Master's Programme in Quantitative Finance

A neural network approach to pricing of a European Call Option with the Heston model

by

Sandra Patricia Guerrero Torres,
sandrapa.guerrero@urosario.edu.co

Abstract In this thesis, we implement deep learning to option pricing. A data-driven approach is proposed, through an Artificial Neural Network (ANN), to calculate the price of European call options with the Heston stochastic volatility model, in order to accelerate the numerical methods and show the ability of Artificial Neural Network to "learn" the model from the dataset.

Keywords: neural networks; option pricing; Heston Model, Finite Differences

Master's Thesis (5 credits)
November 2019
Supervisor: Hugo Eduardo Ramirez

Contents

Introduction

There has been recent considerable interest in the development of artificial neural networks (ANNs) for solving a variety of problems, since the universal approximation theorem states that a feedback network can approximate almost any known function. This thesis examines the application of neural networks in the context of pricing options.
We test this approach on two different types of solvers for the valuation of a European call option with the Heston stochastic volatility model (1993), including valuation using the Fourier-based approach (Lewis, 2001) and using finite difference methods. This thesis is divided into three parts. The first part is concerned with the derivation of the Heston model. The second part covers a number of topics for the option pricing through finite difference methods (FDM). The third part finally covers the major aspects related to train an optimized ANN on a data set generated with a data-driven approach.

1.1 Research Problem

The traditional approach to the pricing of European-style options on an underlying asset assumes that the asset price follows a given exogenous process and prices the options using an arbitrage-free hedging argument. The most relevant example is presented in the article of Black-Scholes (1973) (BSM) on the valuation of European-style options, this approach provides a closed-form valuation formula that can be used to efficiently price plain vanilla options. However, this method is based on several assumptions that are not representative of the real world. In particular, the BSM model assumes that volatility is deterministic and remains constant through the option's life, which clearly contradicts the behavior observed in financial markets. While the BSM framework can be adapted to obtain reasonable prices for plain vanilla options, the constant volatility assumption may lead to significant mispricings when used to evaluate options with non-conventional or exotic features. During the last decades several alternatives have been proposed to improve volatility modeling in the context of derivatives pricing. One of such approaches is to model volatility as a stochastic quantity. By introducing uncertainty in the behavior of volatility, the evolution of financial assets can be estimated more realistically. One of the most widely used stochastic volatility models was proposed by Heston in 1993. The Heston model introduces a dynamic for the underlying asset which can take into account the asymmetry and excess kurtosis that are typically observed in financial assets returns, this model is based on a two-dimensional stochastic diffusion process

with two Brownian movements with correlation ρ, which poses the problem of solving a system of two stochastic equations under the neutral risk measure, one of these equations describes the dynamics of the underlying price and the other its variance, since the valuation model is two-dimensional, by adding stochastic volatility, it is not possible to have an analytical solution for the valuation of options with the Heston model, which is why different numerical methods have therefore been developed to solve the corresponding option pricing partial differential equation problems, e.g. finite difference methods, Fourier approaches and Monte Carlo simulation.

In the context of model calibration, thousands of option prices need to be determined in order to fit the asset parameters. However, due to the requirement of a highly efficient computation, certain high-quality asset pricing valuation models are discarded. Efficient numerical computation is also increasingly important in financial risk management, especially when we deal with real-time risk management (e.g., high frequency trading) or counterparty credit risk issues, where a trade-off between efficiency and accuracy seems often inevitable.
Artificial neural networks (ANNs) with multiple hidden layers have become successful machine learning methods to extract features and detect patterns from a large data set, we aim to take advantage of a classical ANN to speed up option valuation by learning the results of an option pricing method. From a computational point of view, the ANN does not suffer much from the so-called curse of dimensionality of classical numerical methods for solving PDEs.

Given the application of neural networks in finance, and the advantages already described, in this thesis we test two different types of solvers for the valuation of a European call option with the Heston stochastic volatility model, including valuation using the Fourier-based approach (Lewis, 2001), knowing the characteristic function of the stochastic process; and on the other hand, using finite difference methods, which are capable of solving problems in which the value of the derivative satisfies an PDE with a temporal variable and a few spatial variables (underlying assets).
The FDM are numerical methods for solving differential equations by approximating them with difference equations, in which finite differences approximate the derivatives. This method is used to solve the PDE associated with the stochastic volatility model, since it is easy to implement, flexible and offers ways to improve the accuracy of the results and reduce the computational cost, using an implicit type of alternative management (ADI) scheme.
Once Heston prices are determined, through the Fourier-based approach and the application of FDM, we train ANN and test its performance for each data set.

Theory

This chapter discusses key ideas, explanations, concepts, models and theories, used as the basis for this research. Most of these concepts were taken from (Eric Chin and Ólafsson, 2014).

2.1 Taylor Series

The most used tool for discretizing is the approximation by Taylor series. In this work we use this theory for numerical approximations in Fourier integration and finite differences, that is why we give an brief description of this tool next.
If $f(x)$ is an analytic function of x, then for small h

$$f(x_0+h) = f(x_0) + f^{'}(x_0)h + \frac{1}{2!}f^{''}(x_0)h^2 + \frac{1}{3!}f^{'''}(x_0)h^3 + \cdots$$

If $f(x,y)$ is an analytic function of x and y, then for small Δx, Δy:

$$\begin{aligned} f(x_0+\Delta x, y_0+\Delta y) = f(x_0,y_0) &+ \frac{\partial f(x_0,y_0)}{\partial x}\Delta x + \frac{\partial f(x_0,y_0)}{\partial y}\Delta y \\ &+ \frac{1}{2!}\left[\frac{\partial^2 f(x_0,y_0)}{\partial x^2}(\Delta x)^2 + 2\frac{\partial^2 f(x_0,y_0)}{\partial x \partial y}\Delta x \Delta y + \frac{\partial^2 f(x_0,y_0)}{\partial y^2}(\Delta y)^2\right] \\ &+ \frac{1}{3!}\left[\frac{\partial^3 f(x_0,y_0)}{\partial x^3}(\Delta x)^3 + 3\frac{\partial^3 f(x_0,y_0)}{\partial x^2 \partial y}(\Delta x)^2 \Delta y \right. \\ &\left. + 3\frac{\partial^3 f(x_0,y_0)}{\partial x \partial y^2}\Delta x(\Delta y)^2 + \frac{\partial^3 f(x_0,y_0)}{\partial y^3}(\Delta y)^3\right] + \cdots \end{aligned}$$

2.2 Itô's formula

In mathematics, Itô's formula (or lemma) is used in stochastic calculus to find the differential of a function of a particular type of stochastic process. In essence, it is the stochastic calculus counterpart of the chain rule in ordinary calculus via a Taylor series expansion.

2.2.1 Theorem (one-dimensional Itô's formula)

Let $\{W_t : t \geq 0\}$ be a standard Wiener process on the probability space $(\Omega, \mathscr{F}, \mathbb{P})$ and let $\mathscr{F}_t$, $t \geq 0$ be the associated filtration. Consider a stochastic process X_t

satisfying the following stochastic differential equation (SDE):

$$dX_t = \mu(X_t, t)dt + \sigma(X_t, t)dW_t$$

or in integrated form,

$$X_t = X_0 + \int_0^t \mu(X_s, s)ds + \int_0^t \sigma(X_s, s)dW_s$$

with $\int_0^t \left[|\mu(X_s, s)| + \sigma(X_s, s)^2\right] ds < \infty$.
Then for any twice differentiable function $g(X_t, t)$, the stochastic process $Y_t = g(X_t, t)$ satisfies:

$$\begin{aligned} dY_t &= \frac{\partial g}{\partial t}dt + \frac{\partial g}{\partial X_t}dX_t + \frac{1}{2}\frac{\partial^2 g}{\partial X_t^2}(dX_t)^2 \\ &= \frac{\partial g}{\partial t}dt + \frac{\partial g}{\partial X_t}(\mu dt + \sigma dW_t) + \frac{1}{2}\frac{\partial^2 g}{\partial X_t^2}(\sigma^2 dW_t^2) \\ &= \left[\frac{\partial g}{\partial t} + \mu\frac{\partial g}{\partial X_t} + \frac{1}{2}\sigma^2\frac{\partial^2 g}{\partial X_t^2}\right] dt + \sigma\frac{\partial g}{\partial X_t}dW_t \end{aligned}$$

where $(dX_t)^2$ is the quadratic variation process, which is commonly simplified according to the rule: $(dW_t)^2 = dt$, $(dt)^2 = dW_t \cdot dt = dt \cdot dW_t = 0$, as above.

2.3 Ornstein - Uhlenbeck Process

The Ornstein-Uhlenbeck process is a diffusion process with applications in financial mathematics and the physical sciences. In finance, it has appeared as a model of the volatility of the underlying asset price process, so it is used as a starting point for the volatility process in the Heston model, that is why we give the following definition.
Let $(\Omega, \mathscr{F}, \mathbb{P})$ be a probability space and let $\{W_t : t \geq 0\}$ be a standard Wiener process. Suppose X_t follows the Ornstein-Uhlenbeck process with SDE

$$dX_t = \kappa(\theta - X_t)dt + \sigma dW_t$$

where κ, θ and σ are constants. By applying Itôs's formula to $Y_t = e^{\kappa t}X_t$ and taking integrals, we can show for $t < T$:

$$X_T = X_t e^{-\kappa(T-t)} + \theta\left[1 - e^{-\kappa(T-t)}\right] + \int_t^T \sigma e^{-\kappa(T-s)}dW_s$$

2.4 The Fourier transform and its inverse

There are several definitions of the Fourier transform $\hat{f}$ of a function f, the one usually encountered in the mathematical literature and used by (Gupta, 2019) is:
Let f be a real valued function such that $f(x)$ and $f'(x)$ are piecewise continuous

in every finite interval and the integral of $|f(x)|$ exists from $-\infty$ to ∞. The Fourier transform of the the function f is:

$$\hat{f}(w) = \int_{-\infty}^{\infty} e^{iwx} f(x) dx$$

where $i = \sqrt{-1}$ is the imaginary unit, w either real or complex and e^{iwx} is called the phase factor.
The original function f can be recovered from $\hat{f}$ via the inverse Fourier transform:

$$f(x) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \hat{f}(w) e^{iwx} dw$$

and the expression on the right-hand side is called the inverse Fourier transform of the function $\hat{f}(w)$.

2.4.1 Fourier transform of derivative

If f is any function for which the Fourier transforms of $f(x)$ and $f'(x)$ exist and $f(x) \to 0$ as $|x| \to \infty$ then:

$$\begin{aligned} \widehat{f'}(w) &= \int_{-\infty}^{\infty} e^{iwx} f'(x) dx \\ &= -iw\hat{f}(w) \end{aligned} \tag{2.1}$$

To verify equation (2.1) we using integration by parts:

$$\begin{aligned} u &= e^{iwx} & dv &= f'(x) \\ du &= iwe^{iwx} dx & v &= f(x) \end{aligned}$$

This yields:

$$\begin{aligned} \widehat{f'}(w) &= \underbrace{e^{iwx} f(x)\Big|_{-\infty}^{\infty}}_{=0} - iw \int_{-\infty}^{\infty} e^{iwx} f(x) dx \\ &= -iw\hat{f}(w) \end{aligned}$$

The first term must vanish, as we assume f is absolutely integrable on $\mathbb{R}$, $f(x) \to 0$ as $|x| \to \infty$ and the complex exponential is bounded.

Applying integration by parts once more shows that $\widehat{f''}(w) = (-iw)^2 \hat{f}(w)$, while a repeated application shows that the Fourier transform of the derivative of order n is $(-iw)^n \hat{f}(w)$.

Heston Model

In this chapter, we present a complete derivation of the European Call price under the Heston model. First we present the model and obtain the various partial differential equations (PDEs) that arise in the derivation. We show that the call price in the Heston model can be expressed as the sum of two terms that each contains an in-the-money probability, but obtained under a separate measure. We show how to obtain the characteristic function for the Heston model, and how to solve the differential equation from which the characteristic function is derived. We then show how to compute the price of a European call.
The key ideas were taken from (Rouah, 2013).

3.1 Model dynamics

The Heston model assumes that the underlying stock price, S_t follows a Black-Scholes-type stochastic process, but with a stochastic variance ν_t that follows a (Cox and Ross, 1985) process.
Hence, the Heston model is represented by the bivariate system of stochastic differential equations (SDEs)

$$\begin{aligned} dS_t &= \mu S_t dt + \sqrt{\nu_t} S_t dW_t^S \\ d\nu_t &= \kappa(\theta - \nu_t)dt + \sigma\sqrt{\nu_t} dW_t^\nu \end{aligned} \tag{3.1}$$

where $\mathbb{E}^{\mathbb{P}}[dW_t^S, dW_t^\nu] = \rho dt$.
Here $\mathbb{P}$ denotes the historical measure, also called the physical measure.
We will sometimes drop the time index and write $S = S_t$, $\nu = \nu_t$, $W^S = W_t^S$, and $W^\nu = W_t^\nu$, for notational convenience.

The parameters of the model are:
μ the drift of the process for the stock;
$\kappa > 0$ the mean reversion speed for the variance;
$\theta > 0$ the mean reversion level for the variance;
$\sigma > 0$ the volatility of the variance;
$\nu_0 > 0$ the initial (time zero) level of the variance;
$\rho \in [-1, 1]$ the correlation between the two Brownian motions W^S and W^ν.

It is important to note that the volatility $\sqrt{\nu_t}$ is not modeled directly in the Heston model, but rather through the variance ν_t. The process for the variance arises from the 2.3. Ornstein - Uhlenbeck for the volatility $h_t = \sqrt{\nu_t}$ given by:

$$dh_t = -\beta h_t dt + \delta dW_t^\nu \tag{3.2}$$

Applying 2.2.1. Itô's lemma, $\nu_t = h_t^2$ follows the process $\nu_t = g(h_t)$

$$\begin{aligned} d\nu_t &= \frac{\partial g}{\partial h_t} dh_t + \frac{1}{2}\frac{\partial^2 g}{\partial h_t^2}(dh_t)^2 \\ &= 2h_t dh_t + \frac{1}{2}\cdot 2\,(dh_t)^2 \\ &= 2h_t(-\beta h_t dt + \delta dW_t^\nu) + (\beta^2 h_t^2 \underbrace{(dt)^2}_{=0} - 2\beta\delta h_t \cdot \underbrace{dt \cdot dW_t^\nu}_{=0} + \delta^2 \underbrace{(dW_t^\nu)^2}_{=dt}) \\ &= -2\beta h_t^2 dt + 2\delta h_t dW_t^\nu + \delta^2 dt \\ &= (\delta^2 - 2\beta\nu_t)dt + 2\delta\sqrt{\nu_t} dW_t^\nu \end{aligned} \tag{3.3}$$

Defining $\kappa = 2\beta$, $\theta = \frac{\delta^2}{2\beta}$, and $\sigma = 2\delta$:

$$\begin{aligned} d\nu_t &= (\kappa\theta - \kappa\nu_t)dt + \sigma\sqrt{\nu_t} dW_t^\nu \\ &= \kappa(\theta - \nu_t)dt + \sigma\sqrt{\nu_t} dW_t^\nu \end{aligned}$$

Therefore, it can expresses $d\nu_t$ from equation (3.1) as (3.3).

3.2 The Heston PDE

In this section, we explain how to derive the PDE for the Heston model. The argument is similar to the hedging argument that uses a single derivative to derive the Black-Scholes PDE. In the Black-Scholes model, a portfolio is formed with the underlying stock, plus a single derivative which is used to hedge the stock and lender the portfolio riskless. In the Heston model, however, an additional derivative is required in the portfolio, to hedge the volatility.
Hence, we form a portfolio consisting of one option $V = V(S, \nu, t)$, Δ_1 units of the non-dividend paying stock, and Δ_2 units of another option $U(S, \nu, t)$ for the volatility hedge.
The portfolio has value

$$\Pi = V + \Delta_1 S + \Delta_2 U$$

where the t subscripts are omitted for convenience.
Assuming the portfolio is self-financing, the change in portfolio value is

$$d\Pi = dV + \Delta_1 dS + \Delta_2 dU \tag{3.4}$$

The strategy is similar to that for the Black-Scholes case. We apply 2.2.1. Itô's lemma to obtain the processes for U and V, which allows us to find the process for Π. We then find the values of Δ_1 and Δ_2 that makes the portfolio riskless, and we use the result to derive the Heston PDE.
First apply Itô's lemma to $V(S, \nu, t)$:

$$dV = \frac{\partial V}{\partial t}dt + \frac{\partial V}{\partial S}dS + \frac{\partial V}{\partial \nu}d\nu + \frac{1}{2}\frac{\partial^2 V}{\partial S^2}(dS)^2 + \frac{1}{2}\frac{\partial^2 V}{\partial \nu^2}(d\nu)^2 + \frac{\partial^2 V}{\partial S \partial \nu}dSd\nu$$

Here, $(dS)^2 = \nu S^2 dt$, $(d\nu)^2 = \sigma^2\nu dt$, $dSd\nu = \sigma\rho\nu S dt$
Hence

$$dV = \frac{\partial V}{\partial t}dt + \frac{\partial V}{\partial S}dS + \frac{\partial V}{\partial \nu}d\nu + \frac{1}{2}\nu S^2\frac{\partial^2 V}{\partial S^2}dt + \frac{1}{2}\sigma^2\nu\frac{\partial^2 V}{\partial \nu^2}dt + \sigma\rho\nu S\frac{\partial^2 V}{\partial S \partial \nu}dt \tag{3.5}$$

Applying Itô's lemma to the second derivative, $U(S,\nu,t)$, produces an expression identical to (3.5), but in terms of U. Substituting these two expressions into (3.4), the change in portfolio value can be written as:

$$\begin{aligned} d\Pi &= dV + \Delta_1 dS + \Delta_2 dU \\ &= \left[\frac{\partial V}{\partial t} + \frac{1}{2}\nu S^2\frac{\partial^2 V}{\partial S^2} + \frac{1}{2}\sigma^2\nu\frac{\partial^2 V}{\partial \nu^2} + \sigma\rho\nu S\frac{\partial^2 V}{\partial S\partial\nu}\right]dt \\ &+ \Delta_2\left[\frac{\partial U}{\partial t} + \frac{1}{2}\nu S^2\frac{\partial^2 U}{\partial S^2} + \frac{1}{2}\sigma^2\nu\frac{\partial^2 U}{\partial \nu^2} + \sigma\rho\nu S\frac{\partial^2 U}{\partial S\partial\nu}\right]dt \\ &+ \left[\frac{\partial V}{\partial S} + \Delta_2\frac{\partial U}{\partial S} + \Delta_1\right]dS + \left[\frac{\partial V}{\partial \nu} + \Delta_2\frac{\partial U}{\partial \nu}\right]d\nu \end{aligned} \tag{3.6}$$

In order for the portfolio to be hedged against movements in both the stock and volatility, the last two terms in equation (3.6) must be zero. This implies that the hedge parameters must be

$$\Delta_2 = -\frac{\partial V}{\partial \nu}\Big/\frac{\partial U}{\partial \nu}, \qquad \Delta_1 = -\Delta_2\frac{\partial U}{\partial S} - \frac{\partial V}{\partial S}$$

Substitute these values of Δ_2 and Δ_1 into (3.6) to produce

$$\begin{aligned} d\Pi &= \left[\frac{\partial V}{\partial t} + \frac{1}{2}\nu S^2\frac{\partial^2 V}{\partial S^2} + \frac{1}{2}\sigma^2\nu\frac{\partial^2 V}{\partial \nu^2} + \sigma\rho\nu S\frac{\partial^2 V}{\partial S\partial\nu}\right]dt \\ &+ \Delta_2\left[\frac{\partial U}{\partial t} + \frac{1}{2}\nu S^2\frac{\partial^2 U}{\partial S^2} + \frac{1}{2}\sigma^2\nu\frac{\partial^2 U}{\partial \nu^2} + \sigma\rho\nu S\frac{\partial^2 U}{\partial S\partial\nu}\right]dt \end{aligned} \tag{3.7}$$

The condition that the portfolio earn the risk-free rate r, implies that the change in portfolio value is $d\Pi = r\Pi dt$.
Equation (3.4) thus becomes

$$d\Pi = r(V + \Delta_1 S + \Delta_2 U)dt = (rV - rS\Delta_2\frac{\partial U}{\partial S} - rS\frac{\partial V}{\partial S} + \Delta_2 rU)dt \tag{3.8}$$

Joining equations (3.7) and (3.8), drop the dt term, using the fact that $\Delta_2 = -\frac{\partial V}{\partial \nu}\Big/\frac{\partial U}{\partial \nu}$ and re-arrange. This yields

$$\begin{aligned} &\frac{\left[\frac{\partial V}{\partial t} + \frac{1}{2}\nu S^2\frac{\partial^2 V}{\partial S^2} + \frac{1}{2}\sigma^2\nu\frac{\partial^2 V}{\partial \nu^2} + \sigma\rho\nu S\frac{\partial^2 V}{\partial S\partial\nu}\right] - rV + rS\frac{\partial V}{\partial S}}{\frac{\partial V}{\partial \nu}} \\ &= \frac{\left[\frac{\partial U}{\partial t} + \frac{1}{2}\nu S^2\frac{\partial^2 U}{\partial S^2} + \frac{1}{2}\sigma^2\nu\frac{\partial^2 U}{\partial \nu^2} + \sigma\rho\nu S\frac{\partial^2 U}{\partial S\partial\nu}\right] - rU + rS\frac{\partial U}{\partial S}}{\frac{\partial U}{\partial \nu}} \end{aligned} \tag{3.9}$$

The left-hand side of equation (3.9) does not depend on U but is a function of S, ν and t, only. We denote this function by $f : \mathbb{R}^2_+ \times [0,T] \to \mathbb{R}$ and following (Heston, 1993), specify this function as:

$$f(S,\nu,t) = -\kappa(\theta - \nu) + \lambda(S,\nu,t)$$

where the function $\lambda(S, \nu, t)$ is called market price of volatility risk and is not uniquely determined, taking into account that Heston furthermore assumes that the market price of volatility risk is of the form $\lambda(S, \nu, t) = \lambda\nu$ with a constant λ and (Gatheral, 2006) assumes the market price of volatility risk to be zero, so that $\lambda(S, \nu, t) = 0$. That characterise an incomplete market where one cannot replicate derivatives with an portfolio only consisting of the money market account and the underlying. However, if one accepts the prices for plain vanilla call/put options (V) observed in the market and just wants to price other options (U), e.g. barrier options one gets a unique solution by finding an appropriate function λ which matches the prices for V. In practise one simply sets $\lambda(S, \nu, t) = 0$ and calibrates the parameters of the underlying to the observed prices for plain vanilla options, therefore we set $\lambda(S, \nu, t) = 0$ and the left-hand side of equation (3.9) can be written as:

$$\frac{\left[\frac{\partial V}{\partial t} + \frac{1}{2}\nu S^2 \frac{\partial^2 V}{\partial S^2} + \frac{1}{2}\sigma^2 \nu \frac{\partial^2 V}{\partial \nu^2} + \sigma\rho\nu S \frac{\partial^2 V}{\partial S \partial \nu}\right] - rV + rS\frac{\partial V}{\partial S}}{\frac{\partial V}{\partial \nu}}$$

$$= -\kappa(\theta - \nu)$$

Rearrange to produce the Heston PDE expressed in terms of the price S

$$\frac{\partial V}{\partial t} + \frac{1}{2}\nu S^2 \frac{\partial^2 V}{\partial S^2} + \frac{1}{2}\sigma^2 \nu \frac{\partial^2 V}{\partial \nu^2} + \sigma\rho\nu S \frac{\partial^2 V}{\partial S \partial \nu} - rV + rS\frac{\partial V}{\partial S} + \kappa(\theta - \nu)\frac{\partial V}{\partial \nu} = 0 \quad (3.10)$$

The following boundary conditions on the PDE in equation (3.10) hold for a European call option with maturity T and strike K. At maturity, the call is worth its intrinsic value

$$V(S, \nu, T) = max(0, S - K)$$

When the stock price is zero, the call is worthless. As the stock price increases, delta approaches one, and when the volatility increases, the call option becomes equal to the stock price. This implies the following three boundary conditions:

$$V(0, \nu, t) = 0, \quad \frac{\partial V}{\partial S}(\infty, \nu, t) = 1, \quad V(S, \infty, t) = S \quad (3.11)$$

We can define the log price $x = \ln S$ and express the PDE in equation (3.10) in terms of (x, ν, t) instead of (S, ν, t). This simplification requires the following derivatives: By the chain rule:

$$\frac{\partial V}{\partial S} = \frac{\partial V}{\partial x}\frac{1}{S}, \quad \frac{\partial^2 V}{\partial S \partial \nu} = \frac{\partial}{\partial \nu}\left(\frac{1}{S}\frac{\partial V}{\partial x}\right) = \frac{1}{S}\frac{\partial^2 V}{\partial \nu \partial x}$$

Using the product rule,

$$\frac{\partial^2 V}{\partial S^2} = \frac{\partial}{\partial S}\left(\frac{1}{S}\frac{\partial V}{\partial x}\right) = -\frac{1}{S^2}\frac{\partial V}{\partial x} + \frac{1}{S}\frac{\partial^2 V}{\partial S \partial x} = -\frac{1}{S^2}\frac{\partial V}{\partial x} + \frac{1}{S^2}\frac{\partial^2 V}{\partial x^2}$$

Substituting these expressions into the Heston PDE in (3.10) all the S terms cancel, and we obtain the Heston PDE in terms of the log price $x = \ln S$.

$$\frac{\partial V}{\partial t} + \frac{1}{2}\nu \frac{\partial^2 V}{\partial x^2} + \left(r - \frac{1}{2}\nu\right)\frac{\partial V}{\partial x} + \frac{1}{2}\sigma^2 \nu \frac{\partial^2 V}{\partial \nu^2} + \sigma\rho\nu \frac{\partial^2 V}{\partial \nu \partial x} - rV + \kappa(\theta - \nu)\frac{\partial V}{\partial \nu} = 0 \quad (3.12)$$

3.3 The European Call Price

In this section, we show that the call price in the Heston model can be expressed in a manner which resembles the call price in the Black-Scholes model. The time - t price of a European call on a non-dividend paying stock with spot price S_t, when the strike is K and the time to maturity is $\tau = T - t$, is the discounted expected value of the payoff under the risk neutral measure.

$$\begin{aligned} C(K) &= e^{-r\tau}\mathbb{E}^{\mathbb{Q}}\left[(S_T - K)^+\right] \\ &= e^{-r\tau}\mathbb{E}^{\mathbb{Q}}\left[(S_T - K)\cdot \mathbb{1}_{\{S_T>K\}}\right] \\ &= e^{-r\tau}\mathbb{E}^{\mathbb{Q}}\left[S_T\cdot \mathbb{1}_{\{S_T>K\}}\right] - Ke^{-r\tau}\mathbb{E}^{\mathbb{Q}}\left[\mathbb{1}_{\{S_T>K\}}\right] \end{aligned} \tag{3.13}$$

where $\mathbb{1}$ is the indicator function and $\mathbb{Q}$ is the risk neutral measure.
To obtain the expression of the first term in the last line of equation (3.13) we need to change the original measure $\mathbb{Q}$ to another measure $\mathbb{Q}^S$. Consider the Radon-Nikodym derivative

$$\frac{d\mathbb{Q}}{d\mathbb{Q}^S} = \frac{B_T/B_t}{S_T/S_t} = \frac{\mathbb{E}^{\mathbb{Q}}[e^{x_T}]}{e^{x_T}} \tag{3.14}$$

where B_t is the money market and satisfies $B_t = \exp\left(\int_0^t r du\right) = e^{rt}$.
In (3.14), we have written $S_t e^{r(T-t)} = S_t e^{r\tau} = \mathbb{E}^{\mathbb{Q}}[e^{x_T}]$, since under $\mathbb{Q}$ assets grow at the risk-free rate, r. The first expectation in the third line of (3.13) can therefore be written as

$$\begin{aligned} e^{-r\tau}\cdot\mathbb{E}^{\mathbb{Q}}[S_T\cdot\mathbb{1}_{\{S_T>K\}}] &= S_t\mathbb{E}^{\mathbb{Q}}\left[\frac{S_T/S_t}{B_T/B_t}\cdot\mathbb{1}_{\{S_T>K\}}\right] \\ &= S_t\mathbb{E}^{\mathbb{Q}^s}\left[\frac{S_T/S_t}{B_T/B_t}\cdot\mathbb{1}_{\{S_T>K\}}\cdot\frac{d\mathbb{Q}}{d\mathbb{Q}^{\mathbb{S}}}\right] \\ &= S_t\mathbb{E}^{\mathbb{Q}^s}[\mathbb{1}_{\{S_T>K\}}] \\ &= S_t\mathbb{Q}^s(S_T>K) \end{aligned}$$

The second expectation in the third line of (3.13) can be written as

$$\mathbb{E}^{\mathbb{Q}}[\mathbb{1}_{\{S_T>K\}}] = \mathbb{Q}(S_T>K)$$

This implies that the European call price of equation (3.13) can be written in terms of both measures as

$$\begin{aligned} C(K) &= S_t\mathbb{Q}^S(S_T>K) - Ke^{-r\tau}\mathbb{Q}(S_T>K) \\ &= S_tP_1 - Ke^{-r\tau}P_2 \end{aligned} \tag{3.15}$$

We have denoted $P_1 = \mathbb{Q}^S(S_T>K)$ and $P_2 = \mathbb{Q}(S_T>K)$ where the measure $\mathbb{Q}$ uses the bond B_t as the numeraire, while the measure $\mathbb{Q}^S$ uses the stock price S_t. The last line in equation (3.15) is the "Black-Scholes-like" call price formula, with P_1 replacing $\phi(d_1)$ and P_2 replacing $\phi(d_2)$ in the Black-Scholes call price.
The quantities P_1 and P_2 each represent the probability of the call expiring in-the-money, conditional on the value $S_t = e^{x_t}$ of the stock and on the value ν_t of the volatility at time t.

3.3.1 The PDE for probabilities in equation for European Call Price

Using $x = x_t = \ln S_t$

$$C(K) = e^x P_1 - Ke^{-r\tau} P_2 \tag{3.16}$$

Equation (3.16) expresses $C(K)$ in terms of the in-the-money probabilities $P_1 = \mathbb{Q}^S(S_T > K)$ and $P_2 = \mathbb{Q}(S_T > K)$. Since the European Call satisfies the PDE 3.12 we can find the required derivatives of equation (3.16), substitute them into the PDE, and express the PDE in terms of P_1 and P_2. The derivative of $C(K)$ with respect to t is:

$$\frac{\partial C}{\partial t} = e^x \frac{\partial P_1}{\partial t} - Ke^{-r\tau}\left[rP_2 + \frac{\partial P_2}{\partial t}\right] \tag{3.17}$$

with respect to x:

$$\frac{\partial C}{\partial x} = e^x \left[P_1 + \frac{\partial P_1}{\partial x}\right] - Ke^{-r\tau}\frac{\partial P_2}{\partial x} \tag{3.18}$$

The second derivative of $C(K)$ with respect to x:

$$\begin{aligned}\frac{\partial^2 C}{\partial x^2} &= e^x \left[P_1 + \frac{\partial P_1}{\partial x}\right] + e^x \left[\frac{\partial P_1}{\partial x} + \frac{\partial^2 P_1}{\partial x^2}\right] - Ke^{-r\tau}\frac{\partial^2 P_2}{\partial x^2} \\ &= e^x \left[P_1 + 2\frac{\partial P_1}{\partial x} + \frac{\partial^2 P_1}{\partial x^2}\right] - Ke^{-r\tau}\frac{\partial^2 P_2}{\partial x^2}\end{aligned} \tag{3.19}$$

with respect to ν:

$$\frac{\partial C}{\partial \nu} = e^x \frac{\partial P_1}{\partial \nu} - Ke^{-r\tau}\frac{\partial P_2}{\partial \nu} \tag{3.20}$$

The second derivative of $C(K)$ with respect to ν:

$$\frac{\partial^2 C}{\partial \nu^2} = e^x \frac{\partial^2 P_1}{\partial \nu^2} - Ke^{-r\tau}\frac{\partial^2 P_2}{\partial \nu^2} \tag{3.21}$$

with respect to ν and x:

$$\frac{\partial^2 C}{\partial x \partial \nu} = e^x \left[\frac{\partial P_1}{\partial \nu} + \frac{\partial^2 P_1}{\partial x \partial \nu}\right] - Ke^{-r\tau}\frac{\partial^2 P_2}{\partial x \partial \nu} \tag{3.22}$$

As mentioned earlier, since the European call $C(K)$ is a financial derivative, it also satisfies the Heston PDE in (3.12).
To obtain the Heston PDE for P_1 and P_2, Heston (1993) argues that the PDE in (3.12) holds for any contractual features of $C(K)$, in particular, for any strike price $K \geq 0$, for any value of $S \geq 0$, and for any value $r \geq 0$ of the risk-free rate. Setting $K = 0$ and $S = 1$ in the call price in equation (3.15) produces an option whose price is simply P_1. This option will also follow the PDE in (3.12). Similarly, setting $S = 0$, $K = 1$, and $r = 0$ in (3.15) produces an option whose price is $-P_2$. Since $-P_2$ follows the PDE, so does P_2.

Substituting terms in equation (3.17) to (3.22) in equation (3.12) we obtain:

$$
\begin{aligned}
&e^x\frac{\partial P_1}{\partial t} - Ke^{-r\tau}\left(rP_2 + \frac{\partial P_2}{\partial t}\right) + \frac{1}{2}\nu\left[e^x\left(P_1 + 2\frac{\partial P_1}{\partial x} + \frac{\partial^2 P_1}{\partial x^2}\right) - Ke^{-r\tau}\frac{\partial^2 P_2}{\partial x^2}\right] \\
&+ \left(r - \frac{1}{2}\nu\right)\left[e^x\left(P_1 + \frac{\partial P_1}{\partial x}\right) - Ke^{-r\tau}\frac{\partial P_2}{\partial x}\right] + \frac{1}{2}\sigma^2\nu\left(e^x\frac{\partial^2 P_1}{\partial \nu^2} - Ke^{-r\tau}\frac{\partial^2 P_2}{\partial \nu^2}\right) \\
&+ \sigma\rho\nu\left[e^x\left(\frac{\partial P_1}{\partial \nu} + \frac{\partial^2 P_1}{\partial x \partial \nu}\right) - Ke^{-r\tau}\frac{\partial^2 P_2}{\partial x \partial \nu}\right] - r\left(e^x P_1 - Ke^{-r\tau}P_2\right) \\
&+ \kappa(\theta - \nu)\left(e^x\frac{\partial P_1}{\partial \nu} - Ke^{-r\tau}\frac{\partial P_2}{\partial \nu}\right) = 0
\end{aligned}
$$

The brown terms do not depend on P_2 but they are a function of x, ν and t, only; and the other terms depend only on P_2. The last equality is satisfied if each equation of P_1 and P_2 is equal to zero, separately. Regroup common terms to P_1, cancel e^x to obtain:

$$
\begin{aligned}
&\frac{\partial P_1}{\partial t} + \frac{1}{2}\nu\left(P_1 + 2\frac{\partial P_1}{\partial x} + \frac{\partial^2 P_1}{\partial x^2}\right) + \left(r - \frac{1}{2}\nu\right)\left(P_1 + \frac{\partial P_1}{\partial x}\right) + \frac{1}{2}\sigma^2\nu\frac{\partial^2 P_1}{\partial \nu^2} \\
&+ \sigma\rho\nu\left(\frac{\partial P_1}{\partial \nu} + \frac{\partial^2 P_1}{\partial x \partial \nu}\right) - rP_1 + \kappa(\theta - \nu)\frac{\partial P_1}{\partial \nu} = 0
\end{aligned} \tag{3.23}
$$

Simplifying (3.23) becomes

$$
\frac{\partial P_1}{\partial t} + \left(r + \frac{1}{2}\nu\right)\frac{\partial P_1}{\partial x} + \frac{1}{2}\nu\frac{\partial^2 P_1}{\partial x^2} + \sigma\rho\nu\frac{\partial^2 P_1}{\partial x \partial \nu} + \left[\sigma\rho\nu + \kappa(\theta - \nu)\right]\frac{\partial P_1}{\partial \nu} + \frac{1}{2}\sigma^2\nu\frac{\partial^2 P_1}{\partial \nu^2} = 0 \tag{3.24}
$$

Similarly, regroup terms common to P_2, cancel $-Ke^{-r\tau}$:

$$
\begin{aligned}
&rP_2 + \frac{\partial P_2}{\partial t} + \frac{1}{2}\nu\frac{\partial^2 P_2}{\partial x^2} + \left(r - \frac{1}{2}\nu\right)\frac{\partial P_2}{\partial x} + \frac{1}{2}\sigma^2\nu\frac{\partial^2 P_2}{\partial \nu^2} + \sigma\rho\nu\frac{\partial^2 P_2}{\partial x \partial \nu} \\
&- rP_2 + \kappa(\theta - \nu)\frac{\partial P_2}{\partial \nu} = 0
\end{aligned} \tag{3.25}
$$

Simplifying (3.25) becomes

$$
\frac{\partial P_2}{\partial t} + \left(r - \frac{1}{2}\nu\right)\frac{\partial P_2}{\partial x} + \frac{1}{2}\nu\frac{\partial^2 P_2}{\partial x^2} + \sigma\rho\nu\frac{\partial^2 P_2}{\partial x \partial \nu} + \kappa(\theta - \nu)\frac{\partial P_2}{\partial \nu} + \frac{1}{2}\sigma^2\nu\frac{\partial^2 P_2}{\partial \nu^2} = 0 \tag{3.26}
$$

For notational convenience, combine equations (3.24) and (3.26) into a single expression

$$
\frac{\partial P_j}{\partial t} + (r + u_j\nu)\frac{\partial P_j}{\partial x} + \frac{1}{2}\nu\frac{\partial^2 P_j}{\partial x^2} + \sigma\rho\nu\frac{\partial^2 P_j}{\partial x \partial \nu} + (a - b_j\nu)\frac{\partial P_j}{\partial \nu} + \frac{1}{2}\sigma^2\nu\frac{\partial^2 P_j}{\partial \nu^2} = 0 \tag{3.27}
$$

for $j = 1, 2$ and where $u_1 = \frac{1}{2}$, $u_2 = -\frac{1}{2}$, $a = \kappa\theta$, $b_1 = \kappa - \sigma\rho$, and $b_2 = \kappa$.
With x_τ defined as $x_\tau = \ln S_\tau$, the PDE for P_j equation (3.27) becomes

$$
-\frac{\partial P_j}{\partial \tau} + (r + u_j\nu)\frac{\partial P_j}{\partial x} + \frac{1}{2}\nu\frac{\partial^2 P_j}{\partial x^2} + \sigma\rho\nu\frac{\partial^2 P_j}{\partial x \partial \nu} + (a - b_j\nu)\frac{\partial P_j}{\partial \nu} + \frac{1}{2}\sigma^2\nu\frac{\partial^2 P_j}{\partial \nu^2} = 0 \tag{3.28}
$$

The transformation of the PDE from t to the time to maturity $\tau = T - t$ explains the minus sign in front of the maturity derivative in equation (3.28).
Consider the Fourier transform $\hat{P}_j$ of the probabilities $P_j = P_j(x, \nu, \tau)$

$$\hat{P}_j(m, \nu, \tau) = \int_{-\infty}^{\infty} e^{-imx} P_j(x, \nu, \tau) dx$$

Remembering differentiation in subsection 2.4.1 (Fourier transform of derivative) $\hat{P}_j$ with respect to x corresponds to multiplication by im, the PDE in equation (3.28) for $\hat{P}_j$ is

$$-\frac{\partial \hat{P}_j}{\partial \tau} + (r + u_j \nu) im \hat{P}_j - \frac{1}{2}\nu m^2 \hat{P}_j + \rho\sigma\nu im \frac{\partial \hat{P}_j}{\partial \nu} + (a - b_j\nu)\frac{\partial \hat{P}_j}{\partial \nu} + \frac{1}{2}\sigma^2 \nu \frac{\partial^2 \hat{P}_j}{\partial \nu^2} = 0 \quad (3.29)$$

Heston (1993) postulates that the characteristic functions (P_j) for the logarithm of the terminal stock price, $x_T = \ln S_T$ have the log linear form:

$$\hat{P}_j(m; x, \nu, \tau) = E[e^{imx_T}] = \exp\left[A_j(m, \tau) + B_j(m, \tau)\nu + imx_t\right] \quad (3.30)$$

where $i = \sqrt{-1}$ is the imaginary unit, A_j and B_j are coefficients.
The following derivatives are required to evaluate equation (3.29):

$$\frac{\partial \hat{P}_j}{\partial \tau} = \left[\frac{\partial A_j}{\partial \tau} + \nu \frac{\partial B_j}{\partial \tau}\right] \hat{P}_j$$
$$\frac{\partial \hat{P}_j}{\partial \nu} = B_j \hat{P}_j$$
$$\frac{\partial^2 \hat{P}_j}{\partial \nu^2} = B_j \frac{\partial \hat{P}_j}{\partial \nu} = B_j^2 \hat{P}_j$$

Substitute these derivatives into (3.29) and drop the $\hat{P}_j$ terms produces

$$-\frac{\partial A_j}{\partial \tau} - \nu \frac{\partial B_j}{\partial \tau} + (r + u_j\nu) im - \frac{1}{2}\nu m^2 + \rho\sigma\nu im B_j + (a - b_j\nu)B_j + \frac{1}{2}\sigma^2 \nu B_j^2 = 0$$

or equivalently

$$\nu\left(-\frac{\partial B_j}{\partial \tau} + \frac{1}{2}\sigma^2 B_j^2 + \rho\sigma im B_j - b_j B_j + imu_j - \frac{1}{2}m^2\right) - \frac{\partial A_j}{\partial \tau} + rim + aB_j = 0$$

Since $\nu_t > 0$, this last equality is satisfied if:

$$\frac{\partial B_j}{\partial \tau} - \frac{1}{2}\sigma^2 B_j^2 + (b_j - \rho\sigma im)B_j - imu_j + \frac{1}{2}m^2 = 0 \quad (3.31)$$
$$\frac{\partial A_j}{\partial \tau} - aB_j - rim = 0 \quad (3.32)$$

write, $\alpha_j = imu_j - \frac{1}{2}m^2$, $\beta_j = -b_j + \rho\sigma im$, $\gamma = \frac{\sigma^2}{2}$ and substituting $\delta = a$, $\epsilon = rim$ in (3.31) and (3.32) obtains:

$$\frac{\partial B_j}{\partial \tau} - \gamma B_j^2 - \beta_j B_j - \alpha_j = 0 \quad (3.33)$$
$$\frac{\partial A_j}{\partial \tau} = \delta B_j + \epsilon \quad (3.34)$$

$\Rightarrow$

$$\begin{aligned}
\frac{\partial B_j}{\partial \tau} &= \gamma B_j^2 + \beta_j B_j + \alpha_j \\
&= \frac{(\gamma B_j)^2 + \beta_j(\gamma B_j) + \alpha_j \gamma}{\gamma} \\
&= \frac{(\gamma B_j - \pi_+)(\gamma B_j - \pi_-)}{\gamma} \\
&= \frac{\gamma(B_j - \pi_+/\gamma) \cdot \gamma(B_j - \pi_-/\gamma)}{\gamma} \\
&= \gamma(B_j - \pi_+/\gamma)(B_j - \pi_-/\gamma)
\end{aligned}$$

$\pi_\pm = \frac{-\beta_j \pm \sqrt{\beta_j^2 - 4\alpha_j \gamma}}{2}$ is the quadratic formula

Solving the differential equation for B_j

$$\int \frac{dB_j}{(B_j - \pi_+/\gamma)(B_j - \pi_-/\gamma)} = \int \gamma d\tau \tag{3.35}$$

By partial fractions

$$\frac{1}{(B_j - \pi_+/\gamma)(B_j - \pi_-/\gamma)} = \frac{D}{B_j - \pi_+/\gamma} + \frac{E}{B_j - \pi_-/\gamma}$$

D, E constants.
Then $D(B_j - \pi_-/\gamma) + E(B_j - \pi_+/\gamma) = 1$ implies that:

$$\begin{aligned}
&(D+E)B_j = 0 \\
&-D\pi_-/\gamma - E\pi_+/\gamma = 1
\end{aligned}$$

We, obtain

$$D = \frac{-1 - E\pi_+/\gamma}{\pi_-/\gamma}, \qquad E = \frac{1 + E\pi_+/\gamma}{\pi_-/\gamma}$$

Then

$$\begin{aligned}
E(\pi_-/\gamma - \pi_+/\gamma) &= 1 & D &= -E \\
&= \frac{-1}{\pi_+/\gamma - \pi_-/\gamma}, & &= \frac{1}{\pi_+/\gamma - \pi_-/\gamma}
\end{aligned}$$

Substituting D and E into the left side of the equation (3.35) obtains:

$$\begin{aligned}
&\int \frac{dB_j}{(B_j - \pi_+/\gamma)(B_j - \pi_-/\gamma)} \\
&= \frac{1}{\pi_+/\gamma - \pi_-/\gamma} \int \frac{dB_j}{B_j - \pi_+/\gamma} - \frac{1}{\pi_+/\gamma - \pi_-/\gamma} \int \frac{dB_j}{B_j - \pi_-/\gamma} \\
&= \frac{1}{\pi_+/\gamma - \pi_-/\gamma} \left[\ln(B_j - \pi_+/\gamma) - \ln(B_j - \pi_-/\gamma)\right]
\end{aligned}$$

Therefore

$$\frac{1}{\pi_+/\gamma - \pi_-/\gamma} \ln\left(\frac{B_j - \pi_+/\gamma}{B_j - \pi_-/\gamma}\right) = \gamma\tau + C, \quad C \text{ constant}$$

At maturity ($\tau = 0$), the value of $x_T = \ln S_T$ is know, so the expectation in equation 3.30 will disappear, and consequently the right hand side will reduce to simply $\exp(imx_T)$. This implies that the initial conditions at maturity are $A_j(m,0) = 0$ and $B_j(m,0) = 0$.

So taking $\tau = 0$, obtains $C = \frac{1}{\pi_+/\gamma - \pi_-/\gamma} \ln\left(\frac{\pi_+}{\pi_-}\right)$

So

$$\ln\left(\frac{B_j - \pi_+/\gamma}{B_j - \pi_-/\gamma}\right) = \gamma\tau(\pi_+/\gamma - \pi_-/\gamma) + \ln\left(\frac{\pi_+}{\pi_-}\right)$$

Then

$$\begin{aligned}
\frac{B_j - \pi_+/\gamma}{B_j - \pi_-/\gamma} &= \frac{\pi_+}{\pi_-} \cdot e^{\tau(\pi_+ - \pi_-)} \\
B_j - \pi_+/\gamma &= \frac{\pi_+}{\pi_-}(B_j - \pi_-/\gamma) \cdot e^{\tau(\pi_+ - \pi_-)} \\
B_j \left[1 - \frac{\pi_+}{\pi_-} \cdot e^{\tau(\pi_+ - \pi_-)}\right] &= -\frac{\pi_+}{\gamma} e^{\tau(\pi_+ - \pi_-)} + \frac{\pi_+}{\gamma}
\end{aligned}$$

Thus

$$B_j = \frac{\frac{\pi_+}{\gamma}\left(1 - e^{\tau(\pi_+ - \pi_-)}\right)}{1 - \frac{\pi_+}{\pi_-} \cdot e^{\tau(\pi_+ - \pi_-)}}$$

Write $\tilde{\pi} = \frac{\pi_+}{\pi_-}$ and $\eta = (\pi_+ - \pi_-) = \sqrt{\beta_j^2 - 4\alpha_j\gamma}$, obtains

$$\boxed{B_j(m,\tau) = \frac{(\pi_+/\gamma)(1 - e^{\eta\tau})}{1 - \tilde{\pi}e^{\eta\tau}}} \tag{3.36}$$

Now solving for $A_j(m,\tau)$, $B_j(m,\tau)$ can be written as:

$$\begin{aligned}
B_j(m,\tau) &= (\pi_+/\gamma)\left(\frac{1}{1 - \tilde{\pi}e^{\eta\tau}} - \frac{e^{\eta\tau}}{1 - \tilde{\pi}e^{\eta\tau}}\right) \\
&= (\pi_+/\gamma)\left(\frac{e^{-\eta\tau}}{e^{-\eta\tau} - \tilde{\pi}} - \frac{e^{\eta\tau}}{1 - \tilde{\pi}e^{\eta\tau}}\right)
\end{aligned}$$

Multiplying the fraction on the left by $e^{-\eta\tau}$

Given that $\frac{\partial A_j}{\partial \tau} = \delta B_j + \epsilon$, integrating we obtains:

$$\begin{aligned}
\int dA_j &= \delta \int B_j(m,\tau)d\tau + \int \epsilon d\tau \\
&= \delta(\pi_+/\gamma)\left(\int \frac{e^{-\eta\tau}}{e^{-\eta\tau} - \tilde{\pi}} d\tau - \int \frac{e^{\eta\tau}}{1 - \tilde{\pi}e^{\eta\tau}} d\tau\right) + \int \epsilon d\tau
\end{aligned}$$

Taking the substitution

$$u = e^{-\eta\tau} - \tilde{\pi} \;\; du = -\eta e^{-\eta\tau} d\tau, \;\; v = 1 - \tilde{\pi}e^{\eta\tau} \;\; dv = -\tilde{\pi}\eta e^{\eta\tau} d\tau$$

$\Rightarrow$

$$A_j = -\frac{\delta(\pi_+/\gamma)}{\eta}\left[\ln(e^{-\eta\tau}-\tilde{\pi}) - \frac{1}{\tilde{\pi}}\ln(1-\tilde{\pi}e^{\eta\tau})\right] + \epsilon\tau + C, \;\; C \text{ constant.}$$

When $\tau = 0$, by boundary condition $A_j(m,0) = 0$.

$$0 = -\frac{\delta(\pi_+/\gamma)}{\eta}\left[\ln(1-\tilde{\pi}) - \frac{1}{\tilde{\pi}}\ln(1-\tilde{\pi})\right] + C$$
$$0 = -\frac{\delta(\pi_+/\gamma)}{\eta}\cdot\ln(1-\tilde{\pi})\cdot\left(1-\frac{1}{\tilde{\pi}}\right) + C$$

Then

$$C = \frac{\delta(\pi_+/\gamma)}{\eta}\left(\frac{\tilde{\pi}-1}{\tilde{\pi}}\right)\ln(1-\tilde{\pi})$$

And therefore

$$\begin{aligned}
A_j(m,\tau) &= -\frac{\delta(\pi_+/\gamma)}{\eta}\left[\ln(e^{-\eta\tau}-\tilde{\pi}) - \frac{1}{\tilde{\pi}}\ln(1-\tilde{\pi}e^{\eta\tau}) - \left(\frac{\tilde{\pi}-1}{\tilde{\pi}}\right)\ln(1-\tilde{\pi})\right] + \epsilon\tau \\
&= -\frac{\delta(\pi_+/\gamma)}{\eta}\left[\ln(e^{-\eta\tau}-\tilde{\pi}) - \ln(1-\tilde{\pi}) - \frac{1}{\tilde{\pi}}\left\{\ln(1-\tilde{\pi}e^{\eta\tau}) - \ln(1-\tilde{\pi})\right\}\right] + \epsilon\tau \\
&= -\frac{\delta(\pi_+/\gamma)}{\eta}\left[\ln\left(\frac{e^{-\eta\tau}-\tilde{\pi}}{1-\tilde{\pi}}\right) - \frac{1}{\tilde{\pi}}\ln\left(\frac{1-\tilde{\pi}e^{\eta\tau}}{1-\tilde{\pi}}\right)\right] + \epsilon\tau \\
&= -\frac{\delta(\pi_+/\gamma)}{\eta}\left[\ln\left(\frac{e^{-\eta\tau}-\tilde{\pi}}{1-\tilde{\pi}}\right) + \ln(e^{\eta\tau}) - \frac{1}{\tilde{\pi}}\ln\left(\frac{1-\tilde{\pi}e^{\eta\tau}}{1-\tilde{\pi}}\right) - \ln(e^{\eta\tau})\right] + \epsilon\tau \\
&= -\frac{\delta(\pi_+/\gamma)}{\eta}\left[\ln\left(\frac{1-\tilde{\pi}e^{\eta\tau}}{1-\tilde{\pi}}\right) - \frac{1}{\tilde{\pi}}\ln\left(\frac{1-\tilde{\pi}e^{\eta\tau}}{1-\tilde{\pi}}\right) - \eta\tau\right] + \epsilon\tau \\
&= -\frac{\delta(\pi_+/\gamma)}{\eta}\left[\left(\frac{\tilde{\pi}-1}{\tilde{\pi}}\right)\cdot\ln\left(\frac{1-\tilde{\pi}e^{\eta\tau}}{1-\tilde{\pi}}\right) - \eta\tau\right] + \epsilon\tau
\end{aligned}$$

Thus

$$\boxed{A_j(m,\tau) = -\frac{\delta(\pi_+/\gamma)}{\eta}\left[\left(\frac{\tilde{\pi}-1}{\tilde{\pi}}\right)\cdot\ln\left(\frac{1-\tilde{\pi}e^{\eta\tau}}{1-\tilde{\pi}}\right) - \eta\tau\right] + \epsilon\tau} \tag{3.37}$$

Therefore

$$\hat{P}_j(m;x,\nu,\tau) = \exp\left[A_j(m,\tau) + B_j(m,\tau)\nu + imx_t\right] \tag{3.38}$$

with $A_j(m,\tau)$ and $B_j(m,\tau)$ are given in equations (3.37) and (3.36), respectively. We use the functions P_1 and P_2 to implement the model in Python, heston_char_function returns the characteristic functions.

The following code snippet shows the implementation in Python:

```
def heston_char_function(m, x, t_m, r, kappa, theta, sigma,
                         rho, v0):
    """ Valuation of European call option in Heston model
        Fourier-based approach: characteristic function. """
    u1, u2 = 0.5, -0.5
    a = kappa * theta
    b1, b2 = kappa - sigma * rho, kappa
    alpha1 = 1j * m * u1 - 0.5 * m ** 2
    alpha2 = 1j * m * u2 - 0.5 * m ** 2
    beta1 = -b1 + rho * sigma * 1j * m
    beta2 = -b2 + rho * sigma * 1j * m
    gamma = 0.5 * sigma ** 2
    delta, epsilon = a, r * 1j * m
    pi_plus1 = 0.5*(-beta1 + (b1 ** 2 - 4 * alpha1 * gamma) **
        0.5)
    pi_plus2 = 0.5*(-beta2 + (b2 ** 2 - 4 * alpha2 * gamma) **
        0.5)
    pi_minus1 = 0.5*(-beta1 - (b1 ** 2 - 4 * alpha1 * gamma) **
        0.5)
    pi_minus2 = 0.5*(-beta2 - (b2 ** 2 - 4 * alpha2 * gamma) **
        0.5)
    pi_hat_1, pi_hat_2 = pi_plus1 / pi_minus1, pi_plus2 /
        pi_minus2
    eta1, eta2 = pi_plus1 - pi_minus1, pi_plus2 - pi_minus2

    B1 = (pi_plus1 * (np.exp(-eta1 * t_m) - 1)) / (gamma * (np.
        exp(-eta1 * t_m) - pi_hat_1))
    B2 = (pi_plus2 * (np.exp(-eta2 * t_m) - 1)) / (gamma * (np.
        exp(-eta2 * t_m) - pi_hat_2))

    A1 = -(delta * pi_plus1) / (eta1 * gamma) * \
         (((pi_hat_1 - 1) / pi_hat_1) * (cmath.log((np.exp(-eta1
             * t_m) - pi_hat_1) / (1 - pi_hat_1)) + eta1 * t_m)
          - eta1 * t_m) + epsilon * t_m
    A2 = -(delta * pi_plus2) / \
         (eta2 * gamma) * (((pi_hat_2 - 1) / pi_hat_2) * (cmath.
            log((np.exp(-eta2 * t_m) - pi_hat_2) / (1 - pi_hat_2
            )) + eta2 * t_m) - eta2 * t_m) + epsilon*t_m

    char_func_value1 = np.exp(A1 + B1 * v0 + 1j * m * x)
    char_func_value2 = np.exp(A2 + B2 * v0 + 1j * m * x)

    return char_func_value1, char_func_value2
```

Given that the characteristic functions $\hat{P}_j(m; x, \nu, \tau)$ are known, each in the money probability P_j can be recovered from the characteristic function using inversion theorem, as

$$P_j = \Pr(\ln S_T > \ln K) = \frac{1}{2} + \frac{1}{\pi} \int_0^\infty \mathrm{Re}\left[\frac{e^{-im \ln K} \hat{P}_j(m; x, \nu, \tau)}{im}\right] dm \tag{3.39}$$

Writing $k = \ln K$, suppressing the j index and denote $\phi(m)$ to be the characteristic

function for $\ln S_T$ evaluated at m. Equation (3.39) can be written as:

$$\Pr(\ln S_T > k) = \frac{1}{2} + \frac{1}{\pi}\int_0^\infty \text{Re}\left[\frac{e^{-imk}\phi(m)}{im}\right] dm \tag{3.40}$$

To verify this equation, the probabilities expressed in the form of equation (3.40) were derived by Gil-Pelaez (1951) using the inversion theorem for Fourier transforms. The sign function sign(α) plays in an important role in this derivation. It is defined as

$$\text{sign } \alpha = \begin{cases} \alpha/|\alpha| & \alpha \neq 0 \\ 0 & \alpha = 0 \end{cases} \tag{3.41}$$

The sign function has the integral representation

$$\text{sign } \alpha = \frac{1}{\pi}\int_{-\infty}^{\infty} \frac{\sin \alpha x}{x} dx$$

Denote $f(x)$ to be the density of $\ln S_T$, and $F(x)$ to be its distribution, and using the definition of the sign function in (3.41), we can write for a fixed y

$$\begin{aligned}\int_{-\infty}^{\infty} \text{sign}(x-y)f(x)dx &= \int_y^\infty \text{sign}(x-y)f(x)dx - \int_{-\infty}^{y} \text{sign}(y-x)f(x)dx \\ &= [1-F(y)] - F(y) = 1 - 2F(y)\end{aligned} \tag{3.42}$$

To evaluate (3.42), we have broken up the integration range at y and we have exploited the sign of $x-y$ over each region.
To begin the derivation of equation (3.40), note that f(x) can be recovered from $\phi(m)$ as in sub-section 2.4 (Fourier inversion formula) as:

$$f(x) = \frac{1}{2\pi}\int_{-\infty}^{\infty} e^{-imx}\phi(m)dm. \tag{3.43}$$

We can express $\Pr(\ln S_T > k)$ using the density, and then substitute equation (3.43) to obtain:

$$\begin{aligned}\Pr(\ln S_T > k) &= \int_k^\infty f(x)dx = \frac{1}{2\pi}\int_k^\infty \left(\int_{-\infty}^{\infty} e^{-imx}\phi(m)dm\right) dx \\ &= \frac{1}{2\pi}\int_{-\infty}^{\infty} \phi(m)\left(\int_k^\infty e^{-imx}dx\right) dm.\end{aligned} \tag{3.44}$$

The last equality is obtained by reversing the order of integration, as a consequence from Fubini's theorem. Now evaluate the inner integral in the second line of (3.44), which results in

$$\int_k^\infty e^{-imx}dx = -\frac{1}{im}e^{-imx}\Big|_k^\infty = -\frac{1}{im}\lim_{R\to\infty} e^{-imR} + \frac{1}{im}e^{-imk}$$

Then

$$\Pr(\ln S_T > k) = \frac{1}{2\pi}\int_{-\infty}^{\infty} \phi(m)\frac{e^{-imk}}{im}dm - \frac{1}{2\pi}\lim_{R\to\infty}\int_{-\infty}^{\infty} \phi(m)\frac{e^{-imR}}{im}dm \tag{3.45}$$

In the second integrand of (3.45), express $\phi(m)$ as a Fourier transform and apply the results developed earlier in this section. This produces

$$\begin{aligned}
&\frac{1}{2\pi}\lim_{R\to\infty}\int_{-\infty}^{\infty}\left(\int_{-\infty}^{\infty}e^{imx}f(x)dx\right)\frac{e^{-imR}}{im}dm \\
&\quad=\frac{1}{2\pi}\lim_{R\to\infty}\int_{-\infty}^{\infty}f(x)\left(\int_{-\infty}^{\infty}\frac{e^{im(x-R)}}{im}dm\right)dx \qquad (3.46)\\
&\quad=\frac{1}{2\pi}\lim_{R\to\infty}\int_{-\infty}^{\infty}\pi\text{sign}(x-R)f(x)dx \\
&\quad=\frac{1}{2}\lim_{R\to\infty}(1-2F(R))=-\frac{1}{2}
\end{aligned}$$

By equation 3.42

To obtain the second line in equation (3.46), we have applied (3.42), and we have used the fact the inner integrand can be written using Euler's identity in (3.47)

$$\frac{e^{im(x-R)}}{im}=\frac{1}{i}\frac{\cos(m(x-R))}{m}+\frac{\sin(m(x-R))}{m} \qquad (3.47)$$

The first term is an odd function in m, so it will disappear when integrated over $(-\infty,\infty)$, while the second term will integrate to $\pi\text{sign}(x-R)$.
Substituting the result of (3.46) into equation (3.45) produces

$$\Pr(\ln S_T>k)=\frac{1}{2}+\frac{1}{2\pi}\int_{-\infty}^{\infty}\phi(m)\frac{e^{-imk}}{im}dm. \qquad (3.48)$$

By applying Euler's identity to both $\phi(m)$ and e^{-imk}, we can see that the integrand in (3.48) is odd in its imaginary part and even in its real part. Hence, we can use the real part only, restrict the integration range to $(0,\infty)$ and multiply the result by 2, and we obtain the desired expression for the probability in equation (3.39).

3.3.2 Consolidating the integrals

It is possible to regroup the integrals for the probabilities P_1 and P_2 into a single integral, which will speed up the numerical integration required in the call price calculation. Substituting the expression for P_j from equation (3.39) into the call price (3.16) and rearranging produces:

$$\begin{aligned}
C(K)&=e^xP_1-Ke^{-r\tau}P_2 \\
&=e^x\left[\frac{1}{2}+\frac{1}{\pi}\int_0^{\infty}\text{Re}\left(\frac{e^{-im\ln K}\hat{P}_1(m;x,\nu,\tau)}{im}\right)dm\right] \\
&-Ke^{-r\tau}\left[\frac{1}{2}+\frac{1}{\pi}\int_0^{\infty}\text{Re}\left(\frac{e^{-im\ln K}\hat{P}_2(m;x,\nu,\tau)}{im}\right)dm\right] \\
&=\frac{1}{2}e^x-\frac{1}{2}Ke^{-r\tau}+\frac{1}{\pi}\int_0^{\infty}\text{Re}\left[\frac{e^{-im\ln K}}{im}\left(e^x\hat{P}_1-Ke^{-r\tau}\hat{P}_2\right)\right]dm
\end{aligned}$$

heston_int_func returns the integrand in the last equation. The following code snippet shows the implementation in Python:

```
def heston_int_func(m, x, k, t_m, r, kappa, theta, sigma, rho,
    v0):
    char_func_value = heston_char_function(m, x, t_m, r, kappa,
        theta, sigma, rho, v0)
    int_func_value = (np.exp(-1j * m * np.log(k)) / (1j * m) *
                      (np.exp(x) * char_func_value[0] - k * np.
                          exp(-r * t_m) * char_func_value[1])).
                          real

    return int_func_value
```

Finally, the heston_call_value function returns the price of a European call

```
def heston_call_value(s0, k, t_m, r, kappa, theta, sigma, rho,
    v0):
    x = np.log(s0)
    int_value = quad(lambda m: heston_int_func(m, x, k, t_m, r,
        kappa, theta, sigma, rho, v0),
                     0, np.infty, limit=250)[0]
    call_value = max(0, 0.5 * s0 - 0.5 * k * np.exp(-r * t_m) +
        1 / np.pi * int_value)
    return call_value
```

This completes the derivation of the Heston model.

3.4 Summary of the Heston Model

In this chapter, we have presented the derivation of the Heston model, including the PDEs of the model. From equation 3.12, we have obtained the equation which we call henceforth the Heston p.d.e.

$$\frac{\partial V}{\partial t} + \frac{1}{2}\nu\frac{\partial^2 V}{\partial x^2} + \left(r - \frac{1}{2}\nu\right)\frac{\partial V}{\partial x} + \frac{1}{2}\sigma^2\nu\frac{\partial^2 V}{\partial \nu^2} + \sigma\rho\nu\frac{\partial^2 V}{\partial \nu \partial x} - rV + \kappa(\theta - \nu)\frac{\partial V}{\partial \nu} = 0$$

In addition we have obtained the call price, from equation 3.16 is of the form:

$$C(K) = e^x P_1 - Ke^{-r\tau} P_2$$

where the in-the-money probabilities (P_j) of equation 3.39 are the form:

$$P_j = \Pr(\ln S_T > \ln K) = \frac{1}{2} + \frac{1}{\pi}\int_0^\infty \mathrm{Re}\left[\frac{e^{-im\ln K}\hat{P}_j(m; x, \nu, \tau)}{im}\right] dm$$

and the characteristic functions $(\hat{P}_j)$ of the equation 3.38 are the form:

$$\hat{P}_j(m; x, \nu, \tau) = \exp\left[A_j(m, \tau) + B_j(m, \tau)\nu + imx_t\right]$$

The Heston model has become the most popular stochastic volatility model for pricing equity options. This is in part due to the fact that the call price in the model is available in closed form. Some authors refer to the call price as being in "semi-closed" form because of the numerical integration required to obtain P_1 and P_2. But the Black-Scholes model also requires numerical integration, to obtain $\Phi(d_1)$

and $\Phi(d_2)$. In this sense, the Heston model produces call prices that are no less closed than those produced by the Black-Scholes model. The difference is that programming languages often have built-in routines for calculating the standard normal cumulative distribution function, $\Phi(\cdot)$ (usually by employing a polynomial approximation), whereas the Heston probabilities are not built-in and must be obtained using numerical integration, here we have used quadrature method of integration. In the next chapter, we present finite difference methods that approximate the Heston partial differential equation (3.12).

Finite difference method

In this chapter, we present methods to obtain the European call price by solving the Heston PDE in (3.12) along a two-dimensional grid representing the stock price and the volatility. We first show how to construct non-uniform grids for the discretization of the stock price and the volatility, and present formulas for finite difference approximations to the derivatives in the Heston PDE. We then present the Crank-Nicolson scheme, which shows better results in terms of accuracy, consistency, stability, convergence, and performance than to the explicit scheme.

Recall from Section (3.2) the Heston PDE for the value $V(S, \nu, t)$ of an option when the spot price is S, the volatility is ν, and when the maturity is τ, is the following equation:

$$-\frac{\partial V}{\partial \tau}+\frac{1}{2}\nu\frac{\partial^2 V}{\partial x^2}+\left(r-\frac{1}{2}\nu\right)\frac{\partial V}{\partial x}+\frac{1}{2}\sigma^2\nu\frac{\partial^2 V}{\partial \nu^2}+\sigma\rho\nu\frac{\partial^2 V}{\partial \nu \partial x}-rV+\kappa(\theta-\nu)\frac{\partial V}{\partial \nu}=0 \quad (4.1)$$

Recall also that, the choice of $\tau = T - t$ instead of t in equation (3.12) explains the minus sign in front of the maturity derivative in equation (4.1).
Using $V(t) = V(S, \nu, t)$ as compact notation we can express the PDE (4.1) as:

$$\frac{\partial V}{\partial \tau} = \mathrm{L}\, V(t)$$

where the operator L is defined as:

$$\mathrm{L} = \frac{1}{2}\nu\frac{\partial^2}{\partial x^2} + \left(r-\frac{1}{2}\nu\right)\frac{\partial}{\partial x} + \frac{1}{2}\sigma^2\nu\frac{\partial^2}{\partial \nu^2} + \sigma\rho\nu\frac{\partial^2}{\partial \nu \partial x} - r + \kappa(\theta-\nu)\frac{\partial}{\partial \nu} \qquad (4.2)$$

Finite difference methods are techniques to find a numerical approximation to the PDE. To implement finite differences, we first need a discretization grid for the two state variables (the stock price and the variance), and a discretization grid for the maturity. These grids can have equally or unequally spaced increments. Second, we need discrete approximations to the continuous derivatives that appear in the PDE. Finally, we need a finite difference methodology to solve the PDE.

4.1 Generating non-uniform grids

A grid of a subset $\Omega \subset \mathbb{R}^1$ is a finite set of points $\Omega_h := \{x^{(i)} : i \in \{1, \ldots, m\}\} \subset \Omega$ with points $x^{(i)} \in \mathbb{R}$ in strictly increasing order, i.e. $x^{(i)} < x^{(2)} < \cdots < x^{(m)}$.

A structured grid (sometimes also called tensor grid) of the closure of a region $\bar{\Omega} \in \mathbb{R}^2$, this two-dimensional case, since we need a grid for the state variables (S, ν), is the set

$$\bar{\Omega}_h := \{x^{(k)} = (x_1^{(k_1)}, x_2^{(k_2)})\} \subset \bar{\Omega}$$

based on 2 dimensional meshes $\{x_i^{(1)}, \ldots, x_i^{(m)}\}$, $i = 1, 2$. To distinguish between boundary and inner points we define with Ω_h the set of all inner and with Γ_h the set of all boundary grid points.
The choice of an appropriate mesh is of great importance since it directly influes the error we are making by approximating a continuous function with the function values in only grid points of the mesh.
The strategy is generating a function's mapping from a uniform to a non-uniform grid, the idea is very simple. One has to specify an appropriate function $g : [0, 1] \to [0, 1]$ which is continuously differentiable, bijective and strictly monotonic increasing. Now given a uniform grid on $[0, 1]$ one applies the mapping g to the grid points and scale if necessary, i.e. the resulting non-uniform mesh is then defined by $\{y_i\}_{i=0}^n$ with

$$y_i = cg(x_i) + d, \;\; x_i = \frac{i}{n}, \;\; i = 1, \ldots, n.$$

$\Delta x = x_{i+1} - x_i = \frac{1}{n}$, implies that $x_{i+1} = x_i + \Delta x$, Δx is the distance between two adjacent grid points in the uniform grid, and

$$\Delta y = y_{i+1} - y_i = cg(x_{i+1}) - cg(x_i). \tag{4.3}$$

is the distance between two adjacent grid points in the non-uniform grid.
Using the first order Taylor approximation in section 2.1 (Taylor series), $g(x_{i+1})$ can be written as

$$\begin{aligned} g(x_{i+1}) &= g(x_i) + g'(x_i)\Delta x + R_2(\Delta x) \\ \frac{g(x_{i+1}) - g(x_i)}{\Delta x} &= g^{'}(x_i) \end{aligned}$$

Here $R_2(\Delta x)$ denotes the sum of the factors of order greater or equal than $(\Delta x)^2$, which approximate to zero.
Thus equation (4.3) can be written as:

$$\Delta y \approx cg'(x_i)\Delta x = cg'(g^{-1}(y_i))\Delta x$$

Motivated by this result we introduce a distance ratio function

$$\begin{aligned} r : [0, 1] \to \mathbb{R}^+ \;\; by \\ r(y) = g'(g^{-1}(y)) \end{aligned} \tag{4.4}$$

which characterises the ratio between the distances of two adjacent points of the non-uniform grid and the uniform grid.
Thus g must satisfy the ordinary differential equation (o.d.e)

$$g'(x) = r(g(x))$$

This o.d.e can be solved by the separation of variables approach:

$$g'(x) = \frac{dg(x)}{dx} = r(g(x))$$
$$\frac{dg(x)}{r(g(x))} = dx$$

Integrating

$$\int_0^x \frac{dg(z)}{r(g(z))} dz = \int_0^x dz$$

substituting: $y = g(z), \quad dy = dg(z) \cdot dz$ obtains:

$$\int_0^{g(x)} \frac{dy}{r(y)} = x \tag{4.5}$$

By definition of r in (4.4), $r(y) > 0$, and $r \in C[0,1]$, i.e. r is continuously differentiable. If $x = 1$ then $g(1) = 1$, ($g : [0,1] \to [0,1]$ is bijective an strictly monotonic increasing). Thus r satisfies the relation:

$$\int_0^1 \frac{dy}{r(y)} = 1$$

Before choosing a distance ratio function one needs to be aware of the grid structure one would like to have, e.g. the following questions need to be answered: Is one concentration point sufficient or are there more points where the grid should be finer? and how strong should the distance between two adjacent grid points increase as we go away from the concentration points?. As a simple example we consider the distance ratio function:

$$r(y) = \sqrt{c^2 + p^2(y - y^*)^2}.$$

The parameter y^* can be viewed as the centre of the grid point concentration with c as a measure of the intensity because r assumes its minimum at y^* with $r(y^*) = c$. For big values of y, the function is almost linear since $r(y) = \sqrt{c^2 + p^2(y - y^*)^2} \approx \sqrt{p^2 y^2} = |py|$.
The parameter p has to be set appropriately so that the property of a density function is satisfied. A big advantage of the function r is that we are able to find an analytic solution for the grid generating function g by solving the o.d.e. $g' = r(g)$. From equation (4.5) it follows

$$\int_0^{g(x)} \frac{1}{\sqrt{c^2 + p^2(y - y^*)^2}} dy = x \tag{4.6}$$

To find solution to integral in equation (4.6):

Let $x = \frac{1}{\sqrt{a}} \cdot \operatorname{arcsinh}\left(\frac{2ay + b}{\sqrt{4ac_1 - b^2}}\right)$, with $a > 0$ and $4ac_1 - b^2 > 0$, a, b, c_1 constants.

Solving for y, we have

$$\begin{aligned} \sqrt{4ac_1 - b^2} \cdot \sinh(\sqrt{a}x) &= 2ay + b \\ \frac{\sqrt{4ac_1 - b^2} \cdot \sinh(\sqrt{a}x) - b}{2a} &= y \end{aligned} \tag{4.7}$$

Now using differentiation, we obtain:

$$\begin{aligned} \frac{d}{dy}\left(\frac{\sqrt{4ac_1 - b^2} \cdot \sinh(\sqrt{a}x) - b}{2a}\right) &= \frac{d}{dy}y \\ \frac{\sqrt{4ac_1 - b^2}\sqrt{a}}{2a}\cosh(\sqrt{a}x) \cdot \frac{dx}{dy} &= 1 \\ \frac{\sqrt{4ac_1 - b^2}}{2\sqrt{a}}\cosh(\sqrt{a}x) \cdot \frac{dx}{dy} &= 1 \\ \frac{dx}{dy} &= \frac{2\sqrt{a}}{\sqrt{4ac_1 - b^2} \cdot \cosh(\sqrt{a}x)} \end{aligned}$$

Given that $\cosh x > 0$ for all x, and applying the hyperbolic trigonometric identity $\cosh^2 x - \sinh^2 x = 1$, we have $\cosh(\sqrt{a}x) = \sqrt{1 + \sinh^2(\sqrt{a}x)}$. Thus gives

$$\begin{aligned} \frac{2\sqrt{a}}{\sqrt{4ac_1-b^2}\cdot\cosh(\sqrt{a}x)} &= \frac{2\sqrt{a}}{\sqrt{4ac_1-b^2}\cdot\sqrt{1+\sinh^2(\sqrt{a}x)}} \\ &= \frac{2\sqrt{a}}{\sqrt{4ac_1-b^2+(4ac_1-b^2)\sinh^2(\sqrt{a}x)}} \\ &= \frac{2\sqrt{a}}{\sqrt{4ac_1-b^2+(2ay+b)^2}} \\ &= \frac{2\sqrt{a}}{\sqrt{4ac_1-b^2+4a^2y^2+4aby+b^2}} \\ &= \frac{2\sqrt{a}}{\sqrt{4a(ay^2+by+c_1)}} \\ &= \frac{1}{\sqrt{ay^2+by+c_1}} \end{aligned}$$

From (4.7) $(2ay+b)^2 = (4ac_1 - b^2)\cdot\sinh^2(\sqrt{a}x)$

Thus

$$\begin{aligned} \frac{dx}{dy} &= \frac{1}{\sqrt{ay^2+by+c_1}} \\ dx &= \frac{1}{\sqrt{ay^2+by+c_1}}dy \\ x &= \int \frac{1}{\sqrt{ay^2+by+c_1}}dy \end{aligned}$$

Therefore

$$\int \frac{1}{\sqrt{ay^2+by+c_1}}dy = \frac{1}{\sqrt{a}} \cdot \operatorname{arcsinh}\left(\frac{2ay+b}{\sqrt{4ac_1-b^2}}\right).$$

Given that

$$\begin{aligned} c^2 + p^2(y-y^*)^2 &= c^2 + p^2(y^2 - 2y^*y + y^{*2}) \\ &= p^2y^2 - 2p^2y^*y + c^2 + y^{*2}p^2. \end{aligned}$$

Let $a = p^2, \quad b = -2p^2y^*, \quad c_1 = c^2 + y^{*2}p^2$ obtains:

$$\begin{aligned}\frac{1}{\sqrt{a}} \cdot \operatorname{arcsinh}\left(\frac{2ay+b}{\sqrt{4ac_1-b^2}}\right) &= \frac{1}{p} \cdot \operatorname{arcsinh}\left(\frac{2p^2y - 2p^2y^*}{\sqrt{4p^2(c^2+y^{*2}p^2) - 4p^4y^{*2}}}\right) \\ &= \frac{1}{p} \cdot \operatorname{arcsinh}\left(\frac{2p^2(y-y^*)}{2p\sqrt{c^2+y^{*2}p^2 - p^2y^{*2}}}\right) \\ &= \frac{1}{p} \cdot \operatorname{arcsinh}\left(\frac{p}{c}(y-y^*)\right).\end{aligned}$$

So we conclude that

$$\int_0^{g(x)} \frac{1}{\sqrt{c^2+p^2(y-y^*)^2}} dy = \frac{1}{p}\left[\operatorname{arcsinh}\left(\frac{p}{c}(g(x)-y^*)\right) - \operatorname{arcsinh}\left(\frac{-py^*}{c}\right)\right] = x$$

and thus

$$\begin{aligned}\operatorname{arcsinh}\left(\frac{p}{c}(g(x)-y^*)\right) &= px + \operatorname{arcsinh}\left(\frac{-py^*}{c}\right) \\ g(x) &= \frac{c}{p}\sinh\left[px + \operatorname{arcsinh}\left(\frac{-py^*}{c}\right)\right] + y^*. \qquad (4.8)\end{aligned}$$

4.2 Space Discretization

Uniform grids are those which have equally spaced increments for the two state variables. These grids have two advantages: first, they are easy to construct, and second, since the increments are equal, the finite difference approximations to the derivatives in the PDE take on a simple form. Non-uniform grids are more complicated to construct, and the finite difference approximations to the derivatives are more complicated. These grids, however, can be made finer around certain points, in particular, around the region $(S, \nu) = (K, 0)$, as in practice this is the region in the (S, ν)-domain where one wishes to obtain option prices. Hence, non-uniform grids are often preferable since they produce more accurate prices with fewer grid points, and consequently, with less computation time.

We denote the maximum values of S, ν, and t as $S_{\max}$, $\nu_{\max}$ and $t_{\max} = \tau$ (the maturity), and the minimum values as $S_{\min}$, ν_{min}, and $t_{\min} = 0$. We denote by $V_{i,j}^n = V(S_i, \nu_j, t_n)$ the value of a European call at time t_n when the stock price is S_i and the volatility is ν_j. We use $N_S + 1$ points for the stock price, $N_V + 1$ points for the volatility, and $N_T + 1$ points for the maturity. For convenience, sometimes we write simply the other way around $V(S_i, \nu_j)$ for $V_{i,j}^n$.

Using the minimum values $S_{\min} = \nu_{\min} = 0$ and following (Kluge, 2002), who describes a non-uniform grid that is finer around the strike price K and around the spot volatility $\nu_0 = 0$, the grid of size $N_S + 1$ for the stock price is:

$$S_i = c\sinh\left[i\Delta\zeta + \arcsin\left(\frac{-K}{c}\right)\right] + K, \quad i = 0, 1, \ldots, N_S \qquad (4.9)$$

with

$$\Delta\zeta = \frac{1}{N_S}\left[\arcsin\left(\frac{S_{\max}-K}{c}\right) - \arcsin\left(\frac{-K}{c}\right)\right]$$

Equation (4.9) follows from (4.8) with $p = 1$, $y^* = K$ and the parameter c, which controls the fraction of mesh points S_i that lie in the neighborhood of the strike K. We set $c = K/5$, which follows from the numerical experiments by (Hout and S.Foulon, 2010).

We define a non-uniform mesh in the ν-direction analogous to S-direction. Let a constant $d > 0$, the grid of size $N_V + 1$ for the volatility is:

$$\nu_j = d\sinh(j\Delta\eta), \quad j = 0, 1, \ldots, N_V$$

with

$$\Delta\eta = \frac{1}{N_V}\arcsin\left(\frac{\nu_{\max}}{d}\right)$$

In (Hout and S.Foulon, 2010) use $d = \nu_{\max}/500$, and a uniform grid for t. Figure 4.1 illustrates a non-uniform grid using their settings, along with $N_S = 50$, $N_V = 25$, $S_{\max} = 8$, $\nu_{\max} = 5$ and $K = 1$. The grid for the stock price is represented by horizontal points, and the volatility by vertical points.

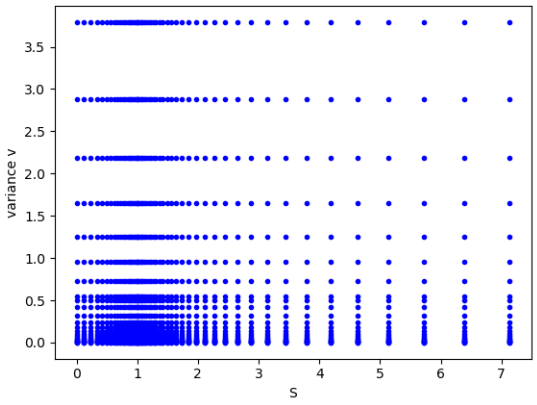


Figure 4.1: Non-uniform grid.
Source: Own elaboration.

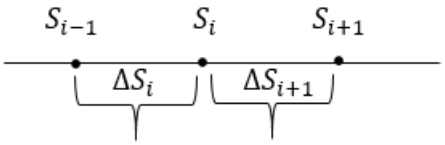
Note that in stock price dimension the grid is finest around the strike price, while in the volatility dimension the grid becomes finer as we progress towards zero.

4.3 Approximation of derivatives

The general approach to approximate derivatives of a function $V : \Omega \subset \mathbb{R}^2 \to \mathbb{R}$, $V \in C^3(\Omega)$, in a certain grid point $x^{(k)} \in \Omega_h$ is to use a weighted sum of the function values of adjacent grid points. For the first order derivatives in equation (4.2) (Operator L) the approximation is made using a scheme with only three points, and the idea is to approximate the derivatives using the following equation:

$$\frac{\partial V}{\partial S}(S_i) \approx \sum_{k=-1}^{1} a_k V(S_{i+k}) \tag{4.10}$$

Applying Taylor series expansion around the grid point S_i, with the abbreviations: $V_i = V(S_i)$, $V_i^{'} = V_S = \frac{\partial}{\partial S}V(S_i, \nu_j)$, $V_i^{''} = V_{SS} = \frac{\partial^2}{\partial S^2}V(S_i, \nu_j)$, $\Delta S_i = S_i - S_{i-1}$.



$\Delta S_i = S_i - S_{i-1}$ and $\Delta S_{i+1} = S_{i+1} - S_i$

$$V(S_{i-1}) = V(S_i - \Delta S_i) = V_i - \Delta S_i V_i^{'} + \frac{1}{2!}(\Delta S_i)^2 V_i^{''} - \frac{1}{3!}(\Delta S_i)^3 V_i^{'''} + \cdots$$
$$V(S_{i+1}) = V(S_i + \Delta S_{i+1}) = V_i + \Delta S_{i+1} V_i^{'} + \frac{1}{2!}(\Delta S_{i+1})^2 V_i^{''} + \frac{1}{3!}(\Delta S_{i+1})^3 V_i^{'''} + \cdots$$

The above expressions are commonly named backward and forward differencing, respectively.
From equation (4.10) we obtain

$$\begin{aligned}
V_i^{'} &= a_{-1}V(S_{i-1}) + a_0 V(S_i) + a_1 V(S_{i+1}) \\
&= a_{-1}\left(V_i - \Delta S_i V_i^{'} + \frac{1}{2}(\Delta S_i)^2 V_i^{''} + R_3(-\Delta S_i)\right) + a_0 V_i \\
&+ a_1\left(V_i + \Delta S_{i+1} V_i^{'} + \frac{1}{2}(\Delta S_{i+1})^2 V_i^{''} + R_3(\Delta S_{i+1})\right) \\
&= (a_{-1} + a_0 + a_1)V_i + (-a_{-1}\Delta S_i + a_1 \Delta S_{i+1})V_i^{'} + \left(\frac{1}{2}a_{-1}(\Delta S_i)^2 + \frac{1}{2}a_1(\Delta S_{i+1})^2\right)V_i^{''} \\
&+ a_{-1}R_3(-\Delta S_i) + a_1 R_3(\Delta S_{i+1}) \\
&= \begin{pmatrix} V_i & V_i^{'} & V_i^{''} \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 \\ -\Delta S_i & 0 & \Delta S_{i+1} \\ \frac{1}{2}(\Delta S_i)^2 & 0 & \frac{1}{2}(\Delta S_{i+1})^2 \end{pmatrix} \begin{pmatrix} a_{-1} \\ a_0 \\ a_1 \end{pmatrix} + a_{-1}R_3(-\Delta S_i) + a_1 R_3(\Delta S_{i+1})
\end{aligned}$$

Here $R_3(-\Delta S_i)$ and $R_3(\Delta S_{i+1})$ denotes the sum of the factors of order greater than $(-\Delta S_i)^2$ and $(\Delta S_{i+1})^2$ respectively, which approximate to zero.
In order to approximate the first derivative, we have to choose the factors a_{-1}, a_0, a_1 so that factors before the function value V_i and its second derivative $V_i^{''}$ are zero and the factor before the first derivative $V_i^{'}$ is one. In general, the following linear equations have to be solved, where exactly one of the $\gamma' s$ is one and the others are zero depending on which derivative has to be approximated.

$$\begin{pmatrix} 1 & 1 & 1 \\ -\Delta S_i & 0 & \Delta S_{i+1} \\ \frac{1}{2}(\Delta S_i)^2 & 0 & \frac{1}{2}(\Delta S_{i+1})^2 \end{pmatrix} \begin{pmatrix} a_{-1} \\ a_0 \\ a_1 \end{pmatrix} = \begin{pmatrix} \gamma \\ \gamma_S \\ \gamma_{SS} \end{pmatrix}$$

- With $\gamma_S = 1$, $\gamma = \gamma_{SS} = 0$

$$\begin{array}{ll} (1) & \left\{ \begin{array}{rcr} a_{-1} + a_0 + & & a_1 = 0 \\ -\Delta S_i a_{-1} & + & \Delta S_{i+1} a_1 = 1 \\ \frac{1}{2}(\Delta S_i)^2 a_{-1} & + \frac{1}{2}(\Delta S_{i+1})^2 a_1 = 0 \end{array} \right. \\ (2) & \\ (3) & \end{array}$$

It follows from the equation (3) that $a_1 = -a_{-1}\frac{(\Delta S_i)^2}{(\Delta S_{i+1})^2}$
Substituting a_1 into equation (2):

$$-\Delta S_i a_{-1} - \frac{(\Delta S_i)^2}{\Delta S_{i+1}} a_{-1} = 1$$

$$a_{-1} = \frac{-1}{\Delta S_i + \frac{(\Delta S_i)^2}{\Delta S_{i+1}}}$$

$$a_{-1} = \frac{-\Delta S_{i+1}}{\Delta S_i \left(\Delta S_{i+1} + \Delta S_i\right)}$$

Therefore we have:

$$a_1 = \frac{\Delta S_i}{\Delta S_{i+1} \left(\Delta S_{i+1} + \Delta S_i\right)}$$

Substituting a_{-1} and a_1 into equation (1):

$$\frac{-\Delta S_{i+1}}{\Delta S_i \left(\Delta S_{i+1} + \Delta S_i\right)} + a_0 + \frac{\Delta S_i}{\Delta S_{i+1} \left(\Delta S_{i+1} + \Delta S_i\right)}$$

Thus we have:

$$-\left(\Delta S_{i+1}\right)^2 + \Delta S_i \Delta S_{i+1} \left(\Delta S_{i+1} + \Delta S_i\right) a_0 + \left(\Delta S_i\right)^2 = 0$$

$$\begin{aligned} a_0 &= \frac{\left(\Delta S_{i+1}\right)^2 - \left(\Delta S_i\right)^2}{\Delta S_i \Delta S_{i+1} \left(\Delta S_{i+1} + \Delta S_i\right)} \\ &= \frac{\left(\Delta S_{i+1} - \Delta S_i\right) \cancel{\left(\Delta S_{i+1} + \Delta S_i\right)}}{\Delta S_i \Delta S_{i+1} \cancel{\left(\Delta S_{i+1} + \Delta S_i\right)}} \\ &= \frac{\Delta S_{i+1} - \Delta S_i}{\Delta S_i \Delta S_{i+1}} \end{aligned}$$

- With $\gamma_{SS} = 1$, $\gamma = \gamma_S = 0$

$$\begin{array}{ll} (1) & \left\{ \begin{array}{rcr} a_{-1} + a_0 + & & a_1 = 0 \\ -\Delta S_i a_{-1} & + & \Delta S_{i+1} a_1 = 0 \\ \frac{1}{2}(\Delta S_i)^2 a_{-1} & + \frac{1}{2}(\Delta S_{i+1})^2 a_1 = 1 \end{array} \right. \\ (2) & \\ (3) & \end{array}$$

It follows from the equation (2) that $a_1 = a_{-1}\frac{\Delta S_i}{\Delta S_{i+1}}$.
Substituting a_1 into equation (3):

$$\frac{1}{2}(\Delta S_i)^2 a_{-1} + \frac{1}{2}\Delta S_i \Delta S_{i+1} a_{-1} = 1$$

$$a_{-1} = \frac{2}{\Delta S_i(\Delta S_{i+1} + \Delta S_i)}$$

Therefore we have:

$$a_1 = \frac{2}{\Delta S_{i+1}\left(\Delta S_{i+1} + \Delta S_i\right)}$$

Substituting a_{-1} and a_1 into equation (1):

$$\frac{2}{\Delta S_i(\Delta S_{i+1} + \Delta S_i)} + a_0 + \frac{2}{\Delta S_{i+1}\left(\Delta S_{i+1} + \Delta S_i\right)} = 0$$

Thus we have:

$$\begin{aligned} 2\Delta S_{i+1} + \Delta S_i \Delta S_{i+1}(\Delta S_i + \Delta S_{i+1})a_0 + 2\Delta S_i &= 0 \\ a_0 = \frac{-2\cancel{(\Delta S_i + \Delta S_{i+1})}}{\Delta S_i \Delta S_{i+1}\cancel{(\Delta S_i + \Delta S_{i+1})}} \\ = \frac{-2}{\Delta S_i \Delta S_{i+1}} \end{aligned}$$

- With $\gamma = 1$, $\gamma_S = \gamma_{SS} = 0$ it is easy to check that $a_{-1} = a_1 = 0$ and $a_0 = 1$

We summarise the result in table 4.1

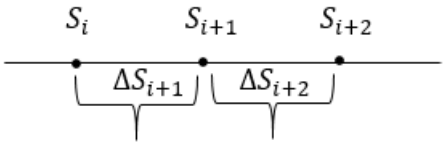
a_k	V	V'	V''
a_{-1}	0	$\frac{-\Delta S_{i+1}}{\Delta S_i(\Delta S_i+\Delta S_{i+1})}$	$\frac{2}{\Delta S_i(\Delta S_i+\Delta S_{i+1})}$
a_0	1	$\frac{\Delta S_{i+1}-\Delta S_i}{\Delta S_i \Delta S_{i+1}}$	$\frac{-2}{\Delta S_i \Delta S_{i+1}}$
a_1	0	$\frac{\Delta S_i}{\Delta S_{i+1}(\Delta S_i+\Delta S_{i+1})}$	$\frac{2}{\Delta S_{i+1}(\Delta S_i+\Delta S_{i+1})}$

Table 4.1: Central approximation of derivatives, inner points

The central difference scheme can not be applied on boundaries. For convection dominated parabolic p.d.e.s the finite method with central difference approximation exhibits an oscillating behaviour. That is why we also need to discuss left and right hand side approximations.
Beginning with right hand side approximation, the idea is to approximate the derivatives using the following equation:

$$\frac{\partial V}{\partial S}(S_i) \approx \sum_{k=0}^{2} a_k V(S_{i+k})$$

Here $\Delta S_{i+1} = S_{i+1} - S_i$, and $\Delta S_{i+2} = S_{i+2} - S_{i+1}$



Then

$$\sum_{k=0}^{2} a_k V(S_{i+k}) = a_0 V(S_i) + a_1 V(S_{i+1}) + a_2 V(S_{i+2}) \tag{4.11}$$

therefore, we need to get an expression for $V(S_{i+1})$ and $V(S_{i+2})$.

$$V(S_{i+1}) = V(S_i + \Delta S_{i+1}) = V_i + \Delta S_{i+1} V_i^{'} + \frac{1}{2}(\Delta S_{i+1})^2 V_i^{''} + R_3(\Delta S_{i+1})$$
$$V(S_{i+2}) = V(S_{i+1} + \Delta S_{i+2}) = V(S_i + \Delta S_{i+1} + \Delta S_{i+2})$$
$$= V_i + (\Delta S_{i+1} + \Delta S_{i+2}) V_i^{'} + \frac{1}{2}(\Delta S_{i+1} + \Delta S_{i+2})^2 V_i^{''} + R_3(\Delta S_{i+1} + \Delta S_{i+2})$$

From equation (4.11) we obtain

$$\begin{aligned}
V_i^{'} &= a_0 V(S_i) + a_1 V(S_{i+1}) + a_2 V(S_{i+2}) \\
&= a_0 V_i + a_1 \left(V_i + \Delta S_{i+1} V_i^{'} + \frac{1}{2}(\Delta S_{i+1})^2 V_i^{''} + R_3(\Delta S_{i+1}) \right) \\
&+ a_2 \left(V_i + (\Delta S_{i+1} + \Delta S_{i+2}) V_i^{'} + \frac{1}{2}(\Delta S_{i+1} + \Delta S_{i+2})^2 V_i^{''} + R_3(\Delta S_{i+1} + \Delta S_{i+2}) \right) \\
&= (a_0 + a_1 + a_2) V_i + \left[a_1 \Delta S_{i+1} + a_2 (\Delta S_{i+1} + \Delta S_{i+2}) \right] V_i^{'} \\
&+ \left[\frac{1}{2} a_1 (\Delta S_{i+1})^2 + \frac{1}{2} a_2 (\Delta S_{i+1} + \Delta S_{i+2})^2 \right] V_i^{''} + a_1 R_3(\Delta S_{i+1}) + a_2 R_3(\Delta S_{i+1} + \Delta S_{i+2}) \\
&= \begin{pmatrix} V_i & V_i^{'} & V_i^{''} \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 \\ 0 & \Delta S_{i+1} & \Delta S_{i+1} + \Delta S_{i+2} \\ 0 & \frac{1}{2}(\Delta S_{i+1})^2 & \frac{1}{2}(\Delta S_{i+1} + \Delta S_{i+2})^2 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix}
\end{aligned}$$

Applying the same method described above results in the system:

$$\begin{pmatrix} 1 & 1 & 1 \\ 0 & \Delta S_{i+1} & \Delta S_{i+1} + \Delta S_{i+2} \\ 0 & \frac{1}{2}(\Delta S_{i+1})^2 & \frac{1}{2}(\Delta S_{i+1} + \frac{1}{2}(\Delta S_{i+2})^2)^2 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} \gamma \\ \gamma_S \\ \gamma_{SS} \end{pmatrix}$$

Applying the substitution method when $\gamma_S = 1$ and the other $\gamma' s$ are equal to zero, and on the other hand when $\gamma_{SS} = 1$ and the other $\gamma' s$ are equal to zero, we obtain the result of this equation system given in Table 4.2.

a_k	V	V'	V''
a_0	1	$-\frac{2\Delta S_{i+1} + \Delta S_{i+2}}{\Delta S_{i+1}(\Delta S_{i+1} + \Delta S_{i+2})}$	$\frac{2}{\Delta S_{i+1}(\Delta S_{i+1} + \Delta S_{i+2})}$
a_1	0	$\frac{\Delta S_{i+1} + \Delta S_{i+2}}{\Delta S_{i+1} \Delta S_{i+2}}$	$\frac{-2}{\Delta S_{i+1} \Delta S_{i+2}}$
a_2	0	$\frac{-\Delta S_{i+1}}{\Delta S_{i+2}(\Delta S_{i+1} + \Delta S_{i+2})}$	$\frac{2}{\Delta S_{i+2}(\Delta S_{i+1} + \Delta S_{i+2})}$

Table 4.2: Right hand side approximation of derivatives, left border

The left hand side approximation of derivatives with

$$\frac{\partial V}{\partial S}(S_i) \approx \sum_{k=-2}^{0} a_k V(S_{i+k})$$

Here $\Delta S_{i-1} = S_{i-1} - S_{i-2}$, and $\Delta S_i = S_i - S_{i-1}$
Then

$$\sum_{k=-2}^{0} a_k V(S_{i+k}) = a_{-2} V(S_{i-2}) + a_{-1} V(S_{i-1}) + a_0 V(S_i) \tag{4.12}$$

therefore, we need to get an expression for $V(S_{i-2})$ and $V(S_{i-1})$.

$$\begin{aligned}
V(S_{i-2}) &= V(S_{i-1} - \Delta S_{i-1}) = V(S_i - \Delta S_i - \Delta S_{i-1}) \\
&= V_i - (\Delta S_i + \Delta S_{i-1})V_i^{'} + \frac{1}{2}(\Delta S_i + \Delta S_{i-1})^2 V_i^{''} + R_3(-(\Delta S_i + \Delta S_{i-1})) \\
V(S_{i-1}) &= V(S_i - \Delta S_i) \\
&= V_i - \Delta S_i V_i^{'} + \frac{1}{2}(\Delta S_i)^2 V_i^{''} + R_3(-\Delta S_i)
\end{aligned}$$

From equation (4.12) we obtain

$$\begin{aligned}
V_i^{'} &= a_{-2}V(S_{i-2}) + a_{-1}V(S_{i-1}) + a_0 V(S_i) \\
&= a_{-2}\left[V_i - (\Delta S_i + \Delta S_{i-1})V_i^{'} + \frac{1}{2}(\Delta S_i + \Delta S_{i-1})^2 V_i^{''} + R_3(-(\Delta S_i + \Delta S_{i-1}))\right] \\
&+ a_{-1}\left[V_i - \Delta S_i V_i^{'} + \frac{1}{2}(\Delta S_i)^2 V_i^{''} + R_3(-\Delta S_i)\right] + a_0 V_i \\
&= (a_{-2} + a_{-1} + a_0)V_i + \left[-(\Delta S_i + \Delta S_{i-1})a_{-2} - \Delta S_i a_{-1}\right] V_i^{'} \\
&+ \left[\frac{1}{2}(\Delta S_i + \Delta S_{i-1})^2 a_{-2} + \frac{1}{2}(\Delta S_i)^2 a_{-1}\right] V_i^{''} \\
&+ a_{-2}R_3(-(\Delta S_i + \Delta S_{i-1})) + a_{-1}R_3(-\Delta S_i) \\
&= \begin{pmatrix} V_i & V_i^{'} & V_i^{''} \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 \\ -(\Delta S_i + \Delta S_{i-1}) & -\Delta S_i & 0 \\ \frac{1}{2}(\Delta S_i + \Delta S_{i-1})^2 & \frac{1}{2}(\Delta S_i)^2 & 0 \end{pmatrix} \begin{pmatrix} a_{-2} \\ a_{-1} \\ a_0 \end{pmatrix}
\end{aligned}$$

Applying the same method described above results in the system:

$$\begin{pmatrix} 1 & 1 & 1 \\ -(\Delta S_i + \Delta S_{i-1}) & -\Delta S_i & 0 \\ \frac{1}{2}(\Delta S_i + \Delta S_{i-1})^2 & \frac{1}{2}(\Delta S_i)^2 & 0 \end{pmatrix} \begin{pmatrix} a_{-2} \\ a_{-1} \\ a_0 \end{pmatrix} = \begin{pmatrix} \gamma \\ \gamma_S \\ \gamma_{SS} \end{pmatrix}$$

Applying the substitution method when $\gamma_S = 1$ and the other $\gamma' s$ are equal to zero, and on the other hand when $\gamma_{SS} = 1$ and the other $\gamma' s$ are equal to zero, we obtain the result of this equation system given in Table 4.3.

a_k	V	V'	V''
a_{-2}	0	$\frac{\Delta S_i}{\Delta S_{i-1}(\Delta S_i + \Delta S_{i-1})}$	$\frac{2}{\Delta S_{i-1}(\Delta S_i + \Delta S_{i-1})}$
a_{-1}	0	$-\frac{\Delta S_i + \Delta S_{i-1}}{\Delta S_i \Delta S_{i-1}}$	$\frac{-2}{\Delta S_i \Delta S_{i-1}}$
a_0	1	$\frac{2\Delta S_i + \Delta S_{i-1}}{\Delta S_i(\Delta S_i + \Delta S_{i-1})}$	$\frac{2}{\Delta S_i(\Delta S_i + \Delta S_{i-1})}$

Table 4.3: Left hand side approximation of derivatives, right border

4.4 Crank-Nicolson Scheme

4.4.1 The Weighted method

This is a general method that incorporates other finite difference schemes as special cases. Recall that $V_{i,j}^n$ denotes the value of the European call at the grid points (S_i, ν_j) and at maturity t_n. Since the two-dimensional grid for (S, ν) is of size $N = (N_S + 1)(N_V + 1)$, at each t_n there are N possible values for $V_{i,j}^n$, as indicated in figure 4.2. To apply the weighted method, we must construct a vector V^n of size N with these values, arranged in any way we like. We choose to stack the N_V column vectors $\nu_0, \nu_1, \nu_2, \ldots, \nu_{N_V}$ in figure 4.2 on top of one another, so that $V^n = (\nu_0^T, \nu_1^T, \ldots, \nu_{N_V}^T)^T$ is our vector.

	ν_0	ν_1	$\ldots$	ν_{N_V}
S_0	$V_{0,0}^n$	$V_{0,1}^n$	$\cdots$	V_{0,N_V}^n
S_1	$V_{1,0}^n$	$V_{1,1}^n$	$\cdots$	V_{1,N_V}^n
$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$
S_{N_S}	$V_{N_S,0}^n$	$V_{N_S,1}^n$	$\cdots$	V_{N_S,N_V}^n

Figure 4.2: Value of the European Call along the Stock Price and Variance Grids

The entries of V^n therefore correspond to the following (S_i, ν_j) points

$$\underbrace{(S_0,\nu_0),(S_1,\nu_0),\ldots,(S_{N_S},\nu_0)}_{\text{Values of } S \text{ for } \nu_0},\ \underbrace{(S_0,\nu_1),(S_1,\nu_1),\ldots,(S_{N_S},\nu_1)}_{\text{Values of } S \text{ for } \nu_1},\ldots,$$
$$\underbrace{(S_0,\nu_{N_V-1}),(S_1,\nu_{N_V-1}),\ldots,(S_{N_S},\nu_{N_V-1})}_{\text{Values of } S \text{ for } \nu_{N_V-1}},\ \underbrace{(S_0,\nu_{N_V}),(S_1,\nu_{N_V}),\ldots,(S_{N_S},\nu_{N_V})}_{\text{Values of } S \text{ for } \nu_{N_V}},\ldots,$$

The weighted method, also called the θ-method, is defined via the relationship

$$\frac{V^{n+1} - V^n}{dt} = \mathrm{L}(\theta V^{n+1} + (1-\theta)V^n)$$

where L is sparse matrix of dimension $N \times N$. This matrix is based on the operator defined in equation (4.2). The initial condition V^0 is known, since it represents the value of the call at expiry. Hence, we can work from expiry, starting with the initial value V^0, and we use L to obtain V^1, V^2, and so forth, until we reach V^{N_T}. This is done by solving, at each time, the system

$$(\mathrm{I} - \theta dt\, \mathrm{L})\, V^{n+1} = (\mathrm{I} + (1-\theta)\, dt\, \mathrm{L})\, V^n \tag{4.13}$$

where I is the identity matrix of size N. The system can be solved by taking the inverse of the matrix on the left-hand side, so that

$$V^{n+1} = (\mathrm{I} - \theta dt\, \mathrm{L})^{-1}\, (\mathrm{I} + (1-\theta)\, dt\, \mathrm{L})\, V^n$$

The vector V^0 depends on the option being priced. For a call option, it will contain $S-K$ in the components of V that correspond to $S > K$, and zero in the components that correspond to $S < K$. The order in which these appear in the vector will depend on how the components of V are arranged. A number of finite difference schemes arise as a special case of equation (4.13), depending on the value of θ. Setting $\theta = 0$ produces the explicit scheme, $\theta = 1/2$ produces the Crank-Nicolson scheme, and $\theta = 1$ produces the implicit scheme.
We decompose L into three matrices A_0, A_1, and A_2 each of size $N \times N$, so that

$$L = A_0 + A_1 + A_2$$

where A_0 contains all entries of L corresponding to the mixed derivative $\partial^2 V/\partial S \partial \nu$, A_1 contains all entries corresponding to $\partial V/\partial S$ and $\partial^2 V/\partial S^2$, and A_2 contains all entries corresponding to $\partial V/\partial \nu$ and $\partial^2 V/\partial \nu^2$. The entries of L corresponding to $rV^n_{i,j}$ are split evenly between A_1 and A_2. Hence, we construct the matrices as:

$$\begin{aligned}
A_0 &= \sigma\rho\nu S \left(\frac{\partial V}{\partial S \partial \nu}\right)_{N\times N} \\
A_1 &= rS \left(\frac{\partial V}{\partial S}\right)_{N\times N} + \frac{1}{2}\nu S^2 \left(\frac{\partial^2 V}{\partial S^2}\right)_{N\times N} - \frac{1}{2} r(V)_{N\times N} \\
A_2 &= \kappa(\theta - \nu) \left(\frac{\partial V}{\partial \nu}\right)_{N\times N} + \frac{1}{2}\sigma^2 \nu \left(\frac{\partial^2 V}{\partial \nu^2}\right)_{N\times N} - \frac{1}{2} r(V)_{N\times N}.
\end{aligned}$$

4.5 Summary of the FDM

In this chapter, we have presented finite difference methods, which are commonly used to obtain European prices in the Heston model. We present the necessary steps to apply the method, as follows: Generate a non-uniform grid, build the space discretization, approximate of derivatives and use the Crank-Nicolson scheme as a solver.

Artificial Neural Network

5.1 Defining Neural Networks

Neural networks are machine learning models inspired by the biological layout of animal and more specifically human brains. They contain many connected units, which mimic the way neurons are laid out in the brain and data is fed through units. Each unit performs a task on the data, for example it could round a number, decide if a result is positive or negative, and so on. This is a graphical representation of what a single unit neuron:

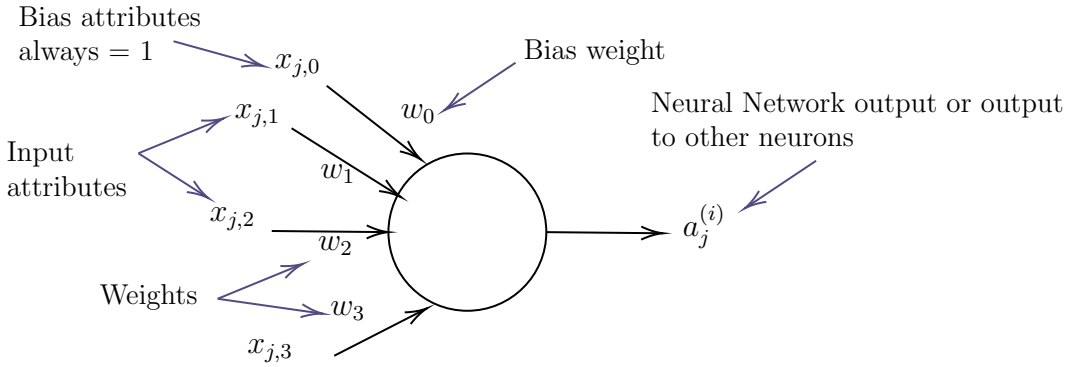


Figure 5.1: Single unit neuron. Source: own elaboration

The unit has inputs on the left, weights that influence choices, a bias attribute, which is an additional set of weights in a neural network that require no input, and this corresponds to the output of an artificial neural network when it has zero inputs. Bias represents an extra neuron included with each pre-output layer and stores the value of "1", for each action. Bias units are not tied to any previous layer in the network, so they don't represent any form of activity, but are treated the same as any other weight. Bias is a fundamental aspect because without a bias node, no layer would be able to produce an output for the next layer that differs from 0 if the feature values were 0.

This is a basic representation of a 3-layer neural network with two inputs, two hidden layers of 3 neurons each and one output layer with 2 neurons.

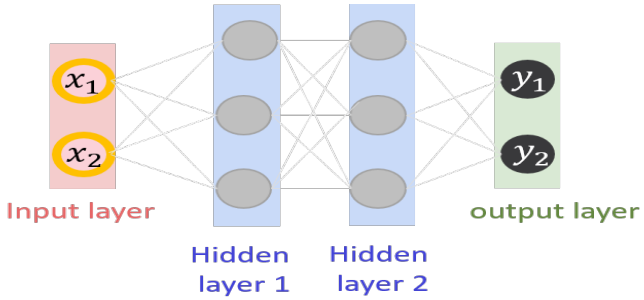
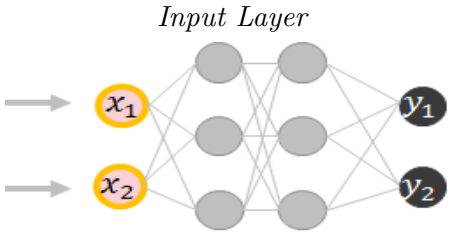


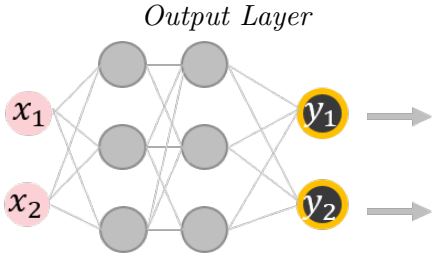
Figure 5.2: Graph of a 3-layer neural network.
Source: own elaboration

Notice that when we say N-layer neural network, we do not count the input layer. Therefore, a single-layer neural network describes a network with no hidden layers (input directly mapped to output). In that sense, you can sometimes hear people say that logistic regression is simply a special case of single-layer Neural Networks. You may also hear these networks interchangeably referred to as "Artificial Neural Networks" (ANN) or "Multi-Layer Perceptrons" (MLP). Many people do not like the analogies between Neural Networks and real brains and prefer to refer to neurons as units.

The input layer of a neural network is composed of artificial input neurons, and brings the initial data into the system for further processing by subsequent layers of artificial neurons, each input unit deals with a different attribute. The input layer is the very beginning of the workflow for the artificial neural network.



The output layer is the final layer of the network and is responsible for providing the answer to a question. Usually is taken to represent the class scores (e.g. in classification), which are arbitrary real-valued numbers, or some kind of real-valued target (e.g. in regression), often, there are multiple output units, which require some sort of transformation before giving an answer. For example, Softmax functions (a common occurrence in the last layer of neural networks) normalizes data before providing an answer.



Finally, hidden layers are any layer between the input and output layers.

So, to continue, we define a notation from (Shuaiqiang Liu and Bohte, 2019) article. A Feedforward Neural Network is a set of many layers of neurons connected together. Then it takes in an input, that input "trickles" through the network and then the neural network returns an output vector.
More formally, call z_j^l the activation of the j^{th} neuron in the l-th layer, where z_j^0 is the j^{th} element in the input vector. Then we can relate consecutive layers via the following relation:

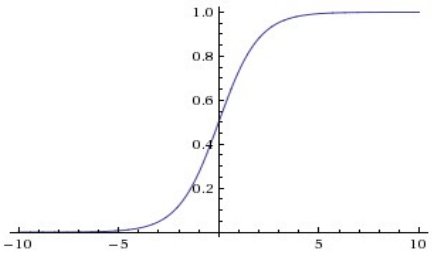
$$z_j^{(l)} = \phi^{(l)} \left(\sum_k w_{jk}^{(l)} \cdot z_k^{(l-1)} + b_j^{(l)} \right)$$

where
$\phi(\cdot)$ is the activation function,
$w_{jk}^{(l)}$ is the weight from the k^{th} neuron in the $(l-1)$-th layer to the j^{th} neuron in the l-th layer,
$b_j^{(l)}$ is the bias of the j^{th} neuron in the l^{th} layer, and
$z_j^{(l-1)}$ is the output value of the j^{th} neuron in the $(l-1)$-th layer.
For more concise notation we can write $z^{(l)} = \phi(w^{(l)} \cdot z^{(l-1)} + b^{(l)})$.

5.2 Activation Functions

In this section, we'll discuss the common types of activation functions. In computing, activation functions are used to determine the output based on inputs. A simple example would be the true or false paradigm of Boolean logic.
In neural networks, perceptrons are algorithms representing a single true or false decision, and they are known as linear classifiers forming a chain of true or false determinations. So, Why are the activation functions needed? Well, the function is a linear relationship, and if we only consider linear relationships, a model won't explore potentially better relationships. Therefore, activation functions give a model the potential to learn complex relationships.
Every activation function (or non-linearity) takes a single number and performs a certain fixed mathematical operation on it. There are several activation functions you may encounter in practice:

Sigmoid Function

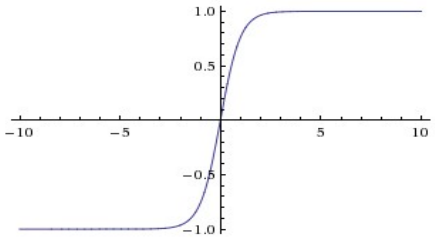


The sigmoid function has the mathematical form:

$$\phi(x) = \frac{1}{1 + e^{-x}}$$

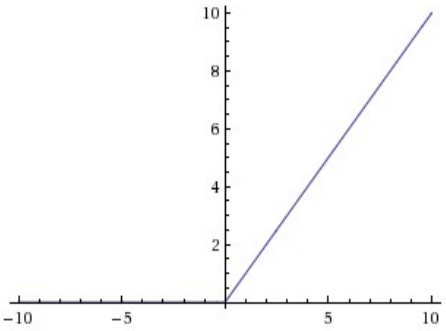
This function takes a real-valued number and "squeezes" it into range between 0 and 1. The sigmoid function has seen frequent use historically since it has a nice interpretation as the firing rate of a neuron: from not firing at all (0) to fully-saturated firing at an assumed maximum frequency (1).

Tanh Function



The tanh squeezes a real-valued number to the range $[-1, 1]$. Like the sigmoid neuron, its activations saturate, but unlike the sigmoid neuron its output is zero-centered. Therefore, in practice the tanh non-linearity is always preferred to the sigmoid non-linearity. It is important to note that the tanh neuron is simply a scaled sigmoid neuron, in particular the following holds: $\tanh(x) = 2\phi(2x) - 1$

ReLU Function



Rectified Linear Unit, or ReLU, is a popular activation function. It computes the function $f(x) = \max(0, x)$. In other words, the activation is simply zero when $x < 0$ and then linear with slope 1 when $x > 0$.
The fact that ReLU is effectively a function that is zero for negative inputs and identity for positive inputs means that it is easy to have zeros as outputs and this leads to dead neurons. However, dead neurons may sound bad, therefore, we simply say that ReLU does a similar job of what an L1 regularization would do which would bring some weights to zero.
L1 regularization is also referred as L1 norm. In L1 norm we shrink the parameters to zero, when input features have weights close to zero, which is useful to feature selection and helps to solve over fitting problem in machine learning.

5.3 Cost Functions for Neural Networks

In this section we discus how cost functions are used to train neural networks. In neural network design, a cost function is a measure of "how good" a neural network did with respect to it's given training sample and the expected output. Simply put, answers the question: How well a network performed in its decision process? It is a way of incentivizing a neural network to make better and ultimately the correct decisions. The term correct is entirely subjective, because in the real world, decisions are not always based on absolute input and so neural networks use cost functions.
A cost function is a single value, not a vector, because it rates how good the neural network did as a whole.
According to the Universal Approximation Theorem (Cybenko, 1989), a single-hidden-layer ANN with a sufficient number of neurons can approximate any continuous function. The distance between two functions is measured by the norm of a function $||\cdot||$,

$$D(f(x), F(x)) = ||f(x) - F(x)||,$$

where $f(x)$ is the objective function, and $F(x)$ is the neural network approximated function.
Within supervised learning, the loss function is equivalent to the above distance, specifically, a cost function is of the form

$$J(\Theta) := D(f(x), F(x|\Theta))$$

where $\Theta = (\mathbf{W}_1, \mathbf{b}_1, \mathbf{W}_2, \mathbf{b}_2, \ldots, \mathbf{W}_L, \mathbf{b}_L)$, is the parameters vector, $\mathbf{W}_j$ is a weight matrix, and $\mathbf{b}_j$ is the bias vector in the L-th neural layer.
There are conditions for neural network cost functions: They must be able to be written as an average taking all inputs into account and they should only depend on the output and not any hidden activation values. We choose mean squared error (MSE) to evaluate the averaged accuracy, MSE, calculates how much error a model has, and is defined as:

$$J_{\text{MSE}}(f(x), F(x|\Theta)) = \frac{1}{N}\sum_{j}^{N}(f(x_j) - F(x_j|\Theta))^2$$

Where N is the number of samples.

5.4 Optimization Algorithms

The training process aims to learn the optimal weights and biases in a function $y(x) = F(x|\Theta)$ to make the loss function as small as possible. The process can be formulated as an optimization problem,

$$\arg\min_{\theta} J(\Theta|(x, y)), \tag{5.1}$$

given the known input-output pairs (x, y) and a loss function $J(\Theta)$.
Several back-propagation gradient descent methods have been successfully applied to solve equation (5.1), for instance, Stochastic Gradient Descent (SGD) and its variants Adam and RMSprop. These optimization algorithms start with initial values and move in the direction in which the loss function decreases. The formula for updating the parameters reads,

$$\begin{cases} \mathbf{W} \leftarrow \mathbf{W} - \eta(i)\frac{\partial J}{\partial \mathbf{W}}, \\ \mathbf{b} \leftarrow \mathbf{b} - \eta(i)\frac{\partial J}{\partial \mathbf{b}}, \\ i = 0, 1, 2, \ldots, \end{cases}$$

where η is a learning rate, which may vary during the iterations. The learning rate plays an important role during the training, as a "large" learning rate value causes the ANN's convergence to oscillate, where as a small one results in ANNs learning slowly, and even getting trapped in local optimal regions.

5.5 Hyper-Parameters Optimization

Training deep neural networks involves numerous choices for the commonly called "hyper-parameters". These include the number of layers, neurons, and the specific activation function, etc. First, determining the depth (the number of hidden layers) and the width (the number of neurons) of the ANN is a challenging problem. Once the MLP architecture is determined, the other hyper-parameters are optimized using automatic machine learning. There are different techniques to implement the automatic search. In a grid search technique, all candidate parameters are systematically parameterized on a pre-defined grid, and all possible candidates are explored in a brute-force way.
During this model selection process, over-fitting can be reduced by adopting the k-fold cross validation. In k-fold cross validation we split our data into k different subsets (or folds). We use $k-1$ subsets to train our data and leave the last subset (or the last fold) as test data. We then average the model against each of the folds and then finalize our model. After that we test it against the test set.

Data

The option prices dataset used in this thesis consists of a call European style option obtained by a data-driven approach. Theoretically, an arbitrary number of samples can be generated since the mathematical model is known, but the quality of the data has an important role in the training of the model and, therefore, the way of generating the data has an impact on the performance of the resulting model. A sampling technique with good space-filling properties should be preferable, so in this thesis Latin hypercube sampling (LHS) is used, in order to generate enough samples of the set of parameters involved in the Heston model, the reason for using LHS is that this technique is able to generate random samples of the parameter values from a multidimensional distribution, which results in a better representation of the parameter space. When sampling across the entire range, each variable has the opportunity to show up as important, if it indeed is important. If an input variable is not important, then the method of sampling is of little or no concern. When the sample data set for the input parameters is available, we select the appropriate numerical methods to generate the training results. For the Heston model, prices are calculated by applying the Fourier-based approach (Lewis, 2001), knowing the characteristic function of the stochastic processes that govern the evolution of the underlying, and by applying FDM with a non-uniform grid and using the Crank-Nicolson scheme.

In table 6.1, we list the range of the six input parameters of the Heston model $(r, \kappa, \theta, \sigma, \rho, \nu_0)$ as well as the two option contract-related parameters (τ, m), here m represents moneyness, and we take a fixed strike price, $K = 1$. We generate around one million data points through Latin hypercube sampling, to obtain the training dataset for the call price with Fourier-based approach, and around five thousand when applying the FDM, the latter because one of the advantages of the FDM is that the two-dimensional grid for (S, ν) is of size $N = (N_S + 1)(N_V + 1)$, at each t_n there are N possible values for $V_{i,j}^n$, as indicated in figure 4.2. Additionally, it is guaranteed that the $V_{i,j}^n$ values obtained correspond to the values of moneyness and initial variance that belong to the respective ranges shown in table 6.1, that is, the results obtained by finite differences were filtered so that the parameters are within the defined range. In this case, we have taken $N_S = 50$ and $N_V = 25$, so we have $1,326$ option prices and to keep the moneyness and the initial variance in the range defined in table 6.1 we have filtered the output prices following the rule: $0.6 < S_0 < 1.4$ and $0.05 < \nu_0 < 0.5$. After making the filter, we obtain around

200 values, for this reason we execute the FDM $5,000$ times to obtain a data set of around one million data points.
Once Heston prices are determined, the entire dataset is randomly divided into two groups, 80% will be training and 20% the test set.

ANN	**Parameters**	**Range**	**Method**
Input	Moneyness, $m = S_0/K$	$(0.6, 1.4)$	LHS
	Time to maturity, τ	$(0.1, 1.4)$ (year)	LHS
	Risk free rate, r	$(0.0\%, 10\%)$	LHS
	Correlation, ρ	$(-0.95, 0.0)$	LHS
	Reversion speed, κ	$(0.0, 2.0)$	LHS
	Long average variance, θ	$(0.0, 0.5)$	LHS
	Volatility of volatility, σ	$(0.0, 0.5)$	LHS
	Initial variance, ν_0	$(0.05, 0.5)$	LHS
Output	European call price, V	$(0, 0.66)$	Fourier Method

Table 6.1: The Heston parameter ranges for training the ANN.

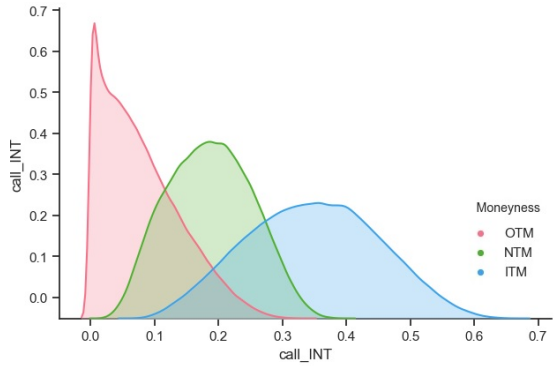
The following table show the data distribution from applying the two approaches.

	Moneyness distribution		**Initial variance distribution**		**Call value distribution**	
	Fourier-based approach	**FDM**	**Fourier-based approach**	**FDM**	**Fourier-based approach**	**FDM**
mean	1.000012	0.9961206	0.275007	0.2009332	0.2072096	0.1863310
std	0.230935	0.2054110	0.129901	0.1209586	0.1518449	0.1138063
min	0.600000	0.6000769	0.050000	0.0500433	8.882e-16	1.363e-10
25%	0.800019	0.8386924	0.162511	0.1042255	0.07264893	0.0684755
50%	1.000013	1.009769	0.275007	0.1814014	0.1820689	0.1631307
75%	1.200006	1.153146	0.387503	0.3153996	0.3281735	0.2843789
max	1.400000	1.399923	0.500000	0.4999567	0.6627153	0.6740285

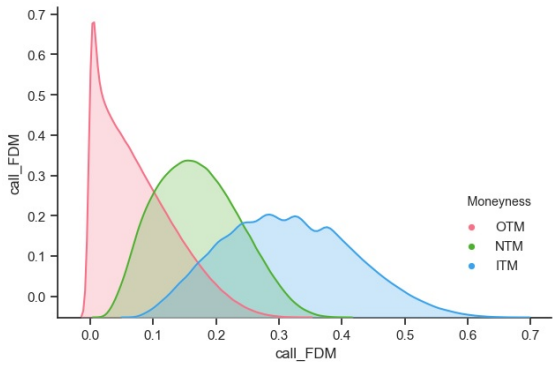
Table 6.2: Data distribution of moneyness, the initial variance and the call value.

The above table show only the distribution of moneyness, the initial variance and the call vale, the values on the left correspond to the call value obtained by applying the Fourier-based approach and the values on the right correspond to the call value obtained using FDM, it is seen the distribution is similar for each variable, although the values obtained by FDM are slightly lower than the values obtained by Fourier-based approach. The other distribution parameters are not shown, because the respective distribution is very similar. The latter because all the parameters of the Fourier-based approach are obtained through LHS and not all FDM parameters are obtained through LHS, but the stock price and the initial variance are obtained from the discretization in the construction of the non-uniform grid.

The following figures show the distribution of call values for the out-the-money (OTM), near-the-money (NTM) and in-the-money (ITM) call option observations, the moneyness criterion as follows: $(S/K) < 0.97$, $0.97 \leq (S/K) \leq 1.05$ and $(S/K) > 1.05$, respectively, values $m = 0.97$ and $m = 1.05$ were chosen as boundary values for the subset division, this is done in the same way as in the (Ramazan Gencay and Kukolj, 2008) article.



(a) Distribution of call values by moneyness Fourier-based approach

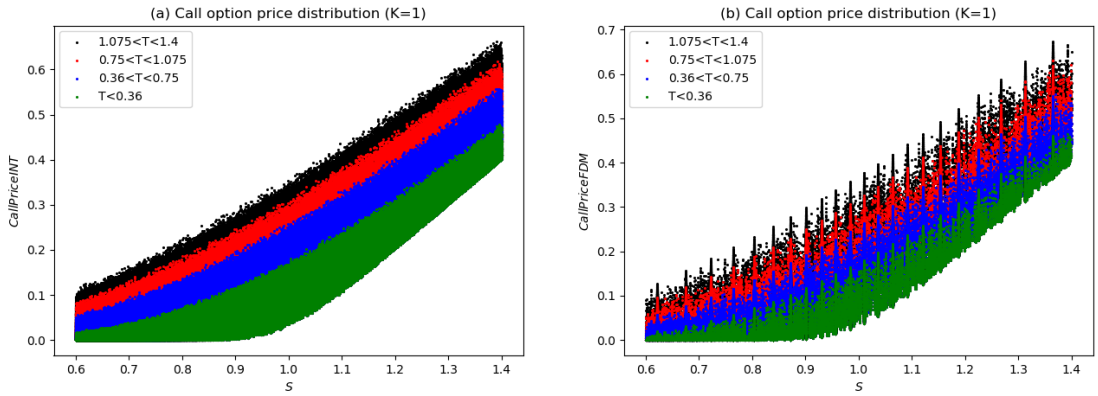


(b) Distribution of call values by moneyness FDM

The distribution is similar for the call values in subfigures (a) and (b), although the number of observations for ITM values, obtained by FDM are slightly less than the values obtained by the Fourier based approach. The following table shows the percentage of participation of moneyness for each data set.

Moneyness	**Fourier-based approach**		**FDM**	
	Nº	%	**Nº**	%
OTM	462,468	46.25%	488,503	45.43%
ITM	437,500	43.75%	441,672	41.08%
NTM	100,000	10.00%	145,080	13.49%
Total	999,968	100%	1,075,255	100%

In the following figures, the ramp functional form which is typical for call options is shown in the data structure. Moreover, the subfigures also depict the maturity value of the options as each individual colour distinguish the different time to maturity (in the following figures denoted as T).



In the above figure, the subfigures (a), (b) show the data for the call option prices which have strike price 1. Figure (a) corresponds to the call price options generated with Fourier method and figure (b) corresponds to call price option generated with FDM, the values $T = 0.36$, $T = 0.75$ and $T = 1.075$ were chosen as boundary values for the subset division of the options according to the time to maturity τ (in figures denoted T).

The difference between the distributions may be due to the fact that the parameters for the Fourier-based approach are obtained 100% with LHS and the parameters of finite difference are obtained with a combination of LHS and the parameters discretized in the grid.

It can be noticed, here the moneyness $m = (S/K)$ matches with S.

Empirical Analysis

7.1 Training Neural Networks

ANNs generally constitute three levels of components, i.e. neurons, layers and the architecture. The architecture is determined by a combination of different layers, that are made up of numerous artificial neurons, by connecting the neurons of adjacent layers, output signals of a previous layer enter a next layer as input signal. By stacking layers on top of each other, signals travel from the input layer through the hidden layers to the output layer potentially through cyclic or recurrent connections, and the ANN builds a mapping among input-output pairs.
We experimentally find that a MLP architecture with four hidden layers has an optimal capacity of approximating option pricing formulas of our current interest. Built on a four hidden layer architecture, the other hyper-parameters are optimized as follows (Algorithm 1)

Algorithm 1 k-fold cross validation

- Split the training data set into k different subsets,
- Select one set as the validation data set,
- Train the model on the remaining k-1 subsets,
- Calculate the metric by evaluating the trained model on the validation part,
- Continue the above steps by exploring all subsets,
- Calculate the final metric which is averaged over k cases,
- Explore the next set of hyper-parameters,
- Rank the candidates according to their averaged metric.

There are two stages to complete the hyper-parameter optimization. In the first stage, we use a random search combined with a 3-fold cross validation to find the initial hyper-parameter configurations for the neural network. As shown in Table 7.1, each model is trained following (Shuaiqiang Liu and Bohte, 2019), with 200 epochs using MSE as the loss metric. An epoch is the moment when the model has processed the whole training data set. It is found that the prediction accuracy increases with the training data set size growing (more related details will be discussed below). Therefore, the random search is implemented on a small data set (50,000 samples were selected by simple random sampling), and then the selected ANN is trained in larger data sets in the application.

Parameters	Options or Range
Activation	ReLU, Sigmoid
Neurons	[200,700]
Initialization	uniform, Glorot_uniform
Optimizer	SGD, Adam
Batch size	[256, 3000]

Table 7.1: The setting of random search for hyper-parameters optimization.

Parameters	Options
Hidden layers	4
Activation	ReLU
Neurons	700-350-233-175
Initialization	Glorot_uniform
Optimizer	Adam
Batch size	256

Table 7.2: The selected model after the random search.

In the second stage, we tried to train the model with a standardized data set, because it is almost always a good practice to standardize the data when we compare measurements that have different units.
Before modeling it using a neural network model, we test the model's performance by increasing the size of the data set of training.

We use scikit-learn's Pipeline framework, which is a procedure that allows performing a series of different transformations on a dataset before modeling, to perform the standardization during the model evaluation process, with 5-fold of cross validation and using $50,000$ samples selected by simple random sampling of the train set. The table 7.3 shows the results for each fold and the average performance.

Dataset	MSE (5-Fold)	Mean (MSE)	std (MSE)
No standardized dataset	1.9180e-05, 2.3101e-05, 2.9004e-05, 4.4750e-04, 2.4739e-05	1.0871e-04	1.6943e-04
Standardized dataset	1.9180e-05, 2.3101e-05, 2.9004e-05, 4.4750e-04, 2.4739e-05	3.7171e-05	1.3813e-05

Table 7.3

Running the neural network with a standardized data set provides improved performance over the baseline model without standardized data, which reduces the error.

In order to investigate the relation between the prediction accuracy and the size of the training set, we increase the number of training samples from 1/8 to 16 times the baseline set, as shown in Table 7.4.

Case	1	2	3	4	5	6	7	8
Training size (×25.000)	1/8	1/4	1/2	1	2	4	8	16

Table 7.4: The different sizes of training data set when training the ANN.

Meanwhile, the test data is kept unchanged. The example here is learning the Heston call value by applying the Fourier-based approach. We first train the ANN on each data set separately using an the architecture described in Table 7.2., and repeat the training stage for each case 3 times to obtain the accuracy metric (MSE) as an average of the performance of the three models. As shown in Figure 7.1, with an increasing data size, the prediction accuracy increases and the gap between the MSE obtained in the training and test set decreases. The training and validation losses remain close, which indicates that there is no over-fitting.

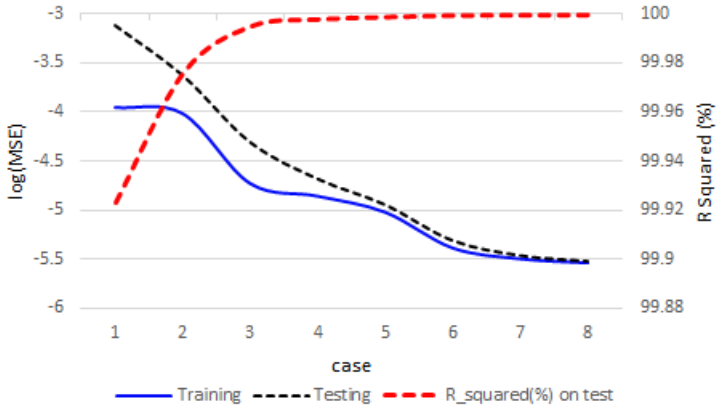


Figure 7.1: R^2 and MSE vs. size of the training set. (R^2 on test set). Source: own elaboration

7.2 Numerical Results

We show the performance of the ANNs for solving the financial models, based on the following accuracy metrics (which forms the basis for the training),

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2,$$

where y_i is the actual value and $\hat{y}_i$ is the ANN predicted value. For completeness, however, we also report the other well-known metrics,

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|,$$

$$MAPE = \frac{1}{N}\sum_{i=1}^{N}\left|\frac{y_i - \hat{y}_i}{y_i}\right|$$

The MSE is used as the training metric to update the weights, and all above metrics are employed to evaluate the selected ANN.
The following tables show the ANN performance for two approaches, both Fourier base approach and FDM ANN solver has been well trained, avoiding over-fitting and approximating the prices accurately.

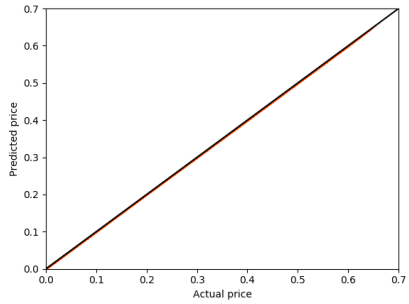
Fourier - based approach	**MSE**	**MAE**	**MAPE**	R^2
Training	1.4178e-07	9.30e-05	8.41e-04	0.99999858
Testing	1.4495e-07	9.39e-05	6.46e-04	0.99999855

Table 7.5: The trained Fourier - based approach - ANN performance.

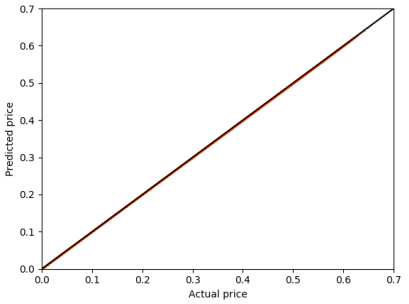
Finite Difference Method	**MSE**	**MAE**	**MAPE**	R^2
Training	9.2651e-07	2.05e-04	1.70e-03	0.99999073
Testing	1.0082e-06	2.09e-04	1.68-03	0.99998993

Table 7.6: The trained FDM - ANN performance.

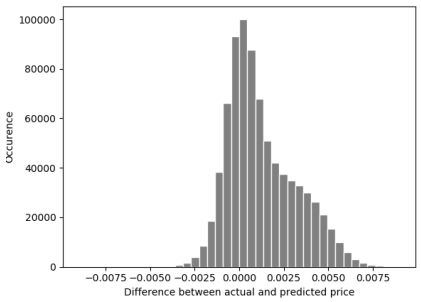
In order to demonstrate the accuracy of the prediction by our choices of architecture, we represent the prediction measures in the following figures. Both figures show that the neural network is very successful in learning the correct functional form and that its expected price almost coincides with the actual price obtained by applying the respective scheme to generate the data set. In both figures, four subfigures are shown that show the prediction of the value of a call option and the histogram of the difference between the actual and expected price, both the train and test set.
Figure 7.2 shows both the train and test sample performance of the ANN trained in the data set generated by Fourier-based scheme. The difference between the actual and expected price distribution shows a small deviations of the actual prices, where the maximum deviation is around 7×10^{-3}, and most of values of call option are equal their actual values.
Figure 7.3 shows both the train and test sample performance of the ANN trained in the data set generated by FDM. The difference between the actual and expected price distribution approximately follows a normal distribution, and most of values of call option are equal their actual values.



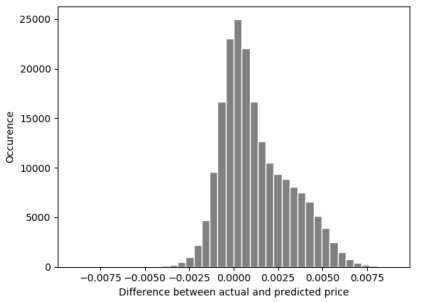
(a) Prediction precision - Train set



(b) Prediction precision - Test set

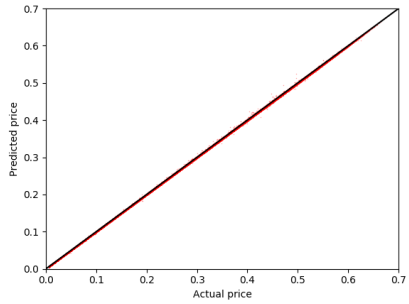


(c) Histogram of difference - Train set

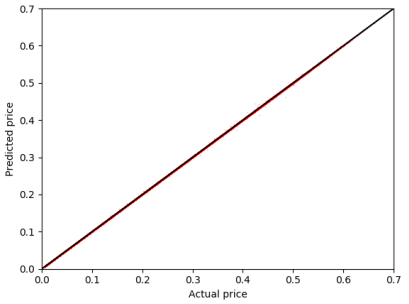


(d) Histogram of difference - Test set

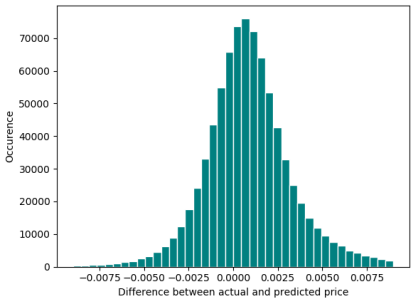
Figure 7.2: Call options price prediction for network with data generated by Fourier - based approach



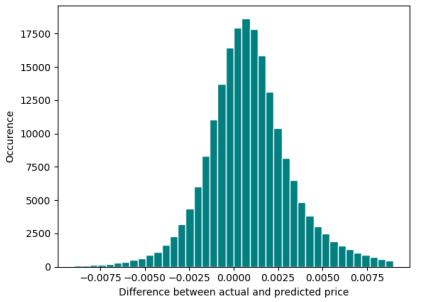
(a) Prediction precision - Train set



(b) Prediction precision - Test set



(c) Histogram of difference - Train set



(d) Histogram of difference - Test set

Figure 7.3: Call options price prediction for network with data generated by Finite Difference Method

Conclusion

- In this thesis we have proposed an Artificial Neural Network(ANN) approach to calculate the price of a European Call Option with the Heston stochastic volatility model, in order to accelerate the corresponding numerical methods and show the ability of Artificial Neural Network to "learn" the model from the dataset. We test the ANN approach on two different solvers, including valuation using the Fourier-based approach and using finite difference methods. Our numerical results show that the ANN can compute option prices efficiently and accurately in a robust way.

- One of the advantages of the FDM is that the two-dimensional grid for (S, ν) is of size $N = (N_S + 1)(N_V + 1)$, at each t_n there are N possible values for $V_{i,j}^n$, this leads to a lower computational cost since in this thesis the finite difference method algorithm had to be run $5,000$ times compared to one million for the Fourier-based approach algorithm.

- We have shown that the sampling technique plays an important role in the implementation of an ANN and has an impact on the performance of the ANN model. In this thesis we used LHS as a sampling technique, and it was shown that the network performance was better with the data obtained by Fourier-based approach than with the data obtained by FDM (tables 7.5 and 7.6), due to the fact that the parameters for the Fourier-based approach were obtained with LHS only, and the parameters of finite difference are obtained with a combination of LHS and the discretization on the non-uniform grid.

- We have shown that the size of a training set has an impact on the performance of the ANN model. With an increasing data size, the prediction accuracy increases and the gap between the MSE obtained in the training and test set decreases.

8.1 Future Research

Although we focus on European call options in this work, it should be possible to extend the approach to pricing more complex options, like American, Bermuda or exotic options.
Additionally, in a future work might change the configuration of the Neural Network (i.e the number of hidden layers and the number of neurons in the hidden layer and also the type of transfer function for a feed-forward NN, so that error become minimum. Also the model accuracy can be further improved, for example, by using deeper neural networks, more complex NN architectures.
Regarding to the use of a non-uniform grid in the FDM, the possibility of better adapting the combination of LHS and the discretization of parameters included in the non-uniform grid could be evaluated to obtain a better distribution of the parameters and, and therefore obtain better performance of the ANN.

Bibliography

I. J. Cox, J.C. and S. Ross. A theory of the term structure of interest rates. Econometrica, 53(2):385–408., 1985.

G. Cybenko. Approximation by superpositions of a sigmoidal function. Mathematics of Control, Signals and Systems, 1989.

D. N. Eric Chin and S. Ólafsson. Problems and solutions in mathematical finance (volume 1: Stochastic calculus). John Wiley & Sons, Ltd, 2014.

J. Gatheral. The volatility surface: A practitioner's guide. John Wiley & Sons, Ltd, 2006.

P. Gupta. Topics in laplace and fourier transforms. Laxmi Publications Pvt.Ltd., 2019.

S. L. Heston. A closed-form solution for options with stochastic volatility with applications to bond and currency options. Review of Financial Studies, 6:327–43., 1993.

K. Hout and S.Foulon. Adi finite difference schemes for option pricing in the heston model with correlation. International Journal of Numerical Analysis and Modeling, 7(2):303–320., 2010. URL http://www.math.ualberta.ca/ijnam/Volume-7-2010/No-2-10/2010-02-06.pdf.

T. Kluge. Pricing derivatives in stochastic volatility models using the finite difference method. Web page, 2002. URL https://pdfs.semanticscholar.org/d6d5/21334abb537929fe200568c537a6c330f84d.pdf?_ga=2.168673697.1428853933.1562967970-1817431915.1562768618.

A. L. Lewis. A simple option formula for general jump-diffusion and other exponential lévy processes. Envision Financial Systems and OptionCity.net, 2001.

N. G. Ramazan Gencay and D. Kukolj. Option pricing with modular neural networks. JEL No. C45; G12, 2008. URL https://www.researchgate.net/publication/24188373_Option_Pricing_With_Modular_Neural_Networks.

F. D. Rouah. The heston model and its extensions in matlab and c#. John Wiley & Sons, Inc., Hoboken, New Jersey, 2013.

C. W. O. Shuaiqiang Liu and S. M. Bohte. Pricing options and computing implied volatilities using neural networks. Risks, 7(1) (2019), 2019. URL https://www.mdpi.com/2227-9091/7/1/16/pdf.

Appendix A: Python scripts FDM

make_grid: Generate non-uniform grid.

```
import matplotlib.pyplot as plt
from math import sinh, asinh
import numpy as np


def g(xi, k, c): return k + c * sinh(xi)


def h(xi, d): return d * sinh(xi)


def make_grid(ns, s_max, s0, k, c, nv, v_max, v0, d):
    delta_zeta_i = (1.0 / ns) * (asinh((s_max - k) / c) - asinh
        (-k / c))
    zeta_s = [asinh(-k / c) + i * delta_zeta_i for i in range(ns
        + 1)]
    vec_s = [g(zeta_s[i], k, c) for i in range(ns + 1)]
    vec_s.append(s0)
    vec_s.sort()
    vec_s.pop(-1)
    delta_s = [vec_s[i + 1] - vec_s[i] for i in range(ns)]

    delta_eta = (1.0 / nv) * asinh(v_max / d)
    eta_v = [i * delta_eta for i in range(nv + 1)]
    vec_v = [h(eta_v[i], d) for i in range(nv + 1)]
    vec_v.append(v0)
    vec_v.sort()
    vec_v.pop(-1)
    delta_v = [vec_v[i + 1] - vec_v[i] for i in range(nv)]

    x, y = np.meshgrid(vec_s, vec_v)

    # grid checking
    plt.plot(x, y, '.', color='blue')
    plt.show()

    return vec_s, delta_s, vec_v, delta_v, x, y
```

Approximation to derivatives

```
# Central approximation of derivatives, inner points
def central_coefficients1(i, index, delta):  # V'
    if index == -1:
        return -delta[i + 1] / (delta[i] * (delta[i] + delta[i +
            1]))
    elif index == 0:
        return (delta[i + 1] - delta[i]) / (delta[i] * delta[i +
            1])
    elif index == 1:
        return delta[i] / (delta[i + 1] * (delta[i] + delta[i +
            1]))


def central_coefficients2(i, index, delta):  # V''
    if index == -1:
        return 2 / (delta[i] * (delta[i] + delta[i + 1]))
    elif index == 0:
        return -2 / (delta[i] * delta[i + 1])
    elif index == 1:
        return 2 / (delta[i + 1] * (delta[i] + delta[i + 1]))


# Left hand side approximation of derivatives, right border
def backward_coefficients(i, index, delta):  # V'
    if index == -2:
        return delta[i] / (delta[i - 1] * (delta[i] + delta[i -
            1]))
    elif index == -1:
        return (-delta[i] - delta[i - 1]) / (delta[i] * delta[i
            - 1])
    elif index == 0:
        return (2 * delta[i] + delta[i - 1]) / (delta[i] * (
            delta[i] + delta[i - 1]))


# Right hand side approximation of derivatives, left border
def forward_coefficients(i, index, delta):  # V'
    if index == 0:
        return (-2 * delta[i + 1] - delta[i + 2]) / (delta[i +
            1] * (delta[i + 1] + delta[i + 2]))
    elif index == 1:
        return (delta[i + 1] + delta[i + 2]) / (delta[i + 1] *
            delta[i + 2])
    elif index == 2:
        return -delta[i + 1] / (delta[i + 2] * (delta[i + 1] +
            delta[i + 2]))
```

Make Matrices

```
def make_matrices(ns, nv, rho, sigma, r_d, kappa, eta, vec_s,
    vec_v, delta_s, delta_v):
    n = (1 + ns)*(1 + nv)
    a_0, a_1, a_2 = np.zeros((n, n)),  np.zeros((n, n)), np.
        zeros((n, n))

    # Definition of a_0
    for j in range(1, nv):
        for i in range(1, ns):
            c = rho * sigma * vec_s[i] * vec_v[j]
            for k in [-1, 0, 1]:
                for l in [-1, 0, 1]:
                    a_0[i + j * (ns + 1), (i + k) + (j + l) * (
                        ns + 1)] += c * central_coefficients1(i -
                         1, k, delta_s)\
                         * central_coefficients1(j - 1, l,
                            delta_v)

    a_0 = csc_matrix(a_0)

    # Definition of a_1
    for j in range(nv + 1):
        for i in range(1, ns):
            b = r_d * vec_s[i]
            c = 0.5 * vec_s[i] ** 2 * vec_v[j]
            for k in [-1, 0, 1]:
                a_1[i + j * (ns + 1), (i + k) + j * (ns + 1)] +=
                     (c * central_coefficients2(i - 1, k, delta_s
                    ) +  b * central_coefficients1(i - 1, k,
                    delta_s))
            a_1[i + j * (ns + 1), i + j * (ns + 1)] += - 0.5 *
                r_d
        a_1[ns + j * (ns + 1), ns + j * (ns + 1)] += - 0.5 * r_d

    a_1 = csc_matrix(a_1)

    # Definition of a_2
    for j in range(nv - 1):
        for i in range(ns + 1):
            d = kappa * (eta - vec_v[j])
            e = 0.5 * sigma ** 2 * vec_v[j]
            if vec_v[j] > 1.:
                for k in [-2, -1, 0]:
                    a_2[i + (j + 1) * (ns + 1), i + (ns + 1) * (
                        j + 1 + k)] +=\
                         d * backward_coefficients(j, k, delta_v)
                for k in [-1, 0, 1]:
                    a_2[i + (j + 1) * (ns + 1), i + (ns + 1) * (
                        j + 1 + k)] += \
                         e * central_coefficients2(j - 1, k,
                            delta_v)
            if j == 0:
                for k in [0, 1, 2]:
                    a_2[i, i + (ns + 1) * k] += d *
```

```
                        forward_coefficients(j, k, delta_v)
            else:
                for k in [-1, 0, 1]:
                    a_2[i + j * (ns + 1), i + (ns + 1) * (j + k)
                        ] += \
                        (d * central_coefficients1(j - 1, k,
                            delta_v) +
                         e * central_coefficients2(j - 1, k,
                             delta_v))
            a_2[i + j * (ns + 1), i + j * (ns + 1)] += - 0.5 *
                r_d

    a_2 = csc_matrix(a_2)

    a = a_0 + a_1 + a_2
    a = csc_matrix(a)

    return a
```

Make boundaries

```
def make_boundaries(ns, nv, r_d, vec_s):
    n = (1 + ns)*(1 + nv)
    b_0, b_1, b_2 = [0.] * n, [0.] * n, [0.] * n

    # Boundary when s = s_max
    for j in range(nv + 1):
        b_1[ns * (j + 1)] = r_d * vec_s[-1]

    # Boundary when v = v_max
    for i in range(1, ns + 1):
        b_2[n - ns - 1 + i] = -0.5 * r_d * vec_s[i]

    b_0 = np.array(b_0)
    b_1 = np.array(b_1)
    b_2 = np.array(b_2)

    b = b_0 + b_1 + b_2

    return b
```

Crank-Nicolson Scheme

```
def f(omega, l, b):
    return l * omega + b


def cn_scheme(n, n_tm, u_0, delta_t, l, b):
    u = u_0
    identity = np.identity(n)
    lhs = csc_matrix(identity - 0.5 * delta_t * l)
    inv_lhs = inv(lhs)
    for i in range(1, n_tm + 1):
        u = inv_lhs * (u + 0.5 * delta_t * (f(u, l, b) + b))

    return u
```

FDM

```
def fdm(moneyness, time_to_maturity, risk_free_rate, reversion_speed,
        long_average_variance, vol_of_vol, correlation, initial_variance):
    # parameters
    K = 1
    S_0 = moneyness
    S = 8 * K
    V_0 = initial_variance
    V = 5.
    T = time_to_maturity

    r_d = risk_free_rate  # domestic interest rate
    rho = correlation
    sigma = vol_of_vol
    kappa = reversion_speed
    eta = long_average_variance

    # grid [0, S] x [0, V]
    ns = 50  # S
    nv = 25  # V
    n = (ns + 1) * (nv + 1)  # matrix A and vector U size
    c = K / 5
    d = V / 500

    # line [0, T]
    N = 20
    delta_t = T / N

    # model setup
    Vec_s, Delta_s, Vec_v, Delta_v, X, Y = make_grid(ns, S, S_0,
        K, c, nv, V, V_0, d)

    A = make_matrices(ns, nv, rho, sigma, r_d, kappa, eta, Vec_s
       , Vec_v, Delta_s, Delta_v)
    B = make_boundaries(ns, nv, r_d, N, Vec_s, delta_t)

    # pricing

    UU_0 = np.array([[max(Vec_s[i] - K, 0) for i in range(ns +
       1)] for _ in range(nv + 1)])
    U_0 = UU_0.flatten()

    price_cn = cn_scheme(n, N, U_0, delta_t, A, B)
    price_cn = np.reshape(price_cn, (nv + 1, ns + 1))

    return Vec_v, Vec_s, price_cn
```

Appendix B: Python scripts ANN

Baseline Neural Network Model

```
glorot_uniform = keras.initializers.glorot_uniform(seed=None)


def build_model(inp_size, hidden_layers=1, nodes=50):

    model = Sequential()
    model.add(Dense(nodes, input_dim=inp_size,
                    kernel_initializer=glorot_uniform,
                       activation='relu'))
    if hidden_layers > 1:
        for i in range(hidden_layers - 1):
            nodes_ = int(nodes/(i+2))
            # nodes_ = int(nodes - 100 * (i + 1))
            model.add(Dense(nodes_, kernel_initializer=
               glorot_uniform, activation='relu'))

    model.add(Dense(1, kernel_initializer='normal'))

    model.compile(loss='mean_squared_error', optimizer='adam',
       metrics=['mean_squared_error'])
    return model
```

Standardized datased

```
def get_dataset(trainX, trainy, testX, testy):
    # fit scaler
    SScaler = StandardScaler()
    SScaler.fit(trainX)
    # transform training dataset
    trainX = SScaler.transform(trainX)
    # transform test dataset
    testX = SScaler.transform(testX)
    SScaler.fit(trainy)
    # transform training dataset
    trainy = SScaler.transform(trainy)
    # transform test dataset
    testy = SScaler.transform(testy)
    return trainX, trainy, testX, testy
```

Train neural network

```
# Split into training and test set
X = df.drop(columns=['call_value'])
Y = df.call_value.ravel()

# Split into training and test set
X_train, X_test, y_train, y_test =
train_test_split(X, Y, test_size=0.20, random_state=123)

# Standardized data
trainX, trainY, testX, testy =
get_dataset(X_train, pd.DataFrame(y_train), X_test,
            pd.DataFrame(y_test))

# create model
def model1():
    return build_model(8, hidden_layers=4, nodes=700)

model = model1()

# Fit the model
model.fit(trainX, trainY, epochs=200, batch_size=256, verbose=1)
```