] -= dp * v_j[i];
          }
        }
      }
    }
  }
}
82    h_f[k] = norm(v_k);
83
84    // in case of tiny h_f[k]; very rarely hits here
85    if ((av + 1e-3 * h_f[k]) == av) {
86      for (unsigned j = 0; j < k; j++) {
87        DArray v_j = &V[j*N];
88        double dp = dot(v_k, v_j);
89        h_f[j] += dp;
90      }
91      for (unsigned i = 0; i < N; i++) {
92        v_k[i] -= dp * v_j[i];
93      }
94    }
95
96    h_f[k] = norm(v_k);
97
98    if ( h_f[k] != 0.0 ) {
99      for (unsigned i = 0; i < N; i++ ) {
100        v_k[i] /= h_f[k];
101      }
102    }
103  }
104
105  if(k > 1) {
106    for (unsigned i = 0; i <= k; i++) {
107      y[i] = h_f[i];
108    }
109    for (unsigned j = 0; j < k-1; j++) {
110      mult_Givens(csin[j], sine[j], &y[j]);
111    }
112  }
113
114  //Given's rotation
115  double one_over_mu = 1./sqrt(h_f[k-1]*h_f[k-1] + h_f[k]*h_f[k]);
116  csin[k-1] = h_f[k-1] * one_over_mu;
117  sine[k-1] = -h_f[k]   * one_over_mu;
118  h_f[k-1] = csin[k-1] * h_f[k-1] - sine[k-1] * h_f[k];
119  h_f[k] = 0;
120  mult_Givens(csin[k-1], sine[k-1], &g[k-1]);
121
122  rho = fabs(g[k]);
123  if ( rho <= rho_tole && rho <= tole_abs ) {
124    cout << " MGMRES: Converged after " << iter << " outer and " << k << " inner iterations, rho = " << rho << endl;
125    break;
126  }
127
128  } // end inner iteration
129
130  if ( rho <= rho_tole && rho <= tole_abs ) {
131    cout << " Total iterations = " << itr_total << " Final residual = " << rho << endl;
132    break;
133  }
134
135  //cout<< Total iterations = "<<itr_total<<" Final residual = "<<rho<<endl;
136  delete [] g;
137  delete [] y;
138  delete [] r;
139  delete [] V;
140  delete [] H;
141  delete [] csin;
142  delete [] sine;
void MGMRES::mult_Givens(double csin, double sine, double g[2])
    double g1 = csin * g[0] - sine * g[1];
    double g2 = sine * g[0] + csin * g[1];
    g[0] = g1;
    g[1] = g2;
}

double MGMRES::dot (double x[], double y[]) {
    double sum = 0.0;
    for (unsigned i = 0; i < N; i++) {
        sum += x[i] * y[i];
    }
    return sum;
}

double MGMRES::norm (double x[]) {
    double sum = 0.0;
    for (unsigned i = 0; i < N; i++) {
        sum += x[i] * x[i];
    }
    return sqrt(sum);
}

void MGMRES::zeros(double x[], unsigned n) {
    for (unsigned i = 0; i < n; i++) {
        x[i] = 0.0;
    }
}
/* heston.h: header for analytic solution for HESTON's model
   *
   * author: Gunter Winkler (gw)
   *         gunter.winkler@mathematik.tu-chemnitz.de
   * *
   * changes:
   * gw 2001-08-07 new
   * *
   */

#ifndef HESTONCLOSEDFORM_H_
#define HESTONCLOSEDFORM_H_

#include <iostream>
#include <cmath>
#include <complex>
#include "Parameter.h"

typedef std::complex<double> heston_complex;

#ifdef DEBUG
#define DEBUG_HESTON(x)     \
   x
#else
#define DEBUG_HESTON(x)     \
#endif

const heston_complex imag = heston_complex(0,1);

struct MARKETDATA {
  double rd, rf; /* (logarithmic) domestic/foreign interest rates */
  double rho, kappa, theta, xi; /* stochastic volatility parameters */
  double lambda; /* price of volatility risk */
};

double Pj ( const int j,
            const double x,
            const double v,
            const double tau,
            const double logstrike,
            MARKETDATA market );

double Heston_Call(const Parameter& P);

/*********** last line of heston.h **************/
#endif
/* heston.C: implementation for analytic solution of HESTON's Model
   for stochastic volatility
   Copyright (C) 2001 Gunter Winkler (guwi17@gmx.de)

   This program is free software; you can redistribute it and/or modify
   it under the terms of the GNU General Public License as published by
   the Free Software Foundation; either version 2 of the License, or
   (at your option) any later version.

   This program is distributed in the hope that it will be useful,
   but WITHOUT ANY WARRANTY; without even the implied warranty of
   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
   GNU General Public License for more details.

   You should have received a copy of the GNU General Public License
   along with this program; if not, write to the Free Software
   Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

   author: Gunter Winkler (gw)
   gunter.winkler@mathematik.tu-chemnitz.de

   paper: 'A closed-form solution for options with stochastic volatility'
   by Heston, L., 1995 Volatility/Risk Publications

   changes:
   gw 2001-08-07 new
   gw 2001-08-23 made public first time
   */

#include "HestonClosedForm.h"
#define Pi 3.14159265358979323846264338328
#define Pi2 1.57079632679489661923132169164

/* names: phi = 1: call, phi = -1: put */

/* some defines */
#define _2U(j) ((j==1)?(1):(-1))
#define _A   (market.kappa * market.theta)
#define _B(j)  ((j==1)?(market.kappa+market.lambda-market.rho*market.xi):(market.kappa+market.lambda))

static heston_complex Fj( const int j, /* j == 1 or j != 1 */
         const double x,
         const double v,
         const double tau,
         const double phi,
         const MARKETDATA market)
{ /* helper function f_j(x,v,tau;phi) from paper */

const double x, v, tau, phi;
const double phi2, xi2, d;
const double xi2_by_xi,
const MARKETDATA market;

/* input: x, v, tau, phi */

rp = market.rho * market.xi * phi;
x12 = market.xi * market.xi;
z1 = heston_complex(_B(j), -rp);

d = z12 + z1 * heston_complex(-phi*phi, _2U(j) * phi);
d = sqrt(d);

DEBUGHESTON(std::cout << " input: " << x << " , " << v << " , " << tau << " , " << phi << endl;)

DEBUGHESTON(std::cout << " input: " << x << " , " << v << " , " << tau << " , " << phi << endl;)

DEBUGHESTON(std::cout << " input: " << x << " , " << v << " , " << tau << " , " << phi << endl;)

DEBUGHESTON(std::cout << " input: " << x << " , " << v << " , " << tau << " , " << phi << endl;)

D = z12 * heston_complex(-phi*phi, _2U(j) * phi);

d = sqrt(d);

DEBUGHESTON(std::cout << " d = " << d << endl;)

DEBUGHESTON(std::cout << " d = " << d << endl;)

DEBUGHESTON(std::cout << " D = " << D << endl;)

/* names: phi = 1: call, phi = -1: put */

/* some defines */
#define _2U(j) ((j==1)?(1):(-1))
#define _A   (market.kappa * market.theta)
#define _B(j)  ((j==1)?(market.kappa+market.lambda-market.rho*market.xi):(market.kappa+market.lambda))

static heston_complex Fj( const int j, /* j == 1 or j != 1 */
         const double x,
         const double v,
         const double tau,
         const double phi,
C = heston_complex((double)0.0,(market.rd-market.rf)*phi*tau)
    + _A/xi2 * ( z2 * tau - (double)2.0 * log(((double)1.0-g*exp(d*tau))/((double)1.0-g)) );

DEBUG_HESTON(std::cout << " C " << C << endl;)

return exp( C + D * v + heston_complex(0.0, phi*x) );

double Pj ( const int j, const double x, const double v, const double tau, const double logstrike, MARKETDATA market )

/*
conditional probability P_j(x,v,tau;phi) from paper
*/
{
const double phimin = 0.0;
const double phimax = 100.0*Pi;
const int  steps  = 100;
const double stepsize = (phimax-phimin)/steps;

const int  N = 3;

const double x_i[N] = { .11270165379258311482073460022,
                      .5,
                      .8872983462074168517926539978 };
const double alpha_i[N] = { .277777777777777777777777777778,
                          .444444444444444444444444444444,
                          .277777777777777777777777777778 };

heston_complex integrand;

double  phi, sum;
int   i,k;

sum = 0;
for (i=steps-1;i>=0;--i)
{
  for (k=N-1;k>=0;--k)
  {
    phi = (i + x_i[k]) * stepsize;
    integrand = (heston_complex) std::polar(1/phi,-phi*logstrike) * (heston_complex(0,-1.0)) * Fj(j,x,v,tau,phi,market);
    sum += alpha_i[k] * integrand.real();
    DEBUG_HESTON (std::cout << " phi " << phi << " int " << integrand << " sum " << sum << endl;)
  }
}
sum *= stepsize;
return 0.5 + sum / Pi;

}

double Heston_Call (const Parameter& P)

/* formula for vanilla call (same as Black-Scholes) */
{
const double spot = P.S0;
const double v = P.v0;
const double tau = P.Tm;
const double strike = P.Ko;
const double kappa = P.kp;
const double lambda = 0.0;
const double rd = P.ir;
const double rf = P.dr;
const double rho = P.rho;
const double theta = P.th;
const double xi = P.xi;
double x, is, pl, p2;
MARKETDATA market;
market.kappa = kappa;
market.lambda = lambda;
market.rd = rd;
market.rf = rf;
market.rho = rho;
market.theta = theta;
market.xi = xi;

x = log(spot);
ls = log(strike);

p1 = Pj(1, x, v, tau, ls, market);
p2 = Pj(2, x, v, tau, ls, market);

return spot * exp(-market.rf * tau) * p1 - strike * exp(-market.rd * tau) * p2;