Project Title	Application of Machine Learning in Heston Model Calibration
Name	Lin Yanjie
Student ID	A0133903M
Itemized Workload	
1. Brainstorming, research, and literature review	25 hours
2.1 Programming: study library APIs and set up cloud environment	10 hours
2.2 Programming: data generation and gathering	10 hours (+ 5 hours background runtime)
2.3 Programming: neural network building and training	10 hours (+ 30 hours background runtime)
2.4 Programming: Heston model calibration test	10 hours (+ 10 hours background runtime)
3. Result analysis	5 hours
4. Report writing	35 hours
Total Workload	105 hours (+ 45 hours background runtime)
Original contribution	The study proposes an innovative framework to calibrate Heston model using feedforward neural networks and differential evolution. The framework can achieve the same accuracy level as that using the traditional calibration method based on the analytical solution. Moreover, the framework can achieve significant reduction of calibration time required, which makes it competitive in the real-world application. The study tests different neural network specifications and identifies a parameter-IV type of model with moderate complexity as the optimal model. The study also examines the application of proposed framework in calibrating other financial models and its possible limitations.

Application of Machine Learning in Heston Model Calibration

Lin Yanjie

1. Introduction

Pricing is one of the key daily activities for derivative traders. Due to the complexity of derivative structures and the quick response required, practitioners often rely on financial models to help them generate reasonable price estimates within a short period of time. In terms of option pricing, one of the most widely known financial models is Black-Scholes model (Black & Scholes, 1973), which gives a theoretical estimate of European option prices based on the volatility of underlying assets and the time to maturity. This model has led to a boom in option trading and provided mathematical legitimacy to the activities of option markets around the world (MacKenzie, 2006).

Nevertheless, with the evolution of option markets, the original Black-Scholes model has become less useful in recent years. One of its key shortcomings is that the model assumes a constant volatility for the underlying asset, which is irrelevant to either strike price or time to maturity. In contrast to that assumption, it is observed in the real market that the implied volatility of vanilla options always increases when the strike price deviates from the spot price. People give a name to this pattern - volatility smile.

To address the volatility smile, researchers have proposed multiple alternative models with the assumption of varying volatility. Most of those models can be categorized into either a local volatility model or a stochastic volatility model. A local volatility model treats volatility as a function of time (t) and the corresponding asset price (S_t). One example is Dupire's equation for both continuous cases (Dupire, 1994) and discrete cases (Derman & Kani, 1994). On the other hand, a stochastic volatility model treats volatility as a random variable with certain distribution patterns. Some examples include Heston model (Heston, 1993), constant elasticity of variance (CEV) model (Cox, 1974), and GARCH model (Bollerslev, 1986).

Every financial model has a set of model parameters, which define how the model works under different market scenarios. The process of finding those parameters is called model calibration. Model calibration is usually formulated as an optimization problem where the objective is to minimize the difference between the market observations and the predictions obtained from the model. It has been a critical task

for derivative traders, which is performed for multiple times every day. Hence, many researchers are interested in this area, and the focus is to find a fast, accurate and robust calibration process that could be applied commonly for most financial models (Deng et al., 2008; Cui et al., 2017).

In recent years, there is a boom of using machine learning and neural networks to solve complicated regression and classification problems, given the advancement of computing power and optimized algorithms. Thus, many researchers have worked on applying machine learning techniques in financial model calibration. For example, Spiegeleer et al. (2018) employed Gaussian process regression in the fitting of sophisticated Greek curves and implied volatility surfaces. Liu et al. (2019) proposed a Calibration Neural Network (CaNN) to solve the high-dimensional calibration problems of the Heston and Bates models.

The key advantage of applying neural networks in model calibration is that it can significantly speed up the computation, compared with analytical solution or other numerical methods (Hernandez, 2016). There are mainly two reasons for its high computational performance: 1) the development of modern processing units (e.g., GPUs) that are dedicated for accelerated computing purposes; 2) the development of open-source scientific libraries (e.g., Tensorflow) that can fully utilize the computing power of GPUs.

In addition to computation speed, researchers also pay attention to the robustness of calibration process. Mrázek and Pospíšil (2017) tested and compared various optimizers available in MATLAB, such as genetic algorithm, simulated annealing, and lsqnonlin, for their robustness in calibrating Heston model with real market data. The key focus of calibration robustness is to avoid the algorithm being stuck at the local minimum. As a result, global optimizers are often preferred in the calibration process instead of local optimization algorithms.

The objective of this study is to develop a machine learning based framework to calibrate Heston model, a popular stochastic volatility model. The framework consists of two parts:

- Train a feedforward neural network that closely approximates the analytical solution of Heston model. The trained neural network takes the market and model parameters as its input and generates the corresponding implied volatility or implied volatility surface as its output.
- Use the trained neural network, and differential evolution, a global optimization algorithm, to calibrate the Heston model parameters, with the observed implied volatility surfaces.

We will demonstrate that the proposed calibration framework can achieve satisfactory accuracy and superior performance in terms of computation speed.

2. Theoretical Background

2.1 Heston Model

2.1.1 Model description

Heston model (Heston, 1993) is one of the most popular stochastic volatility models. It models the movement of asset price and volatility as two stochastic processes with correlation. The model equations are presented as following:

$$dS_t = (r - q)S_t dt + \sqrt{V_t} S_t dW_t^S$$

$$dV_t = \kappa(\theta - V_t)dt + \sigma\sqrt{V_t} dW_t^V$$

$$dW_t^S dW_t^V = \rho dt$$

In the above equations, S_t and $\sqrt{V_t}$ represent the asset price and volatility, respectively. The first equation is like Black–Scholes model, except that the volatility is a time dependent variable instead of a constant. The second equation has the same form as Cox-Ingersoll-Ross model (Cox, Ingersoll, & Ross 1985). It implies that the volatility is a mean reverting process. When t approaches infinity, the expected value of V_t will converge to θ. The third equation describes the correlation between two stochastic processes. It is observed in the real market that the volatility tends to increase when the asset price drops.

To ensure that V_t is strictly positive, Feller (1951) proved that the following condition should be obeyed:

$$2\kappa\theta > \sigma^2$$

In terms of model calibration, the parameters of interest include:

- V_0: Initial asset price variance
- θ: Long term asset price variance
- κ: Rate of variance reversion
- σ: Volatility of asset price volatility
- ρ: Correlation between asset price and volatility

On the other hand, r (risk free rate) and q (dividend rate) are observable from the market, and thus not included in the calibration process.

2.1.2 Option pricing with Heston model

Under Heston model, the price of a European call option can be obtained by a genetic probabilistic approach:

$$C_0 = S_0\Pi_1 - e^{(r-q)T}K\Pi_2$$

In the above equation, Π_1 and Π_2 are two probability related quantities. Π_1 is the option delta, and Π_2 is the risk-neutral probability of option exercise (i.e., $\widetilde{\mathbb{P}}[S_T > K]$). Heston (1993), Gatheral (2006), and Crissstomo (2014) proposed analytical methods to compute the Π_1 and Π_2. The detailed equations proposed by Crissstomo (2014) are presented in the Appendix A.1.

Nevertheless, those analytical solutions involve sophisticated operations such as Fourier transform and complex number calculations. It is not easy to implement those solutions without pre-coded libraries, and the execution speed is generally slow. Hence, practitioners sometimes choose to use numerical methods in option pricing, such as Monte-Carlo simulation.

2.2 Feedforward Neural Networks

2.2.1 Model description

Artificial neural network (ANN) is one of the most popular machine-learning algorithms in recent years, based on which many applications have been developed in the areas of text classification, image recognition, sentiment analysis, robotics, etc. As the name implies, the algorithm is inspired by the biological neural circuits. Artificial neural networks are constructed with a collection of connected units called artificial neurons. Similar as a biological neuron, an artificial neuron can receive the input from the upstream neurons, process the input, and pass an output to the downstream neurons. In the ANN, neurons are typically aggregated into layers, and how those neuron layers are connected defines the architecture of an ANN.

Feedforward neural network is the simplest type of artificial neural network devised (Schmidhuber, 2015). In this type of ANN, the neuron layers are stacked in order. The information, in the form of a 1-D vector, moves in one direction from the first layer (input layer) to the last layer (output layer). The layers in between are called hidden layers. In contrast with other sophisticated ANNs like recurrent neural network, there are no loops in the feedforward neural network that can reverse the information direction.

In feedforward neural networks, the process of a layer receiving the input, processing the input, and passing the output to the successive layer is called forward propagation. It involves two major operations. Firstly, the input vector is multiplied with a weight matrix and added a bias term, to generate an intermediate vector. Secondly, an activation function is applied to every element in the intermediate vector, to generate the final output. The following mathematical expression describes how forward propagation works in one layer:

$$z^{[l]} = W^{[l]}a^{[l-1]} + b^{[l]}$$

$$a^{[l]} = g^{[l]}(z^{[l]})$$

where $a^{[l-1]}$ is the output received from the previous layer $l-1$; $W^{[l]}$ is the weight matrix of layer l; $b^{[l]}$ represents the bias term of layer l; $z^{[l]}$ is the output of layer l before activation function; $g^{[l]}$ is the activation function of layer l; $a^{[l]}$ is the output of layer l that is passed to the next layer $l+1$.

Note that the activation function is usually a non-linear function, which helps reduce the number of neurons required for neural networks to compute nontrivial problems. Some common activation functions include:

- Sigmoid: $f(x) = \frac{1}{1+e^{-x}}$
- Tanh: $f(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$
- ReLu: $f(x) = \max(0, x)$

The universal approximation theorem, proved by Cybenko (1989), suggests that every continuous function can be approximated by a feedforward neural network, and a single hidden layer is sufficient for such an approximation. Nevertheless, the theorem does not indicate how many neurons in a single hidden layer are sufficient to make such an approximation, and whether a single hidden layer structure is optimal. Hence, in practice, researchers tend to add multiple layers when the function to be approximated is highly non-linear and involves sophisticated operations.

2.2.2 Training of neural networks

Machine learning tasks can be categorized into supervised learning and unsupervised learning. A supervised learning task is to generate a model that maps an input with an output. Models (e.g., neural networks) are trained with labeled examples, and each example is a pair of input vector (x) and its desired output values (y). On the other hand, an unsupervised learning task to learn the pattern of data (e.g., clustering). Models are trained with unlabeled data, which do not include explicit input-output pairs. In the context of this study, a supervised learning approach is adopted.

Training of a supervised learning model is essentially an optimization problem, which adjusts the model parameters to minimize the difference between the model predicted outputs and the real outputs. For feedforward neural networks, the parameters to be adjusted are the weight matrices and bias terms of each layer. To formulate the optimization problem, the output of a loss function, which quantifies the prediction error, is defined as the minimization target. For regression problems, the loss function is usually mean squared error, while for classification problems, the loss function is usually cross-entropy loss. Researchers also proposed other kinds of loss functions such as Huber loss (Huber, 1964), Kullback–Leibler divergence (Kullback & Leibler, 1951), etc.

To find the minimum of loss function, gradient descent algorithm is implemented, which carries repeated adjustments to the model parameters based on the partial derivatives of loss function with respect to each parameter. In feedforward neural networks, the gradients are calculated layer by layer in a backward direction from the output layer to the input layer. Hence, the process is also called backpropagation. The following mathematical expression describes how backpropagation works in one layer:

$$dz^{[l]} = W^{[l+1]T} dz^{[l+1]} * g^{[l]'}(z^{[l]})$$

$$dW^{[l]} = dz^{[l]} a^{[l-1]}$$

$$db^{[l]} = dz^{[l]}$$

where $dz^{[l]}$ is the gradient of loss function with respect to the intermediate vector of layer l; $dW^{[l]}$ is the gradient of loss function with respect to the weight matrix of layer l; $db^{[l]}$ is the gradient of loss function with respect to the bias term of layer l; $g^{[l]'}$ is the first order derivative of activation function of layer l; W^T represents the transpose of matrix W; $*$ represents element wise multiplication.

After all the gradients are obtained, the algorithm completes one iteration by updating the weight matrices and bias terms:

$$W_{i+1}^{[l]} = W_i^{[l]} - \alpha_i dW_i^{[l]}$$

$$b_{i+1}^{[l]} = b_i^{[l]} - \alpha_i db_i^{[l]}$$

where i is the index of current iteration; α_i is the learning rate, which is an important tuning parameter in model training.

Theoretically, gradient descent with a defined loss function is sufficient to complete the training process. However, due to the complex nature of neural networks, the training process is usually slow and erratic. There are a few techniques that can be applied to improve the efficiency of training process.

Overfitting and regularization

Overfitting is an issue where the trained model is good at predicting the training data but performs poorly with the test data. It often happens when the model is too complex, or the training data are not enough. As a common practice, if the overfitting problem is spotted, researchers should consider feeding more data or reducing the model complexity (e.g., remove some layers).

Large values in the weight matrices are a sign of overfitting, and regularization techniques are usually applied in the training process to prevent this phenomenon. Some common regularization methods include L2 regularization and dropout. L2 regularization adds an additional term in the loss function,

which is proportional to the sum of L2 norms of weight matrices. Through L2 regularization, large weights are penalized, and thus less favored in gradient descent. Dropout randomly removes a certain percentage of neurons from the network. Since any neuron is possible to be removed, dropout prevents the total weights being "concentrated" at certain neurons. Instead, the total weights are "spread out", making the average level of weights smaller.

Feature Scaling

Feature scaling is a technique applied to the input so that the range of each independent variable is at the same level. It is to ensure that each variable, which represents one kind of feature, has a proportional contribution to the final prediction. It can also make gradient descent converges faster. Some common feature scaling methods include min-max scaling and z score standardization.

Batch normalization

When training deep neural networks with many hidden layers, the issue of internal covariate shift occurs. The distribution of each layer's input changes during the training process, as the parameters of the previous layers change (Ioffe & Szegedy, 2015). This issue slows down the training speed. To resolve the issue, Ioffe and Szegedy (2015) proposed batch normalization technique, in which the output of each layer is scaled per mini batch. By doing so, the distribution of input for the successive layer is fixed, and thus the ill effect of internal covariate shift is removed.

Currently, there are many well-supported open-source machine-learning libraries, such as Tensorflow, Keras, and PyTorch. With the help of those libraries, researchers can conveniently build, train, and test various types of neural networks with different specifications.

2.3 Differential Evolution

Differential evolution, developed by Storn and Price (1997), is a global optimization algorithm, which iteratively improves a pool of candidate solutions with regards to a given measure of quality. The method requires neither derivative calculation nor the assumptions of objective function.

The algorithm comprises the following steps:

- Initialize a pool of N_p candidate solutions: $\theta^{[1]}, \theta^{[2]}, \dots, \theta^{[N_p]}$.
- For each candidate $\theta^{[i]}$,
    - Randomly pick another three candidates $\theta^{[a]}, \theta^{[b]}, \theta^{[c]}$ from the pool, which satisfy the condition $i \neq a \neq b \neq c$.
    - Generate a trail candidate $\theta^{[i]\prime}$ by the following rules:

        - For each variable $\theta_j^{[i]}$ in $\theta^{[i]}$, generate a uniformly distributed variable $r_j \sim U(0,1)$. Compare r_j with a present parameter $C_r \in [0,1]$. If $r_j < C_r$, $\theta_j^{[i]'} = \theta_j^{[a]} + F \times (\theta_j^{[b]} - \theta_j^{[c]})$, where F is a preset parameter and $F \in [0,2]$. Otherwise $\theta_j^{[i]'} = \theta_j^{[i]}$.
    - Compare $f(\theta_i)$ and $f(\theta_i')$ from the objective function. If $f(\theta_i') < f(\theta_i)$ replace $\theta^{[i]}$ with $\theta^{[i]'}$.
- Repeat the above step until the exit criterion is met.

Since random variables, instead of gradients, are used in the algorithm, differential evolution has an advantage over local optimization algorithm that it can jump out of local minimum and reach global minimum. The choice of preset parameters (N_p, F and C_r) can largely impact the optimization performance, and there are rules of thumb developed by Storn (1996), and Liu and Lampinen (2002).

3. Methodology

3.1 Overview

The objective of this study is to develop a machine learning based framework to calibrate Heston model. The framework consists of two major parts: forward pass (feedforward neural network) and backward pass (calibration of Heston model parameters).

Forward pass (feedforward neural network)

A feedforward neural network is trained to closely approximate the analytical solution of Heston model with regards to European call option pricing. The input vector of neural network includes market parameters (e.g., risk free rates), option parameters (e.g., moneyness) and Heston model parameters ($V_0, \theta, \kappa, \rho, \sigma$). There are two kinds of outputs. One is implied volatility (IV), which is the derived volatility back calculated from option price using Black-Scholes model. IV is chosen as the output instead of option price, because it is a popular measure of option price used by derivative traders, and it reflects market consensus of future volatility. Another kind of output is implied volatility surface (IVS), which consists of multiple implied volatility under a fixed grid of option parameters. As a collection of IVs at different strikes and maturities, IVS presents more information to derivative traders. It helps option trader recognize the pattern of volatility smile and construct the volatility term structure.

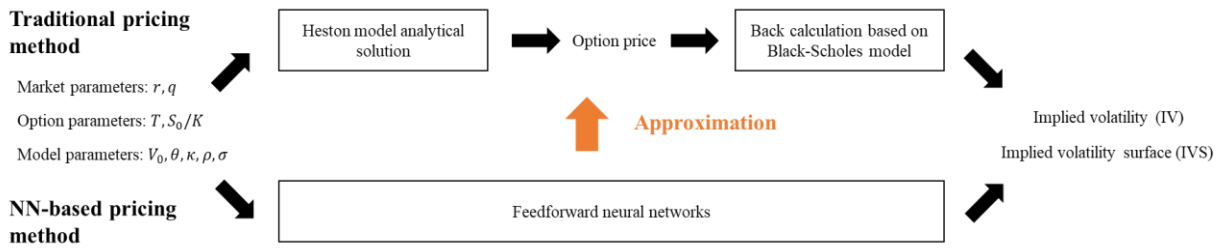

Figure 1: Forward pass of calibration framework

Backward pass (calibration of Heston model parameters)

After obtaining the neural network, we use it to calibrate Heston model parameters based on the observed implied volatility surface. It is formulated as an optimization problem to minimize the difference between the IVS obtained from the neural network and the real IVS, with differential evolution as the optimization algorithm.

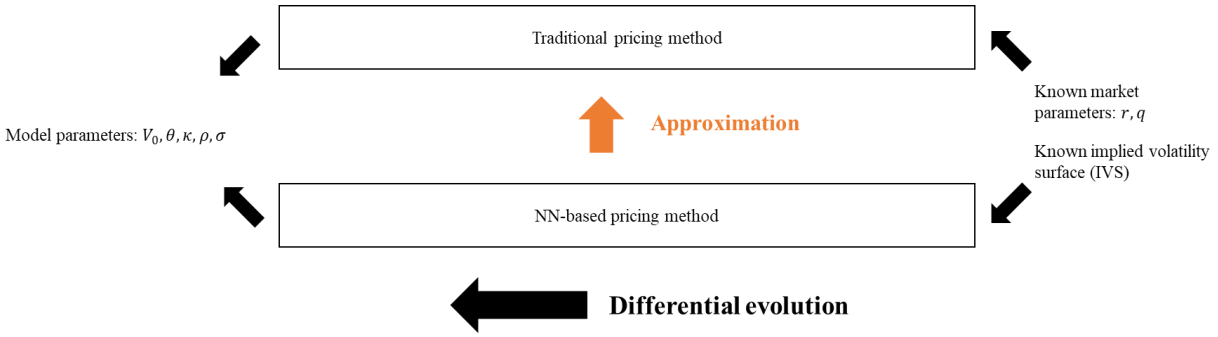

Figure 2: Backward pass of calibration framework

This study involves the assessment of computational performance. Hence, the details of our testing environments (hardware and software) are provided in Appendix A.2 and A.3.

3.2 Data Generation and Processing

3.2.1 Neural network training data

As the purpose of neural network is to approximate the analytical solution of Heston model, its training data consist of millions of theoretical results from the analytical solution of Heston model for European call option pricing, with input parameters randomly chosen from certain ranges. There are two sets of training data, used for training neural networks with two kinds of input-output pairs: parameter-IVS pairs and parameter-IV pairs.

Parameter-IVS

In this data set, the inputs are market parameters and model parameters, and the output is implied volatility surface. The inputs are randomly generated within the following ranges:

Name	**Type**	**Range**	**Distribution**
Risk free rate (r)	Market parameter	[0, 0.05]	Uniform
Dividend rate (d)	Market parameter	[0, 0.02]	Uniform
Initial price variance (V_0)	Heston model parameter	[0.04, 0.36]	Uniform in square root
Long-term price variance (θ)	Heston model parameter	[0.04, 0.36]	Uniform in square root
Rate of reversion (κ)	Heston model parameter	[0, 5]	Uniform
Volatility of price volatility (σ)	Heston model parameter	[0.1, 0.8]	Uniform
Correlation (ρ)	Heston model parameter	[-0.9, 0]	Uniform

Table 1: Input ranges of parameter-IVS training data

The IVS output consists of multiple implied volatilities under a fixed grid of option parameters, which is specified as following:

Name	**Type**	**No. of values**	**Values**
Moneyness (S_0/K)	Option parameter	13	0.8, 0.85, 0.9, 0.95, 0.975, 0.99, 1, 1.01, 1.025, 1.05, 1.1, 1.15, 1.2
Maturity (T)	Option parameter	7	1 week, 2 weeks, 1 month, 2 months, 3 months, 6 months, 1 year

Table 2: IVS grid settings of parameter-IVS training data

From Table 1 and 2, the number of independent variables in one input sample is 7 and the number of dependent variables in one output sample is 91 (13 × 7). There are totally 500,000 pairs generated.

Parameter-IV

In this data set, the inputs are market parameters, option parameters, and model parameters. The output is implied volatility. The inputs are randomly generated within the following ranges:

Name	**Type**	**Range**	**Distribution**
Risk free rate (r)	Market parameter	[0, 0.05]	Uniform
Dividend rate (d)	Market parameter	[0, 0.02]	Uniform
Moneyness (S_0/K)	Option parameter	[0.8, 1.2]	Uniform
Maturity in days (T)	Option parameter	[1, 365]	Uniform and discrete

Initial price variance (V_0)	Heston model parameter	[0.04, 0.36]	Uniform in square root
Long-term price variance (θ)	Heston model parameter	[0.04, 0.36]	Uniform in square root
Rate of reversion (κ)	Heston model parameter	[0, 5]	Uniform
Volatility of price volatility (σ)	Heston model parameter	[0.1, 0.8]	Uniform
Correlation (ρ)	Heston model parameter	[-0.9, 0]	Uniform

Table 3: Input ranges of parameter-IV training data

From Table 3, the number of independent variables in one input sample is 9 and the number of dependent variables in one output sample is 1. There are totally 5,000,000 pairs generated.

3.2.2 Calibration test data

After training the neural network, its performance for Heston model calibration is evaluated with two sets of test data:

- 100 theoretical parameter-IVS pairs
- Observed IVS in the real market

The 100 theoretical parameter-IVS pairs are generated in the same way as that in Section 3.2.1, with the same input ranges and output grid settings as those in Table 1 and 2.

Observed IVS data are obtained from the real option market. Two stocks listed in the New York Stock Exchange, AAPL and FB, are selected, because their historical long-term price volatilities are within the input range in Table 1 and their options have large trading volumes. We obtain the raw option IVS of the two stocks from Yahoo Finance at the market closing time on 2021/02/12, 2021/02/19, and 2021/02/26.

The raw option IVS data are further processed to interpolate IV at the same moneyness and maturity levels as those in Table 2. The data processing involves the following operations:

- Remove the data that are not updated in the last trading hour, because the stock price generally has large fluctuations near the market closing time.
- Interpolate IV at a certain moneyness level using the options at the nearest two strike levels.
- Approximate IV at a certain maturity using the options at the nearest maturity or interpolate it using the options at the nearest two maturities.

After data processing, the final IVS usually consists of only 40 ~ 60 observed IVs, due to lack of valid market data at certain moneyness and maturity levels. Those levels will be excluded in the model calibration process.

3.3 Building and Training Neural Networks

After obtaining the training data in Section 3.2.1, feedforward neural networks are built and trained, aiming to closely approximate the analytical solution of Heston model for European call option pricing. In total, five neural networks are trained, which have different specifications in terms of input-output sizes, network architectures, and training settings. The five neural networks are named Model A, B, C, D, and E, respectively. The detailed neural network specifications are presented as following:

	A	**B**	**C**	**D**	**E**
Input size	7	7	7	9	9
Output size	91	91	91	1	1
Training data	Parameter-IVS pairs			Parameter-IV pairs	
Train-test split	0.8 : 0.2				
Feature Scaling	Yes. Use min-max scaling.				
No. of layers	4	6	8	3	4
No. of neurons per layer	128	256	512	64	128
No. of parameters	47,687	294,087	1,641,415	5,381	35,973
Activation function	ReLu for input and hidden layers. Sigmoid for output layer.				
Batch normalization	Yes, for every layer.				
Dropout	No, for every layer.				
Loss function	Mean squared error				
Optimization algorithm	Adam				
Mini batch size	1024				
No. of epochs	5000				
Learning rate schedule	Reduce to 0.2 × original if no improvement for consecutive 400 epochs				

Table 4: Neural network specifications

All the neural networks are built and trained using Tensorflow and Keras. Given the large size of training data, the training is done in the GPU-enabled environment (Environment 2 in Appendix A.2).

3.4 Calibration of Heston Model Parameters

The trained neural networks are used in the calibration of Heston model parameters, with the test data obtained in Section 3.2.2. The calibration process is formulated as an optimization problem to minimize the difference between the calibrated IVS from the neural networks and the real IVS. The objective

function, which quantifies the error between the calibrated IVS and the real IVS, is mean absolute percentage difference:

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

where $\hat{y}_i$ is the ith calibrated IV on the IVS; y_i is the ith real IV on the IVS; N is the total number of IVs on the IVS.

Differential evolution is the optimization algorithm, which is implemented using SciPy with the following settings:

Population size (N_p)	75
Differential weight (F)	0.5
Crossover probability (C_r)	0.7
Maximum iteration	100
Tolerance	0.01

Table 5: Differential evolution settings

For each sample of calibration test data, one set of optimal calibrated Heston model parameters $(V_0, \theta, \kappa, \rho, \sigma)$ and their corresponding IVS (i.e., the calibrated IVS) are obtained as the result of model calibration, which is further used for evaluation.

Besides the proposed calibration framework using neural networks, the traditional method using the analytical solution is also performed, and its result serves as a benchmark in the evaluation section. The tradition calibration method is implemented based on the codes provided by Balaraman (2016).

3.5 Result Evaluation

The performance of model calibration is assessed by two key criteria: calibration accuracy and computation speed.

Calibration accuracy

Calibration accuracy is related to the error between the calibrated parameters/IVS and the real parameters/IVS. For theoretical test data, both real parameters and IVS can be obtained. Hence, the calibration framework is assessed by both parameter and IVS errors. For each model parameter, its error is calculated as the absolute difference scaled by the corresponding range specified in Table 1. IVS error is calculated as the mean absolute percentage difference of all the IVs on the IVS (the same as the target of differential evolution), which will be called in short the percentage difference of IVS in Section 4.

For real market data, there are no real Heston model parameters provided. Hence, only IVS error is calculated, using the same method above.

Computation speed

For all the calibration tests, the computation time and the number of differential evolution iterations are recorded down. To ensure fairness of comparison, all the calibration tests are done in the same testing environment with only CPU enabled (Environment 1 in Appendix A.2)

4. Results and Discussion

4.1 Findings in Neural Networks Training

4.1.1 Training speed

The five neural network models are trained in the GPU-enabled environment. For Model A, B, and C with 500,000 training samples, the training time per epoch is around 4 seconds, corresponding to 5.5-hour total training time with 5000 epochs. For Model D and E with 5,000,000 training samples, the training time per epoch is around 7 seconds, corresponding to 10-hour total training time with 5000 epochs.

Due to the optimization of GPU performance in Tensorflow, the training speed of complex neural networks using GPU is significantly faster than that using CPU. We perform a "test training" of Model B and C in the CPU-only environment. For Model B, it takes 25 seconds per epoch. For Model C, it takes 80 seconds per epoch. As a result, the total training time becomes much longer as 1.5 days and 4.5 days, respectively.

For the industrial-level application, practitioners should analyze the benefit of speed improvement using GPU and the cost associated with it (e.g., the cost of specialized cloud services), and make the optimal decision accordingly.

4.1.2 Model loss

As discussed in Section 2.2.2, the loss function quantifies the prediction errors of the trained neural network. In other words, its output can be used to assess the predicting capability of the neural network.

In the training process, data are split into two parts – one for training the model and one for testing the model. Hence, there are two outputs obtained from the loss function, when the training is completed.

- In-sample loss: the error of prediction with training samples
- Out-of-sample loss: the error of prediction with test samples

In most scenarios, in-sample loss is lower than out-of-sample loss, because it is the direct optimization target of gradient descent. Thus, out-of-sample loss is usually considered a better indicator of model predicting capability, and a big difference between in-sample and out-of-sample losses suggests that the trained neural network has the issue of overfitting.

The losses (mean squared error) of the trained neural networks are presented in the following Table 6.

	A	**B**	**C**	**D**	**E**
In-sample loss	2.048×10^{-6}	2.203×10^{-6}	2.954×10^{-6}	1.677×10^{-6}	6.824×10^{-7}
Out-of-sample loss	2.059×10^{-6}	2.119×10^{-6}	2.855×10^{-6}	2.334×10^{-6}	1.459×10^{-6}

Table 6: Losses of trained neural networks

In terms of out-of-sample loss, Model E has the lowest one among the five neural networks. It suggests that Model E is the closest approximation to the analytical solution of Heston model for European call option pricing.

In terms of the difference between in-sample and out-of-sample losses, Model A, B, and C have nearly no difference. Hence, there is no indication of overfitting. Model D and E, on the other hand, have some degree of difference between in-sample and out-of-sample losses. Nevertheless, the differences are not order-of-magnitude, and thus we believe that the overfitting is still acceptable.

It is interesting to note that, Model A, B, and C, which map the parameters with an entire IVS, do not perform better when the model complexity increases. On the other hand, Model D and E, which map the parameters with a single IV value, perform better when the model complexity increases. This observation suggests that a parameter-IV model has the potential to improve its approximation accuracy by increasing the model complexity, which could be considered an advantage over a parameter-IVS model.

4.2 Heston Model Calibration Performance

4.2.1 Calibration with theoretical data

All the five trained neural networks, together with the Heston model analytical solution, are used in the calibration of Heston model parameters for the theoretical parameter-IVS data. The results are evaluated based on calibration accuracy and computation speed.

Calibration accuracy

Calibration accuracy is assessed by both parameter and IVS errors. In terms of parameter error, the absolute differences (scaled by the parameter ranges) between the calibrated parameters and the real parameters are presented in Table 7. Note that Table 7 only presents the average figures of the 100 calibration test results. The detailed statistics is presented in Appendix A.4.

	A	**B**	**C**	**D**	**E**	**Benchmark**
V_0	8.43×10^{-4}	7.96×10^{-4}	9.66×10^{-4}	1.36×10^{-3}	6.83×10^{-4}	3.59×10^{-5}
κ	2.01×10^{-2}	1.25×10^{-2}	1.34×10^{-2}	3.69×10^{-2}	2.44×10^{-2}	4.58×10^{-3}
θ	2.26×10^{-2}	1.54×10^{-2}	2.30×10^{-2}	3.58×10^{-2}	2.58×10^{-2}	4.97×10^{-3}
σ	3.79×10^{-2}	1.89×10^{-2}	3.28×10^{-2}	5.15×10^{-2}	3.16×10^{-2}	1.98×10^{-3}
ρ	3.33×10^{-2}	2.24×10^{-2}	3.67×10^{-2}	6.04×10^{-2}	3.36×10^{-2}	1.39×10^{-3}

Table 7: Average absolute errors (scaled by parameter ranges) of calibrated parameters vs. real parameters

In terms of IVS error, the percentage differences between the calibrated IVS and the real IVS, with detailed statistics, are presented in Table 8.

	A	**B**	**C**	**D**	**E**	**Benchmark**
Mean	0.112%	0.101%	0.121%	0.144%	0.0993%	0.0514%
Std	0.164%	0.155%	0.232%	0.106%	0.0673%	0.0583%
Min	0.0216%	0.0218%	0.0246%	0.0396%	0.0256%	0.00252%
25%	0.0423%	0.0361%	0.0478%	0.0749%	0.0511%	0.0162%
50%	0.0607%	0.0514%	0.0597%	0.117%	0.0794%	0.0324%
75%	0.109%	0.0923%	0.0997%	0.160%	0.126%	0.0640%
Max	1.16%	1.16%	1.79%	0.616%	0.375%	0.301%

Table 8: Percentage errors of calibrated IVS vs. real IVS

Overall, Table 7 and 8 demonstrate that the proposed calibration framework using neural networks can achieve a satisfactory accuracy level. Table 7 shows that, using NN-based calibration framework, the average parameter error of V_0 can be controlled at 10^{-4} level, corresponding to a magnitude of 0.01% parameter range. The average parameter errors of other parameters can be controlled at 10^{-2} level, corresponding to a magnitude of 1% parameter range. Table 8 shows that, using NN-based calibration framework, the average percentage IVS error is around 0.1%, and it is very close to 0.05% error achieved by the analytical solution-based calibration. Note that the neural network is only an approximation to the analytical solution of Heston model, which means that it can never achieve the same accuracy level as the

analytical solution. Nevertheless, the error is so close that it could be negligible in the real-world application.

Among the five neural networks, Model B performs the best in terms of parameter accuracy, while Model E performs the best in terms of IVS accuracy. Nevertheless, it is worth noting that the distribution of percentage IVS errors is highly skewed for Model A, B, and C. There are extreme cases where the percentage IVS error can reach more than 1%. On the other hand, Model D and E show a normal-like distribution of percentage IVS errors, which is more satisfactory. Hence, we consider Model E as a better model. The comparison between two types of neural networks could also suggest that a parameter-IV model is overall a better approach than a parameter-IVS model.

Computation speed

Computation speed is assessed by the average calibration time of the 100 calibration tests. The results, which are further decomposed into the calibration time per iteration and the number of iterations, are presented in Table 9.

	A	**B**	**C**	**D**	**E**	**Benchmark**
Average time (s)	16.14	25.85	37.11	11.07	15.24	54.24
Average iterations	73.53	73.69	72.76	60.16	63.20	78.64
Time per iteration (s)	0.2195	0.3508	0.5100	0.1839	0.2411	0.6897
Ratio of time per iteration w.r.t benchmark	0.318	0.509	0.739	0.267	0.350	-

Table 9: Average calibration time for theoretical data

Overall, Table 9 shows that the proposed calibration framework using neural networks can significantly reduce the calibration time required. Based on the complexity of neural networks, the calibration time per iteration is reduced to 25 ~ 75% of that using analytical solution method. The number of iterations is also slightly reduced using the NN-based framework, which makes the calibration time required even shorter.

To further understand the reason why it takes shorter time when calibrating with neural networks, we perform a test to obtain the time taken for one pass of objective function in differential evolution.

	A	**B**	**C**	**D**	**E**	**Benchmark**
One-pass time (ms)	2.656	3.937	5.623	2.156	2.919	7.786
Ratio of one-pass time w.r.t benchmark	0.341	0.506	0.722	0.277	0.375	-

Table 10: One-pass time

It is known that the time-consuming step of objective function is to calculate the IVS with trail parameters (i.e., the forward pass of neural networks). Hence, Table 10 shows that it takes shorter time for the neural networks to calculate the IVS than the analytical solution. Moreover, it is noticed that the ratios of time per iteration in Table 9 and one-pass time in Table 10 are close. It suggests that the shorter one-pass time with neural networks contributes largely to the faster computation speed in the model calibration process.

The enhanced computation speed of neural network can be partially attributed to its parallel computing architecture. For the analytical solution, calculating each IV on the IVS requires a loop, and the time consumed increases proportionally with respect to the number of IVs. For Model A, B, and C, the parameters are directly mapped with the IVS, so it is one-time calculation without a loop. For Model D and E, although the neural network has only one IV as the output, it can run multiple predictions in parallel, which significantly reduces the time required for calculating the IVS. That is why it is observed that Model A and E, which have similar model complexity (four 128-neuron layers) and different numbers of predictions required (1 vs. 91), end up with similar processing time in Table 9 and 10.

It should be noticed that all the calibration tests in this study run in the CPU-only environment (Section 3.5). On the other hand, it is suggested by many researchers that GPU has a better capability to do parallel processing with a large population size. Hence, the proposed NN-based calibration framework, especially the one with a parameter-IV model, can take greater advantages when calibrating IVS with more observations (i.e., options with higher liquidity).

4.2.2 Calibration with real market data

Based on the combined results in Section 4.2.1, we identify Model A and Model E as the optimal parameter-IVS and parameter-IV models, which can achieve satisfactory accuracy and fast calibration speed among their respective model groups. The two models, together with the Heston model analytical solution, are used in the calibration of Heston model parameters for the real market IVS.

Calibration accuracy

For real market data, since the real parameters are not observable, calibration accuracy is only assessed by IVS error. The percentage differences between the calibrated IVS and the real IVS of each market IVS data are presented in Table 11.

	A	**E**	**Benchmark**
AAPL@2021/02/12, 48 IV observations	7.16%	3.16%	3.31%
AAPL@2021/02/19, 58 IV observations	7.31%	3.54%	3.60%
AAPL@2021/02/26, 58 IV observations	4.78%	2.69%	2.73%

FB@2021/02/12, 40 IV observations	8.13%	3.89%	3.95%
FB@2021/02/19, 59 IV observations	6.98%	3.50%	3.72%
FB@2021/02/26, 46 IV observations	10.45%	6.10%	6.13%
Average percentage error	5.73%	3.81%	3.91%

Table 11: Percentage errors of calibrated IVS vs. real IVS

In Table 11, the percentage IVS errors are much larger than those in Table 8, and calibration using Model E can achieve the same accuracy level as that using the analytical solution. The larger calibration error can be attributed to Heston model itself, which cannot perfectly model the real-world price movement. As a result, the approximation error of neural networks with respect to the analytical solution is no longer a major source of calibration error.

Compared to Model E, Model A gives much worse accuracy performance. It is possibly because the market observations do not constitute a full IVS that match the size of Model A output. As a result, some parts of the neural network, which calculate the IVs with no real observations, are no use in the calibration process. The overall predicting capability of the neural network is thus reduced, impacting the accuracy of calibration. This result, again, could suggest that a parameter-IV model is overall a better approach than a parameter-IVS model.

Computation speed

The calibration time and the number of iterations for each market IVS data are presented in Table 12.

	A	**E**	**Benchmark**
AAPL@2021/02/12, 48 IV observations	6.57s / 28 iterations	9.04s / 34 iterations	9.90s / 24 iterations
AAPL@2021/02/19, 58 IV observations	7.59s / 32 iterations	8.59s / 35 iterations	10.59s / 22 iterations
AAPL@2021/02/26, 58 IV observations	6.34s / 27 iterations	9.18s / 38 iterations	9.62s / 38 iterations
FB@2021/02/12, 40 IV observations	8.75s / 30 iterations	7.45s / 24 iterations	9.37s / 27 iterations
FB@2021/02/19, 59 IV observations	8.70s / 30 iterations	6.95s / 25 iterations	16.22s / 23 iterations
FB@2021/02/26, 46 IV observations	4.87s / 17 iterations	5.16s / 22 iterations	7.38s / 17 iterations
Average time per iteration	0.261s / iteration	0.261s / iteration	0.418s / iteration
Ratio of time per iteration w.r.t. benchmark	0.624	0.624	-

Table 12: Calibration time for real market data

Table 12 shows that calibration time using Model A and E is lower than that using the analytical solution method, leading to the same conclusion as that in Section 4.2.1. Nevertheless, it is noticed that the ratio of

time per iteration becomes higher, indicating the advantage of time saving is reduced. It can be attributed to fewer IV observations per real market IVS, which benefits more for the method with no parallel computing. This observation reminds the practitioners that the number of available observations may decide which method is optimal to achieve time saving (i.e., the analytical solution is not always the slower one).

Combining the results in Section 4.2.1 and 4.2.2, we select Model E as the optimal model among the five neural networks.

4.3 Framework Limitations

The results in Section 4.2 have shown that the proposed NN-based calibration framework can achieve satisfactory accuracy for both theoretical and real market data, and faster calibration speed than the traditional calibration method. Moreover, although we only demonstrate its application for calibrating Heston model, the same machine learning based approach can be extended to other financial models.

Nevertheless, it should be acknowledged that the calibration framework has a few limitations. Firstly, the training of neural networks uses theoretical data generated by the model parameters within certain ranges. Those ranges limit the minimum and maximum parameters that could be obtained from the calibration process. Hence, researchers need to identify the reasonable ranges of model parameters, which requires a deeper understanding of financial model itself and current market condition. Note that the training samples out of reasonable parameter ranges could have an adverse effect on the model predictivity. In the real world, practitioners should consider timely renewing the model to adjust the parameter ranges based on the current market condition and developing different models for different kinds of assets.

Secondly, feedforward neural networks may not always be the optimal type of neural network architecture for approximating other financial models. The simplicity of one-directional feedforward architecture leads to inherent limitations when addressing certain kinds of problems. For example, a feedforward neural network does not process vectors with multiple dimensions, which limits its capability when dealing with sequential data, such as time series samples. Hence, researchers should select a suitable neural network architecture based on the nature of the financial model to be approximated. For example, recurrent neural networks could be a better choice for approximating the models dealing with time series data.

5. Conclusion

In summary, this study develops a machine learning based framework to calibrate Heston model, and demonstrates its superior performance in terms of calibration accuracy and computation speed. The framework consists of a feedforward neural network, which serves as a close approximation of the analytical solution, and a backward calibration algorithm based on the trained neural network and differential evolution.

The study examines the calibration performance with different neural network specifications, using both theoretical and real market data. It is found that it is optimal to train a parameter-IV type model with moderate complexity (Model E), which has the model and market parameters as input and a single implied volatility as output. The calibration framework with the optimal neural network, can achieve the same accuracy level as calibration through the analytical solution, and significantly reduce the calibration time required to 37.5% (theoretical data) and 62.4% (real market data).

The proposed calibration framework can be extended further to other financial models. Nevertheless, it also has certain limitations. Further works can be done to improve the framework so that it can be applied in the real-world trading scenarios.

References

Balaraman, G. (2016). *Heston Model Calibration Using QuantLib Python and Scipy Optimize*. Retrieved from http://gouthamanbalaraman.com/blog/heston-calibration-scipy-optimize-quantlib-python.html

Black, F., & Scholes, M. (1973). The Pricing of Options and Corporate Liabilities. *The Journal of Political Economy, 81*(3):637–654.

Bollerslev, T. (1986). Generalized Autoregressive Conditional Heteroskedasticity. *Journal of Econometrics. 31*(3): 307–327.

Cox, J. (1975). Notes on option pricing I: Constant elasticity of diffusions. *Unpublished Draft, Stanford University*.

Cox, J. C., Ingersoll, J. E., & Ross, S. A. (1985). A Theory of the Term Structure of Interest Rates. *Econometric*a, *53*(2), 385.

Crissstomo, R. (2014). An Analysis of the Heston Stochastic Volatility Model: Implementation and Calibration Using Matlab. *SSRN Electronic Journal*.

Cui, Y., del Bano Rollin, S., & Germano, G. (2017). Full and fast calibration of the Heston stochastic volatility model. *European Journal of Operational Research, 263*(2):625-638.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals, and systems, 2*(4): 303–314.

Deng, Z. C., Yu, J. N., & Yang, L. (2008). An inverse problem of determining the implied volatility in *option pricing. Journal of Mathematical Analysis and Applications, 340*(1):16 - 31.

Derman, E., & Kani, I. (1994). Riding on a Smile. *Risk, 7*, 32-39.

Dupire, B. (1994). Pricing with a smile. *Risk, 7*(1):18–20, 32, 33.

Feller, W. (1951). Two Singular Diffusion Problems. *The Annals of Mathematics, 54*(1), 173.

Gatheral, J. (2006). *The Volatility Surface: A Practitioner's Guide* (1st ed.). Wiley.

Hernandez, A. (2016). Model Calibration with Neural Networks. *SSRN Electronic Journal*.

Heston, S. (1993). A Closed-Form Solution for Options with Stochastic Volatility with Applications to Bond and Currency Options. *Review of Financial Studies, 6*, 327-343.

Huber, P. J. (1964). Robust Estimation of a Location Parameter. *The Annals of Mathematical Statistics, 35*(1), 73–101.

Ioffe, S. & Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *Proceedings of the 32nd International Conference on Machine Learning, in Proceedings of Machine Learning Research 37*:448-456

Kullback, S., & Leibler, R. A. (1951). On Information and Sufficiency. *The Annals of Mathematical Statistics, 22(*1), 79–86.

Liu, J., & Lampinen, J. (2002). On setting the control parameter of the differential evolution method. *Proceedings of the 8th International Conference on Soft Computing (MENDEL)*, 11-18

Liu, S., Borovykh, A., Grzelak, L. A., & Oosterlee, C. W. (2019). A neural network-based framework for financial model calibration. *Journal of Mathematics in Industry*, *9*(1).

MacKenzie, D. (2006). *An Engine, not a Camera: How Financial Models Shape Markets*. Cambridge, MA: MIT Press. ISBN 0-262-13460-8.

Mrázek, M., & Pospíšil, J. (2017). Calibration and simulation of Heston model. *Open Mathematics*, *15*(1), 679–704.

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural Networks*, *61*, 85–117.

Spiegeleer, J. D., Madan, D. B., Reyners, S., & Schoutens, W. (2018). Machine learning for quantitative finance: fast derivative pricing, hedging, and fitting. *Quantitative Finance*, *18*(10):1635-1643.

Storn, R. (1996). On the usage of differential evolution for function optimization. *Proceedings of North American Fuzzy Information Processing*, 519-523.

Storn, R., & Price, K. (1997). Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, *11*(4):341-359.

Appendix - A

A.1 Characteristic functions for Π_1 and Π_2 by Crissstomo (2014)

$$\Pi_1 = \frac{1}{2} + \frac{1}{\pi}\int_0^{\infty} Re\left[\frac{e^{-i\omega \ln K}\Psi_{\ln S_T}(\omega - i)}{i\omega\Psi_{\ln S_T}(-i)}\right]d\omega$$

$$\Pi_2 = \frac{1}{2} + \frac{1}{\pi}\int_0^{\infty} Re\left[\frac{e^{-i\omega \ln K}\Psi_{\ln S_T}(\omega)}{i\omega}\right]d\omega$$

where

$$\Psi_{\ln S_T}(\omega) = e^{C(t,\omega)\theta + D(t,\omega)V_0 + i\omega \ln[S_0 e^{(r-q)t}]}$$

$$C(t,\omega) = \kappa\left[r_- t - \frac{2}{\sigma^2}\ln\left(\frac{1 - ge^{-ht}}{1-g}\right)\right]$$

$$D(t,\omega) = r_- \frac{1 - e^{-ht}}{1 - ge^{-ht}}$$

$$r_{\pm} = \frac{\beta \pm h}{\sigma^2}, h = \sqrt{\beta^2 - 4\alpha\gamma}, g = \frac{r_-}{r_+}$$

$$\alpha = -\frac{\omega^2}{2} - \frac{i\omega}{2}, \beta = a - \rho\sigma i\omega, \gamma = \frac{\sigma^2}{2}$$

A.2 Hardware infrastructure

	Environment 1	**Environment 2**
Machine	Acer Swift SF314-54	AWS Deep Learning AMI g4dn.xlarge
OS	Windows 10	Windows Server 2016
RAM	8GB	16GB
CPU	Intel(R) Core(TM) i5-8250U	Inter(R) Xeon(R) Platinum 8259CL
CPU speed	1.8 ~ 3.4 GHz	2.5 ~ 3.5 GHz
No. of processors	8 (4 cores × 2 threads)	4 (vCPUs)
GPU	-	NVIDIA Tesla T4 16GB

Table A.1 Hardware infrastructure settings

A.3 Library versions

	Environment 1	**Environment 2**
Python	3.8.2	3.7.6
Conda	-	4.8.2
Tensorflow	2.4.1	2.0.0

Keras	2.4.3	-
Numpy	1.19.5	1.18.1
Pandas	1.2.1	1.2.2
Scikit-learn	0.24.1	0.23.2
Matplotlib	3.3.4	3.3.4
Quantlib	1.21	1.18

Table A.2 Scientific library versions

A.4 Parameter errors for calibrating theoretical data, with detailed statistics

	V_0	κ	θ	σ	ρ
Mean	0.000843	0.020114014	0.0225588	0.037890589	0.033268646
Std	0.001453564	0.019985908	0.072287146	0.087700911	0.055979121
Min	1.23E-06	9.45E-05	0.000104472	0.000171036	0.000162359
25%	0.000270225	0.006694185	0.001427538	0.005842901	0.00982156
50%	0.000554468	0.013725022	0.003647266	0.012854694	0.018105836
75%	0.00097092	0.023930279	0.011998495	0.038362023	0.032225859
Max	0.013852869	0.092357783	0.638934811	0.749239295	0.455793323

Table A.3 Errors of calibrated parameters vs. real parameters (scaled by parameter ranges) for Model A

	V_0	κ	θ	σ	ρ
Mean	0.000795656	0.012482147	0.01537296	0.018927324	0.022354712
Std	0.001099773	0.014109753	0.046053069	0.030134303	0.028140921
Min	4.89E-07	4.89E-05	0.000125234	7.48E-05	0.000672132
25%	0.000290158	0.003081452	0.001153358	0.002857798	0.008575766
50%	0.000539442	0.006829439	0.003152121	0.009453405	0.013749485
75%	0.000908525	0.017276367	0.008591333	0.021461655	0.025718593
Max	0.009937238	0.070249296	0.350474329	0.174634186	0.207527891

Table A.4 Errors of calibrated parameters vs. real parameters (scaled by parameter ranges) for Model B

	V_0	κ	θ	σ	ρ
Mean	0.000966141	0.013407139	0.022983902	0.032790039	0.036682863
Std	0.001778569	0.017959009	0.077470878	0.050086786	0.041808812
Min	9.14E-06	0.000138393	1.28E-05	6.28E-05	0.00025363
25%	0.000276497	0.003471851	0.001401847	0.003769721	0.007779301
50%	0.000613315	0.008955579	0.003305391	0.012877045	0.021837935
75%	0.001119817	0.015138652	0.009519545	0.034968966	0.046549197
Max	0.01701306	0.116341806	0.569775647	0.23125082	0.224456329

Table A.5 Errors of calibrated parameters vs. real parameters (scaled by parameter ranges) for Model C

	V_0	κ	θ	σ	ρ
Mean	0.001359033	0.036889234	0.035758902	0.051527324	0.06038394
Std	0.001099063	0.038111345	0.084091701	0.08144488	0.070400789

Min	4.29E-06	0.001238006	4.34E-05	2.29E-05	0.001034831
25%	0.000453929	0.009241892	0.002744773	0.00888695	0.020757572
50%	0.001244378	0.025855734	0.008218555	0.021935414	0.03552569
75%	0.001812231	0.04550712	0.02609581	0.062230142	0.070961465
Max	0.004919563	0.200774468	0.615625772	0.509518153	0.389845603

Table A.6 Errors of calibrated parameters vs. real parameters (scaled by parameter ranges) for Model D

	V_0	κ	θ	σ	ρ
Mean	0.000683408	0.02441708	0.025828941	0.031679113	0.033622923
Std	0.000652061	0.035483572	0.073151294	0.046494097	0.037001545
Min	4.34E-06	2.29E-05	5.75E-05	0.000240257	4.99E-05
25%	0.000218239	0.006463309	0.001582975	0.006140904	0.0089414
50%	0.000524642	0.014616139	0.004673914	0.01329475	0.023121258
75%	0.000934723	0.027465736	0.015525714	0.04054721	0.041239091
Max	0.003051271	0.239549807	0.608911776	0.272957649	0.212286236

Table A.7 Errors of calibrated parameters vs. real parameters (scaled by parameter ranges) for Model E

	V_0	κ	θ	σ	ρ
Mean	3.59E-05	0.004580419	0.004965908	0.001979289	0.001388234
Std	4.33E-05	0.003741599	0.01120456	0.005227607	0.002049759
Min	5.24E-08	2.83E-05	2.42E-05	5.55E-06	3.12E-06
25%	9.16E-06	0.001576533	0.0006273	8.20E-05	0.000276586
50%	1.84E-05	0.004007417	0.001809042	0.000230644	0.000685676
75%	4.37E-05	0.006764534	0.003620623	0.00124235	0.001511269
Max	0.000221368	0.025032736	0.079576792	0.035159046	0.010072232

Table A.8 Errors of calibrated parameters vs. real parameters (scaled by parameter ranges) for benchmark

Appendix - B

B.1 List of source code files

File name	Directory	Purpose
IVGen.py	-	Generate theoretical parameter-IV pairs for neural network training
IVSGen.py	-	Generate theoretical parameter-IVS pairs for neural network training (also generate data for calibration test)
xxx.npy	data	Training and calibration test data
NNTrain_xxx.py	-	Build and train neural network
xxx.h5	model	Trained neural networks
NNLoss.py	-	Check loss of trained neural networks
NNCalib.py NNCalib_batch.py QLCalib.py	-	Heston model parameter calibration test using theoretical data
xxx.npy	sol	Calibration test results
NNExecTime.py NNExecTime_batch.py QLExecTime.py	-	Check execution time for one pass of objective function in differential evolution
Analysis.py	-	Data analysis of calibration results
queryMktData.py	aapl, fb	Query real market implied volatility data
xxx.npy	aapl, fb	Real Market implied volatility data
Calib_aapl.py Calib_appl_batch.py Calib_fb.py Calib_fb_batch.py	-	Heston model parameter calibration test using real market data
Pricing.py Calibration.py	QuantLib	Sample option pricing and parameter calibration codes

Table B.1 List of source code files