Contents

Abstract **4**

Introduction **5**

1 Option Pricing and Asset Models **6**

1.1 The Black-Scholes Equation 6

1.1.1 Model Assumptions 6

1.1.2 Mathematical model 7

1.1.3 Implied Volatility 8

1.2 The Heston Model 9

1.2.1 Mathematical Model 10

1.2.2 Techniques for price approximation 11

1.3 Implied Volatility 12

1.3.1 Bisection Method 12

1.3.2 Newton-Raphson Method 12

1.3.3 Brent Method 13

1.4 Artificial Neural Networks 14

1.4.1 Structure of ANNs 15

1.4.2 Multi-Layer Perceptron (MLP) 15

2 Implementation **17**

2.1 Black and Scholes model implementation 17
2.1.1 Dataset generation and analysis 17
2.1.2 Neural Networks Parameters 20
2.2 Heston model implementation 22
2.2.1 Dataset generation and analysis 22
2.2.2 Neural Networks Parameters 22

3 Results 23
3.1 Metrics results from model training (mae, mse, mape and score) 23
3.1.1 Metrics meaning 23
3.1.2 Metrics values 24
3.2 Model loss graph 24
3.3 Performance metrics for Heston-ANN model 25
3.4 error distribution on the wide test datasets 26
3.4.1 Error distribution for BS-ANN 26
3.4.2 Error distribution for IV-ANN 26
3.4.3 Error distribution for Heston-IV-ANN 27

4 Conclusion 28

Abstract

Neural networks are playing an increasingly vital role in the financial market, offering solutions to challenges of stationarity and non-linearity while also providing robustness and predictive power. Options and their pricing are crucial areas of focus for investment banks, hedge funds, and trading firms. A key element in these activities is implied volatility, an essential parameter that offers a numerical estimation of risk, forming the foundation for modeling and risk management. Various numerical algorithms are used to determine implied volatility from given data sets. However, these algorithms often involve trade-offs between convergence, robustness, and computational efficiency. Using artificial neural networks to determine implied volatility aims to overcome these issues, offering a more suitable, stable, and computationally efficient method.

Keywords: Option Pricing, Implied Volatility, Artificial Neural Network, Black-Scholes, Heston.

Introduction

Risk management is a crucial objective for any participant in financial markets, and a significant part of this involves hedging positions. The introduction of the Black-Scholes model brought mathematical rigor to option pricing, enabling a new way to trade risk. However, accurate option pricing remains a critical component of any financial market model, balancing the need for precision with computational speed. The Black-Scholes model has persisted despite numerous attempts to enhance it, as many improvements aimed at increasing accuracy often compromise pricing speed.

One notable extension to the Black-Scholes model is the relaxation of the restrictive assumption of constant volatility, allowing volatility to be defined as a stochastic process. This leads to more complex models like the Heston model, which also assumes stochastic volatility but lacks analytical solutions for many options, necessitating the use of iterative approximation methods such as finite difference methods or Monte Carlo simulations. These methods, while potentially more accurate, often incur substantial computational costs, negating their benefits.

In this report, we explore the use of neural networks to address these computational challenges. Artificial neural networks (ANNs) offer solutions to the problems of stationarity and non-linearity in financial markets, providing robustness and predictive power. By training neural networks on data generated from sophisticated financial models, we can leverage their ability to approximate complex functions quickly and efficiently.

We begin by introducing stochastic modeling for financial markets in chapter 1, delving into the mathematical concepts of the Black-Scholes and Heston models. Chapter 2 offers an introduction to neural networks and highlights important properties relevant to our work. In chapter 3, we discuss the implementation of neural networks in option pricing and the computation of implied volatility. Our goal is to demonstrate how neural networks can mitigate computational costs while maintaining accuracy, thus enhancing the efficiency of option pricing models.

Chapter 1

Option Pricing and Asset Models

Options are flexible financial contracts that come in various forms; however, this report will focus on the primary and most common options contracts, namely vanilla options, which are available to all investors. The basis of these contracts is as follows: The buyer of the option has the contractual right, but not the obligation, to buy or sell an underlying asset at a specified strike price during a specified date range. Conversely, the seller has the obligation to honor this contract if the owner of the option exercises their right. In each of these scenarios, an investor could speculate that the price of an underlying asset will increase or decrease during a certain time range. Thus, the investor could choose to invest in a put or a call option, betting that the stock will decrease or increase, respectively. There are many models used to price these derivatives, the most famous of which are the Black-Scholes and Heston models.

1.1 The Black-Scholes Equation

The Black–Scholes model is a mathematical description of financial markets and derivative investment instruments. The model develops partial differential equations whose solution, the Black–Scholes formula, is widely used in the pricing of European-style options [2].

1.1.1 Model Assumptions

The Black–Scholes model of the market for a particular equity makes the following explicit assumptions:

- It is possible to borrow and lend cash at a known constant risk-free interest rate.
- The stock price follows a geometric Brownian motion with constant drift and volatility.
- There are no transaction costs, taxes or bid-ask spread.
- The underlying security does not pay a dividend.
- All securities are infinitely divisible (i.e., it is possible to buy any fraction of a share)
- There are no restrictions on short selling.
- There is no arbitrage opportunity.

From these conditions, it is possible to create a hedged position, consisting of a long position in the stock and a short position in (calls on the same stock), whose value will not depend on the price of the stock.

1.1.2 Mathematical model

Per the model assumptions above, we assume that the price of the underlying asset (typically a stock) follows a geometric Brownian motion. That is:

$$dS_t = \mu S_t dt + \sigma S_t dW \tag{1.1}$$

Where:

- W: Brownian. The term dW here stands in for any and all sources of uncertainty in the price history of the stock.
- S_t: Stock price.
- μ: The drift rate of S_t, annualized.
- σ: The volatility of the stock's returns.
- t: Time in years; we generally use: now = 0, expiry = T.

A European option contract on the underlying stock price can be valued via the Black-Scholes PDE, which can be derived from Itô's Lemma under a replicating portfolio approach or via the martingale approach. Denoting the option price by $V(t,S)$, the Black-Scholes equation reads:

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS\frac{\partial V}{\partial S} - rV = 0 \tag{1.2}$$

with time t until to maturity T, and r the risk-free interest rate. The PDE is accompanied by a final condition representing the specific payoff, for example, the European call option payoff at time T

$$V(t = T, S) = (S_0 - K)^+ \tag{1.3}$$

where K is the option's strike price.

An analytic solution to 1.2, 1.3 exists for European plain vanilla options:

$$V_c(t,S) = SN(d_1) - Ke^{-r\tau}N(d_2) \tag{1.4}$$

Where:

$$d_1 = \frac{log(S/K) + (r - 0.5\sigma^2\tau}{\sigma\sqrt{\tau}} \tag{1.5}$$

$$d_2 = d_1 - \sigma\sqrt{\tau} \tag{1.6}$$

And $\tau := T - t$, $V_c(t,S)$ is the European call option value at time t for stock value S, and $N()$ represents the normal distribution. This solution procedure is denoted by $V() = BS()$.

1.1.3 Implied Volatility

Implied volatility is considered an important quantity in finance. Given an observed market option price V^{mkt}, the Black-Scholes implied volatility σ^* can be determined by solving:

$$BS(\sigma^*;S;K;\tau;r) = V^{mkt} \tag{1.7}$$

The monotonicity of the Black-Scholes equation for the volatility guarantees the existence of $\sigma^* \in [0,+\infty))$. We can write the implied volatility as an implicit formula:

$$\sigma^*(K,T) = BS^{-1}(V^{mkt};S;K;\tau;r) \tag{1.8}$$

Where BS^{-1} denotes the inverse Black-Scholes function. Moreover, by adopting moneyness, $m = \frac{S_t}{K}$, and time to maturity $\tau = T - t$, one can express the implied volatility as $\sigma^*(m,\tau)$. For simplicity, we denote here $\sigma^*(m,\tau)$ by σ^*. An analytic solution for Equation 1.8 does not exist. The value of σ^* is determined by means of a numerical iterative technique, since Equation 1.8 can be converted into a root-finding problem:

$$g(\sigma^*) = BS(S,\tau,K,r,\sigma^*) - V^{mkt}(S,\tau,K) = 0 \tag{1.9}$$

This will form the basis of the iterative techniques outlined in the following sections.

1.2 The Heston Model

One of the limitations of using the Black-Scholes model is the assumption of a constant volatility σ. A major modeling step away from the assumption of constant volatility in asset pricing was made by modeling the volatility/variance as a diffusion process. The resulting models are the stochastic volatility models. The idea to model volatility as a random variable is confirmed by practical financial data which indicates the variable and unpredictable nature of the stock price's volatility. The most significant argument to consider the volatility to be stochastic is the implied volatility smile/skew, which is present in the financial market data and can be accurately recovered by SV models, especially for options with a medium to long time to the maturity date T. With an additional stochastic process, which is correlated with the asset price process S_t, we deal with a system of SDEs, for which option valuation is more computationally expensive than for a scalar asset price process. The most popular stochastic volatility model is the Heston model.

1.2.1 Mathematical Model

In 1993, Steven Heston proposed the following formulas to describe the movement of asset prices, where an asset's price and volatility follow random, Brownian motion processes:

$$dS_t = rS_t dt + \sqrt{v_t} S_t dW_t^s, S_{t_0} = S_0, \tag{1.10}$$

$$dv_t = \kappa(\bar{v} - v_t)dt + \gamma\sqrt{v_t} dW_t^v, v_{t_0} = v_0, \tag{1.11}$$

$$dW_t^s dW_t^v = \rho dt \tag{1.12}$$

The variables of the system are defined as follows:

- S_t: the asset price at time t
- r: risk-free interest rate - the theoretical interest rate on an asset that carries no risk.
- $\sqrt{V_t}$: volatility (standard deviation) of the asset price.
- σ: volatility of the volatility $\sqrt{V_t}$
- $\bar{v}$: long-term price variance.
- κ: rate of reversion to the long-term price variance.
- dt: indefinitely small positive time increment
- W_t^s: Brownian motion of the asset price
- W_t^v: Brownian motion of the asset's price variance.
- ρ: Correlation coefficient for W_t^s and W_t^v
- v_0: the t_0-value of the variance.

By the martingale approach, we arrive at the following multi-dimensional Heston option pricing PDE:

$$\frac{\partial V}{\partial t} + rS\frac{\partial V}{\partial S} + \kappa(\bar{v} - v)\frac{\partial V}{\partial v} + \frac{1}{2}vS^2\frac{\partial^2 V}{\partial S^2} + \rho\gamma Sv\frac{\partial^2 V}{\partial S \partial v} + \frac{1}{2}\gamma^2 v\frac{\partial^2 V}{\partial v^2} - rV = 0 \tag{1.13}$$

The typically observed implied volatility shapes in the market, e.g., smile or skew, can be reproduced by varying the above parameters $\{\kappa, \rho, \gamma, v_0, \bar{v}\}$. In general, the parameter v_0 impacts the kurtosis of the asset return distribution, and the coefficient ρ controls its asymmetry. The Heston model does not have analytic solutions and is thus solved numerically.

1.2.2 Techniques for price approximation

There are mainly three different techniques for approximating the prices stated by the Heston Stochastic Volatility model.

- **Finite differences (FD)** Finite differences for the PDE problem are often used for free boundary problems, as they occur when valuing American options, or for certain exotic options like barrier options. The derivatives of the option prices (the so-called option Greeks) are accurately computed with finite differences.

- **Monte Carlo methods, MCM:** Monte Carlo simulation and numerical integration rely on the Feynman-Kac Theorem, which essentially states that (European) option values can be written as discounted expected values of the option's payoff function at the terminal time T, under the risk-neutral measure. Monte Carlo methods are often employed in this context for the valuation of path-dependent high-dimensional options, and also for the computation of all sorts of valuation adjustments in modern risk management. However, Monte Carlo methods are typically somewhat slow to converge, and particularly in the context of model calibration this can be an issue.

- **Different Fourier transformation techniques:** The availability of the asset price's characteristic function is a pre-requisite to using Fourier techniques. One of the efficient techniques in this context is the COS method, which utilizes Fourier-cosine series expansions to approximate the asset price's probability density function. The COS method can be used to compute European option values under the Heston model highly efficiently. However, for many different, modern asset models the characteristic function is typically not available.

1.3 Implied Volatility

1.3.1 Bisection Method

The bisection method is applied to continuous functions where intervals are repeatedly bisected into two subintervals, selecting the one whose value changes sign and must therefore contain the root. Hence, this method is the simplest alternative and provides stability and robustness at a cost of computational efficiency. More formally, the bisection method makes use of the intermediate value theorem, which states that a continuous function f whose domain contains an interval $I = [a,b]$, will take on any of the given values between $f(a)$ and $f(b)$. Thus, a corollary of Balzano's Theorem states, that if the values of a continuous function change sign between that interval, then a function has a root in that interval. To determine these roots, the bisection method algorithm is implemented by an iterative procedure as follows:

1. Choose a suitable interval $I = [a,b]$ and choose two starting values $[a_0, b_0$ as a sub-interval within those values which satisfy $f(a_0)f(b_0) < 0$.
2. Determine m_0, the midpoint of that interval, and the function value at this point $f(m_0)$.
3. Choose a suitable tolerance ε where if $|a_n - b_n| < \varepsilon$, $0 \leq n \leq N$, the iterative process will break and the value of m_n will be returned.
4. If the tolerance is not breached, evaluate the next sub-interval $[a_{n+1}, b_{n+1}]$ where if:
   (a) $f(a_n)f(m_n) < 0$, set $a_{n+1} = a_{n+1}$ and $b_{n+1} = m_n$
   (b) $f(b_n)f(m_n) < 0$, set $a_{n+1} = m_n$ and $b_{n+1} = b_n$
5. Iterate the above process N times or until the tolerance ε is breached and return the value of m_n, $n \leq N$.

This method is robust at determining an accurate approximation of the roots of $f(x)$ yielding an absolute error of :

$$|m_{true} - m_n| \leq \frac{b-a}{2^{n+1}} \tag{1.14}$$

1.3.2 Newton-Raphson Method

The most popular alternative is to use the Newton-Raphson (NR) method due to its computational efficiency, though it does rely on the derivative of the function to exist. Consider a

differentiable function $f(x)$ and choose a point x_0 which is near the solution to the equation $f(x_0) = 0$. If this point is suitable, it is possible to approximate the value of the function by drawing a tangent to this line which is expressed as follows:

$$y = f'(x_0)(x - x_0) + f(x_0) = 0 \tag{1.15}$$

The x-intercept is thus the solution to the equation :

$$f'(x_0)(x_1 - x_0) + f(x_0) = 0 \rightarrow x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} \tag{1.16}$$

By iteration and recursion, this equation can then be generalized as follows:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{1.17}$$

which converges very quickly provided the derivative to the function is sufficiently large.

It must be noted that the derivative in question when utilizing the NR-method is a popular option metric denoted as Vega ν. Vega measures the rate of change of the options price for volatility and in some cases, if Vega is sufficiently small the NR-algorithm will not converge. This will occur when the slope of the line is flat, in regions that are deep ITM and OTM.

1.3.3 Brent Method

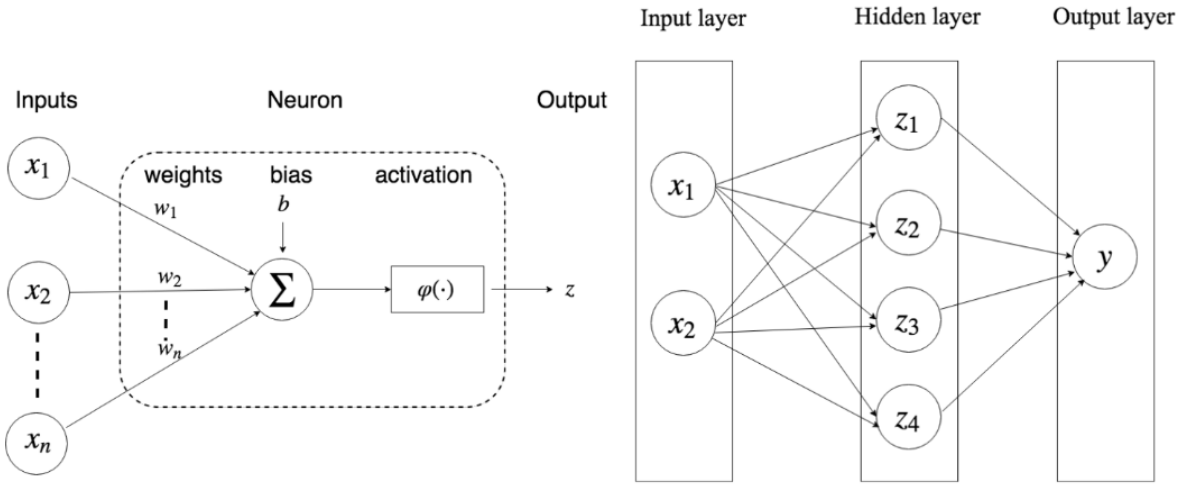
The Brent method is very popular from a practical point of view as it provides the most suitable combination of robustness and computational efficiency. The method combines the Secant and bisection methods with a few additional features such as an inverse quadratic method, approximating the function with an inverse quadratic function instead of a line. By using this approximation, the rate of convergence will increase and decrease the computational time. It must be noted, that this approximation is also liable to fail. In such cases, the algorithm reverts to a secant method. Furthermore, the Brent method is capable of monitoring convergence where the algorithm can revert to a bisection step in scenarios where the function evaluations are diverging or the convergence rate is too slow. More formally, this is expressed by:

$$x_{k+1} = \frac{x_k f(x_{k-1}) f(x_{k-2})}{(f(x_k) - f(x_{k-1}))(f(x_k) - f(x_{k-2}))} + \frac{x_{k-1} f(x_{k-2}) f(x_k)}{(f(x_{k-1}) - f(x_{k-2}))(f(x_{k-1}) - f(x_k))} + \frac{x_{k-2} f(x_{k-1}) f(x_k)}{(f(x_{k-2}) - f(x_{k-1}))(f(x_{k-2}) - f(x_k))} \tag{1.18}$$

where the quadratic interpolation method is replaced by the numerical approximation of the derivative in cases where $x_i = x_{i-1}$, $\forall i, i = 1, ...k$.

1.4 Artificial Neural Networks

Artificial Neural Networks (ANNs) are computational models inspired by the human brain's structure and function. These models are designed to recognize patterns, learn from data, and make decisions or predictions. ANNs have become a cornerstone of modern artificial intelligence (AI) and machine learning (ML), with applications spanning across various fields such as image and speech recognition, natural language processing, and predictive analytics.



1.4.1 Structure of ANNs

ANNs consist of interconnected layers of nodes, or neurons, which are analogous to the neurons in a biological brain. The primary components of an ANN include:

1. **Input Layer:** This layer receives the initial data or features that the network will process.
2. **Hidden Layers:** These layers perform intermediate computations and transformations on the input data. ANNs can have one or more hidden layers, which contribute to their ability to model complex relationships.
3. **Output Layer:** This layer produces the final output or prediction based on the computations of the hidden layers.

Each neuron in a layer is connected to neurons in the subsequent layer through weighted connections. These weights are adjusted during the training process to minimize the error in the network's predictions.

1.4.2 Multi-Layer Perceptron (MLP)

MultiLayer Perceptrons (MLPs) are a class of feedforward artificial neural networks that consist of multiple layers of neurons, typically organized into an input layer, one or more hidden layers, and an output layer. MLPs are one of the simplest and most widely used types of neural networks, serving as the foundation for more complex architectures in deep learning. They are particularly well-suited for tasks where data can be represented in a structured, tabular format.

A multi-layer perceptron (MLP), consisting of at least one hidden layer is the simplest version of an ANN. Mathematically, MLPs are defined by the following parameters :

$$\theta = (W_1, b_1, W_2, b_2, ..., W_L, b_L) \tag{1.19}$$

A MLP with "one-hidden-layer" presents its mathematical formula :

$$y = \phi^{(2)}(\sum_j w_j^{(2)} z_j^{(1)} + b^{(2)}$$
$$z_j^{(1)} = \phi^{(1)}(\sum_i w_{ij}^{(1)} x_i + b_j^{(1)}$$

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 $||\cdot||$:

$$D(f(x),F(x)) = ||f(x) - F(x)|| \tag{1.20}$$

where f(x) is the objective function, F(x) is the neural network approximated function.
The loss function is equivalent to the above distance :

$$L(\theta) := D(f(x),F(x|\theta)) \tag{1.21}$$

The training process aims to learn the optimal weights and biases to make the loss function as small as possible. The process can be formulated as an optimization problem :

$$argmin_\theta L(\theta|(x,y)) \tag{1.22}$$

Several back-propagation gradient descent methods have been successfully applied (Stochastic Gradient Descent (SGD) and its variants Adam and RMSprop, ...). The formula for updating the parameters is :

$$W \leftarrow W - \eta(i)\frac{\delta L}{\delta W'} \tag{1.23}$$

$$b \leftarrow b - \eta(i)\frac{\delta L}{\delta b'} \tag{1.24}$$

$$i = 0,1,2,..., \tag{1.25}$$

where η is a learning rate, which may vary during the iterations.

Chapter 2

Implementation

During this chapter we will discuss the datasets that were generated using the different models mentioned above as well as the parameters of choice for the MLPs we trained.

2.1 Black and Scholes model implementation

2.1.1 Dataset generation and analysis

By using the Latin hypercube sampling (LHS), we generated a dataset of 50K samples containing the different parameters used in the Black and Scholes model for calculating option prices.
We therefor used the wide range suggested in the research paper and were unfortunately unable to compare it to the narrow range due to the lack of time we were faced with. the table details the parameters and their ranges : After using the Black-Scholes model for calculating

Parameter	Range
Stock Price (S/K)	[0.4, 1.6]
Time to Maturity	[0.2, 1.1]
Risk free rate	[0.02, 0.1]
Volatility	[0.01, 1.0]

out option prices we get the following result :

	S/K	r	T	sigma	Option Prices
0	0.908537	0.054381	0.323828	0.846186	0.145224
1	0.821742	0.030448	0.728939	0.116410	0.001432
2	1.440178	0.025436	0.656050	0.836684	0.596865
3	1.279796	0.070649	0.960395	0.907837	0.574850
4	0.676531	0.074265	0.820220	0.840793	0.128554

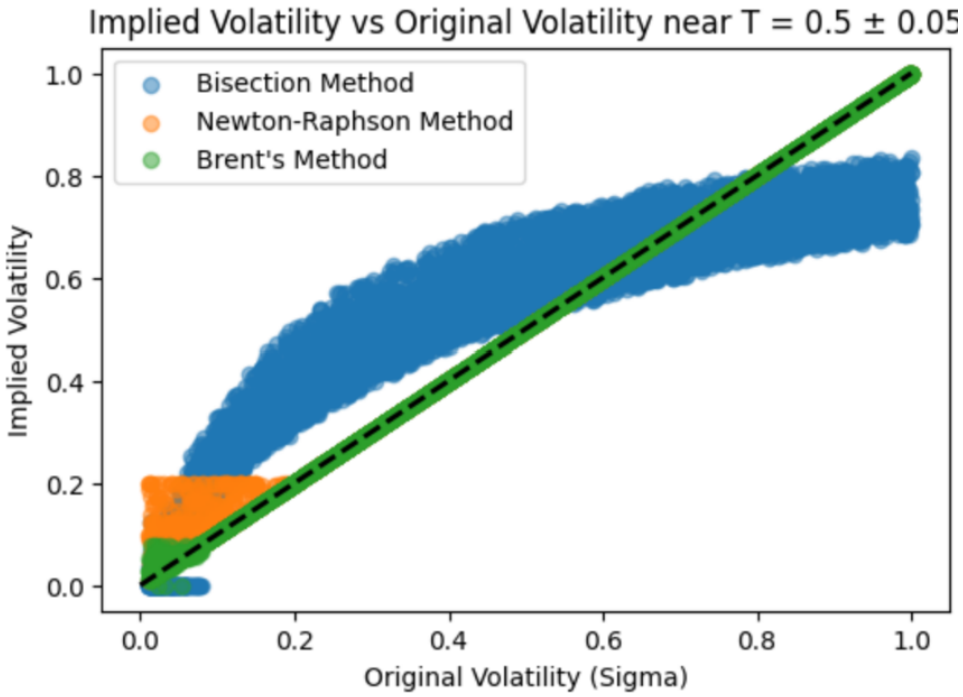
After that we calculated the implied volatility using the three numerical methods :

- Bisection method
- Newton-Raphson method
- Brent method

After getting the implied volatility for each one of them we calculated the difference between the calculated implied volatility and the volatility (sigma) we had in our initial dataset.
these were the results we observed :

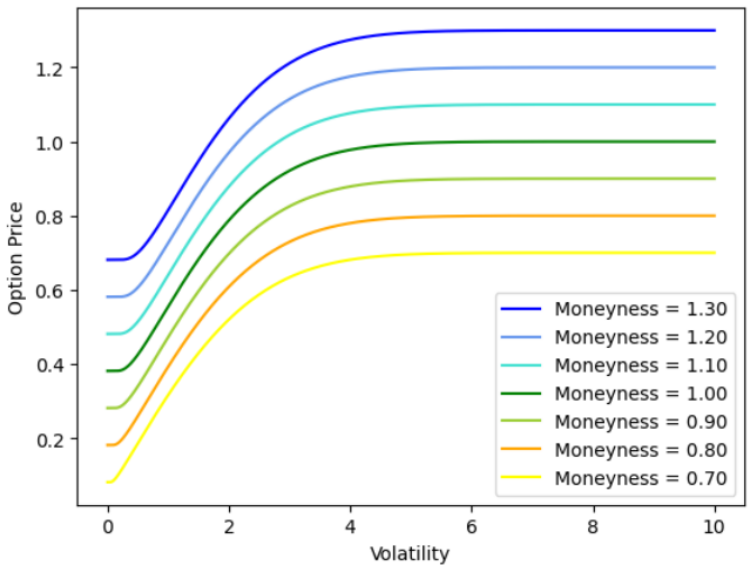
	Bisection Diff	Newton-Raphson Diff	Brent Diff
count	49853.000000	3.621200e+04	4.985300e+04
mean	0.134889	4.564001e-03	4.098225e-04
std	0.082199	1.876669e-02	3.833945e-03
min	0.001005	0.000000e+00	1.610934e-13
25%	0.067633	9.714451e-16	1.163275e-09
50%	0.128907	5.933032e-13	1.235740e-08
75%	0.192505	1.070123e-10	8.095387e-08
max	0.399503	1.896512e-01	1.100314e-01

After that we decided to generate a graph of the distribution of the implied volatility we calculated with the 3 different method's in function of the original volatility we had generated.



this graph demonstrates that Brent's Method gives us the most approximate result in respect of the value of our original volatility.

We then calculated the Option Prices in function of the volatility with different values of Moneyness using the volatility obtained with the Newton-Raphson method :



This made us notice that at some point when the volatility reaches infinity the option prices

are almost always the same value. This shows how innacurate the Newton-Raphson method can be in finding the volatility.

We will thus be using the result of Brent's numerical method to train our neural network.

2.1.2 Neural Networks Parameters

Now that the dataset to be used has been decided and that we generated all the necessary data we needed, it was now primordial to focus on searching for the right parameters for our Neural Network. To do that we had to use a randomized search of our hyper parameters by focusing on the following options :

- **hidden layer size :** 400
- **activation :** relu, tahn, sigmoid, elu
- **solver :** sgd, adam, rmsprop
- **alpha :** $0.0001, 0.001, 0.01$
- **learning rate :** constant, adaptive
- **batch size :** $256, 512, 1024, 2048, 3000$
- **early stopping :** True, False

However, due to the lack of computing power we had and because we wanted to work on a dataset of 50k samples (to be as close to reality as possible) we were not able to run the piece of code we developed in order to do our own hyper parameters search. The following code is the implementation of our hyper parameters search :

```python
from sklearn.model_selection import RandomizedSearchCV
# Parameter grid for random search
param_dist = {
    'hidden_layer_sizes': [(400, 400, 400, 400)],
    'activation': ['relu', 'tanh', 'sigmoid', 'elu'],
    'solver': ['sgd', 'adam', 'rmsprop'],
    'alpha': [0.0001, 0.001, 0.01],
    'learning_rate': ['constant', 'adaptive'],
    'batch_size': [256, 512, 1024, 2048, 3000],
    'early_stopping': [True, False]
}
mlp = MLPRegressor(max_iter=200)
# RandomizedSearchCV
random_search = RandomizedSearchCV(
    estimator=mlp,
    param_distributions=param_dist,
    n_iter=50,
    cv=3,
    n_jobs=-1,
    scoring='neg_mean_squared_error',
    random_state=42
)
random_search.fit(X_train, y_train)
best_params = random_search.best_params_
print(f"Best parameters: {best_params}")
```

In order to continue our implementation, we chose to use the same parameters that were found on the research paper :

Parameters	Options
Hidden layers	4
Neurons(each layer)	400
Activation	ReLU
Dropout rate	0.0
Batch-normalization	No
Initialization	Glorot_uniform
Optimizer	Adam
Batch size	1024

2.2 Heston model implementation

2.2.1 Dataset generation and analysis

the same way we generated the Black and Scholes dataset, we decided to use the LHS method to produce a dataset of 50k samples using again the wide range suggested by the research parper :

Parameter	Range
Stock Price (S/K)	[0.6, 1.4]
Time to Maturity	[0.1, 1.4]
Risk free rate	[0.0, 0.1]
Reversion speed	[0.0, 0.2]
Long average variance	[0.0, 0.5]
Volatility of volatility	[0.0, 0.5]
Correlation	[-0.95, 0.0]
initial variance	[0.05, 0.5]

The next step was to calculate the option prices and implied volatility using the Heston model. This made it possible to get the following dataset :

	S/K	T	r	kappa	theta	sigma	rho	V	Option Prices	Implied Volatility
0	0.998491	0.808872	0.016549	0.400395	0.499249	0.117712	-0.772886	0.208808	NaN	NaN
1	0.693260	1.334909	0.017128	0.354064	0.225360	0.399126	-0.215901	0.495315	0.121693	0.636059
2	0.887845	0.518151	0.088414	0.298450	0.359126	0.029365	-0.127296	0.484277	NaN	NaN
3	0.695816	0.162393	0.010967	1.895403	0.485258	0.151117	-0.308222	0.396328	0.007501	0.632977
4	1.398427	0.195990	0.096160	0.912650	0.349668	0.227912	-0.928034	0.373771	0.433227	0.634178

As we can notice some of our values were NaN, so we had to filter them out. We opted with the option of deleting the rows with NaN values to keep the data as clean as possible. This reduced the size of our dataset from 50k samples to 31k samples.

2.2.2 Neural Networks Parameters

For the same reason we couldn't do a randomized search of the hyper parameters of the Black and Scholes model's ANN, the optimal parameters used for training the Heston-ANN model and the IV-ANN model were the same as the ones that were given in the research paper.

Chapter 3

Results

3.1 Metrics results from model training (mae, mse, mape and score)

3.1.1 Metrics meaning

To evaluate the performance of our models, we have decided on using the following metrics :

- **Mean Absolute Error :** is a measure of errors between paired observations expressing the same phenomenon. It is calculated as the average of the absolute differences between predicted values and actual values.

- **Mean Squared Error :** measures the average of the squares of the errors—that is, the average squared difference between the estimated values and the actual value.

- **Mean Absolute Percentage Error :** measures the size of the error in percentage terms. It is the average of the absolute percentage errors of forecasts.

- **R^2 Score (Score):** The R^2 score, also known as the coefficient of determination, measures how well the predicted values from a regression model approximate the actual data points. It represents the proportion of the variance for a dependent variable that's explained by an independent variable or variables in a regression model.

3.1.2 Metrics values

Using the previously mentioned datasets and hyper parameters for each of our models, we can produce the following results for each metric :

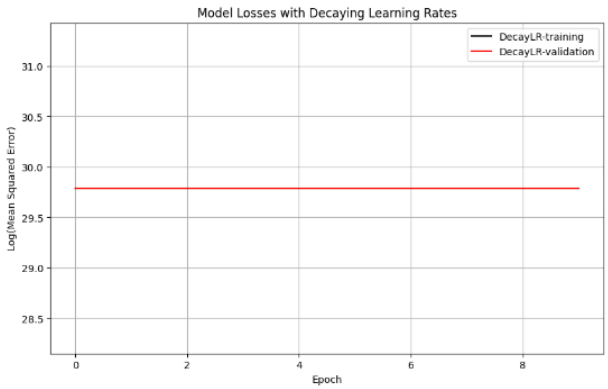
Model	MAE	MSE	MAPE	SCORE
BS-ANN	0.02	0.0006	725958391208	0.994
Heston-ANN	0.004	3.91	1143	0.998
IV-ANN	0.003	3.8	0.008	0.998

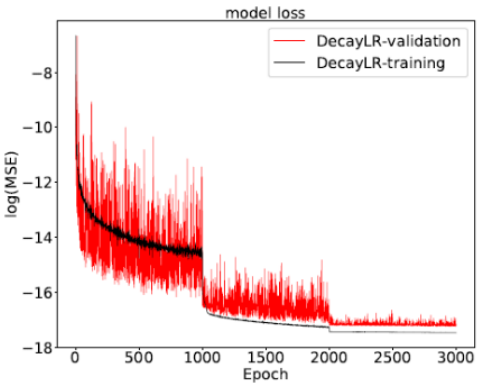
With this table we can come up with these conclusions :

- The R^2 Score of the 3 models might seem very high (almost equal to 1) which could indicate that the models perform well. However the **BS-ANN** model seems to have an **MAPE** that is way too gigh for what it should equal (it should at least be lower than 50 to say that the model can somehow perform in an adequate manner). This leads us to understand that those metrics are not enough to know how the models really perform.
- The only model that could be said performs well out of the three is the **IV-ANN** model, however, as said earlier, we cannot deduce that with those metrics alone.

3.2 Model loss graph

In the research paper a number of model loss graphs were shown. However, Because we had insufficient computing power we couldn't run the calculations that could give us the possibility to print the different model losses based on learning rates. We only were able to print poor results like the one shown here :





3.3 Performance metrics for Heston-ANN model

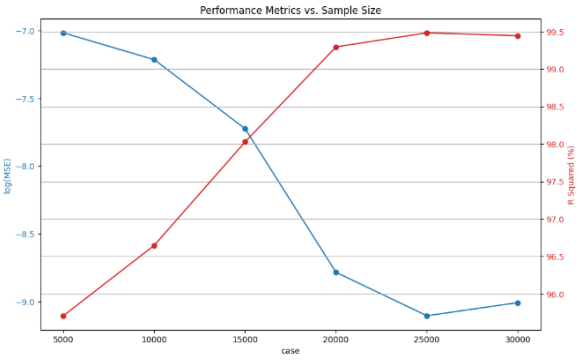
It this section of the research, we are trying to show that each time we increase the sample size the R^2 score of our model increases as well and our MSE lowers.
In the research paper the following steps are taken into consideration to do that :

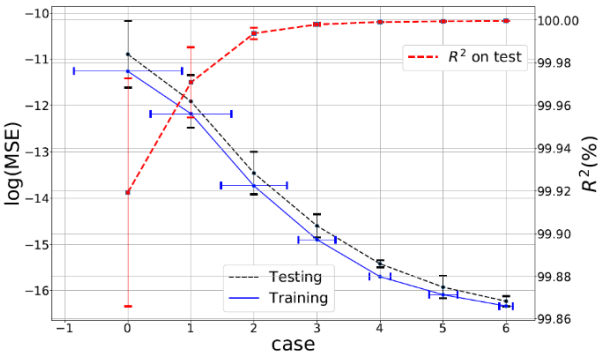
1. A dataset of 150k samples is generated using the Heston model.
2. the study takes into account multiple cases, each case having a coefficient that when multiplied by 24300 gives the number of samples to use for training.
3. for each case the hyper parameters of the model are searched again and the model is retrained
4. as a final step, the R^2 Score and the log(MSE) of each case is calculated.

Since we do not have the computing power to search for the hyper parameters each time and that we cannot generate a dataset of 150k samples we decided to go with these conditions instead :

- We were going to limit our study to a dataset of 30k samples.
- We were going to have 6 cases where the size of the dataset of each case is calculated by multiplying the number of the case by 5k (so case 3 would have 15k samples).
- we were going to use the optimal parameters used earlier in our model training.

We know that our results won't be as accurate as those of the paper, however our goal is not the results but the reproduction of the method used. Thus we got the following results :

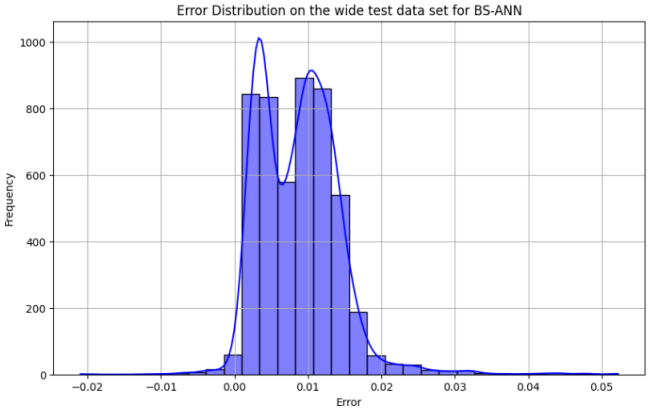


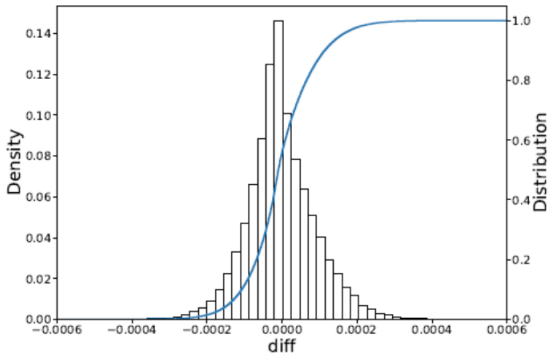


3.4 error distribution on the wide test datasets

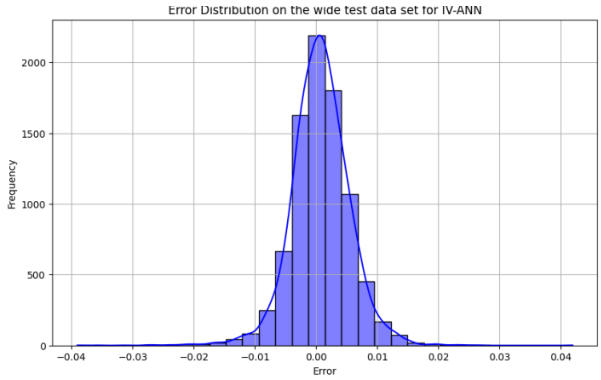
For our final analysis, we decided to showcase the error distribution of the wide test datasets using our 3 ANN models. This resulted into the 3 following graphs :

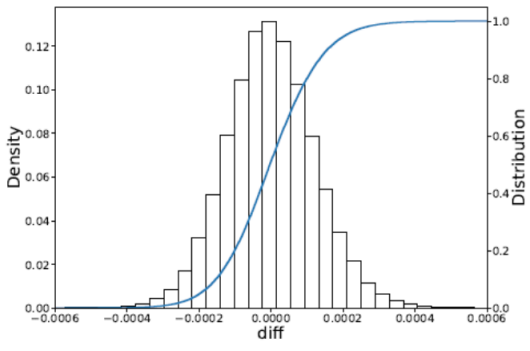
3.4.1 Error distribution for BS-ANN



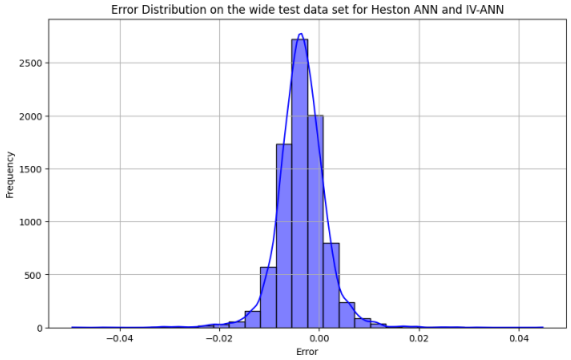


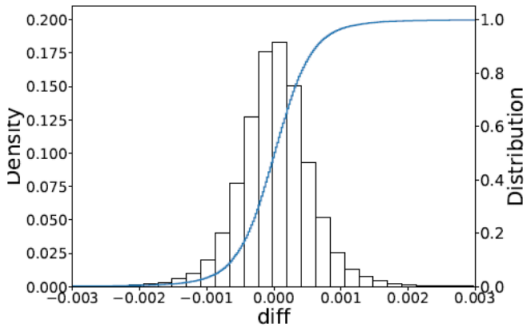
3.4.2 Error distribution for IV-ANN





3.4.3 Error distribution for Heston-IV-ANN





Chapter 4

Conclusion

This study showcased the power of ANNs in making the calculations of implied volatility much quicker than before, but the efficiency of such method is yet to be unveiled. Unfortunately we were not able to tap into the full potential of this technology due to technical issues. However, we were still able to grasp the full length of its importance in optimizing and enhancing performance of calculations from now on.

Bibliography

[1] Shuaiqiang Liu, Cornelis W. Oosterlee, and Sander M. Bohte, "Pricing Options and Computing Implied Volatilities using Neural Networks," 2019.

[2] "Diffusion and Black–Scholes Pricing Formula," Available at: https://courses.seas.harvard.edu/climate/eli/Courses/APM105/Sources/02-second-order-ODEs/Motivation/diffusion-and-Black%E2%80%93Scholes-pricing-formula-Wikipedia.pdf