No-Arbitrage Option Pricing with Neural SDEs

Alexio Phytides

A dissertation submitted to the Faculty of Commerce, University of Cape Town, in partial fulfilment of the requirements for the degree of Master of Philosophy.

October 15, 2023

MPhil in Mathematical Finance,
University of Cape Town.

Declaration

I declare that this dissertation is my own, unaided work. It is being submitted for the Degree of Master of Philosophy at the University of Cape Town. It has not been submitted before for any degree or examination in any other University.

Signed by candidate

October 15, 2023

Abstract

Neural stochastic differential equations (SDEs) represent a significant advancement in the field of machine learning by combining the power of neural networks and SDEs, two influential modelling approaches. SDEs are used to model systems that exhibit randomness or uncertainty and are defined by a set of differential equations that describe the evolution of the system over time, along with a random noise term. Neural SDEs extend this framework by using neural networks to model the SDE's drift and/or diffusion coefficients, resulting in a more flexible and powerful modelling approach.

This dissertation delves into the use of neural SDEs for modelling the complex dynamics of asset price processes. Through a thorough examination of various training methodologies for neural SDEs, we aim to develop a more pragmatic approach to training these models, thereby advancing the understanding of neural SDEs and their potential for modelling financial systems. Through numerical experiments, we compare the performance of neural SDEs to well-established models, such as the Black-Scholes and CEV models, using European call option prices computed from neural SDE generated stock prices. The numerical experiment results suggest that neural SDEs are a promising tool for understanding the behaviour of complex, dynamic systems, and may offer improved accuracy and flexibility compared to traditional option pricing approaches. Overall, this work provides insight into the use of neural SDEs for modelling the intricacies of financial systems and other types of dynamic processes.

Acknowledgements

To the AIFMRM department, I extend my sincerest gratitude for creating a truly exceptional Masters program. I am grateful for being part of the enriching learning experience it provided. Special recognition goes to my supervisor, Dr. Ralph Rudd, for his unwavering guidance and support throughout my dissertation journey. I am deeply grateful for the opportunity to work on such a thought-provoking research topic under his guidance. Lastly, I would like to express my appreciation to my friends and family for their continuous support and encouragement throughout this journey. This accomplishment would not have been possible without the contributions of all who helped me along the way.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Financial option pricing has been a heavily researched topic in the field of quantitative finance over the last few decades. The interest in this topic has surged since 1973, when the trading of financial options was officially standardised, marking the commencement of the modern era of options trading. Since then, options have become extremely prominent in the finance industry, to the extent that in many cases more money is invested in options than in their corresponding underlying assets. The widespread popularity of options is reflected in the 2021 global options trading volume of approximately 38.88 billion contracts, as reported by The World Federation of Exchanges (2022). The appeal of options trading predominately stems from their ability to effectively mitigate risk and provide cost-efficient trading opportunities for investors. While financial option contracts can encompass a wide range of complexity, a vanilla option gives the holder the right, but not the obligation, to buy or sell an underlying asset at a specified price and on a predetermined date in the future. This type of option serves as the basic building block for many more complicated contracts widely adopted in the finance industry.

The study of option pricing within the field of mathematical finance can be traced back to the year 1900, when Bachelier (1900) laid the foundations by introducing the first option pricing model. However, it was not until the 1970s that the field truly gained momentum following the revolutionary work of Black and Scholes (1973), and Merton (1973), who proposed a model for pricing European-style options, which is widely regarded as the cornerstone of modern-day option pricing. Despite the significance of these early works, ongoing research continues to push the boundaries in search of more accurate and computationally efficient methods of option pricing computation. The need for rapid and accurate pricing and hedging of options necessitates ongoing research, as practitioners and researchers work to identify the optimal models that meet the changing needs of the industry.

One of the more recent developments in quantitative finance is the increasing use of machine learning techniques to solve the problem of pricing options. This

has been driven by a combination of theoretical advancements, as well as developments in hardware and software, which have overcome past deficiencies in implementing machine learning, enabling the use of more complex models for option pricing.

This leads to the focus of this dissertation, which draws on a recent development in machine learning techniques, by tackling the problem of financial option pricing with neural stochastic differential equations (SDEs). Neural SDEs are a novel concept at the intersection of the fields of deep learning, stochastic processes and differential equations (Kidger *et al.*, 2021). They can broadly be decomposed into the two dominant modelling paradigms of SDEs and neural networks.

In this dissertation, we delve into the intricacies of neural SDEs, exploring the synergies between the two powerful modelling paradigms that form their foundations, as outlined in Chapter 2. To gain a comprehensive understanding of the various approaches adopted to tackle the problem of pricing financial options with neural SDEs, Chapter 3 conducts a thorough literature review. Building on this foundation, Chapter 4 presents this dissertation's proposed approach for training neural SDEs, complete with a discussion of the constraints imposed to ensure fair pricing without any opportunities for arbitrage. To test the efficacy of the neural SDE methodology, Chapter 5 conducts numerical experiments using known processes with well-understood dynamics and closed-form solutions to option pricing, as benchmark models. We compare the solutions obtained using the proposed neural SDE approach with these benchmarks, using the Black-Scholes model and constant elasticity of variance model as reference points. Finally, Chapter 6 synthesises the key findings and observations made throughout the dissertation, providing a comprehensive summary of this research.

Overall, the purpose of this dissertation is to address the issue of option pricing in finance, by investigating a modern approach, using neural SDEs, that is both theoretically grounded and practically feasible.

Chapter 2

Background

In this chapter, we explore the foundational aspects of neural SDEs. We begin by examining the two key modelling paradigms of SDEs and neural networks. We then proceed to define neural SDEs and explore their utility in pricing financial options. Additionally, we delve into the principle of arbitrage-free pricing and illustrate how neural SDEs can be used to produce accurate and fair option prices, given specific constraints.

In this chapter, we reference the works of Björk (2004) and Bishop (2006) as the primary sources for important definitions and theorems. We simply present the key results without proof, providing these references for readers seeking a deeper understanding of the stated results.

2.1 Stochastic Differential Equations

The application of stochastic processes in mathematical finance has seen phenomenal growth over the last decades. This upsurge of interest stems from the field of option pricing, which often adopts pricing techniques that necessitate the specification of a stochastic process governing the behaviour of the option's underlying asset through time. Put simply, a stochastic process is a collection of random variables indexed by time. In this dissertation, we specifically examine the role of stochastic processes in the context of SDEs. SDEs are mathematical models that can be used to understand and predict the behaviour of systems that exhibit uncertainty or randomness. These models consist of a set of differential equations that describe how the system changes over time, along with a random noise term that captures the inherent unpredictability of the system. SDEs are particularly useful for modelling phenomena such as fluctuating stock prices and other types of random processes, as they provide a framework for understanding and predicting the evolution of such systems. The solution to a SDE is itself a stochastic process, reflecting the inherent randomness of the modelled system.

Definition 2.1.1. We consider a specific form of a SDE, which is a one-dimensional process with a one-dimensional driving Brownian motion, for all $t \geq 0$, given by

$$\begin{aligned} dX_t &= \mu(t, X_t)\, dt + \sigma(t, X_t)\, dW_t, \\ X_0 &= x_0, \end{aligned} \tag{2.1}$$

where $\mu(t, X_t)$ and $\sigma(t, X_t)$ are referred to as the drift and diffusion coefficients respectively, and W_t denotes a Wiener process. This equation should be interpreted as an informal way of expressing the corresponding integral equation,

$$X_t - X_0 = \int_0^t \mu(u, X_u)\, du + \int_0^t \sigma(u, X_u)\, dW_u,$$

which characterises the behaviour of the continuous-time stochastic process X_t.

The price of an asset is often characterised by sudden jumps and fluctuations, making it a non-smooth function of time. Therefore, standard calculus tools cannot be used to effectively model its behaviour through time. As a result, the field of financial asset pricing borrows heavily from a branch of mathematics known as stochastic calculus. A key result in stochastic calculus is Ito's Lemma, which provides a framework to determine the derivative of a time-dependent function of a stochastic process (Itô, 1944). By applying Ito's Lemma, it is possible to derive new SDEs from existing ones, making it a powerful tool in the analysis of financial systems and other dynamic systems that exhibit randomness.

2.2 Neural Networks

Parallel to the widespread adoption of machine learning, there has been considerable interest in the development of neural networks for solving a variety of problems. Neural networks represent a class of robust, non-linear models inspired by the neural architecture of the brain, which are capable of modelling complex non-linear relationships.

2.2.1 Neural Network Architectures

Neural networks are comprised of three distinct layers of neurons[1], an input layer, one or several hidden layers and an output layer. However, linearly separable data does not necessitate the use of any hidden layers in a neural network (Karsoliya,

[1] A neuron is a processing unit that receives inputs, processes them using a set of learned weights and biases, and produces an output. The combination of multiple neurons allows the network to learn complex, non-linear relationships between inputs and outputs.

2012). Deep neural networks form a subset of neural networks that have multiple hidden layers between the input and output layers. The greater the number of layers to be processed, the deeper the network. If for a given layer, all the neurons in one layer are connected to all the neurons in the next layer, we refer to this as a fully-connected network. In the numerical experiments conducted in this dissertation, we adopt a class of fully-connected feed-forward neural networks, also known as multilayer perceptrons. These are specific types of fully-connected neural networks in which the connections between the neurons do not form a cycle, hence information flows in one direction only, from the input layer to the output layer, without any back loops, as illustrated in Figure 2.1.

Figure 2.1 presents an intuitive visualisation of the architecture of a multilayer perceptron. This illustration serves as a visual introduction to the formal definition of a deep neural network model, which is detailed in Definition 2.2.1.

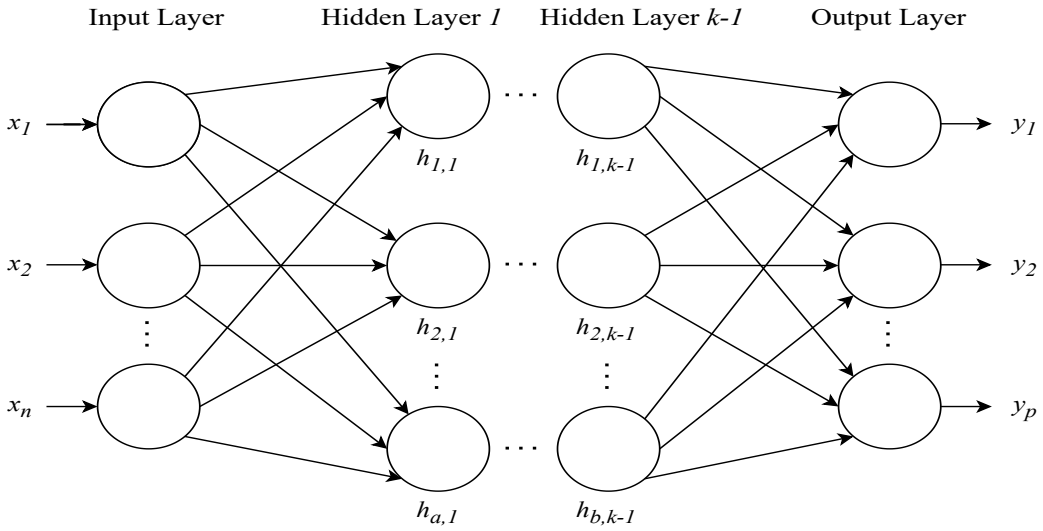

Fig. 2.1: Neural architecture of a multilayer perceptron.

Definition 2.2.1. A k-layer neural network can be represented mathematically as a function $f : \mathbb{R}^n \to \mathbb{R}^p$, which is a composition of multiple non-linear multivariate functions $f_1, f_2, \ldots, f_k$, and g. As a result, a k-layer neural network can be defined as $f(x) = g \circ f_k \circ \cdots \circ f_2 \circ f_1(x)$, where $f_i : \mathbb{R}^{n_i} \to \mathbb{R}^{n_{i-1}}$, for $i \in \{1, 2, \ldots, k\}$, such that $f_i(x) = \gamma(w_i x + b_i)$. This can be represented as

$$\begin{aligned} f(x) &= g \circ f_k \circ \cdots \circ f_2 \circ f_1(x) \\ &= g(\gamma(\ldots \gamma(w_2 \gamma(w_1 x + b_1) + b_2) \cdots + b_k)). \end{aligned}$$

In Definition 2.2.1, n is the dimension of the input layer, p is the dimension of the output layer, g is the output function, which maps the output of the neural network to the desired format, $w_i x + b_i$ is a linear combination of the input x with its coefficients w_i, plus a bias term b_i and finally, γ is the activation function, which introduces non-linearity in the neural network.

In this dissertation, we focus on the class of multilayer perceptrons, as illustrated in Figure 2.1. We specifically examine the architecture of multilayer perceptrons which consist of a single neuron in both the input and output layers. The number of hidden layers and neurons in each hidden layer will be optimised for each numerical experiment. Careful selection of hidden layers and neurons ensures that the neural network can effectively capture the intricacies of the underlying data set, while avoiding over-fitting and excessive computational demands by not having too many neurons in each of the hidden layers (Caruana *et al.*, 2000).

The predictive capacity of a neural network is so compelling, that any class of problems that can be reduced to a function, in theory, can be solved to an arbitrary degree of accuracy with the use of neural networks (Hornik *et al.*, 1989). This result is expressed in the Universal Approximation Theorem for Neural Networks, which demonstrates the ability of neural networks to approximate any continuous function to an arbitrary degree of accuracy, making them unique systems of functions that are universal approximators (Hornik *et al.*, 1989).

2.2.2 Training Neural Networks

Training a neural network refers to the process of modifying the weights such that the network becomes an optimal approximator to the target function. In this dissertation, we consider neural networks in the context of supervised learning, where the role of the network is to predict the output variables given values for the input variables, which are labelled for a particular output (Bishop, 2006). The training process can be simplified, by expressing it as the problem of optimising an objective function represented by $f(\theta)$, where $f : \mathbb{R}^n \to \mathbb{R}$ is defined over some domain $\theta \in \Theta$. Here, $\theta \in \Theta$ represents all the tune-able weights of each network that are adjusted during training, in order to help reconcile the differences between the actual and predicted outcome. This can be formulated as the task of finding the minimiser $\theta^* \in \arg\max_{\theta \in \Theta} f(\theta)$ (Andrychowicz *et al.*, 2016).

In optimisation, the mean square error (MSE) is a widely used objective function, that allows us to determine the extent to which the neural network is able to accurately predict the outcomes. It is calculated by taking the average of the square of the differences between the predicted and actual values. Therefore, the MSE can

be represented mathematically by

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

where n is the size of the training data set, y_i are the observed values, and $\hat{y}_i$ are the predicted values that are output from the neural network.

A common approach to minimising differentiable objective functions is gradient descent. This method iteratively updates the weights of the network, by computing the gradient of the objective function with respect to the weights and moving in the direction of the negative gradient. This process is repeated until the objective function is minimised and the network's performance is optimised, resulting in a sequence of updates given by

$$\theta_{t+1} = \theta_t - \alpha_t \nabla f(\theta_t).$$

If the objective function, $f(\theta)$, is differentiable with respect to its parameters, gradient descent is a relatively efficient optimisation method. However, Glorot and Bengio (2010) show that standard gradient descent from random initialisation produces poor results with deep neural networks. This is because standard gradient descent is hampered by the fact that it only makes use of gradients and ignores second-order information (Andrychowicz *et al.*, 2016). Therefore, when dealing with deep neural networks in this dissertation, we will make use of alternative optimisation algorithms which overcome this shortcoming of the standard gradient descent algorithm. Specifically, we will make use of the Adaptive Moment Estimation (Adam) optimisation algorithm, which is a technique of stochastic optimisation, introduced by Kingma and Ba (2014). Adam overcomes the shortcomings of standard gradient descent, as it uses estimations of the first- and second-order moments of the gradient to adapt the learning rate[2] for each weight of the neural network (Kingma and Ba, 2014). It is an extension of the stochastic gradient descent algorithm that incorporates the benefits of both the Adaptive Gradient Algorithm (AdaGrad) and Root Mean Square Propagation (RMSProp) (Kingma and Ba, 2014). In addition, it also incorporates a momentum term, which helps the optimiser to converge faster and be less sensitive to the choice of hyperparameters, by allowing it to overcome the problem of oscillation around the optimal solution, making it a more robust and efficient optimisation method (Kingma and Ba, 2014).

[2] The learning rate is a hyperparameter that controls the step size at which the optimiser makes updates to the model weights during training.

2.3 Neural SDEs

With the foundational aspects in place, we now explore the focus of this dissertation, neural SDEs. Neural SDEs adopt the framework of SDEs, while incorporating neural networks to model the dynamics of the underlying system. This allows the model to capture complex, nonlinear relationships between the variables in the system. In a neural SDE, the drift and/or diffusion coefficients, which capture the behaviour of the underlying process, are represented using neural networks, allowing for a more flexible and powerful modelling approach.

Definition 2.3.1. We consider a specific form of a neural SDE, which is a one-dimensional process with a one-dimensional driving Brownian motion, given by

$$dX_t^{\phi,\theta} = \mu(t, X_t^{\phi}, \phi)\, dt + \sigma(t, X_t^{\theta}, \theta)\, dW_t. \tag{2.2}$$

From Equation (2.2), we can deduce that a neural SDE takes on the form of a SDE, as given in Equation (2.1), but where the drift and/or the diffusion processes are given by neural networks parameterised by ϕ and/or θ, respectively. In Equation (2.2), ϕ and θ represent all the tune-able weights of each network, that are adjusted during training. An imperative component of using neural SDEs is that we can calibrate a neural SDE to market variables. This allows us to price options on the asset using martingale pricing theory, but without explicitly specifying (and thus potentially oversimplifying) the dynamics of the asset price.

Over time, the finance industry has adopted machine learning to varying degrees of sophistication. However, it is important to highlight that there are still unique barriers to the adoption of such technology. One of the major obstacles to the adoption of machine learning in the finance industry is the opaque nature of many machine learning models, also known as the "black-box" problem. These models can be quite complex and not easily interpretable, making it difficult for practitioners to understand how the model arrives at a particular decision and identify potential biases or errors in the model. This can be a significant barrier for organisations that need to ensure the transparency and explainability of their decision-making processes, especially in regulated industries. This longstanding concern is emphasised by the introduction of the European Commission's proposed Artificial Intelligence (AI) Act, which attempts to regulate a wide range of AI applications (Veale and Borgesius, 2021). This legislation establishes the first law on AI by a major regulator, which could pave the way for the widespread implementation of AI-related regulation across key industries.

Adopting a neural SDE approach to pricing options, presents several advantages. Firstly, this approach offers a fresh perspective on a classical problem in

finance, thus allowing for the consideration of an alternative method for pricing options. Additionally, through the use of neural SDEs to price options, we mitigate to some extent, the concerns that regulators have around the use of "black-box" models, as this approach combines neural networks with well-established risk models based on classical SDEs. This is opposed to more traditional machine learning approaches, that use neural networks to approximate the entire option pricing process. Finally, by design, neural SDEs are data-driven, and are therefore adaptable to a change in the environment, allowing for a more flexible modelling approach.

2.4 No-Arbitrage Pricing

In this dissertation, we ensure an arbitrage-free setting when pricing options. By adhering to the principle of arbitrage-free pricing, it is ensured that the option prices produced under the adopted option pricing methodology do not permit traders in the market to make a risk-free profit through an arbitrage strategy.

To ensure that the neural SDE model does not produce any arbitrage opportunities, we construct a class of models that permit no-arbitrage when pricing options, by specifying certain conditions on the form of the neural SDE, described in Equation (2.2). To ensure that this framework holds, we adhere to one of the main pillars supporting the modern theory of mathematical finance, given in Theorem 2.4.1, when formulating a pricing strategy (Björk, 2004).

Theorem 2.4.1 (First Fundamental Theorem of Asset Pricing)**.** A model does not permit a free lunch with vanishing risk, if and only if there exists an equivalent martingale (risk-neutral) measure.

Definition 2.4.1. A measure $\mathbb{Q}$ is said to be risk-neutral, if under $\mathbb{Q}$, the discounted asset price is a martingale.

Under Theorem 2.4.1, ensuring no-arbitrage ultimately requires that all discounted assets be martingales under the risk-neutral measure $\mathbb{Q}$. Therefore, to ensure an arbitrage-free setting when pricing options with the neural SDE governing the behaviour of the underlying asset, we model assets under the risk-neutral measure. We achieve this by ensuring the discounted asset price is a martingale under the risk-neutral measure, by specifying certain conditions on the form of the neural SDE, which is further detailed in Section 4.1.2.

Chapter 3

Literature Review

This chapter explores the literature on the topic of neural SDEs and the various approaches that different studies adopt to tackle the problem of pricing options with neural SDEs. The aim of this chapter is to provide a comprehensive understanding of the existing literature on methods of using neural SDEs to price options and to position this dissertation's proposed methodology within this context.

3.1 Model Calibration with Neural Networks

At the forefront of the application of neural networks in the calibration process, is a breakthrough paper by Hernandez (2016). Hernandez (2016) proposes a method of using neural networks to learn the calibration map from market data (such as stock or option prices) directly to a parametric model. Hernandez (2016) achieves this by proposing a method to calibrate models using neural networks, which can perform the calibration process significantly faster than traditional calibration methods once the initial training process is complete, regardless of the parametric model. This is a remarkable result as it removes, to some extent, the calibration speed as a consideration for a model's usability, by moving the computationally heavy calibration process offline, thus significantly accelerating the process of calibrating models.

Subsequent to the breakthrough paper by Hernandez (2016), many papers proposed methods to build on this approach. Dimitroff *et al.* (2018), Liu *et al.* (2019) and Stone (2020) are some of the notable examples that proposed innovative methods for using neural networks in the calibration process. However, while these approaches present significant methodologies for the application of neural networks in the calibration process, by primarily focusing on calibrating fixed parametric models, these approaches do not address perhaps even the more important aspects of model selection and model uncertainty (Gierjatowicz *et al.*, 2020).

3.2 Alternative Approaches to Neural SDEs

The introduction of neural SDEs stems from the preceding research in neural ordinary differential equations (ODEs). Neural SDEs have gained significant attention in the field of financial option pricing in recent years. This is due to the ability of neural SDEs to provide a versatile framework for modelling the dynamics of financial systems, as they allow for the modelling of complex and non-linear systems, while also providing a way to handle model uncertainty and model selection. These features make neural SDEs a promising tool for pricing financial options and solving other problems in finance.

The approach taken by pricing financial options with Neural SDEs is fundamentally distinct from the approaches mentioned in Section 3.1. Through the use of neural SDEs, instead of specifying a fixed parametrisation for the model SDEs, we let the data dictate the model, while still keeping a strong prior on the model form. We are able to not specify a fixed parametrisation for the model SDEs, as instead of specifying the model dynamics, we allow the drift and/or diffusion coefficients to be given by neural networks, as shown in Equation (2.2).

3.2.1 Neural SDEs Derived from Neural ODEs

Since their introduction, the research behind neural ODEs has progressed at a rapid pace, prompting proposals of a wide variety of similarly inspired models (Chen *et al.*, 2018). In particular, several authors have introduced the concept of neural SDEs, by building off the breakthroughs made in the field of neural ODEs. This is possible as SDEs provide a link between probability theory and earlier, more developed fields of ordinary and partial differential equations. Therefore, by noting the relationship between ODEs and SDEs, direct links can be formed with neural ODEs and neural SDEs.

Tzen and Raginsky (2019) approached the problem of obtaining neural SDEs from two alternative perspectives. One perspective is by viewing neural SDEs as a continuous limit of deep latent Gaussian models, which is based heavily on the research conducted by Rezende *et al.* (2014). The alternative approach is to view neural SDEs as a stochastic version of the neural ODE proposed by Chen *et al.* (2018), by building off the relationship between ODEs and SDEs.

3.2.2 Generative Modelling

An alternative approach to the problem of constructing neural SDEs, is by connecting neural SDEs to the concept of generative modelling. This approach is adopted by Gierjatowicz *et al.* (2020), who additionally follow the robust finance paradigm,

where instead of computing a single arbitrage-free price corresponding to the option, they provide robust arbitrage-free price bounds. To validate the performance of their adopted neural SDE option pricing methodology, numerical assessments are performed and it is shown that these price bounds are sufficiently tight around the closed-form option pricing solution, providing strong evidence for the accuracy and reliability of the proposed methodology (Cox and Obloj, 2011). The main aim of the methodology proposed by Gierjatowicz *et al.* (2020), is to combine neural networks with risk models based on classical SDEs, where we can find robust bounds for the prices of options. Under this methodology, previously proposed methods are enhanced, by incorporating relevant market data (such as option prices) to tighten the pricing interval.

An alternative approach to the problem of constructing neural SDEs as generative models is explored by Kidger *et al.* (2021). Kidger *et al.* (2021) show that SDEs and generative adversarial networks (GANs) follow a similar formulation. Using this connection, they outline how to directly train neural SDEs as continuous time, infinite-dimensional, time series GANs. Kidger *et al.* (2021) make this connection by noting that SDEs have inherent randomness, and are distributions over paths or across time series, whereas in modern machine learning terminology, we refer to this as a generative model. Kidger *et al.* (2021) uses this connection to show how modern machine learning of GANs and the traditional way of fitting SDEs operate equivalently. Subsequently, they put the pieces together and demonstrated how to train neural SDEs as generative time series models.

DeLise (2021) proposes an alternative approach where it is assumed that the underlying price process is directly modelled by a neural SDE. DeLise (2021) does this by adopting a simple training scheme that is theoretically analogous to the SDE-GAN approach proposed by Kidger *et al.* (2021), but implements the Wasserstein distance metric as a loss function for training, which measures the distance between probability measures over a metric space. DeLise (2021) adopts an approach that does not require option price data for training, and therefore can be considered an unsupervised learning approach to the problem of option pricing with neural SDEs.

Chapter 4

Research Methodology

This chapter describes the approach to option pricing with neural SDEs that will be investigated in this dissertation. This chapter also specifies the constraints that will be placed on the neural SDE, to ensure that we are producing option prices that are arbitrage-free. Furthermore, we provide an outline of the benchmark models used in the numerical experiments conducted to assess the performance of the proposed option pricing methodology.

4.1 Option Pricing Methodology

The pieces are now in place to describe the main contribution of this dissertation, which is a proposed methodology to option pricing with neural SDEs. This dissertation approaches pricing options using neural SDEs by breaking down the problem into several initial steps, to build up to a desired methodology that is both consistent with no-arbitrage option pricing theory and of practical use, by phasing out the assumptions that the initial approaches rely on.

4.1.1 Training the Neural SDE

Training a neural SDE refers to training the neural network components of the neural SDE, which involves using an optimisation algorithm to find a set of weights that best map a collection of inputs to a collection of outputs, as outlined in Section 2.2.2 (Bishop, 2006). The optimisation algorithm essentially tunes the neural network components of the neural SDE in an effort to minimise the objective function.

The first step to breaking down the training methodology of the neural SDE will be to assume that the dynamics of the underlying asset price process take on a known form. This assumption allows us to simplify the problem and focus on using the neural network components of the neural SDE to approximate the known forms of the drift and/or diffusion coefficients, by the Universal Approximation Theorem for Neural Networks. This approach allows us to gradually build up to

a more comprehensive methodology for training neural SDEs and ensures a solid foundation for the later stages of the research.

The next step is to adopt a method of moments training scheme theoretically analogous to the approach presented by Rackauckas *et al.* (2020). This training methodology aims to match the underlying process's first- and second-order moments to the neural SDE. Through implementing this approach, we build upon the initial step, but instead of assuming a specific functional form for the underlying asset price process, we assume the means and variances of the process must either be known or have the capability to be precisely calculated.

The initial approaches lay the foundation to establish a more practical training methodology, by verifying the neural SDE technique and implementation. We propose a training scheme fundamentally distinct from the initial approaches, in the sense that it does not require any assumptions on the form of the underlying asset price process. The proposed training methodology utilises observable option prices to directly train the neural SDE, providing a methodology that is easily adoptable in practice as the training data can be conveniently observed in the market. A key benefit of adopting such an approach is that in addition to the calibration process, model selection and model uncertainty are done simultaneously. In this sense, the model selection process is data-driven (Gierjatowicz *et al.*, 2020).

4.1.2 No-arbitrage Constraints

To ensure that we are implementing the neural SDE option pricing methodology in an arbitrage-free setting, we construct a class of models that permit no arbitrage opportunities. We achieve this by specifying certain constraints on the form of the neural SDE, as detailed in Section 2.4. The approach to specifying these constraints on the neural SDE that this dissertation adopts is theoretically analogous to the no-arbitrage option pricing approach proposed by Gierjatowicz *et al.* (2020).

To ensure the neural SDE model permits no arbitrage opportunities, we reformulate the neural SDE, previously defined by Equation (2.2), as

$$dS_t^\theta = rS_t^\theta\, dt + \sigma^S\left(t, S_t^\theta, \theta\right) dW_t. \tag{4.1}$$

In Equation (4.1), $\sigma^S\left(t, S_t^\theta, \theta\right)$ represents a neural network as the diffusion coefficient of the neural SDE, where θ represents the tune-able weights of that network. We assume an annual continuously compounded risk-free rate of interest, which is assumed to be constant, given by $r \in \mathbb{R}$. Therefore, the no-arbitrage constraints imply that we will be solely concerned with the neural SDE described in Equation (4.1), to govern the dynamics of the underlying asset price process. Under these constraints, we let the drift term be given by a fixed parameterisation rS_t^θ, while the

diffusion coefficient is given by a neural network. To justify that under these constraints an arbitrage-free setting is ensured, through an application of Ito's Lemma, it is demonstrated in Appendix A.1 that the discounted asset price governed by the neural SDE dynamics, is drift-less and hence a (local) martingale. This implies that we are working under the risk-neutral measure by Definition 2.4.1 and hence, by Theorem 2.4.1 the model is arbitrage-free.

4.1.3 Simulating Stock Price Paths

Now that we have specified the neural SDE governing the dynamics of the underlying asset price process, the subsequent step in the option pricing algorithm is to simulate stock price paths from the neural SDE. While this dissertation solely focuses on pricing European call options, which are only concerned with the value of the underlying asset at expiry, we still generate entire stock price paths. This is done as training the neural SDE under the method of moments training scheme, as described in Section 4.1.1, requires sample means and variances of the underlying stock price process to be computed at different time points. Additionally, by generating entire stock price paths from the neural SDE, we demonstrate the versatility of this methodology in being able to price path-dependent options. These are a subset of options whose payoff is dependent on the path the underlying asset's price takes over its life or at certain times during the option's life.

In this section, we consider the problem of generating a sample path of the underlying stock price, up to some future time $T > 0$, for the process determined by the neural SDE, given by Equation (4.1). Specifically, we focus on simulating stock prices for some $N \in \mathbb{Z}^+$, given by the sequence $\{S_{t_0}, S_{t_1}, \ldots, S_{t_N}\}$, at time points $0 = t_0, t_1, \ldots, t_N = T$, with fixed time increments $\Delta t_i = t_i - t_{i-1}$ and where the initial stock price S_0, is known. Here, as we assume that S_0 is known, N is the number of time increments where we compute the price of the underlying stock. Given that the neural SDE in Equation (4.1) does not have a closed-form solution, it is necessary to explore different approximation schemes to simulate stock price paths that are governed by the dynamics of the neural SDE.

The two approximation schemes reviewed in this dissertation are the Euler-Maruyama method and the Milstein method (Kloeden and Platen, 1992). These methods present effective approximation schemes, by providing a balance between accuracy and computational efficiency. While there is substantial literature on alternative higher-order approximation schemes for solving SDEs, in the context of this dissertation, we are only aiming to compute an estimation of the expected value of the solution to the SDE, as we use Monte Carlo methods, which is further detailed in Section 4.1.4. Therefore, as Monte Carlo methods require an estimation

of the expected value of the solution, it is indeed sufficient to be able to accurately sample random trajectories of the SDE, instead of relying on an accurate pathwise approximation of a particular trajectory (Giles, 2008). The former is exactly what a solver with high weak-order provides, making it sufficient to restrict this dissertation to the Euler-Maruyama and Milstein methods for providing approximation schemes to solving SDEs, as opposed to seeking higher-order schemes with more computationally demanding algorithms. However, for further details on higher-order approximation schemes, interested readers are encouraged to see Kloeden and Platen (1992).

Figure 4.1 illustrates the approximate sample paths generated from both the Euler-Maruyama and Milstein approximation schemes, against a sample path corresponding to the analytical solution of the geometric Brownian motion process, given in Equation (4.5). The error plot displays the $\log_{10}$ absolute error of the respective approximation scheme compared to the analytical solution, where the absolute value of this error gives an indication of how close the prediction is to the analytical solution in terms of decimal place accuracy. Evidently, the inclusion of an additional correction term in the Milstein method leads to improved convergence properties and a more accurate approximation to the analytical solution over the entire stock price path, as shown by the error plot in Figure 4.1.

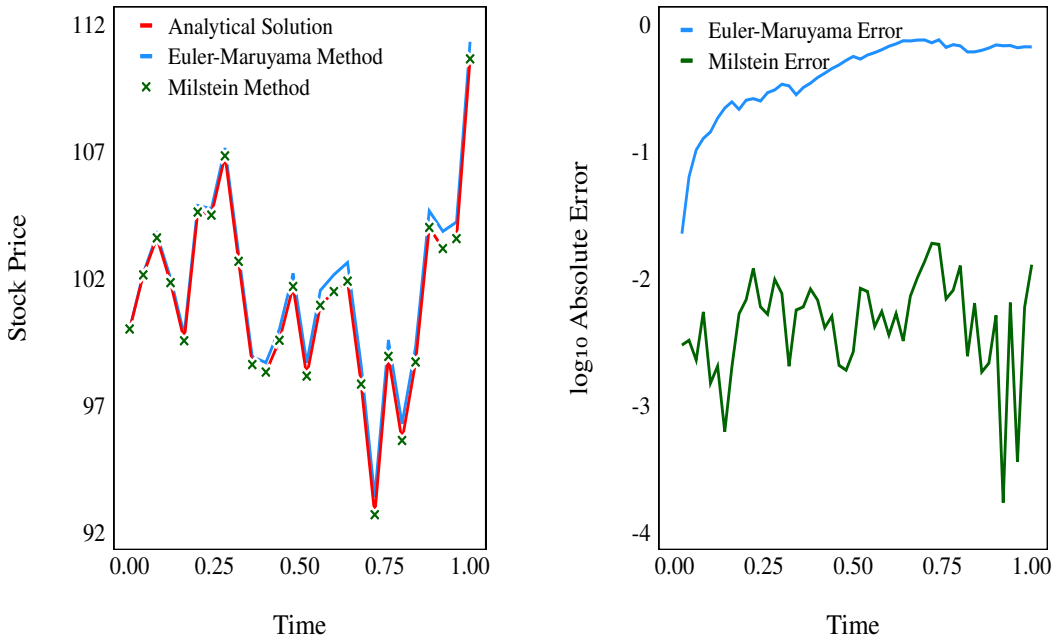

Fig. 4.1: Comparison of approximation schemes to geometric Brownian motion.

4.1.4 Generating European Call Option Prices

In this dissertation, we will be restricted to pricing European call options on a non-dividend paying stock, whose price process is specified by a neural SDE, as defined in Equation (4.1). We implement this restriction for simplicity, as the focus of this dissertation is to establish the underlying asset dynamics through neural SDEs, and ultimately price options using these dynamics. Once the underlying asset's dynamics have been established, pricing any non-path-dependent option follows the same formulation as pricing European call options, with a simple modification of the payoff corresponding to the specific type of option.

We now consider the problem of pricing a European call option on a non-dividend paying stock, whose price process is specified by a neural SDE. Suppose the option is characterised by strike price K, maturity $T > 0$ and that the annual continuously compounded risk-free interest rate is r. For a concise description of the Monte Carlo approach we adopt to pricing options with neural SDEs, a summary of the option pricing methodology is provided below, in Algorithm 1.

Algorithm 1 Risk-neutral call option pricing algorithm.

Let n represent the number of simulated terminal stock prices. We then compute option prices at time $t = 0$, for various strikes, $K \in \mathbb{R}$, by implementing the following steps for each K, while keeping the parameters S_0, r, σ and T fixed.

1. Specify the neural SDE governing the dynamics of the underlying asset price process, given by Equation (2.2).
2. Apply the no-arbitrage constraints on the neural SDE to ensure an arbitrage-free setting, as specified by Equation (4.1).
3. Train the neural SDE according to a training algorithm specified in Section 4.1.1.
4. Simulate n terminal stock prices, given by $\left\{\hat{S}_T^1, \hat{S}_T^2, \ldots, \hat{S}_T^n\right\}$, according to one of the approximation schemes for solving SDEs, as specified in Section 4.1.3.
5. Calculate the payoffs of the European call options corresponding to each of the n terminal stock prices, and discount the payoffs at the risk-free rate r.
6. Calculate the average over the n discounted payoffs, which will produce the Monte Carlo estimate of the call option price $\hat{C}_n(S_0, r, \sigma, T, K)$, which is given by

$$\hat{C}_n(S_0, r, \sigma, T, K) \coloneqq \frac{1}{n} \sum_{i=1}^{n} e^{-rT} \left(\hat{S}_T^i - K\right)^+. \tag{4.2}$$

4.2 Benchmark Models

This dissertation aims to firmly establish the efficacy of using neural SDEs to price options, as described by the methodology outlined in Section 4.1, by considering various proof-of-concept experiments. In order to do this, we conduct numerical experiments by drawing empirical samples from a known process that has well-understood dynamics and a closed-form solution to the option pricing problem. This allows us to execute the methodology for pricing options with neural SDEs and compare the results to a benchmark solution.

In order to investigate the claims of the proposed methodology for pricing options with neural SDEs, the numerical experiments conducted in this dissertation consider the Black-Scholes model, also known as the Black-Scholes-Merton model, and the constant elasticity of variance model, as benchmark models for comparison to the option prices computed under the neural SDE.

4.2.1 The Black-Scholes Model

The Black-Scholes model is a mathematical model for pricing financial instruments. The theory behind the model was introduced by Black and Scholes (1973), and shortly thereafter Merton (1973) extended their work and coined the phrase the Black-Scholes options pricing model. The Black-Scholes model is widely recognised as one of the most important contributions to modern mathematical finance theory, as it allows for analytical simplicity and traceability in pricing European options. Black and Scholes (1973) present the European call option pricing result, given below in Proposition 4.1.

Proposition 4.1 (The Black-Scholes option pricing formula)**.** Let $C(S_t, r, \sigma_{BS}, T, K)$ denote the price at time t of a European call option, given that the current stock price is S_t, the time of maturity is $T > t$, the exercise (strike) price is given by K, the volatility of the underlying asset is given by σ_{BS}, which is assumed to be constant, and r denotes the annual continuously compounded risk-free rate of interest. For such a call option, the price at time t is given by

$$C(S_t, r, \sigma_{BS}, T, K) \coloneqq S_t \Phi\left(d_1\right) - K e^{-r(T-t)} \Phi\left(d_2\right), \tag{4.3}$$

where

$$\begin{aligned} d_1 &\coloneqq \frac{\log \frac{S_t}{K} + \left(r + \frac{1}{2}\sigma_{BS}^2\right)(T-t)}{\sigma_{BS}\sqrt{T-t}}, \\ d_2 &\coloneqq \frac{\log \frac{S_t}{K} + \left(r - \frac{1}{2}\sigma_{BS}^2\right)(T-t)}{\sigma_{BS}\sqrt{T-t}} = d_1 - \sigma_{BS}\sqrt{T-t}. \end{aligned}$$

In the Black-Scholes option pricing formula given in Equation (4.3), we use $\Phi(\cdot)$ to represent the cumulative distribution function of the standard normal distribution.

One of the major drawbacks of the Black-Scholes model is its reliance on several assumptions that have been shown to seldom hold true in practice. However, these assumptions allow it to achieve its analytical simplicity and traceability. Most significantly, the Black-Scholes model assumes that the underlying asset price process follows a geometric Brownian motion (GBM) (Merton, 1973). This means that the underlying asset price process is governed the following SDE

$$dS_t = \mu S_t \, dt + \sigma_{BS} S_t \, dW_t, \quad S_0 \text{ a constant.} \tag{4.4}$$

In the SDE given by Equation (4.4), both μ and σ_{BS} are constants, where μ is called the drift rate of the underlying asset price and σ_{BS} is called its volatility. A noteworthy property of GBM, is that for an arbitrary initial value S_0, the SDE given by Equation (4.4) has the analytical solution, given by

$$S_t = S_0 \exp\left(\left(\mu - \frac{1}{2}\sigma_{BS}^2\right) t + \sigma_{BS} W_t\right). \tag{4.5}$$

The solution to the SDE in Equation (4.4), is a log-normally distributed random variable (i.e. $S_t \sim \log \mathcal{N}\left(\mu, \sigma_{BS}^2\right)$), with expected value and variance given by

$$\mathbb{E}\left[S_t | S_0\right] = S_0 e^{\mu t}, \tag{4.6}$$

$$\mathbb{V}\text{ar}\left[S_t | S_0\right] = S_0^2 e^{2\mu t}\left(e^{\sigma_{BS}^2 t} - 1\right). \tag{4.7}$$

4.2.2 The Constant Elasticity of Variance Model

Although the Black-Scholes model is widely used in practice, it has some limitations, which has led to it being superseded by more advanced models in certain cases. One such model is the constant elasticity of variance (CEV) model. The CEV model was first proposed by Cox (1975) and can be seen as a generalisation of the Black-Scholes model, as it is used to model the price dynamics of stocks or other assets that exhibit non-constant volatility. The CEV model is a simple parametric local volatility model, where the underlying stock price is assumed to evolve over time according to the SDE given by

$$dS_t = \mu S_t \, dt + \sigma_{CEV} S_t^{\beta} \, dW_t, \quad S_0 \text{ a constant,} \tag{4.8}$$

where $\beta \in [0, 1]$ is referred to as the elasticity factor. In the case where $\beta = 1$, we reproduce the Black-Scholes assumption of constant volatility and hence, the underlying asset price process follows a GBM, as detailed in Section 4.2.1. If the CEV

model takes on a value of $\beta < 1$, the underlying stock model, given by Equation (4.8), produces behaviour consistent with the leverage effect, which is commonly observed in equity markets. The leverage effect refers to the observed relationship between stock returns and both the perceived and actual volatility of the stock. This relationship is such that when the stock price decreases, the volatility of the stock tends to increase (Figlewski and Wang, 2000). Therefore, the stock price and its volatility are negatively correlated.

While the elasticity factor β can take on any value in the range $[0, 1]$, there are only certain values for β in which there exists an analytical solution for the stock price S_t (i.e. for any $\beta \in \{0, 0.5, 1\}$) (McWalter, 2022). As a result, it is therefore surprising that closed-form option pricing formulae have been established for the CEV model for any $\beta \in [0, 1]$. McWalter (2022) proposes a variation of the CEV option pricing formulae, which is presented below in Proposition 4.2.

Proposition 4.2 (The CEV call option pricing formula)**.** Let $C(S_0, r, \sigma_{CEV}, T, K, \beta)$ denote the price at time $t = 0$ of a European call option. Given that the current stock price is S_0, the time of maturity is $T > t$, the exercise (strike) price is K and the elasticity factor is given by β, the price at time $t = 0$ for such an option is given by

$$C(S_0, r, \sigma_{CEV}, T, K, \beta) \coloneqq S_0 \left[1 - \mathcal{X}(y; z, x)\right] - Ke^{-rT}\mathcal{X}(x; z - 2, y), \tag{4.9}$$

where $\mathcal{X}(\cdot; d, \lambda)$ is used to denote the non-central chi-square distribution with d degrees of freedom and non-centrality parameter λ, and

$$x \coloneqq \kappa S_0^{2(1-\beta)} e^{2r(1-\beta)T}, \quad y \coloneqq \kappa K^{2(1-\beta)}, \quad \text{and} \quad z \coloneqq 2 + \frac{1}{1-\beta},$$

where

$$\kappa \coloneqq \frac{2r}{\sigma_{CEV}^2 \left(1 - \beta\right)\left(e^{2r(1-\beta)T} - 1\right)}.$$

In this dissertation, we will only be concerned with the CEV model for $\beta = 0.5$. This is known as the square-root diffusion process, which is comparable to the prominent Cox–Ingersoll–Ross process without mean reversion (Cox, 1975). Therefore, under this model, the underlying asset dynamics are given by

$$dS_t = \mu S_t \, dt + \sigma_{CEV} \sqrt{S_t} \, dW_t, \quad S_0 \text{ a constant.} \tag{4.10}$$

We limit the numerical experiments of this dissertation to only consider this model, as it provides an analytical solution for the stock price S_t, which is essential in implementing the initial neural SDE training schemes, as outlined in Section 4.1.1.

The transition density of S_t governed by the square-root diffusion process, as generalised by Glasserman (2004), for $u < t$, can be expressed as

$$S_t = -\frac{\sigma_{CEV}^2\left(1 - e^{\mu(t-u)}\right)}{4\mu}\chi_0'^2\left(-\frac{4\mu e^{\mu(t-u)}}{\sigma_{CEV}^2\left(1 - e^{\mu(t-u)}\right)}S_u\right), \tag{4.11}$$

where $\chi_v'^2(\lambda)$ denotes a non-central chi-square random variable with v degrees of freedom and non-centrality parameter λ. Therefore, given S_u, S_t is distributed as $-\frac{\sigma_{CEV}^2\left(1-e^{\mu(t-u)}\right)}{4\mu}$ multiplied by a non-central chi-square random variable with 0 degrees of freedom, which is defined to be a Poisson mixture of mass at zero together with chi-squared distributions that have even degrees of freedom, and non-centrality parameter $\lambda = -\frac{4\mu e^{\mu(t-u)}}{\sigma_{CEV}^2\left(1-e^{\mu(t-u)}\right)}S_u$ (Siegel, 1979).

From Equation (4.11), we can deduce that the solution to the square-root diffusion process is a random variable with closed-form expressions for the expected value and variance, given by

$$\mathbb{E}\left[S_t|S_0\right] = S_0 e^{\mu t}, \tag{4.12}$$

$$\mathbb{V}\mathrm{ar}\left[S_t|S_0\right] = \frac{S_0\sigma_{CEV}^2}{\mu}\left(e^{2\mu t} - e^{\mu t}\right). \tag{4.13}$$

For a derivation of the aforementioned results, stated in Equations (4.12) and (4.13), see Appendix A.2.

Another key property of the CEV model is that by inspection, we can observe that the local volatility function is given by

$$\sigma(t, S_t) = \sigma_{CEV} S_t^{(\beta-1)}. \tag{4.14}$$

Chapter 5

Option Pricing with Neural SDEs

This chapter outlines the various neural SDE training schemes, presents concise algorithms for implementing these training schemes and conducts numerical experiments[1] to validate the efficacy of these training schemes. The numerical experiments are designed as proof-of-concept experiments in order to investigate the performance of the neural SDE option pricing methodologies proposed by this dissertation. In order to do this, we draw empirical samples from a benchmark process that is well-understood and has a known set of dynamics governing the behaviour of the underlying asset. Furthermore, the availability of a mathematically precise solution for calculating option prices under the benchmark process will aid in the comparison of the results. Specifically, we use the Black-Scholes and CEV models, detailed in Section 4.2, as benchmark models in the numerical experiments.

To conduct the numerical experiments, we utilise the Julia programming language and its open-source machine-learning software, SciML, which provides a collection of tools for solving equations and modelling systems developed in Julia. Within the SciML library, we make use of the following modules: DifferentialEquations.jl (Rackauckas and Nie, 2017) and DiffEqFlux.jl (Rackauckas *et al.*, 2019). These modules provide a powerful and efficient means of solving differential equations and implementing machine-learning algorithms, which allows for a seamless integration of the neural SDE methodology into the option pricing framework.

In order to ensure the validity of the numerical experiments on the proposed training methodologies in this chapter, it is necessary that the parameters of the underlying option pricing models will remain fixed. By maintaining these parameters, we enable a fair comparison of the various training methodologies put forth in this dissertation. In accordance with the description of the Black-Scholes model provided in Section 4.2.1, the values listed in Table 5.1 will be employed to parameterise the Black-Scholes model in this analysis.

[1] The code used to conduct the numerical experiments throughout this dissertation, is available at https://github.com/Alexio-Phytides/Neural-SDE-Option-Pricing.git.

Black-Scholes Model Parameters		
Parameter	Symbol	Value
Initial stock price	S_0	100.00
Risk-free rate of interest	r	8.00%
Volatility	σ_{BS}	20.00%
Time of maturity	T	1 year
Strike	K	$\{80.00, 82.00, \ldots, 118.00, 120.00\}$

Tab. 5.1: Black-Scholes model parameters.

Following the description of the CEV model provided in Section 4.2.2, we adopt the values listed in Table 5.2 to parameterise the CEV model in this analysis.

CEV Model Parameters		
Parameter	Symbol	Value
Initial stock price	S_0	100.00
Risk-free rate of interest	r	8.00%
Volatility	σ_{CEV}	200.00%
Time of maturity	T	1 year
Strike	K	$\{80.00, 82.00, \ldots, 118.00, 120.00\}$
Elasticity factor	β	0.5

Tab. 5.2: CEV model parameters.

The same parameter values used to parameterise the Black-Scholes model given in Table 5.1, will be employed in the CEV model, with the volatility parameter adjusted to ensure that the initial local volatilities are the same under both benchmark models. This ensures a fair comparison of the option prices computed under the neural SDE and both benchmark models. To determine this parameter value, we use the CEV model's local volatility function to calculate the implied volatility of the asset based on its initial price S_0, given by

$$\begin{aligned} \sigma_{BS} S_0 &= \sigma_{CEV} S_0^{\beta}. \\ \implies \sigma_{CEV} &= \sigma_{BS} S_0^{(1-\beta)}. \end{aligned}$$

Finally, by plugging in the parameter values from Tables 5.1 and 5.2, and solving for the CEV volatility parameter σ_{CEV}, we obtain the CEV volatility parameter given in Table 5.2, which ensures the same initial local volatility under both models.

5.1 Training the Diffusion Coefficient in Isolation

5.1.1 Training Methodology

Training the diffusion coefficient in isolation is the initial neural SDE training algorithm we adopt, where we assume that the dynamics of the underlying asset price process are known. This allows us to simplify the problem by using the neural network components of the neural SDE to approximate the known forms of the drift and/or diffusion coefficients, by leveraging the Universal Approximation Theorem for Neural Networks. To ensure that the neural SDE option prices are free of arbitrage, we adhere to the constraints outlined in Section 4.1.2. These constraints dictate that the neural SDE must be of the form specified in Equation (4.1), where the drift term of the neural SDE is given by a fixed parameterisation rS_t^θ, while the diffusion coefficient is given by the neural network $\sigma^S\left(t, S_t^\theta, \theta\right)$. Therefore, we use neural networks to approximate the known form of the underlying asset's diffusion coefficient, which allows us to effectively capture the randomness inherent in the asset's evolution using the expressiveness of neural networks. See Algorithm 2 in Appendix B.1, for an outline of this proposed neural SDE training scheme.

5.1.2 Numerical Experiments

Now that the neural SDE training methodology has been established, we evaluate its implementation by pricing European call options for the range of strikes given in Table 5.1. To evaluate the effectiveness of the training algorithm, we begin the numerical experiments by comparing the option prices produced by the neural SDE option pricing methodology to those of the Black-Scholes model. We use the parameter values provided in Table 5.1 to parameterise the Black-Scholes model, as detailed in Section 4.2.1. By comparing the results of these two methods, we gain insight into the performance of the neural SDE in approximating option prices.

As outlined in Algorithm 2, the initial step in applying the neural SDE training methodology is to define the dynamics of the underlying asset that the neural SDE model aims to replicate. This involves specifying the form of the diffusion coefficient that represents the target function to approximate using neural networks. As the aim of this numerical experiment is to replicate the Black-Scholes model, for the parameters provided in Table 5.1, the underlying asset is governed by the SDE given in Equation (4.4), and thus the form of the diffusion coefficient is given by

$$b(t, S_t) = \sigma_{BS} S_t.$$

This equation specifies the function that the neural network must approximate in order to accurately replicate the dynamics of the underlying asset according to the

Black-Scholes model. Approximating this function with a neural network is a relatively simple task, as this function is linear in the underlying asset S_t.

Given the simplicity of the function we aim to approximate, we define the neural architecture as a simple linear model with no hidden layers. Therefore, the network will consist of an input layer with a single node, an output layer with a single node, and no hidden layers in between. We do not apply an activation function to any of the layers, meaning that we are essentially using a linear regression model. This minimalistic approach provides a sufficient method to accurately replicate the function while avoiding over-fitting and excessive computational requirements.

Figure 5.1 illustrates the performance of the neural network in approximating the known form of the diffusion coefficient, with the results displayed both before and after training. Evidently, the neural network has managed to approximate the diffusion coefficient after training. A table displaying the computational cost of implementing this algorithm is provided in Appendix C.1.

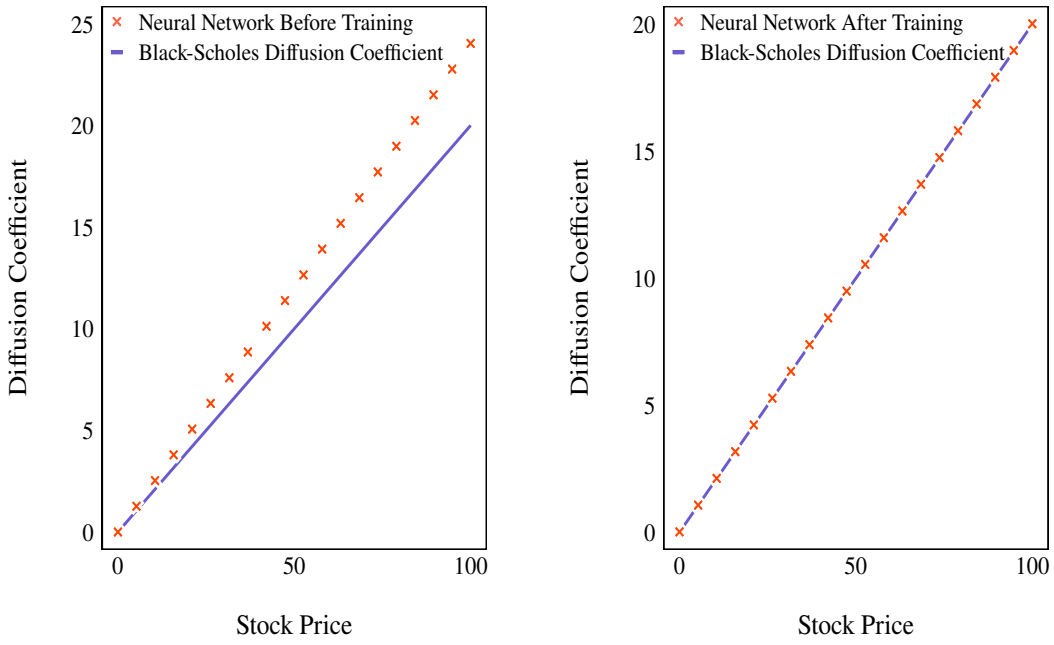

Fig. 5.1: Neural network approximation to Black-Scholes diffusion coefficient.

We now adopt Algorithm 1 to compute option prices using the trained neural SDE. This involves using the trained model to simulate terminal stock prices and using these simulations to calculate Monte Carlo estimates of European call options for the various strikes given in Table 5.1. We implement the Euler-Maruyama method, as specified in Section 4.1.3, to simulate terminal stock values used in the option pricing methodology, where we perform a Monte Carlo simulation with

100 000 realisations of the underlying terminal stock price.

Figure 5.2 compares the performance of the trained neural SDE model in calculating European call option prices, to the Black-Scholes model, for various values of log-moneyness[2] corresponding to the range of strikes in Table 5.1. The first plot displays the option prices produced by each method, while the second plot shows the implied volatility corresponding to these option prices. Figure 5.2 shows that the neural SDE results generally align with those of the Black-Scholes model, with MSE $\approx 7.978 \times 10^{-6}$ in implied volatilities. Greater deviations are observed across the options priced further in-the-money, with the largest error in implied volatility being less than 0.56%, occurring when the option is priced deep in-the-money. This outcome is promising, considering the additional approximation errors introduced by the Monte Carlo simulation and Euler-Maruyama approximation scheme.

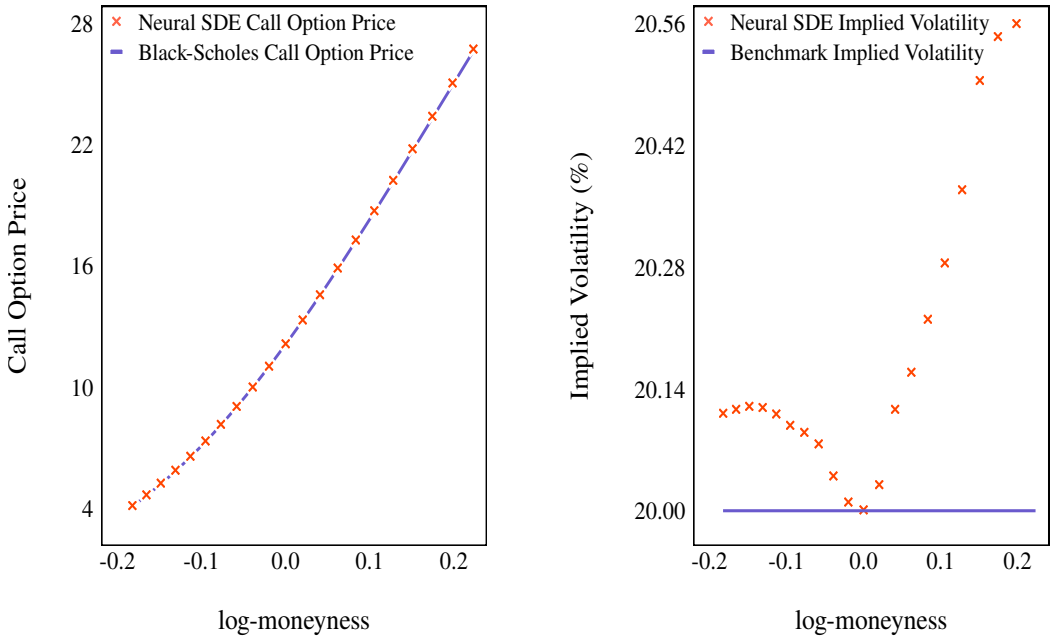

Fig. 5.2: Comparison of neural SDE and Black-Scholes European call option prices.

Now that we have demonstrated the ability of the neural SDE model to reproduce Black-Scholes call option prices by training the diffusion coefficient independently, we conduct a numerical experiment using the same training methodology, with the CEV model parameterised by values listed in Table 5.2, as the benchmark model. We follow the same procedure to training the neural SDE as the numerical experiment under the Black-Scholes model, with the exception that the diffusion

[2] Recall, log-moneyness is defined by $k = \ln\left(\frac{S_t}{K}\right)$. Therefore, when $k = 0$ the call option is priced at-the-money, when $k < 0$ it is priced out-the-money, and when $k > 0$ it is priced in-the-money.

coefficient we aim to approximate using a neural network, is now given by

$$b(t, S_t) = \sigma_{CEV}\sqrt{S_t}.$$

Evidently, the diffusion coefficient is not a simple linear function as with the Black-Scholes model, therefore we carefully design the neural architecture to handle this added complexity while considering the computational demands. Table 5.3 lists the neural architecture we adopt to approximate the square-root diffusion process in this numerical experiment, where the Exponential Linear Unit (ELU) activation function is applied to the input and hidden layers, which is given by

$$\gamma(x) \coloneqq \begin{cases} x & \text{if } x > 0, \\ \alpha\,(e^x - 1) & \text{otherwise.} \end{cases}$$

We also apply the non-linear rectifier Softplus to the output layer, given by

$$\gamma(x) \coloneqq \log(1 + e^x),$$

to ensure a positive output, as restricted by the form of the diffusion coefficient.

Neural Network Architecture		
Layer	Nodes	Activation Function
Input layer	1	ELU, $\alpha = 1$
Hidden layer	16	ELU, $\alpha = 1$
Output layer	1	Softplus

Tab. 5.3: CEV diffusion coefficient neural architecture.

Figure 5.3 illustrates the performance of the neural network in approximating the known form of the diffusion coefficient, both before and after training. Similarly to the Black-Scholes numerical experiment, the results shown in Figure 5.3 indicate that the neural network is able to accurately replicate the function after training. However, the use of a deep neural network, as opposed to a linear regression model, results in a significant increase in computational cost, as displayed in Appendix C.1, which compares the run time of the two numerical experiments.

Figure 5.4 compares the performance of the neural SDE in computing European call option prices, to the CEV model. The results indicate that the neural SDE produces option prices that are in close agreement with those of the CEV model, with MSE $\approx 1.284 \times 10^{-5}$ in implied volatilities. Greater deviations are observed for options priced further in-the-money, with the largest deviation in implied volatility across the range of log-moneyness, observed to be less than 0.91%.

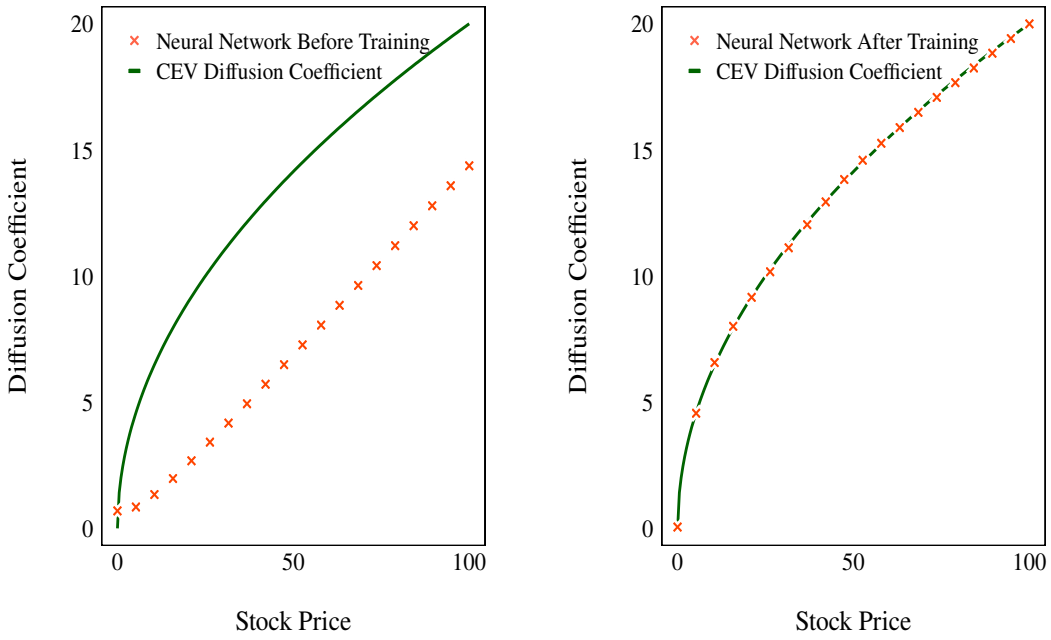

Fig. 5.3: Neural network approximation to CEV diffusion coefficient.

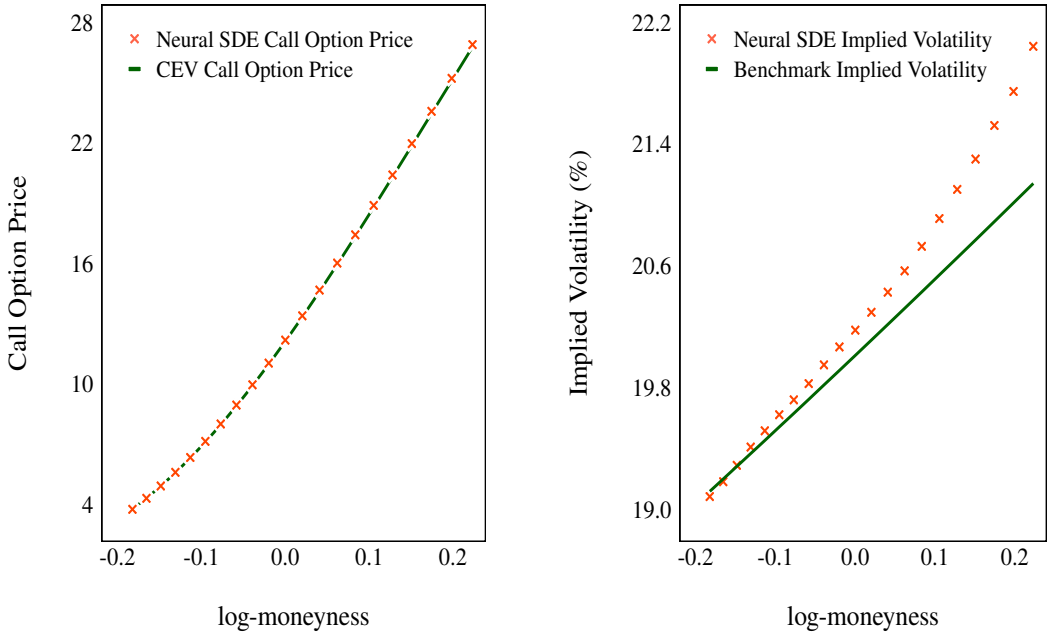

Fig. 5.4: Comparison of neural SDE and CEV European call option prices.

5.2 Method of Moments Training

5.2.1 Training Methodology

Now that we have described the process of training the diffusion coefficient in isolation, the subsequent training methodology uses a method of moments training scheme, that is theoretically analogous to the approach presented by Rackauckas *et al.* (2020). This training approach builds upon the method outlined in Section 5.1, by relaxing assumptions about the specific form of the SDE governing the underlying asset price process. Now, we assume that we either have multiple realisations of the underlying process and can thus compute its first- and second-order moments, or that we have direct access to its first- and second-order moments. We then use this assumption to match the first- and second-order moments of the underlying asset price process to that of the neural SDE. Similarly to the initial approach, these assumptions do not hold in practice, however, this approach serves as a building block for a more practical approach to training the neural SDE.

Consider a stock price path given by the sequence $\{S_{t_0}, S_{t_1}, \dots, S_{t_N}\}$, for some $N \in \mathbb{Z}^+$, at time points $0 = t_0, t_1, \dots, t_N = T$, with fixed time increments given by $\Delta t_i = t_i - t_{i-1}$ and where S_0 is known. Let y_m and y_v denote the set of means and variances of the underlying asset respectively, at each time point. Now, let $\left\{\hat{S}_{t_0}, \hat{S}_{t_1}, \dots, \hat{S}_{t_N}\right\}$ represent a stock price path realisation, generated from the neural SDE, using an approximation scheme described in Section 4.1.3. We simulate n realisations of these stock price paths, which can be depicted by the matrix

$$\begin{bmatrix} \hat{S}^1_{t_0} & \hat{S}^1_{t_1} & \dots & \hat{S}^1_{t_N} \\ \hat{S}^2_{t_0} & \hat{S}^2_{t_1} & \dots & \hat{S}^2_{t_N} \\ \vdots & \vdots & \ddots & \vdots \\ \hat{S}^n_{t_0} & \hat{S}^n_{t_1} & \dots & \hat{S}^n_{t_N} \end{bmatrix}. \tag{5.1}$$

We can now determine the sample means and variances of these simulated stock prices at each time point $0 = t_0, t_1, \dots, t_N = T$, by computing the means and variance of each column in the matrix from Equation (5.1). This will result in the vectors $\hat{y}_m$ and $\hat{y}_v$ of length $N+1$ each, representing the sample means and variances of the simulated stock prices, respectively. In order to match the first- and second-order moments of the underlying asset price process to the neural SDE, we establish a custom loss function $\mathcal{L}(y, \hat{y})$, which is defined as

$$\mathcal{L}(y, \hat{y}) \coloneqq \sum_{i=1}^{N+1} \left(|y^i_m - \hat{y}^i_m|\right) + \sum_{i=1}^{N+1} \left(|y^i_v - \hat{y}^i_v|\right). \tag{5.2}$$

See Algorithm 3 in Appendix B.2, for a description of the neural SDE training methodology outlined in this section.

5.2.2 Numerical Experiments

Similarly to Section 5.1.2, we first compare the neural SDE option prices to the Black-Scholes model, parameterised by the values in Table 5.1. Following Algorithm 3, we initiate the numerical experiment by defining the architecture of the neural SDE. We employ the neural architecture described in Table 5.4 to characterise the neural network specified as the diffusion coefficient of the neural SDE, and let the drift coefficient be given by a fixed parameterisation, as outlined by the no-arbitrage constraints in Section 4.1.2. We employ same the neural architecture as the CEV diffusion coefficient neural architecture, given in Table 5.3, with an additional hidden layer to improve the model's ability to capture intricate features of the system and handle the increased complexity of this training algorithm.

Neural Network Architecture		
Layer	Nodes	Activation Function
Input layer	1	ELU, $\alpha = 1$
Hidden layer	16	ELU, $\alpha = 1$
Hidden layer	16	ELU, $\alpha = 1$
Output layer	1	Softplus

Tab. 5.4: GBM method of moments neural architecture.

We construct a training data set using the closed-form expressions for the mean and variance of the GBM, given in Equations (4.6) and (4.7), respectively. We pre-train the neural SDE using a sample size of $10\,000$ stock price paths, to compute sample means and variances at each time point, in each epoch[3]. We then resume training with a larger sample size, by generating $50\,000$ realisations in each epoch. The pre-training phase consists of 200 epochs, followed by resuming the training process with an additional 300 epochs. The computational cost of this training algorithm is shown in Appendix C.2, highlighting the increased computational expense in undertaking this training approach compared to the methodology proposed in Section 5.1, which is a consequence of simulating multiple realisations of the underlying process in each epoch. While this training methodology presents higher computational demands compared to the approach presented in Section 5.1, it allows us to relax the assumption of prior knowledge of the underlying SDE's dynamics.

Figure 5.5 illustrates the performance of the neural SDE in approximating the moments of the GBM, with the results displayed before and after training. The er-

[3] An epoch is a hyperparameter which represents one full pass through the training data, where the model has seen every training example once.

ror bars indicate an interval of one standard deviation from the GBM mean, with the ribbon displaying the same interval corresponding to the process governed by the neural SDE. In this figure, the sample means and standard deviations are generated from 100 000 simulations from the neural SDE at each time point. The results in Figure 5.5 indicate that the neural SDE is able to reproduce the first- and second-order moments of the GBM after training. It is critical to highlight that the means of the process governed by the neural SDE and that of the GBM, are matched both before and after training, as indicated by the scatter points in Figure 5.5. This is a consequence of the no-arbitrage constraints, which sets the neural SDE's drift coefficient to the same fixed parameterisation as that of the SDE governing the GBM.

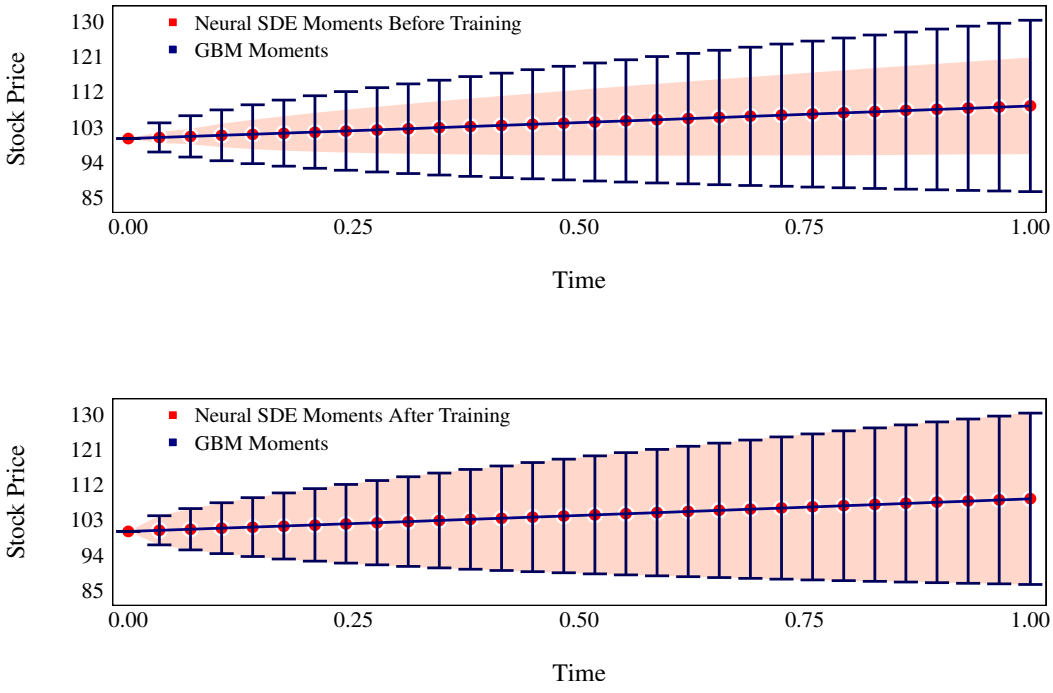

Fig. 5.5: Comparison of neural SDE and GBM moments.

Now that the neural SDE has been trained to match the moments of the underlying asset's price process, we use Algorithm 1 to compute European call option prices using the neural SDE to generate 100 000 stock price paths under the Euler-Maruyama approximation scheme. Figure 5.6 compares the performance of the neural SDE in calculating European call option prices to that of the Black-Scholes model. The first plot displays the option prices produced by each method, while the second plot shows the implied volatility corresponding to these option prices. This figure shows that the neural SDE produces results that largely align with those of the Black-Scholes model, with MSE $\approx 2.199 \times 10^{-6}$ in implied volatilities. Greater

discrepancies are observed for options priced both further in-the-money and out-the-money, where the difference between the implied volatility of the neural SDE and benchmark model, does not exceed 0.26% over the range of log-moneyness.

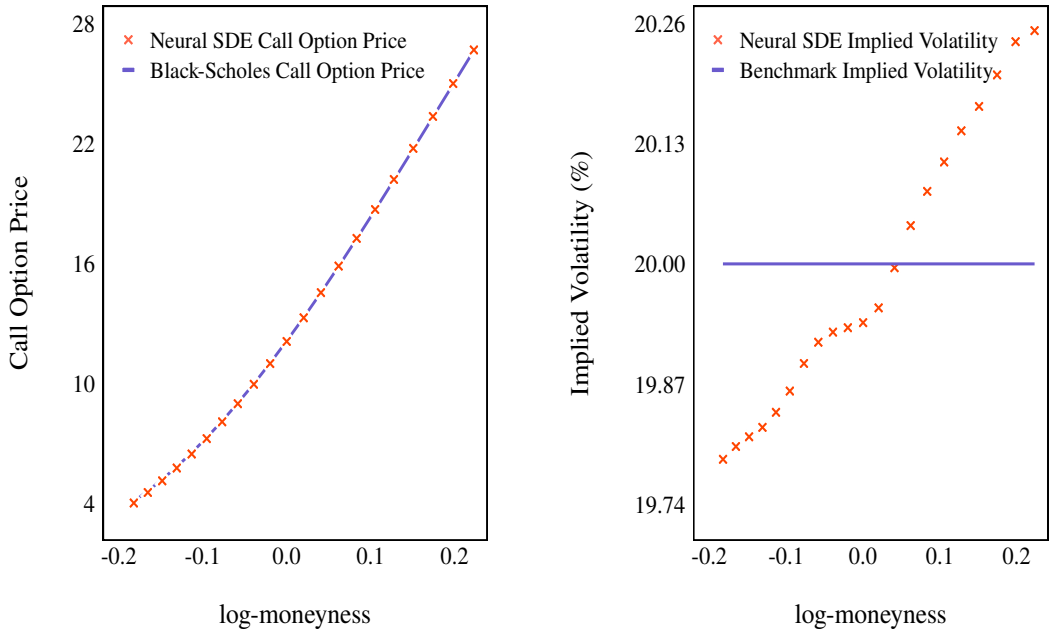

Fig. 5.6: Comparison of neural SDE and Black-Scholes European call option prices.

Now, we adopt the CEV model as the benchmark model, parameterised by the values provided in Table 5.2. We follow the same formulation as the numerical experiment under the Black-Scholes model, employing the same neural architecture described in Table 5.4, however we now build training data using the closed-form expressions for the mean and variance of the solution to the square-root diffusion process, given by Equations (4.12) and (4.13), respectively. These results indicate that the neural SDE is able to accurately replicate option prices under this methodology, while relaxing the assumption of knowledge on the specific form of the underlying SDE, as required in the initial training approach. However, as shown in Appendix C.2, this method has greater computational demands.

Figure 5.7 depicts the ability of the neural SDE to approximate the moments of the solution to the square-root diffusion process. Evidently, the neural SDE is able to match the moments of the benchmark process after training, resulting in neural SDE option prices which closely align with those of the CEV model, with MSE $\approx 1.251 \times 10^{-5}$ in implied volatilities. The largest deviation in implied volatility, depicted in Figure 5.8, is less than 0.53%, demonstrating the ability of the neural SDE to reproduce CEV option prices under this training approach.

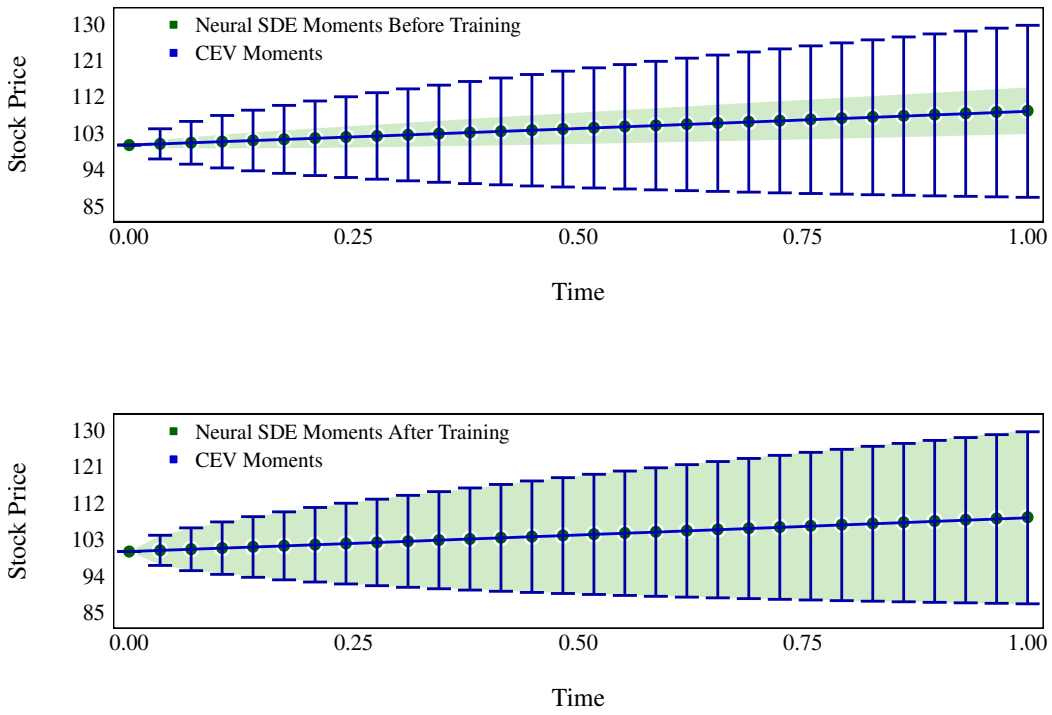

Fig. 5.7: Comparison of neural SDE and CEV moments.

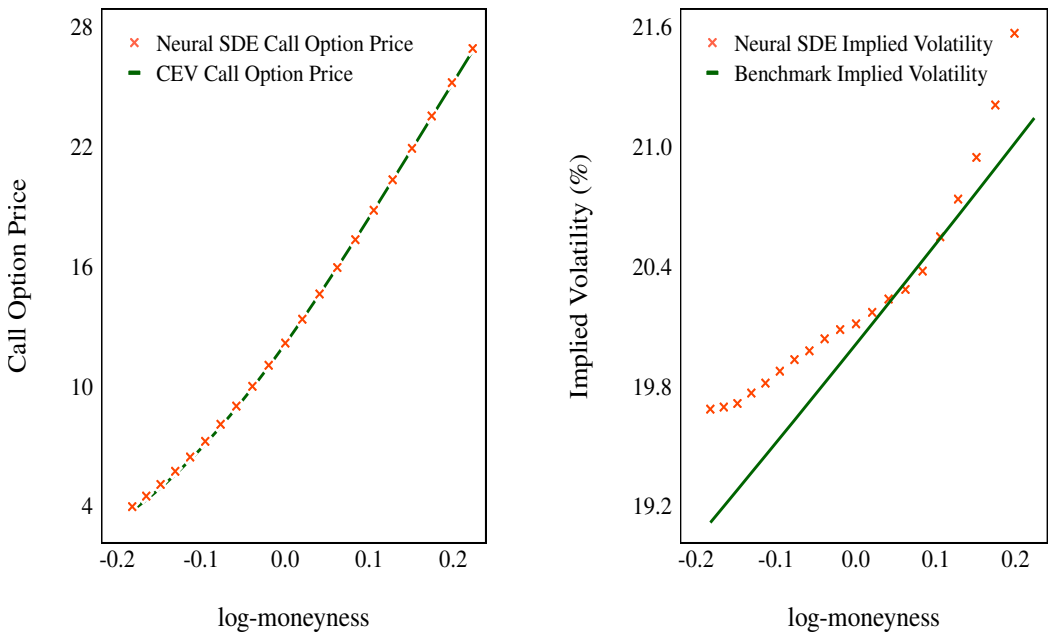

Fig. 5.8: Comparison of neural SDE and CEV European call option prices.

5.3 Training with Option Prices

5.3.1 Training Methodology

Now that we have outlined and implemented the two initial approaches for training the neural SDE, in Sections 5.1 and 5.2, we have built up the necessary foundations to present a more practical neural SDE training scheme, that uses option prices to train the model. This proposed training scheme is fundamentally distinct from the initial approaches, as it eliminates the need for potentially restrictive assumptions on the dynamics or quantitative measures (i.e. moments) of the underlying asset price process. To implement this training scheme, we define a loss function that compares the European call option prices produced by the neural SDE model, to the option prices in the training data. We use this loss function to train the neural SDE, by minimising the difference between the predicted option prices and the training data. This allows us to evaluate the performance of the neural SDE and make adjustments to improve its accuracy. We define a set of European call option prices, denoted as y_c, for a range of strikes $K \in \mathbb{N}$ of length N. The neural SDE produces Monte Carlo estimates of European call option prices for the same range of strikes, denoted as $\hat{y}_c$, using Algorithm 1. To evaluate the performance of the neural SDE, we use the MSE loss function. This loss function can be expressed as

$$\mathcal{L}\left(y, \hat{y}\right) \coloneqq \frac{1}{N} \sum_{i=1}^{N} \left(y_c - \hat{y}_c\right)^2 . \tag{5.3}$$

See Algorithm 4 in Appendix B.3, for a description of the neural SDE training scheme, which trains the neural SDE using option prices, as outlined in this section.

5.3.2 Numerical Experiments

Using the methodology outlined in Section 5.3.1, we train the neural SDE and use it to price European call options for the various strikes listed in Table 5.1. To evaluate the accuracy of this approach, we compare the neural SDE option prices, to the Black-Scholes model, parameterised by the values in Table 5.1. Following Algorithm 4, we begin by defining the neural SDE to model the underlying asset price process and specify its neural architecture. Table 5.3 lists the architecture of the neural network specified as the diffusion coefficient of the neural SDE. This choice of neural architecture is driven by the need to strike a balance between model complexity and performance. It aims to capture non-linearities in the underlying data while avoiding overfitting and ensuring that option prices are predicted within the correct range (positive values). Next, we build training data by computing 100 European call option prices under the Black-Scholes model, within the range of

strikes given in Table 5.1, using the closed-form expression in Equation (4.3). We pre-train the neural SDE with $10\,000$ terminal stock price realisations used to compute Monte Carlo estimates of the European call option price, in each epoch. We then resume training with a larger sample size of $50\,000$ realisations in each epoch. Subsequently, we conduct a final round of training, by generating $100\,000$ realisations in each epoch. In this algorithm, each round of training consists of 50 epochs.

Figure 5.9 depicts $10\,000$ stock price simulations from the trained neural SDE, and the empirical distribution of the terminal values compared to that of the GBM.

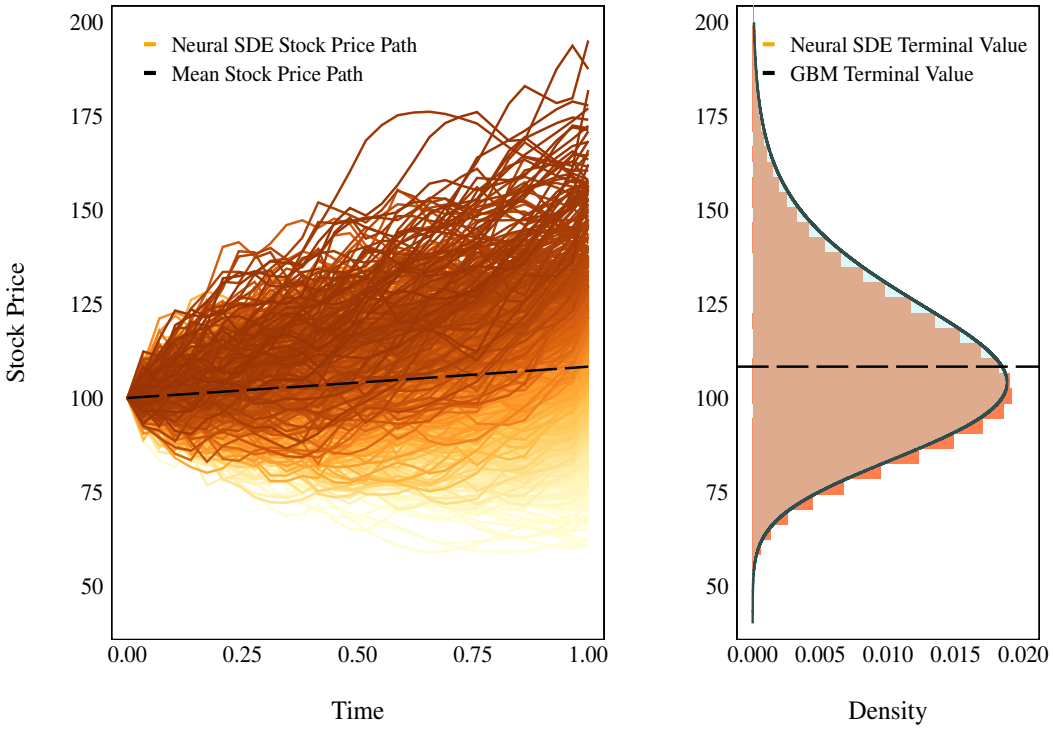

Fig. 5.9: Neural SDE empirical distribution compared to GBM.

Figure 5.9 shows that training the neural SDE using option prices, as described in Section 5.3.1, allows the neural SDE to closely replicate the distribution of the terminal values of the underlying process. The distribution of the underlying process at maturity, given its initial value S_0, is given by a log-normal distribution with mean and variance defined by Equations (4.6) and (4.7), respectively.

Figure 5.10 shows the performance of the neural SDE in approximating European call option prices computed under the Black-Scholes model, using the trained neural SDE to generate $100\,000$ terminal stock prices under the Euler-Maruyama approximation scheme. The results demonstrate that after training, the neural SDE outperforms the previous methodologies in reproducing Black-Scholes option

prices. Under this approach, the difference between the implied volatility of the neural SDE and benchmark model does not exceed more than 0.038%, compared to a maximum deviation of 0.56% and 0.26% observed in the Black-Scholes numerical experiments conducted under the training methodologies in Sections 5.1 and 5.2, respectively. Additionally, from Figure 5.10, we observe a MSE $\approx 4.304\times10^{-8}$ in implied volatilities. This is in contrast to a MSE $\approx 7.978\times10^{-6}$ and MSE $\approx 2.199\times10^{-6}$ observed in the Black-Scholes numerical experiments conducted under the training methodologies in Sections 5.1 and 5.2, respectively. A table displaying the computational cost of implementing this algorithm is provided in Appendix C.3.

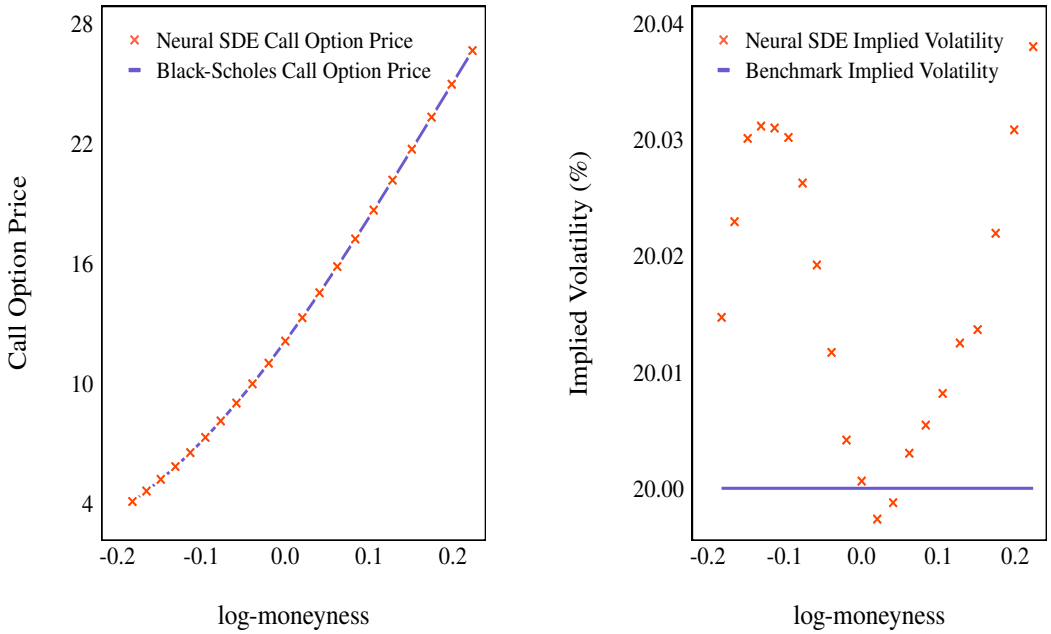

Fig. 5.10: Comparison of neural SDE and Black-Scholes call option prices.

We have demonstrated the neural SDE's ability to reproduce option prices computed by the Black-Scholes model, using option prices to train the neural SDE. Next, we compare the results of the neural SDE to those of the CEV model, using the parameter values listed in Table 5.2. The procedure for this numerical experiment follows similarly to the numerical experiment conducted with the Black-Scholes model as the benchmark model, where we employ the same neural architecture as described in Table 5.4. Additionally, to build the training data, we compute 100 European call option prices under the CEV model, within the range of strikes given in Table 5.2, where we now use the closed-form expression for the CEV option pricing formula, given by Equation (4.9).

Figure 5.11 depicts 10 000 stock price simulations from the neural SDE after the training algorithm has been implemented, and the empirical distribution of the terminal values compared to that of the CEV model.

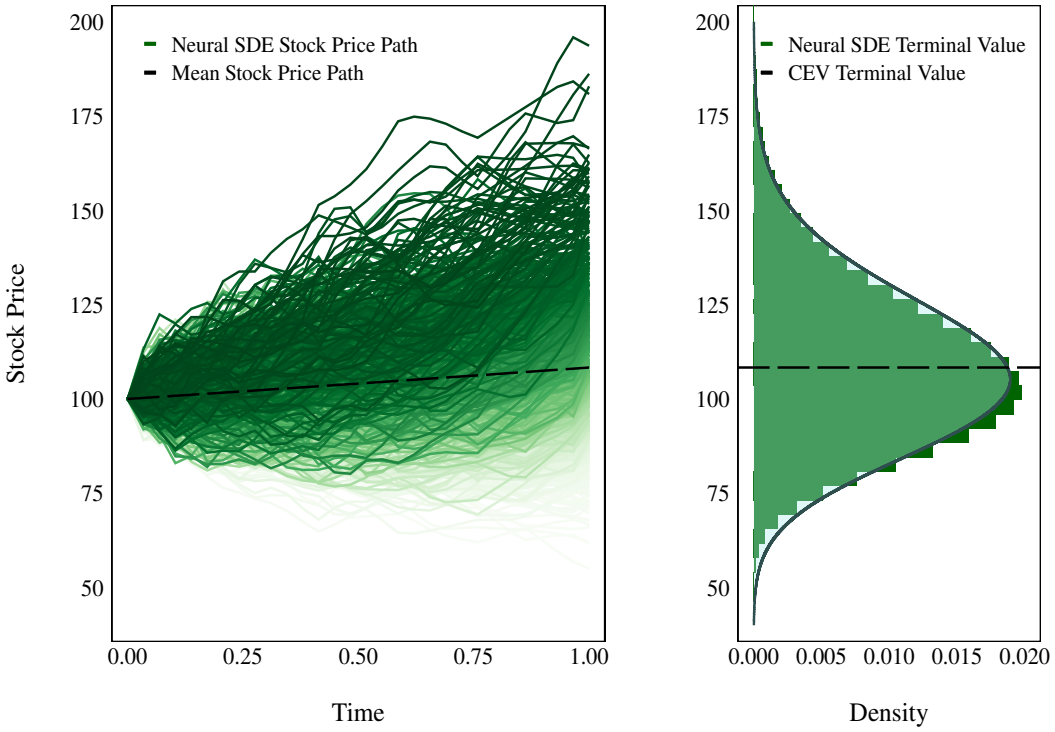

Fig. 5.11: Neural SDE empirical distribution compared to CEV.

Figure 5.11 shows that after adopting the training methodology outlined in Section 5.3.1, the neural SDE closely replicates the distribution of the underlying process at maturity, where the underlying process, given its initial value S_0, is distributed as $-\frac{\sigma^2_{CEV}\left(1-e^{rT}\right)}{4r}$ multiplied by a non-central chi-square random variable with 0 degrees of freedom and non-centrality parameter $\lambda = -\frac{4re^{rT}}{\sigma^2_{CEV}(1-e^{rT})}S_0$. The histogram in Figure 5.11 shows that the empirical distribution of the terminal neural SDE values closely approximates the underlying process's distribution at maturity.

Figure 5.12 shows the performance of the neural SDE in approximating European call option prices computed under the CEV model. Similarly to the previous numerical experiment conducted under the Black-Scholes model, the results in Figure 5.12 demonstrate that after training, the neural SDE outperforms the previous methodologies in reproducing CEV option prices. Under this training approach, the difference between the implied volatility of the neural SDE and benchmark model does not exceed 0.42%, compared to a maximum deviation of 0.91% and 0.53% observed in the CEV numerical experiments conducted under the training

methodologies in Sections 5.1 and 5.2, respectively. Additionally, from Figure 5.10, we observe a MSE $\approx 4.552 \times 10^{-6}$ in implied volatilities. This is in contrast to a MSE $\approx 1.284 \times 10^{-5}$ and MSE $\approx 1.251 \times 10^{-5}$ observed in the CEV numerical experiments conducted under the training methodologies in Sections 5.1 and 5.2, respectively.

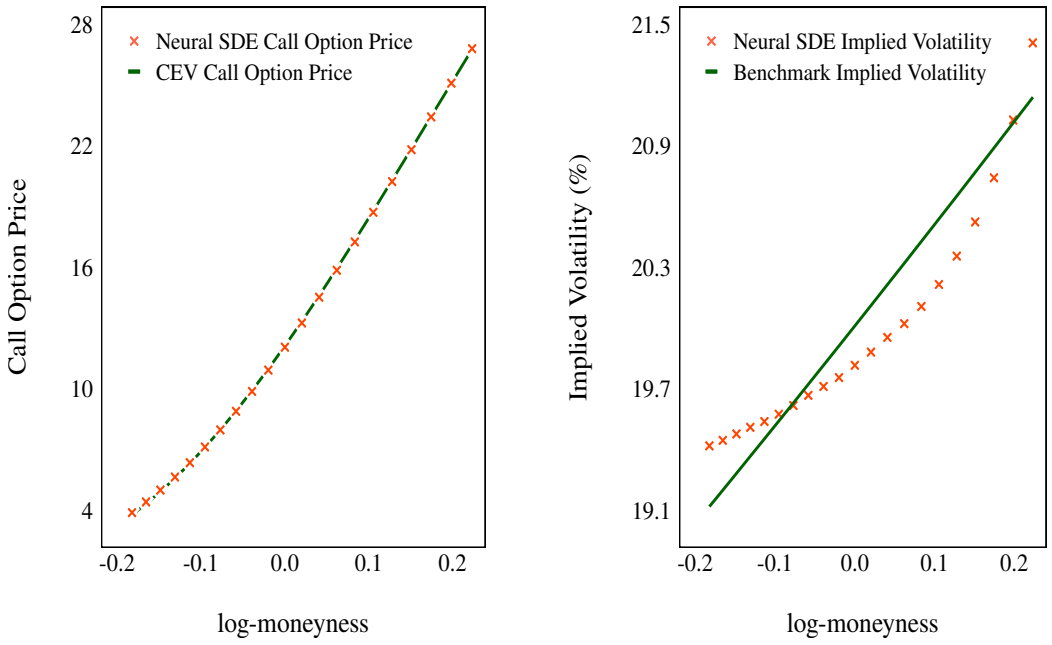

Fig. 5.12: Comparison of neural SDE and CEV European call option prices.

A table displaying the computational cost of implementing this algorithm is provided in Appendix C.3. By comparing the computational cost of this training algorithm under both benchmark models, to that of the method of moments training algorithm, it is evident that this approach is significantly less computationally demanding, while still managing to produce better results. The decrease in computational cost is a consequence of only having to generate terminal stock prices in each epoch to compute Monte Carlo estimates of the neural SDE call option price. This is in contrast to the method of moments approach, which requires entire stock paths to be generated, and the sample means and variances to be computed at each time point, in each epoch. In addition to providing a less computationally demanding algorithm, by eliminating the need for potentially restrictive assumptions about the underlying asset price process, which are central to the two initial approaches, this training scheme allows for a more robust and flexible model that can accurately capture the underlying asset price process. However, it is important to note that this approach requires a significant amount of option price data, which can present a practical limitation for its implementation in real-world applications.

Chapter 6

Conclusion

In this dissertation, we delved into the innovative application of neural SDEs in computing arbitrage-free option prices. The focus of this dissertation was on the class of these models, as neural SDEs allow for the augmentation of classical risk models such as SDEs, with modern predictive analytics. This approach to pricing options provides the benefit of flexibility from the neural networks while staying within the realm of classical models, through the framework of SDEs. This is a valuable approach to pricing options, as by augmenting classical risk models to this methodology, we adopt a component that is well understood by traders, risk managers and regulators (Gierjatowicz *et al.*, 2020). This is a significant advantage of adopting neural SDEs as it addresses concerns from regulators, to some extent, surrounding the use of "black-box" models. Furthermore, this dissertation highlighted a key benefit of adopting such an approach to pricing options, as in addition to the calibration process, model selection and model uncertainty are done simultaneously. In this sense, the neural SDE model selection process is data-driven (Gierjatowicz *et al.*, 2020).

This dissertation presented an approach to developing a training methodology for neural SDEs, by detailing two initial training approaches as building blocks to the objective training methodology that is both of practical use and theoretically consistent. The proposed neural SDE training scheme is fundamentally distinct from the initial training approaches, in the sense that it does not require any assumptions on the dynamics or form of the underlying asset price process, but rather trains the neural SDE using option pricing data. By eliminating the need for these assumptions, we were able to establish a training scheme that is both practical and efficient, allowing for a more robust and flexible model that can accurately capture the underlying asset price process.

We conducted an evaluation of the robustness of the proposed neural SDE methodology for pricing options, by performing numerical experiments under the well-established Black-Scholes and CEV models. The results from the numerical

experiments indicate that this approach to pricing options, while it is highly effective in comparison to the initial approaches in terms of its computational demands and option pricing accuracy, it requires an extensive supply of option price data for the training algorithm. This may be a limitation when compared to traditional option pricing methodologies, which do not have these same requirements. However, the proposed methodology has the advantage of being robust and flexible, as it is able to capture the underlying asset price process without any assumptions on the dynamics or form of the underlying process. This could be seen as a trade-off for the added computational requirements.

Given the demanding computational nature of the proposed neural SDE training methodology, there is room for future research to explore methods for optimising its efficiency. Possible avenues for improvement could involve finding new methodologies to reduce the computational load or exploring alternative algorithms that achieve similar results with fewer resources. These efforts could assist in making the proposed approach more accessible and practical for a wider range of applications in the financial industry. Moreover, a valuable avenue for further research in this area would be the investigation of incorporating more relevant market data beyond option prices, such as implied volatilities, which has the potential to further refine the pricing interval and boost the accuracy of the model.

Overall, this dissertation attempts to offer a thorough understanding of the capabilities and robustness of using neural SDEs to compute arbitrage-free option prices under a variety of market conditions.

Bibliography

Andrychowicz, M., Denil, M., Gomez, S., Hoffman, M. W., Pfau, D., Schaul, T., Shillingford, B. and De Freitas, N. (2016). Learning to learn by gradient descent by gradient descent, *Advances in Neural Information Processing Systems* **29**: 1–8.

Bachelier, L. (1900). Théorie de la spéculation, *Annales Scientifiques de l'École Normale Supérieure*, Vol. 17, pp. 21–86.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*, Vol. 4, Springer.

Björk, T. (2004). *Arbitrage Theory in Continuous Time*, 2 edn, Oxford University Press.

Black and Scholes, Myron, F. (1973). The pricing of options and corporate liabilities, *Journal of Political Economy* **81**(3): 637–654.

Caruana, R., Lawrence, S. and Giles, C. (2000). Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping, *Advances in Neural Information Processing Systems* **13**.

Chen, R. T., Rubanova, Y., Bettencourt, J. and Duvenaud, D. K. (2018). Neural ordinary differential equations, *Advances in Neural Information Processing Systems* **31**.

Cox, A. M. and Obloj, J. (2011). Robust hedging of double touch barrier options, *SIAM Journal on Financial Mathematics* **2**(1): 141–182.

Cox, J. (1975). Notes on option pricing I: Constant elasticity of variance diffusions, *Unpublished note, Stanford University, Graduate School of Business* .

DeLise, T. (2021). Neural options pricing, *arXiv preprint arXiv:2105.13320* .

Dimitroff, G., Roeder, D. and Fries, C. P. (2018). Volatility model calibration with convolutional neural networks, *Available at SSRN 3252432* .

Figlewski, S. and Wang, X. (2000). Is the'leverage effect'a leverage effect?, *Available at SSRN 256109* .

Gierjatowicz, P., Sabate-Vidales, M., Šiška, D., Szpruch, L. and Žurič, Ž. (2020). Robust pricing and hedging via neural SDEs, *arXiv preprint arXiv:2007.04154* .

Giles, M. (2008). Improved multilevel monte carlo convergence using the milstein scheme, *Monte Carlo and Quasi-Monte Carlo Methods 2006*, Springer, pp. 343–358.

Glasserman, P. (2004). *Monte Carlo Methods in Financial Engineering*, Vol. 53, Springer.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, JMLR Workshop and Conference Proceedings, pp. 249–256.

Hernandez, A. (2016). Model calibration with neural networks, *Available at SSRN 2812140* .

Hornik, K., Stinchcombe, M. and White, H. (1989). Multilayer feedforward networks are universal approximators, *Neural Networks* **2**(5): 359–366.

Itô, K. (1944). 109. stochastic integral, *Proceedings of the Imperial Academy* **20**(8): 519–524.

Karsoliya, S. (2012). Approximating number of hidden layer neurons in multiple hidden layer BPNN architecture, *International Journal of Engineering Trends and Technology* **3**(6): 714–717.

Kidger, P., Foster, J., Li, X. and Lyons, T. J. (2021). Neural SDEs as infinite-dimensional gans, *International Conference on Machine Learning*, PMLR, pp. 5453–5463.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization, *arXiv preprint arXiv:1412.6980* .

Kloeden, P. E. and Platen, E. (1992). Stochastic differential equations, *Numerical Solution of Stochastic Differential Equations*, Springer, pp. 103–160.

Liu, S., Borovykh, A., Grzelak, L. A. and Oosterlee, C. W. (2019). A neural network-based framework for financial model calibration, *Journal of Mathematics in Industry* **9**(1): 1–28.

McWalter, T. A. (2022). Numerical methods in finance II , Lecture notes, The University of Cape Town, Department of Finance and Tax.

Merton, R. C. (1973). Theory of rational option pricing, *The Bell Journal of economics and management science* pp. 141–183.

Rackauckas, C., Innes, M., Ma, Y., Bettencourt, J., White, L. and Dixit, V. (2019). Diffeqflux.jl-a Julia library for neural differential equations, *arXiv preprint arXiv:1902.02376* .

Rackauckas, C., Ma, Y., Martensen, J., Warner, C., Zubov, K., Supekar, R., Skinner, D., Ramadhan, A. and Edelman, A. (2020). Universal differential equations for scientific machine learning, *arXiv preprint arXiv:2001.04385* .

Rackauckas, C. and Nie, Q. (2017). Differentialequations.jl–a performant and feature-rich ecosystem for solving differential equations in Julia, *Journal of Open Research Software* **5**(1): 15.

Rezende, D. J., Mohamed, S. and Wierstra, D. (2014). Stochastic backpropagation and approximate inference in deep generative models, *International Conference on Machine Learning*, PMLR, pp. 1278–1286.

Siegel, A. F. (1979). The noncentral chi-squared distribution with zero degrees of freedom and testing for uniformity, *Biometrika* **66**(2): 381–386.

Stone, H. (2020). Calibrating rough volatility models: a convolutional neural network approach, *Quantitative Finance* **20**(3): 379–392.

The World Federation of Exchanges (2022). WFE IOMA derivatives report 2021.

Tzen, B. and Raginsky, M. (2019). Neural stochastic differential equations: Deep latent gaussian models in the diffusion limit, *arXiv preprint arXiv:1905.09883* .

Veale, M. and Borgesius, F. Z. (2021). Demystifying the draft EU artificial intelligence act—analysing the good, the bad, and the unclear elements of the proposed approach, *Computer Law Review International* **22**(4): 97–112.

Appendix A

Relevant Derivations

A.1 Neural SDE Discounted Asset Price Dynamics

We show through an application of Itô's Lemma, that for the asset price dynamics given by the neural SDE in Equation (4.1), the discounted asset price is drift-less and hence a (local) martingale. We assume an annual continuously compounded risk-free rate of interest, $r \in \mathbb{R}$. Therefore, the discounted asset price is given by $f = e^{-rt} S_t^\theta$, where the function f has the following partial derivatives,

$$\begin{aligned}
\frac{\partial f}{\partial t} &= -re^{-rt} S_t^\theta, \\
\frac{\partial f}{\partial S} &= e^{-rt}, \\
\frac{\partial^2 f}{\partial S^2} &= 0.
\end{aligned}$$

To obtain the dynamics of the discounted asset price, we apply Itô's Lemma to the function $f = e^{-rt} S_t^\theta$, which gives us

$$\begin{aligned}
df &= \left(-re^{-rt} S_t^\theta + re^{-rt} S_t^\theta\right) dt + \sigma^S \left(t, S_t^\theta, \theta\right) e^{-rt} \, dW_t \\
&= \sigma^S \left(t, S_t^\theta, \theta\right) e^{-rt} \, dW_t.
\end{aligned}$$

$\therefore$ The discounted asset price $\left(e^{-rt} S_t^\theta\right)$ is drift-less and hence a (local) martingale.

A.2 Properties of the Square-root Diffusion Process

We derive the expected value and variance for the process governed by the square-root diffusion process. From Equation (4.11), we know that under the square-root diffusion process specified in Equation (4.10), given S_0, S_t is distributed as $-\frac{\sigma_{CEV}^2\left(1-e^{\mu t}\right)}{4\mu}$ times a non-central chi-square random variable, characterised by 0 degrees of freedom and non-centrality parameter $\lambda = -\frac{4\mu e^{\mu t}}{\sigma_{CEV}^2(1-e^{\mu t})}S_0$, i.e. for $t > 0$

$$S_t = -\frac{\sigma_{CEV}^2\left(1-e^{\mu t}\right)}{4\mu}\chi_0^{'2}\left(-\frac{4\mu e^{\mu t}}{\sigma_{CEV}^2\left(1-e^{\mu t}\right)}S_0\right),$$

where $\chi_v^{'2}(\lambda)$ denotes a non-central chi-square random variable with v degrees of freedom and non-centrality parameter λ. To derive the mean and variance of this process, we use the mean and variance of $\chi_v^{'2}(\lambda)$, which is given by

$$\mathbb{E}\left[\chi_v^{'2}(\lambda)\right] = v + \lambda, \tag{A.1}$$

$$\mathbb{V}\mathrm{ar}\left[\chi_v^{'2}(\lambda)\right] = 2(v + 2\lambda). \tag{A.2}$$

Therefore, using Equation (A.1), we derive the expected value as

$$\begin{aligned}
\mathbb{E}\left[S_t|S_0\right] &= \mathbb{E}\left[-\frac{\sigma_{CEV}^2\left(1-e^{\mu t}\right)}{4\mu}\chi_0^{'2}\left(-\frac{4\mu e^{\mu t}}{\sigma_{CEV}^2\left(1-e^{\mu t}\right)}S_0\right)\middle|S_0\right] \\
&= -\frac{\sigma_{CEV}^2\left(1-e^{\mu t}\right)}{4\mu}\mathbb{E}\left[\chi_0^{'2}\left(-\frac{4\mu e^{\mu t}}{\sigma_{CEV}^2\left(1-e^{\mu t}\right)}S_0\right)\middle|S_0\right] \\
&= -\frac{\sigma_{CEV}^2\left(1-e^{\mu t}\right)}{4\mu}\left(-\frac{4\mu e^{\mu t}}{\sigma_{CEV}^2\left(1-e^{\mu t}\right)}S_0\right) \\
&= S_0 e^{\mu t}.
\end{aligned}$$

Now, using Equation (A.2), we derive the variance as

$$\begin{aligned}
\mathbb{V}\mathrm{ar}\left[S_t|S_0\right] &= \mathbb{V}\mathrm{ar}\left[-\frac{\sigma_{CEV}^2\left(1-e^{\mu t}\right)}{4\mu}\chi_0^{'2}\left(-\frac{4\mu e^{\mu t}}{\sigma_{CEV}^2\left(1-e^{\mu t}\right)}S_0\right)\middle|S_0\right] \\
&= \frac{\sigma_{CEV}^4\left(1-e^{\mu t}\right)^2}{16\mu^2}\mathbb{V}\mathrm{ar}\left[\chi_0^{'2}\left(-\frac{4\mu e^{\mu t}}{\sigma_{CEV}^2\left(1-e^{\mu t}\right)}S_0\right)\middle|S_0\right] \\
&= \frac{\sigma_{CEV}^4\left(1-e^{\mu t}\right)^2}{16\mu^2}\left(-\frac{16\mu e^{\mu t}}{\sigma_{CEV}^2\left(1-e^{\mu t}\right)}S_0\right) \\
&= \frac{S_0\sigma_{CEV}^2}{\mu}\left(e^{2\mu t}-e^{\mu t}\right).
\end{aligned}$$

Appendix B

Neural SDE Training Algorithms

B.1 Training the Diffusion Coefficient in Isolation

Algorithm 2 Training the Diffusion Coefficient in Isolation Algorithm.

Let $\mathcal{E}$ denote the error between the predicted output and the true output, and n_{epochs} denote the maximum number of epochs.

1. Specify the dynamics of the underlying asset that we aim to replicate with the model. We generalise these dynamics as

$$dS_t = a(t, S_t)\, dt + b(t, S_t)\, dW_t,$$

   where $a(t, S_t) = rS_t$ to keep consistent with the no-arbitrage requirements, and $b(t, S_t)$ is the specific form of the diffusion coefficient we aim to approximate.

2. Define the neural SDE used to approximate the underlying asset's dynamics.

3. Specify the neural SDE's architecture, aiming to minimise the computational requirements while capturing the underlying complexity, to ensure the model can be trained efficiently. Then, randomly initialise the weights (θ) and biases (b).

4. Pre-process the training data and labels, so it is suitable for training.

5. **while** the error $\mathcal{E}$ is above a certain threshold and the maximum number of epochs n_{epochs} has not been reached **do**

   Feed the data through the network to get an output from the output layer.

   Calculate the error $\mathcal{E}$, between the predicted output and the true output.

   Propagate the error back through the network, using back-propagation to calculate the gradients of the weights and biases with respect to the error.

   Update the weights (θ) and biases (b) of the neural network, using an optimisation algorithm.

end while
return θ and b

B.2 Method of Moments Training

Algorithm 3 Method of Moments Training Algorithm.

Let $\mathcal{E}$ denote the error between the predicted output and the true output, and n_{epochs} denote the maximum number of epochs.

1. Define the neural SDE used to approximate the underlying asset's dynamics, by

$$dS_t^\theta = rS_t^\theta \, dt + \sigma^S \left(t, S_t^\theta, \theta\right) dW_t,$$

   keeping consistent with the no-arbitrage conditions set on the neural SDE.

2. Specify the neural SDE's architecture, ensuring it captures the underlying data set's complexity, while avoiding over-fitting and excessive computational requirements.

3. Pre-process the training data consisting of the means and variances of the underlying asset's price process, so that it is suitable for the training algorithm.

4. Pre-train the neural SDE by following Step 5 and generating a smaller sample size of stock price path realisations used to calculate means and variances, setting a higher error threshold and reducing the maximum number of epochs. This will initialise the weights (θ) and biases (b) of the neural SDE in a way that allows the network to learn more effectively when it is fine-tuned on the main data set.

5. **while** the error $\mathcal{E}$ is above a certain threshold and the maximum number of epochs n_{epochs} has not been reached **do**

   Generate n stock price path realisations from the neural SDE at each time point $0 = t_0, t_1, \ldots, t_N = T$, as shown in Equation (5.1).

   Compute the means and variances at each time point from the simulated stock price paths.

   Calculate the error $\mathcal{E}$, between the predicted moments and the true moments, as given by the function in Equation (5.2).

   Propagate the error back through the network, using back-propagation to calculate the gradients of the weights and biases with respect to the error.

   Update the weights (θ) and biases (b) of the neural SDE, using the Adam optimisation algorithm.

   end while
   return θ_t and b

6. Resume the training process in Step 5, while increasing the sample size of generated stock paths, reducing the error threshold and raising the maximum number of epochs.

B.3 Training with Option Prices

Algorithm 4 Training with Option Prices Algorithm.

Let $\mathcal{E}$ denote the error between the predicted output and the true output, and n_{epochs} denote the maximum number of epochs.

1. Define the neural SDE used to approximate the underlying asset's dynamics, by

$$dS_t^\theta = rS_t^\theta \, dt + \sigma^S \left(t, S_t^\theta, \theta\right) dW_t,$$

   keeping consistent with the no-arbitrage conditions set on the neural SDE.

2. To accurately capture the complexity of the underlying data and prevent overfitting, specify the architecture of the neural SDE that strikes the right balance between expressiveness and generalisation. In addition, aim to minimise the computational requirements of the model, to ensure it can be trained and used efficiently.

3. Pre-process the training data consisting of European call option prices for various strikes, so it is suitable for training.

4. Pre-train the neural SDE by following Step 5 and generating a smaller sample size of the terminal stock price realisations used to compute Monte Carlo estimates of the call option price for various strikes, setting a higher error threshold and reducing the maximum number of epochs. This will initialise the weights (θ) and biases (b) of the neural SDE in a way that allows the network to learn more effectively when it is fine-tuned on the main data set.

5. **while** the error $\mathcal{E}$ is above a certain threshold and the maximum number of epochs n_{epochs} has not been reached **do**

   Generate n terminal stock price realisations from the neural SDE at maturity of the European call option (i.e. at time T).

   For a given range of strikes, compute Monte Carlo estimates of the European call option prices $\hat{C}_n(S_0, r, \sigma, T, K)$, using Algorithm 1.

   Feed the data through the network to get an output from the output layer.

   Calculate the error $\mathcal{E}$, between the predicted output and the true output, given by the function in Equation (5.3).

   Propagate the error back through the network, using back-propagation to calculate the gradients of the weights and biases with respect to the error.

   Update the weights (θ) and biases (b) of the neural SDE, using the Adam optimisation algorithm.

   end while
   return θ_t and b

6. Resume the training process in Step 5, while increasing the sample size of the terminal stock price realisations, reducing the error threshold and increasing the maximum number of epochs.

Appendix C

Computational Cost of Training Algorithms

The training algorithms in each numerical experiment, conducted in Chapter 5, were executed on an Apple MacBook Pro with an M1 Pro chip, featuring a 10-core CPU (8 performance cores and 2 efficiency cores), a 16-core GPU, a 16-core Neural Engine and 200GB/s memory bandwidth.

C.1 Training the Diffusion Coefficient in Isolation Numerical Experiment

Computational Cost of Neural SDE Training Algorithm (Seconds)	
Benchmark Model	Total Run Time
Black-Scholes model	2.537
CEV model	345.235

Tab. C.1: Training the diffusion coefficient in isolation algorithm run time.

C.2 Method of Moments Training Numerical Experiment

Computational Cost of Neural SDE Training Algorithm (Minutes)			
Benchmark Model	Pre-training	Training	Total Run Time
Black-Scholes model	7.612	83.081	90.693
CEV model	10.981	128.885	139.866

Tab. C.2: Method of moments training algorithm run time.

C.3 Training with Option Prices Numerical Experiment

Computational Cost of Neural SDE Training Algorithm (Minutes)				
Benchmark Model	Pre-training	1^{st} Round	2^{nd} Round	Total Run Time
Black-Scholes model	3.336	21.547	39.689	64.572
CEV model	3.774	22.253	41.589	67.616

Tab. C.3: Training with option prices algorithm run time.