THE UNIVERSITY *of* EDINBURGH

Quantitative Finance informed Machine Learning

Marc Sabaté Vidales

Doctor of Philosophy
University of Edinburgh
2023

Declaration

I declare that this thesis was composed by myself and that the work contained therein is my own, except where explicitly stated otherwise in the text.

(*Marc Sabaté Vidales*)

Abstract

This PhD thesis consists of two parts. In the first part, we develop and study deep learning-based methods for approximating high-dimensional parabolic (path-dependent) linear PDEs parametrised by the model parameters. The proposed algorithms approximate solutions together with their gradients. Furthermore, we show how to remove the bias in the deep network approximation of the PDE solution by combining obtained approximation of the gradient of the PDE with Monte Carlo simulations. In the context of quantitative finance, our methodology can be used to simultaneously compute the prices and hedging strategies for (path-dependent) derivatives with the underlying modelled by a diffusion model with possible path-dependent coefficients across ranges of parameters and initial conditions.

In the second part of this thesis, we study techniques that enable learning the model using market data while keeping a strong prior on its form. First, we develop Neural SDEs, which is a stochastic differential equation (SDE) where the drift and the diffusion coefficients are parametrised by Neural Networks. This allows us to obtain robust bounds for prices of derivatives while incorporating relevant market data. Neural SDEs can also be seen as an instance of generative models, a class of Machine Learning models that learn to sample from a target distribution (either time marginal or on the path space). Second, we shift our focus to conditional generators for time series data. We introduce the Sig-Wasserstein GAN that relies on a path signature that emerged from rough path theory and enjoyed universal approximation property. All the results of the thesis are underpinned with thorough numerical experiments that show the efficiency of our algorithms.

Contents

Acknowledgements

I am very grateful for the support and the guidance that I have received from my supervisors Lukasz Szpruch and David Siska. We first met around five years ago when I was working as a Machine Learning engineer. The journey since then has been inspiring, and I am very excited about the new projects that we have ahead of us.

I would also like to extend my gratitude to the many amazing people in Edinburgh, Barcelona and Guelph with whom I have shared these years in different moments. Erola, Niamh, Viktoria, Fabian, Toyo, Pau, my family, Tom... The last few years have been a great experience, in great part thanks to all of you.

Chapter 1

Introduction

Deep neural networks trained with stochastic gradient descent have shown a huge success in the last few years at a number of applications such as computer vision, natural language processing, generative modelling (the combination of the last two has very recently brought astonishing chat-bot applications such as ChatGPT3), or reinforcement learning. The success of these methods in seemingly high-dimensional settings has encouraged to investigate their performance in solving high-dimensional PDEs. Starting with the pioneering work [1, 2], where probabilistic representation has been used to learn PDEs using neural networks, recent years have brought an influx of interest in deep PDE solvers.

Similarly, generative Machine Learning models have benefited from combining neural networks with stochastic differential equations models, leading to powerful generative AI models such as score-based generative models, [3, 4].

In this thesis we combine results from stochastic analysis with deep neural networks techniques and provide a new perspective on classical questions that arise in quantitative finance. In the first half of the thesis we propose new PDE solvers based on the probabilistic representation of high-dimensional PDEs and path-dependent PDEs. In the second half of the thesis we propose a generative model to learn time series distributions, as well as a novel GAN-based algorithm based on path signatures. All our numerical experiments show the efficiency of our algorithms in the context of risk-neutral asset pricing in mathematical finance.

In the rest of this introduction we derive the main concepts in risk-neutral asset pricing and we summarise the outline of this thesis.

1.1 Machine learning methods applied to Mathematical Finance - Option pricing

Financial instruments in a market fall into two main categories, a) the *underlying* stocks, such as shares, bonds, foreign currencies, and b) their *derivatives*, which are contracts that guarantee a fixed payment or delivery of the underlying in the future, regardless of any changes in the stock's price. Derivatives can be used to reduce risk of adverse movements of a stock price, but this reduction of risk comes at a price that, in most cases, must be paid between the two parties when the contract is initially signed. The fair price of the derivative is such that none of the parties can build a trading strategy that yields a risk-free profit at the expense of the other party.

A forward contract is a type of derivative that illustrates the concept of fairness in a simple way. When this contract is signed, the *long* party agrees to buy a stock for a fixed price K at a future time t

(the expiration time) regardless of any changes in the stock's price. At expiration time, the long party receives a payoff $S_t - K$, where S_t is the price of the stock at t. Of course, the parties aim to agree on a *fair value* of K. One way to find K is by building a parallel portfolio that ensures that at terminal time, each position can be closed without incurring any loss. The following provides a trading strategy for the short party:

- At initial time, the short party borrows S_0 amount of money and uses it to buy one unit of the stock.
- At expiry time the short party sells the stock to the long party and gets K in return. In addition, they return the loan plus the compounded interests which amounts to $S_0 \cdot e^{rt}$, where r is the interest rate.

If $K < S_0 \cdot e^{rt}$ the short party incurs a loss. Using a symmetric argument, one can see that for $K > S_0 \cdot e^{rt}$ the long party incurs a loss. Thus, for any value different than $S_0 \cdot e^{rt}$, some side of the contract can take advantage of the free money left on the table via the construction of the above sequence of actions. This is called an *arbitrage* opportunity. Rational agents will only enter a forward contract with $K = S_0 \cdot e^{rt}$ and any other value creates an arbitrage opportunity.

In this example, we have been able to find the fair value of K without any further assumptions on S_t. For more complex derivatives, the above procedure cannot be applied anymore and additional assumptions are needed to find the fair value of the contract.

This is illustrated in a second example: a *call option* is a derivative where the long party has the option but not the obligation to buy the stock at expiration time t. At expiration time the holder of the option only buys the stock if $S_t > K$, hence the above procedure cannot be applied as the short party is not guaranteed to be able to sell their stock to get K in return.

Of course, upon entering a call option contract the long party has to pay some amount of money called the *premium* to the short party otherwise it would already constitute a risk-free profit opportunity. The price of the derivative is therefore encoded in the premium, and not in the strike as it was in the case of the forward contract. The life of the option is as follows,

- At initial time, two parties sign a call option contract with strike K. The long party pays some premium v_0 to the short party.
- At expiration time, the long party exercises the option, getting $\max(S_t - K, 0)$ as a payoff.

The fair price of the option v_0 must depend on some future belief of the relative position of S_t with respect to K. In addition, the derivation of v_0 must also include the expiration time and the strike value. It becomes necessary to model the price of the underlying risky assets $(S_t)_{t\in[0,T]}$ as well as the price of some risk-free asset $(B_t)_{t\in[0,T]}$ as a stochastic processes.

It turns out that no-arbitrage pricing of the option is closely linked to the existence of a martingale measure (or risk-neutral measure) via the 1st Fundamental Theorem of Asset Pricing (see for example [5]). This theorem tells us that if we are able to build a probability measure equivalent[1] to the real-world measure under which, in average, it makes no difference to invest all the wealth in the risky asset or in the risk-free asset, then there is no trading strategy that makes a risk-free profit.

An important consequence of this theorem is that in order to keep the model free of arbitrage, the fair price of the option also needs to satisfy the above. In other words, the discounted price of the option is a local martingale under the risk-neutral measure. The following example puts these concepts together.

[1] two measures $\mathbb{P}, \mathbb{Q}$ of a probability space $(\Omega, \mathcal{F})$ are equivalent if non-zero probability events under one of the measure also have non-zero probability under the other measure

Example 1.1.1. *We consider a simple financial market with one risky asset and a risk-free bond. We have a probability space $(\Omega, \mathcal{F}, \mathbb{P})$ and an $\mathbb{R}-$valued Wiener process $(W_t)_{t\in[0,T]}$ generating a filtration $(\mathcal{F}_t)_{t\in[0,T]}$. We consider a derivative with payoff h, with h an $\mathcal{F}_T$-measurable random variable. We call $\mathbb{P}$ the real world measure, and we model the market with one risk-free asset (or bond) and one risky asset with price processes as follows*

$$\begin{aligned} dB_t &= r_t B_t dt, \quad B_0 = 1 \\ dS_t &= \mu_t S_t dt + \sigma_t S_t dW_t, \quad S_0 = s_0. \end{aligned} \tag{1.1}$$

We set $\varphi_t := (\mu_t - r_t)/\sigma_t$, and we define γ_T as

$$\gamma_T := \exp\left(-\int_0^T \varphi_s^2 ds - \int_0^T \varphi_s dW_s \right).$$

If $\mathbb{E}\gamma_T = 1$, then by Girsanov's theorem the measure $\mathbb{Q}$ defined by $\mathbb{Q}(A) = \mathbb{E}^{\mathbb{P}}(\mathbf{1}_A \gamma_T)$ is a risk-neutral measure, i.e. the process $\tilde{S}_t := S_t/B_t$ is a local martingale under $\mathbb{Q}$. Furthermore, in order to keep the model free of arbitrage, the fair price of the derivative with payoff h needs to satisfy

$$v_t = B_t \cdot \mathbb{E}^{\mathbb{Q}}\left[h/B_T \middle| \mathcal{F}_t \right],$$

i.e. $\tilde{v}_t := v_t/B_t$ is also a $\mathbb{Q}$-local martingale.

The above example provides a three-steps algorithm to derive the fair price of the option, under the presence of market data.

Algorithm 1 Financial market calibration and option pricing

a) Handcraft a (parsimonious) model such as (1.1) that best captures the market dynamics.
b) Calibrate the parameters of the model to real-world data. In the above (1.1), r_t can be calibrated using bonds data; the risk parameter σ_t is calibrated using options prices. By working under $\mathbb{Q}$, it becomes unnecessary to calibrate μ_t.
c) Approximate v_0 by Monte Carlo sampling from the discounted payoff under the risk-neutral measure.

If the coefficients of the handcrafted SDE model satisfy the usual regularity assumptions for the SDE to have a strong solution, and if the option payoff belongs to $\mathcal{L}^2(\Omega)$, then Feynman-Kac formula provides an alternative representation of v_0 as the solution of a linear parabolic PDE.

So far we have only considered two simple derivatives, but the type of contracts vary in form and complexity. Pricing by Monte Carlo sampling can become expensive and inaccurate. On the other hand, solving a high dimensional linear parabolic PDE suffers from the curse of dimensionality when one uses finite elements solvers. In this thesis, we explore machine learning methods to solve the above problem in an efficient, accurate and data-driven manner.

1.2 Thesis outline

1.2.1 High-dimensional PDE approximation

The initial project that inspired this PhD was based on [1]. We aimed to extend its algorithm to find the solution of the parametric PDE over the whole space domain (as opposed to [1] that was approximating the solution of the PDE at a single point) and whole parameters domain while being able to control the

approximation error arising from the neural network parameterisation of the solution. The resulting algorithm leverages Monte Carlo sampling from the probabilistic representation of the PDE and deep learning to obtain an unbiased and low-variance approximation of the solution of the PDE.

In Chapters 2 and 3 we propose novel algorithms leveraging deep learning techniques and Monte Carlo methods to provide an unbiased solver of high-dimensional PDEs. In the particular case of PDEs arising from option pricing theory, our algorithms can be applied in the setup where we may want to price an option under a given parsimonious market model. Step (a) of algorithm 1 is therefore given. Furthermore, we assume the PDE to be parametric. This means that our proposed solver is actually solving a family of PDEs, which is particularly convenient if one wants to calibrate the parameters of the market model to real-world data, as our solver provides the gradient of the price in terms of the model's parameters.

Chapter 2 was published as a paper in the journal *Applied Mathematical Finance*, [6].

In Chapter 3 we extend our solvers introduced in Chapter 2 to consider exotic options (derivatives with path-dependent payoffs), as well as more general financial market models. In such settings, the option price is path-dependent. Our parametrization takes this into account by combining recurrent neural networks and path signatures, a path encoder that arises from rough path theory and that keeps a surprising amount of information of a trajectory.

1.2.2 Time series generative methods - Data-driven market generator

Chapters 4 and 5 flip the paradigm in algorithm 1. Instead of pricing an option under the presence of a calibrated handcrafted model, we design a deep learning based model that learns the market dynamics under the risk-neutral measure that in addition provides robust bounds for illiquid option prices not present in the training data. Our model keeps the form of an SDE, but its coefficients are parameterised by neural networks. Depending on the available market data at hand, we provide two different training algorithms.

Chapter 4 was published as a paper in the *Journal of computational finance*, [7]. This paper was jointly written with my supervisors Lukasz Szpruch and David Siska, and the PhD student Patryk Gierjatowicz. In addition to writing the paper, I coded all the proposed algorithms presented in the paper, and I ran the simulations for the Local Stochastic Volatility models. Patryk ran the simulations for the Local Volatility model.

Finally, Chapter 5 studies a novel algorithm to train path generative models such as Neural SDEs. If one has a dataset containing path samples from the target distribution, one can use a GAN (generative adversarial network) approach to optimise the generative model's parameters. Furthermore, we use the powerful properties of the path signature to derive an efficient training algorithm. This paper was accepted and included in the *Proceedings of the Second ACM International Conference on AI in Finance*, [8]. This paper was jointly written with Hao Ni and her group. For the numerical experiments section, I coded the LogSigRNN and the Sig-Wasserstein distance for the Rough Volatility model example.

1.3 Code

A relevant load of work included in this thesis comes from implementing and running the proposed algorithms. All the code that I have produced during these years, including the projects in this thesis is available at `https://github.com/msabvid`.

Chapter 2

Unbiased deep solvers for linear parametric deep PDEs

We develop several deep learning algorithms for approximating families of parametric PDE solutions. The proposed algorithms approximate solutions together with their gradients, which in the context of mathematical finance means that the derivative prices and hedging strategies are computed simultaneously. Having approximated the gradient of the solution one can combine it with a Monte-Carlo simulation to remove the bias in the deep network approximation of the PDE solution (derivative price). This is achieved by leveraging the Martingale Representation Theorem and combining the Monte Carlo simulation with the neural network. The resulting algorithm is robust with respect to quality of the neural network approximation and consequently can be used as a black-box in case only limited a priori information about the underlying problem is available. We believe this is important as neural network based algorithms often require fair amount of tuning to produce satisfactory results. The methods are empirically shown to work for high-dimensional problems (e.g. 100 dimensions). We provide diagnostics that shed light on appropriate network architectures.

2.1 Introduction

Numerical algorithms that solve PDEs suffer from the so-called "curse of dimensionality", making it impractical to apply known discretisation algorithms such as finite difference schemes to solve high-dimensional PDEs. However, it has been recently shown that deep neural networks trained with stochastic gradient descent can overcome the curse of dimensionality [9, 10], making them a popular choice to solve this computational challenge in the last few years.

In this work, we focus on the problem of numerically solving parametric linear PDEs in high dimensions arising from Europen option pricing with many assets. Let $B \subseteq \mathbb{R}^p, p \geq 1$ be a parameter space (for instance, in the Black–Scholes equation with fixed rate, B is the domain of the volatility parameter). Consider $v = v(t, x; \beta)$ satisfying

$$\begin{aligned} &\left[\partial_t v + b\nabla_x v + \frac{1}{2}\text{tr}\left[\nabla_x^2 v\sigma^*\sigma\right] - cv\right](t, x; \beta) = 0\,, \\ &v(T, x; \beta) = g(x; \beta)\,, \;\; t \in [0, T]\,, \;\; x \in \mathbb{R}^d\,, \;\; \beta \in B\,. \end{aligned} \tag{2.1}$$

Here $t \in [0, T]$, $x \in \mathbb{R}^d$ and $\beta \in B$ and b, σ, c and g are functions of $(t, x; \beta)$ which specify the problem.

The Feynman–Kac theorem provides a probabilistic representation for v so that Monte Carlo methods can be used for its unbiased approximation in one single point $(t, x; \beta)$. What we propose in this work is a method for harnessing the power of deep learning algorithms to numerically solve (2.1) in a way that is robust even in edge cases when the output of the neural network is not of the expected quality, that is when the neural network does not generalise well on points that are not seen during the trainig process, by combining them with Monte Carlo algorithms.

From the results in this thesis we observe that neural networks provide an efficient computational device for high dimensional problems. However, we observed that these algorithms are sensitive to the network architecture, parameters and distribution of training data. A fair amount of tuning is required to obtain good results. Based on this we believe that there is great potential in combining artificial neural networks with already developed and well understood probabilistic computational methods, in particular the control variate method for using potentially imperfect neural network approximations for finding unbiased solutions to a given problem, see Algorithm 2.

2.1.1 Main contributions

We propose three classes of learning algorithms for simultaneously finding solutions and gradients to parametric families of PDEs.

i) *Projection solver*: See Algorithm 3. We leverage Feynman–Kac representation together with the fact that conditional expectation can be viewed as an L^2-projection operator. The gradient can be obtained by automatic differentiation of already obtained approximation of the PDE solution.

ii) *Martingale representation solver*: See Algorithm 4. This algorithm was inspired by Cvitanic et. al. [11] and Weinan et. al, Han et. al. [2, 1] and is referred to as deep BSDE solver. Our algorithm differs from [2] in that we approximate solution and its gradient at all the time-steps and across the entire space and parameter domains rather than only one space-time point. Furthermore we propose to approximate the solution-map and its gradient by separate networks.

iii) *Martingale control variates solver*: Algorithms 5 and 6. Here we exploit the fact that martingale representation induces control variate that can produce zero variance estimator. Obviously, such control variate is not implementable but provides a basis for a novel learning algorithm for the PDE solution.

For each of these classes of algorithms we develop and test different implementation strategies. Indeed, one can either take one (large) network to approximate the entire family of solutions of (2.1) or take a number of (smaller) networks, where each of them approximates the solution at a time point in a grid. The former has the advantage that one can take arbitrarily fine time discretisation without increasing the overall network size. The advantage of the latter is that each learning task is simpler due to each network being smaller. One can further leverage the smoothness of the solution in time and learn the weights iteratively by initialising the network parameters to be those of the previous time step. We test both approaches numerically. At a high level all the algorithms work in path-dependent (non-Markovian) setting but there the challenge is an efficient method for encoding information in each path. This problem is solved in chapter 3, also published in [12].

To summarise the key contribution of this chapter are:

i) We derive and implement three classes of learning algorithms for approximation of parametric PDE solution map and its gradient.

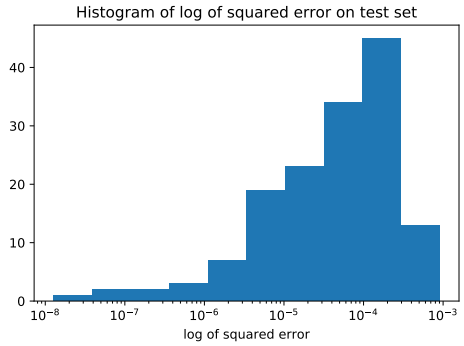


Figure 2.1: Histogram of mean-square-error of solution to the PDE on the test data set.

ii) We propose a novel iterative training algorithm that exploits regularity of the function we seek to approximate and allows using neural networks with smaller number of parameters.

iii) The proposed algorithms are truly black-box in that quality of the network approximation only impacts the computation benefit of the approach and does not introduce approximation bias. This is achieved by combining the network approximation with Monte Carlo as stated in Algorithm 2.

iv) Code for the numerical experiments presented in this paper is being made available on GitHub: `https://github.com/msabvid/Deep-PDE-Solvers`.

We stress the importance of point iii) above by directing reader's attention to Figure 2.1, where we test generalisation error of trained neural network for the 5 dimensional family of PDEs corresponding to pricing a basket option under the Black–Scholes model. We refer reader to Example 2.D.2 for details. We see that while the average error over test set is of order $\approx 10^{-5}$, the errors for a given input varies significantly. Indeed, it has been observed in deep learning community that for high dimensional problems one can find input data such that trained neural network that appears to generalise well (i.e achieves small errors on the out of training data) produces poor results [13].

2.1.2 Literature review

Deep neural networks trained with stochastic gradient descent proved to be extremely successful in number of applications such as computer vision, natural language processing, generative models or reinforcement learning [14]. The application to PDE solvers is relatively new and has been pioneered by Weinan et. al, Han et. al. [2, 1, 15]. See also Cvitanic et. al. [11] for the ideas of solving PDEs with gradient methods and for direct PDE approximation algorithm. PDEs provide an excellent test bed for neural networks approximation because a) there exists alternative solvers e.g Monte Carlo b) we have well developed theory for PDEs, and that knowledge can be used to tune algorithms. This is in contrast to mainstream neural networks approximations in text or images classification.

Apart from growing body of empirical results in literature on "Deep PDEs solvers", [16, 17, 18, 19, 20] there has been also some important theoretical contributions. It has been proved that deep artificial neural networks approximate solutions to parabolic PDEs to an arbitrary accuracy without suffering from the curse of dimensionality. The first mathematically rigorous proofs are given in [21] and [22]. The high level idea is to show that neural network approximation to the PDE can be established by building on Feynman-Kac approximation and Monte-Carlo approximation. By checking that Monte-Carlo simulations do not suffer from the curse of dimensionality one can infer that the same is true for

neural network approximation. Furthermore, it has been recently demonstrated in [23, 24] that noisy gradient descent algorithm used for training of neural networks of the form considered in [21, 22] induces a unique probability distribution function over the parameter space which minimises learning. See [25, 26, 27, 28, 29, 30] for related ideas on convergence of gradient algorithms for overparametrised neural networks. This means that there are theoretical guarantees for the approximation of (parabolic) PDEs with neural networks trained by noisy gradient methods alleviating the curse of dimensionality.

An important application of deep PDE solvers is that one can in fact approximate a parametric family of solutions of a PDE. To be more precise let $B \subseteq \mathbb{R}^p$, $p \geq 1$, be a parameter space. In the context of finance these, for example, might be initial volatility, volatility of volatility, interest rate and mean reversion parameters. One can approximate the parametric family of functions $F(\cdot;\beta)_{\beta\in B}$ for an arbitrary range of parameters. This then allows for swift calibration of models to data (e.g options prices). This is particularly appealing for high dimensional problems when calibrating directly using noisy Monte-Carlo samples might be inefficient. This line of research gained some popularity recently and the idea has been tested numerically on various models and data sets [31, 32, 33, 34, 35, 36, 37]. There are some remarks that are in order. In the context of models calibration, while the training might be expensive one can do it offline, once and for good. One can also notice that training data could be used to produce a "look-up table" taking model parameters to prices. From this perspective the neural network, essentially, becomes an interpolator and a compression tool. Indeed the number of parameters of the network is much smaller than number of training data and therefore it is more efficient to store those. The final remark is that while there are other methods out there, such as Chebyshev functions, neural networks seem robust in high dimensions which make them our method of choice.

2.1.3 Notation

We denote by $\mathcal{DN}$ the set of all fully connected feedforward neural networks (see Appendix 2.C). We also use $\mathcal{R}[f]_\theta \in \mathcal{DN}$ with $\theta \in \mathbb{R}^\kappa$ to denote a neural network with weights θ approximating the function $f : \mathbb{R}^{d_0} \to \mathbb{R}^{d_1}$ for some $d_0, d_1 \in \mathbb{N}$.

2.1.4 Outline

This paper is organised as follows. Section 2.2 provides theoretical underpinning for the derivation of all the algorithms we propose to solve (2.1). More specifically in Section 2.2.2 we combine the approximation of the gradient of the solution of the PDE resulting from the Deep Learning algorithms with Monte Carlo to obtain an unbiased approximation of the solution of the PDE. In Section 2.3, we describe the algorithms in detail.

Finally in Section 2.4 we provide numerical tests of the proposed algorithms. We empirically test these methods on relevant examples including a 100 dimensional option pricing problems, see Examples 2.4.4 and 2.D.3. We carefully measure the training cost and report the variance reduction achieved.

Since we work in a situation where the function approximated by the neural network can be obtained via other methods (Monte-Carlo, PDE solution) we are able to test the how the expressiveness of fully connected artificial neural networks depends on the number of layers and neurons per layer. See Section 2.D.1 for details.

2.2 PDE Martingale control variate

Control variate is one of the most powerful variance reduction techniques for Monte-Carlo simulation. While a good control variate can reduce the computational cost of Monte-Carlo computation by several orders of magnitude, it relies on judiciously chosen control variate functions that are problem specific. For example, when computing price of basket options a sound strategy is to choose control variates to be call options written on each of the stocks in the basket, since in many models these are priced by closed-form formulae. In this chapter, we are interested in black-box-type control variate approach by leveraging the Martingale Representation Theorem and neural networks. The idea of using Martingale Representation to obtain control variates goes back at least to [38]. It has been further studied in combination with regression in [39] and [40].

The bias in the approximation of the solution can be completely removed by employing control variates where the deep network provides the control variate resulting in very high variance reduction factor in the corresponding Monte Carlo simulation.

Let $(\Omega, \mathcal{F}, \mathbb{P})$ be a probability space and consider an $\mathbb{R}^{d'}$-valued Wiener process $W = (W^j)_{j=1}^{d'} = ((W_t^j)_{t\geq 0})_{j=1}^{d'}$. We will use $(\mathcal{F}_t^W)_{t\geq 0}$ to denote the filtration generated by W. Consider a $D \subseteq \mathbb{R}^d$-valued, continuous, stochastic process defined for the parameters $\beta \in B \subseteq \mathbb{R}^p$, $X^\beta = (X^{\beta,i})_{i=1}^d = ((X_t^{\beta,i})_{t\geq 0})_{i=1}^d$ adapted to $(\mathcal{F}_t^W)_{t\geq 0}$ given as the solution to

$$dX_t^\beta = b(t, X_t^\beta; \beta)\, dt + \sigma(t, X_t; \beta)\, dW_t, \;\; t \in [0, T]\,, \;\; X_0^\beta = x \in \mathbb{R}^d. \tag{2.2}$$

We will use $(\mathcal{F}_t^\beta)_{t\geq 0}$ to denote the filtration generated by X^β.

Let $g : \mathbb{R}^d \to \mathbb{R}$ be a measurable function and we assume that there is a (stochastic) discount factor given by

$$D(t_1, t_2; \beta) := e^{-\int_{t_1}^{t_2} c(s, X_s^\beta; \beta)\, ds}$$

for an appropriate function $c = c(t, x; \beta)$. We will omit β from the discount factor notation for brevity. We now interpret $\mathbb{P}$ as some risk-neutral measure and so the $\mathbb{P}$-price of our contingent claim is

$$v(t, x; \beta) := \mathbb{E}\left[D(t, T) g(X_T^\beta) \middle| X_t^\beta = x \right]. \tag{2.3}$$

Say we have iid r.v.s $(X_T^{\beta,i})_{i=1}^N$ with the same distribution as X_T^β, where for each i, $X_t^{\beta,i} = x$. Then the standard Monte-Carlo estimator is

$$v^N(t, x; \beta) := \frac{1}{N} \sum_{i=1}^N D^i(t, T) g(X_T^{\beta,i})\,.$$

Convergence $v^N(t, x; \beta) \to v(t, x; \beta)$ in probability as $N \to \infty$ is granted by the Law of Large Numbers (see for example Chapter 5 in [41]). Moreover the classical Central Limit Theorem (see again Chapter 5 in [41]) tells that

$$\mathbb{P}\left(v(t, x; \beta) \in \left[v^N(t, x; \beta) - z_{\alpha/2} \frac{\sigma}{\sqrt{N}}, v^N(t, x; \beta) + z_{\alpha/2} \frac{\sigma}{\sqrt{N}} \right] \right) \to 1 - \alpha \text{ as } N \to \infty\,,$$

where $\sigma := \sqrt{\mathbb{V}ar\left[D(t, T) g(X_T^\beta) \right]}$ and $z_{\alpha/2}$ is such that $1 - \Phi(z_{\alpha/2}) = \alpha/2$ with Φ being the distribution function (cumulative distribution function) of the standard normal distribution. To decrease the width of the confidence intervals one can increase N, but this also increases the computational cost.

A better strategy is to reduce variance by finding an alternative Monte-Carlo estimator, say $\mathcal{V}^N(t,x;\beta)$, such that

$$\mathbb{E}[\mathcal{V}^N(t,x;\beta)] = v(t,x;\beta) \quad \text{and} \quad \mathbb{V}ar[\mathcal{V}^N(t,x;\beta)] < \mathbb{V}ar[v^N(t,x;\beta)], \tag{2.4}$$

and the cost of computing $\mathcal{V}^N(t,x;\beta)$ is similar to $v^N(t,x;\beta)$.

In the remainder of the chapter we will devise and test several strategies, based on deep learning, to find a suitable approximation for $\mathcal{V}^N(t,x;\beta)$, by exploring the connection of the SDE (2.2) and its associated PDE.

2.2.1 PDE derivation of the control variate

It can be shown that under suitable assumptions on b, σ, c and g, and fixed $\beta \in B$ that $v \in C^{1,2}([0,T] \times D)$. See e.g. [42]. Let $a := \frac{1}{2}\sigma\sigma^*$. Then, from Feynman–Kac formula (see e.g. Th. 8.2.1 in [43]), we get

$$\begin{cases} \left[\partial_t v + \text{tr}(a\partial_x^2 v) + b\partial_x v - cv\right](t,x;\beta) = 0 \quad \text{in } [0,T) \times D\,, \\ v(T,\cdot) = g \quad \text{on } D\,. \end{cases} \tag{2.5}$$

Since $v \in C^{1,2}([0,T] \times D)$ and since v satisfies the above PDE, if we apply Itô's formula then we obtain

$$D(t,T)v(T, X_T^\beta;\beta) = v(t,x;\beta) + \int_t^T D(t,s)\partial_x v(s, X_s^\beta;\beta)\sigma(s, X_s^\beta;\beta)\, dW_s\,. \tag{2.6}$$

Hence Feynman-Kac representation together with the fact that $v(T, X_T^\beta;\beta) = g(X_T^\beta)$ yields

$$v(t,x;\beta) = D(t,T)g(X_T^\beta) - \int_t^T D(t,s)\partial_x v(s, X_s^\beta;\beta)\sigma(s, X_s^\beta;\beta)\, dW_s. \tag{2.7}$$

Provided that

$$\sup_{s\in[t,T]} \mathbb{E}[|D(t,s)\partial_x v(s, X_s^\beta;\beta)\sigma(s, X_s^\beta;\beta)|^2] < \infty\,,$$

then the stochastic integral is a martingale. Thus we can consider the Monte-Carlo estimator:

$$\mathcal{V}^N(t,x;\beta) := \frac{1}{N}\sum_{i=1}^N \left\{ D^i(t,T)g(X_T^{\beta,i}) - \int_t^T D^i(t,s)\partial_x v(s, X_s^{\beta,i};\beta)\sigma(s, X_s^{\beta,i};\beta)\, dW_s^i \right\}. \tag{2.8}$$

To obtain a control variate we thus need to approximate $\partial_x v$. If one used classical approximation techniques to the PDE, such as finite difference or finite element methods, one would run into the curse of the dimensionality - the very reason one employs Monte-Carlo simulations in the first place. Artificial neural networks have been shown to break the curse of dimensionality in specific situations [21]. To be more precise, authors in [10, 44, 22, 45, 46, 21, 47, 48, 49] have shown that there always exist a deep feed forward neural network and some parameters such that the corresponding neural network can approximate the solution of a linear PDE arbitrarily well in a suitable norm under reasonable assumptions (terminal condition and coefficients can be approximated by neural networks). Moreover the number of parameters grows only polynomially in dimension and so there is no curse of dimensionality. However, while the papers above construct the network they do not tell us how to find the "good" parameters. In practice the parameter search still relies on gradient descent-based minimisation over a non-convex landscape. The application of the deep-network approximation to the solution of the PDE as a martingale control variate is an ideal compromise.

If there is no exact solution to the PDE (2.5), as would be the case in any reasonable application, then

we will approximate $\partial_x v$ by $\mathcal{R}[\partial_x v]_\theta \in \mathcal{DN}$.

To obtain an implementable algorithm we discretise the integrals in $\mathcal{V}_t^{\beta,N,v}$ and take a partition of $[0,T]$ denoted $\pi := \{t = t_0 < \cdots < t_{N_{\text{steps}}} = T\}$, and consider an approximation of (2.2) by $(X_{t_k}^{\beta,\pi})_{t_k \in \pi}$. For simplicity we approximate all integrals arising by Riemann sums always taking the left-hand point when approximating the value of the integrand.

The implementable control variate Monte-Carlo estimator is then the form

$$\begin{aligned}\mathcal{V}^{\pi,\theta,\lambda,N}(t,x;\beta) := \frac{1}{N}\sum_{i=1}^{N}\Bigg\{&(D^\pi(t,T))^i g(X_T^{\beta,\pi,i}) \\ -\lambda \underbrace{\sum_{k=1}^{N_{\text{steps}}-1} (D^\pi(t,t_k))^i \mathcal{R}[\partial_x v]_\theta(t_k, X_{t_k}^{\beta,\pi,i};\beta)\sigma(t_k, X_{t_k}^{\beta,\pi,i};\beta)\left(W_{t_{k+1}}^i - W_{t_k}^i\right)}_{=:\mathcal{I}}&\Bigg\},\end{aligned} \tag{2.9}$$

where $D^\pi(t,T) := e^{-\sum_{k=1}^{N_{\text{steps}}-1} c(t_k, X_{t_k}^{\beta,\pi})(t_{k+1}-t_k)}$ and λ is a free parameter to be chosen (because we discretise and use approximation to the PDE it is expected $\lambda \neq 1$). Again, we point out that the only bias of the above estimator comes from the numerical scheme used to solve the forward and backward processes. Nevertheless, $\mathcal{R}[\partial_x v]_\theta$ does not add any additional bias independently of the choice θ. We will discuss possible approximation strategies for approximating $\partial_x v$ with $\mathcal{R}[\partial_x v]_\theta$ in the following section.

Remark 2.2.1. Note that for a fixed θ, one can find the optimal value of λ such that the variance of $\mathcal{V}^{\pi,\theta,\lambda,N}(t,x;\beta)$ is minimised (see chapter 4 of [50]). Furthermore, for this optimal value of λ the strength of the fitted control variate, measured by the variance reduction ratio

$$\frac{\mathbb{V}ar((D^\pi(t,T))g(X_T^{\beta,\pi}) - \lambda\cdot\mathcal{I})}{\mathbb{V}ar((D^\pi(t,T))g(X_T^{\beta,\pi})}$$

is $1-\rho^2$, with $\rho := Corr((D^\pi(t,T))g(X_T^{\beta,\pi}),\mathcal{I})$ (see again chapter 4 of [50] for a detailed calculation). It follows that the value of λ is set after training is complete and θ is fixed.

In this section we have actually derived an explicit form of the Martingale representation (see e.g. [51, Th. 14.5.1]) of $D(t,T)g(X_T^\beta)$ in terms of the solution of the PDE associated to the process X^β, which is given as the solution to (2.2). In Appendix 2.A we provide a more general framework to build a low-variance Monte Carlo estimator $\mathcal{V}_t^N$ for any (possibly non-Markovian) $\mathcal{F}^W$-adapted process X^β.

2.2.2 Unbiased Parametric PDE approximation

After having trained the networks $\mathcal{R}[\partial_x v]_\theta$ (using any of Algorithms 3, 4, 5, 6 that we will introduce in Section 2.3) and $\mathcal{R}[v]_\eta$ (using any of Algorithms 3, 4) that approximate v and $\partial_x v$ respectively one then has two options to approximate $v(t,x_t;\beta)$

i) Directly with $\mathcal{R}[v]_\eta(t,x_t;\beta)$ if Algorithms 3 or 4 were used, which will introduce some approximation bias.

ii) By combining $\mathcal{R}[\partial_x v]_\theta$ with the Monte Carlo approximation of $v(t,x_t;\beta)$ using (2.9), which will yield an unbiased estimator of $v(t,x_t;\beta)$. The complete method is stated as Algorithm 2.

Algorithm 2 Unbiased parametric PDE solver

Input: t, x, β where $t \in \pi$.
Initialisation: θ, N_{trn}
for $i : 1 : N_{\text{trn}}$ **do**
  Generate samples $(x_t^{\beta,\pi,i})_{t\in\pi}$ by using numerical SDE solver of (2.2).
end for
Find the optimal weights $\theta^{*,N_{\text{trn}}}$ of $\mathcal{R}[\partial_x v]_\theta$ using one of Algorithms 3, 4, 5, 6.
return $\mathcal{V}_{t,T}^{\beta,\pi,\theta^*,\lambda,N}$ as defined in (2.9) with θ replaced by $\theta^{*,N_{\text{trn}}}$.

2.3 Deep PDE solvers

In this section we propose two algorithms that learn the PDE solution (or its gradient) and then use it to build control a variate using (2.9). We also include in the Appendix 2.B an additional algorithm to solve such linear PDEs using deep neural networks.

2.3.1 Projection solver

Before we proceed further we recall a well known property of conditional expectations, for proof see e.g. [52, Ch.3 Th. 14].

Theorem 2.3.1. *Let $\mathcal{X} \in L^2(\mathcal{F})$. Let $\mathcal{G} \subset \mathcal{F}$ be a sub σ-algebra. There exists a random variable $Y \in L^2(\mathcal{G})$ such that*

$$\mathbb{E}[|\mathcal{X} - \mathcal{Y}|^2] = \inf_{\eta\in L^2(\mathcal{G})} \mathbb{E}[|\mathcal{X} - \eta|^2].$$

The minimiser, $\mathcal{Y}$, is unique and is given by $\mathcal{Y} = \mathbb{E}[\mathcal{X}|\mathcal{G}]$.

The theorem tells us that conditional expectation is an orthogonal projection of a random variable X onto $L^2(\mathcal{G})$. Instead of working directly with (2.5) we work with its probabilistic representation (2.6). To formulate the learning task, we replace $\mathcal{X}$ by $D(t,T)g(X_T^\beta)$ so that $v(t, X_t^\beta; \beta) = \mathbb{E}[\mathcal{X}|X_t^\beta]$. Hence, by Theorem (2.3.1),

$$\mathbb{E}[|\mathcal{X} - v(t, X_t^\beta; \beta)|^2] = \inf_{\eta\in L^2(\sigma(X_t^\beta))} \mathbb{E}[|\mathcal{X} - \eta|^2]$$

and we know that for a fixed t the random variable which minimises the mean square error is a function of X_t. But by the Doob–Dynkin Lemma [51, Th. 1.3.12] we know that every $\eta \in L^2(\sigma(X_t))$ can be expressed as $\eta = h_t(X_t^\beta)$ for some appropriate measurable h_t. For the practical algorithm we restrict the search for the function h_t to the class that can be expressed as deep neural networks $\mathcal{DN}$. Hence we consider a family of functions $\mathcal{R}_\theta \in \mathcal{DN}$ and set learning task as

$$\theta^* := \arg\min_\theta \mathbb{E}_\beta \left[\mathbb{E}_{(X_t^{\beta,\pi})_{t\in\pi}} \left[\sum_{k=0}^{N_{\text{steps}}} \left(D(t_k,T)g(X_T^{\beta,\pi}) - \mathcal{R}[v]_{\theta_{t_k}}(X_{t_k}^{\beta,\pi};\beta) \right)^2 \right] \right]. \tag{2.10}$$

The inner expectation in (2.10) is taken across all paths generated using numerical scheme on (2.2) for a fixed β and it allows to solve the PDE (2.5) for such β. The outer expectation is taken on β for which the distribution is fixed beforehand (e.g. uniform on B if it is compact), thus allowing the algorithm to find the optimal neural network weights θ^* to solve the parametric family of PDEs (2.5). Automatic differentiation is used to approximate $\partial_x v$. Algorithm 3 describes the method.

Algorithm 3 Projection solver

Initialisation: θ, N_{trn}, distribution of β.
for $i : 1 : N_{\text{trn}}$ **do**
Generate training set:

- Sample different values of β from a fixed distribution (for example Uniform distribution).
- Generate $(x_t^{\beta,\pi,i})_{t\in\pi}$ by using numerical SDE solver on (2.2)

end for
Use SGD to find $\theta^{*,N_{\text{trn}}}$ where

$$\theta^{*,N_{\text{trn}}} = \arg\min_{\theta} \mathbb{E}^{\mathbb{P}^{N_{\text{trn}}}} \left[\sum_{k=0}^{N_{\text{steps}}-1} \left(D(t_k,T) g(X_T^{\beta,\pi}) - \mathcal{R}[v]_{\theta_{t_k}}(X_{t_k}^{\beta,\pi};\beta) \right)^2 \right]$$

Where $\mathbb{E}^{\mathbb{P}^{N_{\text{trn}}}}$ denotes the empirical mean.
Automatic differentiation applied to $\mathcal{R}[v]_{\theta^*_{t_k}}(x_{t_k}^{\beta,\pi};\beta)$ can be used to approximate $\partial_x v$.
return $\theta^{*,N_{\text{trn}}}$.

2.3.2 Probabilistic representation based on Backward SDE

Instead of working directly with (2.5) we work with its probabilistic representation (2.6) and view it as a BSDE. To formulate the learning task based on this we recall the time-grid π so that we can write it recursively as

$$\begin{aligned} v(t_{N_{\text{steps}}}, X^{\beta}_{t_{N_{\text{steps}}}};\beta) &= g(X^{\beta}_{t_{N_{\text{steps}}}})\,, \\ D(t,t_{m+1}) v(t_{m+1}, X^{\beta}_{t_{m+1}};\beta) &= D(t,t_m) v(t_m, X^{\beta}_{t_m};\beta) \\ &\quad + \int_{t_m}^{t_{m+1}} D(t,s)\partial_x v(s, X^{\beta}_s;\beta)\sigma(s, X^{\beta}_s;\beta)\, dW_s \ \text{ for } \ m = 0,1,\ldots,N_{\text{steps}}-1\,. \end{aligned}$$

Next consider deep network approximations for each time step in π and for both the solution of (2.5) and its gradient.

$$\mathcal{R}[v]_{\eta_m}(x;\beta) \approx v(t_m,x;\beta)\,, \quad t_m \in \pi\,, \ \ x \in \mathbb{R}^d$$

and

$$\mathcal{R}[\partial_x v]_{\theta_m}(x;\beta) \approx \partial_x v(t_m,x;\beta)\,, \quad t_m \in \pi\,, \ \ x \in \mathbb{R}^d\,.$$

Approximation depends on weights $\eta_m \in \mathbb{R}^{k_\eta}$, $\theta_m \in \mathbb{R}^{k_\theta}$. We then set the learning task as

$$\begin{aligned} (\eta^*,\theta^*) := \arg\min_{(\eta,\theta)} \mathbb{E}_{\beta,X^\beta} \Bigg[& \left| g(X^{\beta,\pi}_{t_{N_{\text{steps}}}}) - \mathcal{R}[v]_{\eta_{N_{\text{steps}}}}(X^{\beta,\pi}_{t_{N_{\text{steps}}}}) \right|^2 \\ & + \frac{1}{N_{\text{steps}}} \sum_{m=0}^{N_{\text{steps}}-1} |\mathcal{E}^{(\eta,\theta)}_{m+1}|^2 \Bigg], \end{aligned} \tag{2.11}$$

$$\begin{aligned} \mathcal{E}^{(\eta,\theta)}_{m+1} := & D(t,t_{m+1})\mathcal{R}[v]_{\eta_{m+1}}(X^{\beta,\pi}_{t_{m+1}};\beta) - D(t,t_m)\mathcal{R}[v]_{\eta_m}(X^{\beta,\pi};\beta) \\ & - D(t,t_m)\mathcal{R}[\partial_x v]_{\theta_m}(X^{\beta,\pi}_{t_m};\beta)\sigma(t_m, X^{\beta,\pi}_{t_m};\beta)\Delta W_{t_{m+1}}\,, \end{aligned}$$

where

$$\eta = \{\eta_0,\ldots,\eta_{t_{N_{\text{steps}}}}\},\ \theta = \{\theta_0,\ldots,\theta_{t_{N_{\text{steps}}}}\}\,.$$

The complete learning method is stated as Algorithm 4, where we split the optimisation (2.11) in several optimisation problems, one per time step: learning the weights θ_m or η_m at a certain time step $t_m < t_{N_{\text{steps}}}$

only requires knowing the weights η_{m+1}. At $m = N_{\text{steps}}$, learning the weights $\eta_{N_{\text{steps}}}$ only requires the terminal condition g. Note that the algorithm assumes that adjacent networks in time will be similar, and therefore we initialise η_m and θ_m by η^*_{m+1} and θ^*_{m+1}.

Remark 2.3.2. During the time that I was working on this thesis, algorithms similar to 4 were published, such as in [53], where the authors use non-linear Feynman Kac formula to design an algorithm to approximate the solution of a family of non-linear PDEs.

Algorithm 4 Martingale representation solver, iterative

Initialisation: N_{trn}
for $i : 1 : N_{\text{trn}}$ **do**
generate samples $(x_t^{\beta,\pi,i})_{t\in\pi}$ by using numerical SDE solver on (2.2) and sampling from the distribution of β.
end for
Initialisation: $\eta_{N_{\text{steps}}}$
Find $\eta^{*,N_{\text{trn}}}_{N_{\text{steps}}}$ using SGD where

$$\eta^{*,N_{\text{trn}}}_{N_{\text{steps}}} := \arg\min_{\eta} \frac{1}{N_{\text{trn}}} \sum_{i=1}^{N_{\text{trn}}} \left| g(x^{\beta,\pi,i}_{t_{N_{\text{steps}}}}) - \mathcal{R}[v]_{\eta_{N_{\text{steps}}}}(x^{\beta,\pi,i}_{t_{N_{\text{steps}}}}; \beta) \right|^2$$

for $m : N_{\text{steps}} - 1 : 0 : -1$ **do**
Initialise $(\theta_m, \eta_m) = (\theta^{*,N_{\text{trn}}}_{m+1}, \eta^{*,N_{\text{trn}}}_{m+1})$
Find $(\theta^{*,N_{\text{trn}}}_m, \eta^{*,N_{\text{trn}}}_m)$ using SGD where

$$(\theta^{*,N_{\text{trn}}}_m, \eta^{*,N_{\text{trn}}}_m) := \arg\min_{(\eta_m,\theta_m)} \frac{1}{N_{\text{trn}}} \sum_{i=1}^{N_{\text{trn}}} \left| \mathcal{E}^{\beta,\pi,i,(\eta,\theta)}_{m+1} \right|^2$$

where

$$\begin{aligned}\mathcal{E}^{\pi,i,(\eta,\theta)}_{m+1} :=& D^{\pi,i}(t,t_{m+1})\mathcal{R}[v]_{\eta_{m+1}}(x^{\beta,\pi,i}_{t_{m+1}};\beta) - D^{\pi,i}(t,t_m)\mathcal{R}[v]_{\eta_m}(x^{\beta,\pi,i}_{t_m};\beta) \\ &- D^{\pi,i}(t,t_m)\mathcal{R}[\partial_x v]_{\theta_m}(x^{\beta,\pi,i}_{t_m};\beta)\sigma(t_m, x^{\beta,\pi,i}_{t_m};\beta)\Delta W^i_{t_{m+1}}\,.\end{aligned}$$

end for
return $(\theta^{*,N_{\text{trn}}}_m, \eta^{*,N_{\text{trn}}}_m)$ for all $m = 0, 1, \ldots, N_{\text{steps}}$.

2.3.3 Martingale Control Variate deep solvers

So far, the presented methodology to obtain the control variate consists of first learning the solution of the PDE and more importantly its gradient (Algorithms 3, 4) which is then plugged in (2.9). Alternatively, one can directly use the variance of (2.9) as the loss function to be optimised in order to learn the control variate. We expand this idea and design two additional algorithms.

Recall definition of $\mathcal{V}^{\beta,\pi,\theta,\lambda,N}_{t,T}$ given by (2.9). From (2.7) we know that the theoretical control variate Monte-Carlo estimator has zero variance and so it is natural to set-up a learning task which aims to learn the network weights θ in a way which minimises said variance:

$$\theta^{\star,\text{var}} := \arg\min_{\theta} \mathbb{V}\text{ar}\left[\mathcal{V}^{\beta,\pi,\theta,\lambda,N}_{t,T}\right].$$

Setting $\lambda = 1$, the learning task is stated as Algorithm 5.

We include a similar Algorithm in Appendix 2.B. This algorithm uses Remark 2.2.1 to find the stochastic integral with maximum correlation with the Monte Carlo estimator such that, with the optimal

Algorithm 5 Martingale control variates solver: Empirical variance minimisation

Initialisation: θ, N_{trn}
for $i : 1 : N_{\text{trn}}$ **do**
generate samples $(x_t^{\beta,\pi,i})_{t\in\pi}$ by using numerical SDE solver on (2.2) and sampling from the distribution of β.
end for
Find $\theta^{*,N_{\text{trn}}}$ where

$$\theta^{*,N_{\text{trn}}} := \arg\min_{\theta} \mathbb{V}\text{ar}^{N_{\text{trn}}}\left[\mathcal{V}_{t,T}^{\beta,\pi,\theta,\lambda,N_{\text{trn}}}\right],$$

where $\mathbb{V}ar^{N_{\text{trn}}}$ denotes the empirical variance, and $\mathcal{V}_{t,T}^{\beta,\pi,\theta,\lambda,N}$ is obtained from (2.9).
return $\theta^{*,N_{\text{trn}}}$.

value of λ, the variance reduction is optimised. In this case, the value of λ is chosen with empirical samples, see [50].

2.4 Examples and experiments

2.4.1 Options in Black–Scholes model on $d > 1$ assets

Take a d-dimensional Wiener process W. We assume that we are given a symmetric, positive-definite matrix (covariance matrix) Σ and a lower triangular matrix C s.t. $\Sigma = CC^*$. For such a positive-definite Σ we can always use Cholesky decomposition to find C. The risky assets will have volatilities given by σ^i. We will (abusing notation) write $\sigma^{ij} := \sigma^i C^{ij}$, when we don't need to separate the volatility of a single asset from correlations. The risky assets under the risk-neutral measure are then given by

$$dS_t^i = rS_t^i\,dt + \sigma^i S_t^i \sum_j C^{ij} dW_t^j\,. \tag{2.12}$$

All sums will be from 1 to d unless indicated otherwise. Note that the SDE can be simulated exactly since

$$S_{t_{n+1}}^i = S_{t_n}^i \exp\left(\left(r - \frac{1}{2}\sum_j (\sigma^{ij})^2\right)(t_{n+1} - t_n) + \sum_j \sigma^{ij}(W_{t_{n+1}}^j - W_{t_n}^j)\right).$$

The associated PDE is (with $a^{ij} := \sum_k \sigma^{ik}\sigma^{jk}$)

$$\partial_t v(t,S) + \frac{1}{2}\sum_{i,j} a^{ij} S^i S^j \partial_{x_i x_j} v(t,S) + r\sum_i S^i \partial_{S^i} v(t,S) - rv(t,S) = 0\,,$$

for $(t,S) \in [0,T) \times \mathbb{R}^d_{>0}$ together with the terminal condition $v(T,S) = g(S)$ for $S \in R^d_{>0}$.

2.4.2 Deep Learning setting

In this subsection we describe the neural networks used in the four proposed algorithms as well as the training setting, in the specific situation where we have an options pricing problem in Black-Scholes model on $d > 1$ assets.

Learning algorithms 4, 5 and 6 share the same underlying fully connected artificial network which will be different for different t_k, $k = 0, 1, \ldots, N_{\text{steps}} - 1$. At each time-step we use a fully connected artificial neural network denoted $\mathcal{R}[\cdot]_{\theta_k} \in \mathcal{DN}$. The choice of the number of layers and network width is motivated by empirical results on different possible architectures applied on a short-lived options

problem. We present the results of this study in Appendix 2.D.1. The architecture is similar to that proposed in [18].

At each time step the network consists of four layers: one d-dimensional input layer, two $(d+20)$-dimensional hidden layers, and one output layer. The output layer is one dimensional if the network is approximation for v and d-dimensional if the network is an approximation for $\partial_x v$. The non-linear activation function used on the hidden layers is the the linear rectifier `relu`. In all experiments except for Algorithm 4 for the basket options problem we used batch normalisation [54] on the input of each network, just before the two nonlinear activation functions in front of the hidden layers, and also after the last linear transformation.

The networks' optimal parameters are approximated by the Adam optimiser [55] on the loss function specific for each method. Each parameter update (i.e. one step of the optimiser) is calculated on a batch of $5 \cdot 10^3$ paths $(x^i_{t_n})_{n=0}^{N_{\text{steps}}}$ obtained by simulating the SDE. We take the necessary number of training steps until the stopping criteria defined below is met, with a learning rate of 10^{-3} during the first 10^4 iterations, decreased to 10^{-4} afterwards.

During training of any of the algorithms, the loss value at each iteration is kept. A model is assumed to be trained if the difference between the loss averages of the two last consecutive windows of length 100 is less than a certain ϵ.

2.4.3 Evaluating variance reduction

We use the specified network architectures to assess the variance reduction in several examples below. After training the models in each particular example, they are evaluated as follows:

i) We calculate $N_{\text{MC}} = 10$ times the Monte Carlo estimate $\overline{\Xi_T} := \frac{1}{N_{\text{in}}} \sum_{i=1}^{N_{\text{in}}} \Xi_T^i$ and the Monte Carlo with control variate estimate $\bar{\mathcal{V}}_{t,T}^{\pi,\theta,\lambda,N_{\text{steps}}} = \frac{1}{N_{\text{in}}} \sum_{i=1}^{N_{\text{in}}} \mathcal{V}_{t,T}^{\pi,\theta,\lambda,N_{\text{steps}},i}$ using $N_{\text{in}} = 10^6$ Monte Carlo samples.

ii) From Central Limit Theorem, as N_{in} increases the standardised estimators converge in distribution to the Normal. This allows to find the following a 95% confidence interval of the variance of the estimator (which by Central Limit Theorem is assumed to be normally distributed for high N_{MC}), given by

$$\left[\frac{(N_{\text{MC}}-1)S^2}{\chi_{1-\alpha/2,N_{\text{MC}}-1}}, \frac{(N_{\text{MC}}-1)S^2}{\chi_{\alpha/2,N_{\text{MC}}-1}}\right]$$

where S is the sample variance of the N_{MC} controlled estimators $\bar{\mathcal{V}}_{t,T}^{\pi,\theta,\lambda,N_{\text{steps}}}$, and $\alpha = 0.05$. These are calculated for both the Monte Carlo estimate and the Monte Carlo with control variate estimate. See for example [56] for more details.

iii) We use the $N_{\text{MC}} \cdot N_{\text{in}} = 10^7$ generated samples Ξ_T^i and $\mathcal{V}_{t,T}^{\pi,\theta,\lambda,N_{\text{steps}},i}$ to calculate and compare the empirical variances $\tilde{\sigma}^2_{\Xi_T}$ and $\tilde{\sigma}^2_{\mathcal{V}_{t,T}^{\pi,\theta,\lambda,N_{\text{steps}},i}}$.

iv) The number of optimizer steps and equivalently number of random paths generated for training provide a cost measure of the proposed algorithms.

v) We evaluate the variance reduction if we use the trained models to create control variates for options in Black-Scholes models with different volatilities than the one used to train our models.

Example 2.4.1 (Low dimensional problem with explicit solution)**.** *We consider exchange option on two assets. In this case the exact price is given by the Margrabe formula. We take* $d = 2$*,* $S_0^i = 100$*,* $r = 5\%$*,*

$\sigma^i = 30\%$, $\Sigma^{ii} = 1$, $\Sigma^{ij} = 0$ *for* $i \neq j$. *The payoff is*

$$g(S) = g(S^{(1)}, S^{(2)}) := \max\left(0, S^{(1)} - S^{(2)}\right) .$$

From Margrabe's formula (see [57]) we know that

$$v(0,S) = \textit{BlackScholes}\left(\textit{risky price} = \frac{S^{(1)}}{S^{(2)}}, \textit{strike} = 1, T, r, \bar{\sigma}\right) ,$$

where $\bar{\sigma} := \sqrt{\left(\sigma^{11} - \sigma^{21}\right)^2 + \left(\sigma^{22} - \sigma^{12}\right)^2}$.

We organise the experiment as follows: We train our models with batches of 5,000 random paths $(s^i_{t_n})_{n=0}^{N_{steps}}$ *sampled from the SDE 2.12, where* $N_{steps} = 50$. *The assets' initial values* $s^i_{t_0}$ *are sampled from a lognormal distribution*

$$X \sim \exp((\mu - 0.5\sigma^2)\tau + \sigma\sqrt{\tau}\xi),$$

where $\xi \sim \mathcal{N}(0,1), \mu = 0.08, \tau = 0.1$. *The existence of an explicit solution allows to build a control variate of the form* (2.9) *using the known exact solution to obtain* $\partial_x v$. *Recall that from* (2.9)*, if one has* $\partial_x v$ *then the only bias comes from the numerical scheme to calculate the stochastic integral, and no additional bias is added from the function approximation of* $\partial_x v$. *Therefore, for a fixed number of time steps* N_{steps} *this provides an upper bound on the variance reduction an artificial neural network approximation of* $\partial_x v$ *can achieve.*

We follow the evaluation framework to evaluate the model, simulating $N_{MC} \cdot N_{in}$ *paths by simulating* (2.12) *with constant* $(S^1_0, S^2_0)^i = (1,1)$. *We report the following results:*

i) Table 2.1 provides the empirical variances calculated over 10^6 *generated Monte Carlo samples and their corresponding control variates. The variance reduction measure indicates the quality of each control variate method. The variance reduction using the control variate given by Margrabe's formula provides a benchmark for our methods. Table 2.1 also provides the cost of training for each method, given by the number of optimiser iterations performed before hitting the stopping criteria, defined before with* $\epsilon = 5 \times 10^{-6}$. *We add an additional row with the control variate built using automatic differentiation on the network parametrised using the Deep Galerkin Method [15]. The DGM attempts to find the optimal parameters of the network satisfying the PDE on a pre-determined time and space domain. In contrast to our algorithms, the DGM method is not restricted to learn the solution of the PDE on the paths built from the probabilistic representation of the PDE. However, this is what is precisely enhancing the performance of our methods in terms of variance reduction, since they are specifically learning an approximation to the solution of the PDE and its gradient such that the resulting control variate will yield a low-variance Monte Carlo estimator.*

Our experiments show that Algorithm 4 requires smaller number of sample paths compared to the other algorithms in order to achieve the desired accuracy. We hypothesise that this is due to the fact that objective function in Algorithm 4 minimises the error between consecutive timesteps of the Backward probabilistic representation of the PDE. The other algorithms first calculate the full stochastic integral between $[0,T]$ *and then perform the minimisation.*

ii) Table 2.2 provides the confidence intervals for the variances and of the Monte Carlo estimator, and the Monte Carlo estimator with control variate assuming these are calculated on 10^6 *random paths. Moreover, we add the confidence interval of the variance of the Monte Carlo estimator calculated over* N_{in} *antithetic paths where the first* $N_{in}/2$ *Brownian paths generated using* $(Z_i)_{i=1,\ldots,N_{steps}}$ *samples from a normal and the second half of the Brownian paths are generated using the antithetic*

Method	Emp. Var.	Var. Red. Fact.	Train. Paths	Opt. Steps
Monte Carlo	3.16×10^{-2}	-	-	-
Algorithm 3	2.47×10^{-4}	127.7	38×10^{6}	7 600
Algorithm 4	2.59×10^{-4}	121.98	6.945×10^{6}	1380
Algorithm 5	2.39×10^{-4}	132.28	36.055×10^{6}	7211
Algorithm 6	2.40×10^{-4}	131.53	45.61×10^{6}	9122
MC + CV Margrabe	2.12×10^{-4}	149.19	-	-
MC + CV DGM [15]	1.22×10^{-3}	25.8	-	-

Table 2.1: Results on exchange option problem on two assets, Example 2.4.1. Empirical Variance and variance reduction factor

samples $(-Z_i)_{i=1,\dots,N_{steps}}$. See [58, Section 4.2] for more details. All the proposed algorithms in this paper outperform the Monte Carlo estimator and the Monte Carlo estimator with antithetic paths; compared to the latter, our algorithms produce unbiased estimators with variances that two orders of magnitude less.

iii) Figure 2.2 studies the iterative training for the BSDE solver. As it has been observed before, this type of training does not allow us to study the overall loss function as the number of training steps increases. Therefore we train the same model four times for different values of ϵ between 0.01 and 5×10^{-6} and we study the number of iterations necessary to meet the stopping criteria defined by ϵ, the variance reduction once the stopping criteria is met, and the relationship between the number of iterations and the variance reduction. Note that the variance reduction stabilises for $\epsilon < 10^{-5}$. Moreover, the number of iterations necessary to meet the stopping criteria increases exponentially as ϵ decreases, and therefore for our results printed in Tables 2.1 and 2.2 we employ $\epsilon = 5 \times 10^{-6}$.

iv) Figure 2.4 displays the variance reduction after using the trained models on several Black Scholes problem with exchange options but with values of σ other than 0.3 which was the one used for training. We see that the various algorithms work similarly well in this case (not taking training cost into account). We note that the variance reduction is close to the theoretical maximum which is restricted by time discretisation. Finally we see that the variance reduction is still significant even when the neural network was trained with different model parameter (in our case volatility in the option pricing example). The labels of Figure 2.4 can be read as follows:

i) MC + CV Corr op: Monte-Carlo estimate with Deep Learning-based Control Variate built using Algorithm 6.

ii) MC + CV Var op: Monte-Carlo estimate with Deep Learning-based Control Variate built using Algorithm 5.

iii) MC + CV BSDE solver: Monte-Carlo estimate with Deep Learning-based Control Variate built using Algorithm 4.

iv) MC + CV Margrabe: Monte-Carlo estimate with Control Variate using analytical solution for this problem given by Margrabe formula.

Method	Confidence Interval Variance	Confidence Interval Estimator
Monte Carlo	$[2.36 \times 10^{-6}, 4.15 \times 10^{-6}]$	$[0.1187, 0.1195]$
Monte Carlo + antithetic paths	$[1.15 \times 10^{-6}, 2.02 \times 10^{-6}]$	$[0.1191, 0.1195]$
Algorithm 3	$[4.13 \times 10^{-9}, 1.09 \times 10^{-8}]$	$[0.11919, 0.11926]$
Algorithm 4	$[4.12 \times 10^{-9}, 1.09 \times 10^{-8}]$	$[0.11919, 0.11925]$
Algorithm 5	$[4.32 \times 10^{-9}, 1.14 \times 10^{-8}]$	$[0.11919, 0.11926]$
Algorithm 6	$[2.30 \times 10^{-9}, 6.12 \times 10^{-8}]$	$[0.11920, 0.11924]$
MC + CV Margrabe	$[3.10 \times 10^{-9}, 8.23 \times 10^{-9}]$	$[0.11919, 0.11925]$
MC + CV DGM [15]	$[3.10 \times 10^{-9}, 8.23 \times 10^{-9}]$	$[0.11919, 0.11925]$

Table 2.2: Results on exchange option problem on two assets, Example 2.4.1.

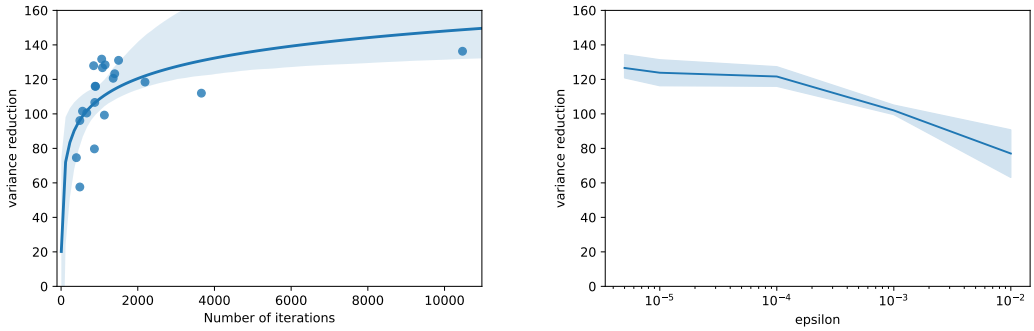


Figure 2.2: Left: Variance reduction in terms of number of optimiser iterations. Right: Variance reduction in terms of epsilon. Both are for Example 2.4.1 and Algorithm 4.

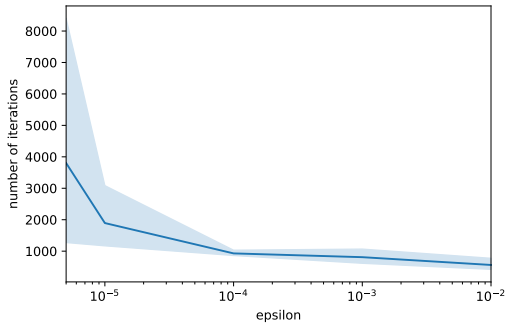


Figure 2.3: Number of optimiser iterations in terms of epsilon for Example 2.4.1 and Algorithm 4.

Example 2.4.2 (Low-dimensional problem with explicit solution - Approximation of Price using PDE solver compared to Control Variate)**.** *We consider exchange options on two assets as in Example 2.4.1. We consider algorithm 4 that can be applied in two different ways:*

i) It directly approximates the solution of the PDE (2.5) *and its gradient in every point.*

ii) We can use $\partial_x v$ *to build the control variate using probabilistic representation of the PDE* (2.6)

We compare both applications by calculating the expected error of the L^2*-error of each of them with respect to the analytical solution given by Margrabe formula. From Margrabe's formula we know that*

$$v(0,S) = \textit{BlackScholes}\left(\textit{risky price} = \frac{S^{(1)}}{S^{(2)}}, \textit{strike} = 1, T, r, \bar{\sigma}\right),$$

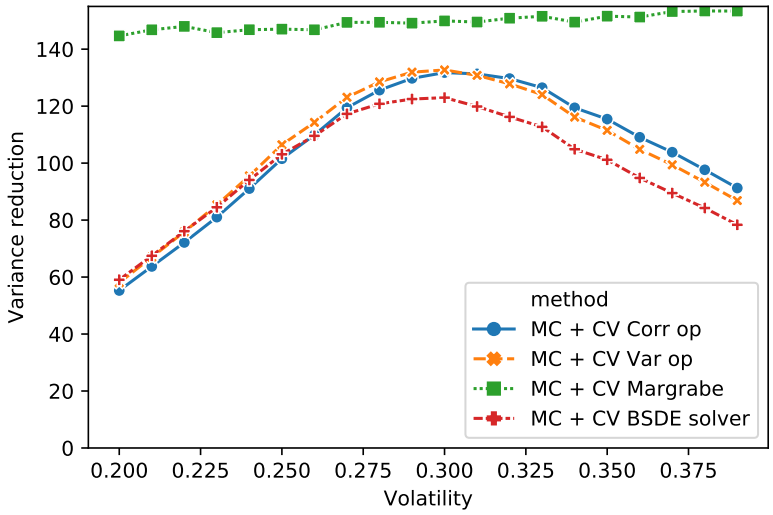


Figure 2.4: Variance reduction achieved by network trained with $\sigma = 0.3$ but then applied in situations where $\sigma \in [0.2, 0.4]$. We can see that the significant variance reduction is achieved by a neural network that was trained with "incorrect" σ. Note that the "MC + CV Margbrabe" displays the optimal variance reduction that can be achieved by using exact solution to the problem. The variance reduction is not infinite even in this case since stochastic integrals are approximated by Riemann sums.

Let $\mathcal{R}[v]_{\eta_0}(x) \approx v(0,x)$ be the Deep Learning approximation of price at any point at initial time, calculated using Algorithm 4, and $\mathcal{R}[\partial_x v]_{\theta_m}(x) \approx \partial_x v(t_m, x)$ be the Deep Learning approximation of its gradient for every time step in the time discretisation. The aim of this experiment is to show how even if Algorithm 4 numerically converges to a biased approximation of $v(0,x)$ (see Figure 2.5 left), it is still possible to use $\mathcal{R}[\partial_x v]_{\theta_m}(x)$ to build an unbiased Monte-Carlo approximation of $v(0,x)$ with low variance.

We organise the experiment as follows.

i) We calculate the expected value of the L^2-error of $\mathcal{R}\eta_0(x)$ where each component of $x \in \mathbb{R}^2$ is sampled from a lognormal distribution:

$$\mathbb{E}[|v(0,x) - \mathcal{R}[v]_{\eta_0}(x)|^2] \approx \frac{1}{N}\sum_{i=1}^{N} |v(0,x^i) - \mathcal{R}[v]_{\eta_0}(x^i)|^2$$

ii) We calculate the expected value of the L^2-error of the Monte-Carlo estimator with control variate where each component of $x \in \mathbb{R}^2$ is sampled from a lognormal distribution:

$$\mathbb{E}[|v(0,x) - \mathcal{V}_{0,T}^{\pi,\theta,\lambda,N_{MC},x}|^2] \approx \frac{1}{N}\sum_{i=1}^{N} |v(0,x^i) - \mathcal{V}_{0,T}^{\pi,\theta,\lambda,N_{MC},x^i}|^2\,,$$

where $\mathcal{V}_{0,T}^{\pi,\theta,\lambda,N_{MC},x}$ is given by 2.9, and is calculated for different values of Monte Carlo samples.

iii) We calculate the expected value of the L^2-error of the Monte-Carlo estimator without control variate where each component of $x \in \mathbb{R}^2$ is sampled from a lognormal distribution:

$$\mathbb{E}[|v(0,x) - \Xi_{0,T}^{\pi,\theta,\lambda,N_{MC},x}|^2] \approx \frac{1}{N}\sum_{i=1}^{N} |v(0,x^i) - \Xi_{0,T}^{\pi,\theta,\lambda,N_{MC},x^i}|^2\,,$$

where

$$\Xi_{0,T}^{\pi,\theta,\lambda,N_{MC},x} := \frac{1}{N_{MC}} \sum_{j=1}^{N_{MC}} D(t,T) g(X_T^i)$$

Figure 2.5 provides one realisation of the described experiment for different Monte-Carlo samples between 10 and 200. It shows how in this realisation, 60 Monte-Carlo iterations are enough to build a Monte-Carlo estimator with control variate having lower bias than the solution provided by Algorithm 4.

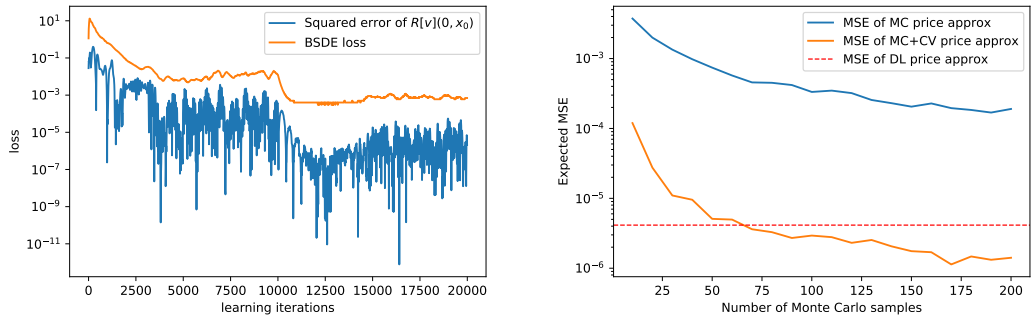


Figure 2.5: Left: Loss of Algorithm 4 and squared error of $\mathcal{R}[v](t, x_0)$ in terms of training iterations. Right: Expected MSE of the two different approaches with respect to analytical solution in terms of number of Monte Carlo samples

Example 2.4.3 (Low-dimensional problem with explicit solution. Training on random values for volatility)**.** *We consider exchange option on two assets. In this case the exact price is given by the Margrabe formula. The difference with respect to the last example is that now we aim to generalise our model, so that it can build control variates for different Black-Scholes models. For this we take* $d = 2$*,* $S_0^i = 100$*,* $r = 0.05$*,* $\sigma^i \sim$ *Unif*$(0.2, 0.4)$*,* $\Sigma^{ii} = 1$*,* $\Sigma^{ij} = 0$ *for* $i \neq j$*.*

The payoff is

$$g(S) = g(S^{(1)}, S^{(2)}) := \max\left(0, S^{(1)} - S^{(2)}\right).$$

We organise the experiment as follows: for comparison purposes with the BSDE solver from the previous example, we train our model for exactly the same number of iterations, i.e. $1,380$ *batches of 5,000 random paths* $(s_{t_n}^i)_{n=0}^{N_{steps}}$ *sampled from the SDE 2.12, where* $N_{steps} = 50$*. The assets' initial values* $s_{t_0}^i$ *are sampled from a lognormal distribution*

$$X \sim \exp((\mu - 0.5\sigma^2)\tau + \sigma\sqrt{\tau}\xi),$$

where $\xi \sim \mathcal{N}(0,1)$*,* $\mu = 0.08$*,* $\tau = 0.1$*. Since now* σ *can take different values, it is included as input to the networks at each time step.*

The existence of an explicit solution allows to build a control variate of the form (2.9) *using the known exact solution to obtain* $\partial_x v$*. For a fixed number of time steps* N_{steps} *this provides an upper bound on the variance reduction an artificial neural network approximation of* $\partial_x v$ *can achieve.*

Figure 2.6 adds the performance of this model to Figure 2.4, where the variance reduction of the Control Variate is displayed for different values of the volatility between 0.2 *and* 0.4*.*

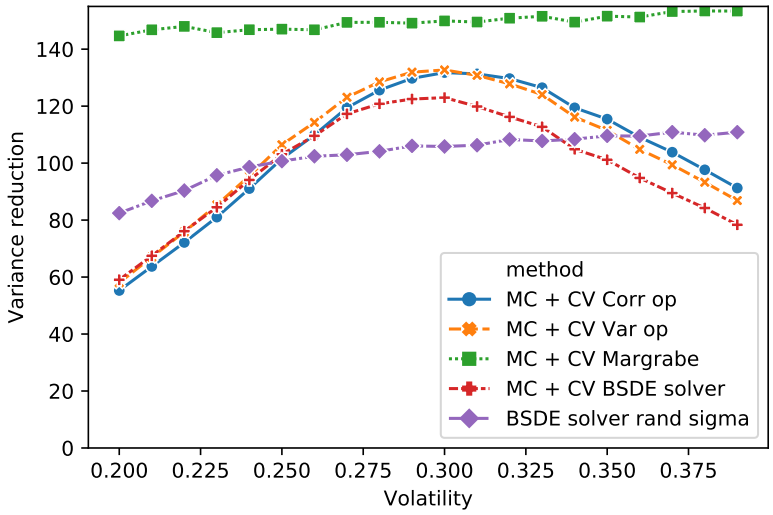


Figure 2.6: Extension of Figure 2.4 with variance reduction achieved by training the model on different Black-Scholes models

Example 2.4.4 (High-dimensional problem, exchange against average)**.** *We extend the previous example to 100 dimensions. This example is similar to* EX_{10E} *from [59]. We will take* $S_0^i = 100$, $r = 5\%$, $\sigma^i = 30\%$, $\Sigma^{ii} = 1$, $\Sigma^{ij} = 0$ *for* $i \neq j$.

We will take this to be

$$g(S) := \max\left(0, S^1 - \frac{1}{d-1}\sum_{i=2}^{d} S^i\right).$$

The experiment is organised as follows: we train our models with batches of $5 \cdot 10^3$ *random paths* $(s^i_{t_n})_{n=0}^{N_{steps}}$ *sampled from the SDE* (2.12)*, where* $N_{steps} = 50$*. The assets' initial values* $s^i_{t_0}$ *are sampled from a lognormal distribution*

$$X \sim \exp((\mu - 0.5\sigma^2)\tau + \sigma\sqrt{\tau}\xi),$$

where $\xi \sim N(0,1)$, $\mu = 0.08$, $\tau = 0.1$.

We follow the evaluation framework to evaluate the model, simulating $N_{MC} \cdot N_{in}$ *paths by simulating* (2.12) *with constant* $S_0^i = 1$ *for* $i = 1, \ldots, 100$*. We have the following results:*

i) Table 2.3 provides the empirical variances calculated over 10^6 *generated Monte Carlo samples and their corresponding control variates. The variance reduction measure indicates the quality of each control variate method. Table 2.3 also provides the cost of training for each method, given by the number of optimiser iterations performed before hitting the stopping criteria with* $\epsilon = 5 \cdot 10^{-6}$*. Algorithm 4 outperforms the other algorithms in terms of variance reduction factor. This is not surprising as Algorithm 4 explicitly learns the discretisation of the Martingale representation (equation* (2.7)*) from which the control variate arises.*

ii) Table 2.4 provides the confidence interval for the variance of the Monte Carlo estimator, and the Monte Carlo estimator with control variate assuming these are calculated on 10^6 *random paths.*

iii) Figures 2.7 and 2.8 study the iterative training for the BSDE solver. We train the same model four times for different values of ϵ *between* 0.01 *and* 5×10^{-6} *and we study the number of iterations*

necessary to meet the stopping criteria defined by ϵ, the variance reduction once the stopping criteria is met, and the relationship between the number of iterations and the variance reduction. Note that in this case the variance reduction does not stabilise for $\epsilon < 10^{-5}$. However, the number of training iterations increases exponentially as ϵ decreases, and therefore we also choose $\epsilon = 5 \times 10^{-6}$ to avoid building a control that requires a high number of random paths to be trained.

Method	Emp. Var.	Var. Red. Fact.	Train. Paths	Opt. Steps
Monte Carlo	1.97×10^{-2}	-	-	-
Algorithm 3	5.94×10^{-3}	33.16	74×10^{6}	14 900
Algorithm 4	1.51×10^{-4}	130.39	14.145×10^{6}	2 829
Algorithm 5	5.29×10^{-4}	37.22	97.265×10^{6}	19 453
Algorithm 6	1.93×10^{-4}	102.05	76.03×10^{6}	15 206

Table 2.3: Results on exchange option problem on 100 assets, Example 2.4.4. Empirical Variance and variance reduction factor and costs in terms of paths used for training and optimizer steps.

Method	Confidence Interval Variance	Confidence Interval Estimator
Monte Carlo	$[1.51 \times 10^{-6}, 2.65 \times 10^{-6}]$	$[0.0845, 0.0849]$
Monte Carlo + antithetic paths	$[8.77 \times 10^{-7}, 1.53 \times 10^{-6}]$	$[0.0845, 0.0848]$
Algorithm 3	$[3.04 \times 10^{-8}, 2.14 \times 10^{-7}]$	$[0.0848, 0.08493]$
Algorithm 4	$[5.32 \times 10^{-9}, 1.41 \times 10^{-8}]$	$[0.08485, 0.8492]$
Algorithm 5	$[4.13 \times 10^{-9}, 1.09 \times 10^{-8}]$	$[0.08484, 0.08490]$
Algorithm 6	$[3.80 \times 10^{-9}, 1.0 \times 10^{-8}]$	$[0.08487, 0.08493]$

Table 2.4: Results on exchange option problem on 100 assets, Example 2.4.4.

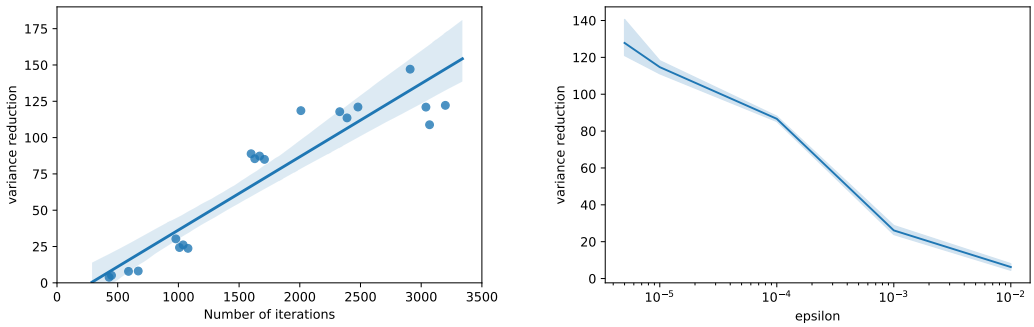


Figure 2.7: Left: Variance reduction in terms of number of optimiser iterations. Right: Variance reduction in terms of epsilon. Both for Example 2.4.4 and Algorithm 4.

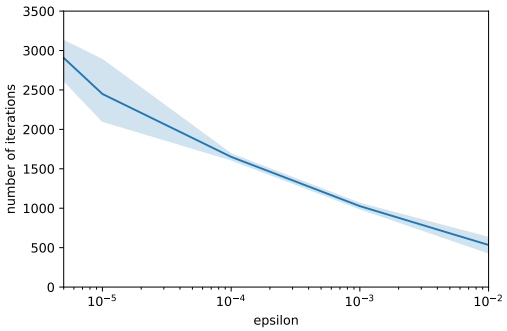


Figure 2.8: Number of optimiser iterations in terms of ϵ for Example 2.4.4 and Algorithm 4.

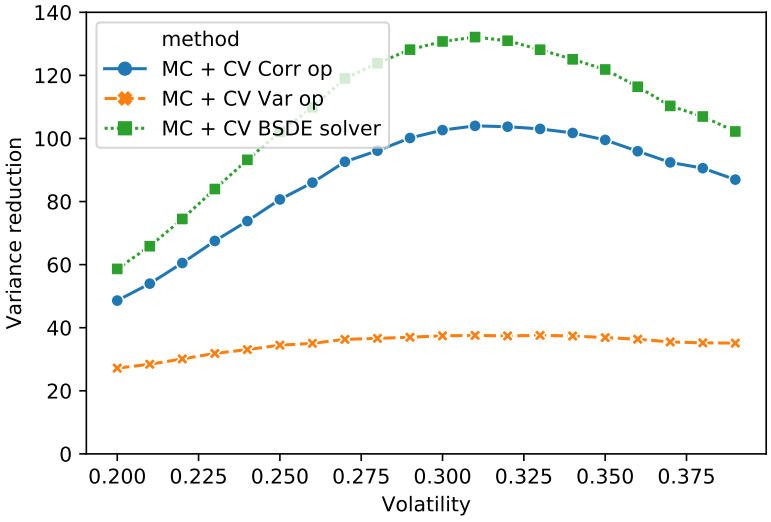


Figure 2.9: Variance reduction with network trained with $\sigma = 0.3$ but applied for $\sigma \in [0.2, 0.4]$ for the model of Example 2.4.4. We see that the variance reduction factor is considerable even in the case when the network is used with "wrong" σ. It seems that Algorithm 5 is not performing well in this case. This can be caused by many factors related to training, such as local minima being met or lack of training samples.

2.A Martingale Control Variate

Let $(\Omega, \mathcal{F}, \mathbb{P})$ be a probability space and consider an $\mathbb{R}^{d'}$-valued Wiener process $W = (W^j)_{j=1}^{d'} = ((W_t^j)_{t\geq 0})_{j=1}^{d'}$. We will use $(\mathcal{F}_t^W)_{t\geq 0}$ to denote the filtration generated by W. Consider a $D \subseteq \mathbb{R}^d$-valued, continuous, stochastic process defined for the parameters $\beta \in B \subseteq \mathbb{R}^p$, $X^\beta = (X^{\beta,i})_{i=1}^d = ((X_t^{\beta,i})_{t\geq 0})_{i=1}^d$ adapted to $(\mathcal{F}_t^W)_{t\geq 0}$.

Let $g : C([0,T], \mathbb{R}^d) \to \mathbb{R}$ be a measurable function. We shall consider path-dependent contingent claims of the form $g((X_s^\beta)_{s\in[0,T]})$. Finally we assume that there is a (stochastic) discount factor given by

$$D(t_1, t_2; \beta) = e^{-\int_{t_1}^{t_2} c(s, X_s^\beta; \beta) ds}$$

for an appropriate function $c = c(t, x; \beta)$. We will omit the β from the discount factor notation. Let

$$\Xi_T^\beta := D(t,T) g((X_s^\beta)_{s\in[0,T]}) \,.$$

We now interpret $\mathbb{P}$ as some risk-neutral measure and so the $\mathbb{P}$-price of our contingent claim is

$$V_t^\beta = \mathbb{E}\left[\Xi_T^\beta | \mathcal{F}_t^\beta\right] = \mathbb{E}\left[D(t,T) g((X_s^\beta)_{s\in[0,T]}) \big| \mathcal{F}_t^\beta\right] \,.$$

By assumption Ξ_T^β is $\mathcal{F}_T^W$ measurable and $\mathbb{E}[|\Xi_T^\beta|^2] < \infty$. Hence, from the Martingale Representation Theorem, see e.g. [51, Th. 14.5.1], there exists a unique process $(Z_t^\beta)_t$ adapted to $(\mathcal{F}_t^W)_t$ with $\mathbb{E}[\int_0^T |Z_s^\beta|^2 ds] < \infty$ such that

$$\Xi_T^\beta = \mathbb{E}[\Xi_T^\beta | \mathcal{F}_0^W] + \int_0^T Z_s^\beta \, dW_s \,. \tag{2.13}$$

The proof of the existence of the process $(Z_t^\beta)_t$, is non-constructive. In the setup of the paper, we used the Markovian property of Ξ_t^β to approximate Z_t^β via the associated linear PDE. In the more general non-Markovian setup, [60] provides a numerical method to construct the martingale representation.

Observe that in our setup, $\mathcal{F}_0 = \mathcal{F}_0^W$, $\mathcal{F}_t^\beta \subseteq \mathcal{F}_t^W$ for $t \geq 0$. Hence tower property of the conditional expectation implies that

$$\mathbb{E}[\Xi_T^\beta \big| \mathcal{F}_t^\beta] = \mathbb{E}[\Xi_T^\beta \big| \mathcal{F}_0^W] + \int_0^t Z_s^\beta \, dW_s \,. \tag{2.14}$$

Consequently (2.13) and (2.14) imply

$$\mathbb{E}[\Xi_T^\beta \big| \mathcal{F}_t^\beta] = \Xi_T^\beta - \int_t^T Z_s^\beta \, dW_s \,.$$

We then observe that

$$V_t^\beta = \mathbb{E}[\Xi_T^\beta \big| \mathcal{F}_t^\beta] = \mathbb{E}\left[\Xi_T^\beta - \int_t^T Z_s^\beta \, dW_s \Big| \mathcal{F}_t^\beta\right] \,.$$

If we can generate iid $(W^i)_{i=1}^N$ and $(Z^{\beta,i})_{i=1}^N$ with the same distributions as W and Z respectively then we can consider the following Monte-Carlo estimator of V_t^β:

$$\mathcal{V}_t^{\beta,N} := \frac{1}{N} \sum_{i=1}^N \left(\Xi_T^{\beta,i} - \int_t^T Z_s^{\beta,i} \, dW_s^i \right) \,.$$

In chapter 3 and [12] we provide deep learning algorithms to price path-dependent options in the risk neutral measure by solving the corresponding path-dependent PDE, using a combination of Recurrent Neural networks and path signatures to parametrise the process Z^β.

2.B Martingale Control Variate Deep Solvers

2.B.1 Empirical correlation maximisation

This method is based on the idea that since we are looking for a good control variate we should directly train the network to maximise the variance reduction between the vanilla Monte-Carlo estimator and the control variates Monte-Carlo estimator by also trying to optimise λ.

Recall we denote $\Xi_T = D(t,T) g((X_s)_{s\in[t,T]})$. We also denote as $M_{t,T}$ as the stochastic integral that

arises in the martingale representation of Ξ_T. The optimal coefficient $\lambda^{*,\theta}$ that minimises the variance $\mathbb{V}\text{ar}[\Xi_T - \lambda M^\theta_{t,T}]$ is

$$\lambda^{*,\theta} = \frac{\mathbb{C}\text{ov}[\Xi_T, M^\theta_{t,T}]}{\mathbb{V}\text{ar}[M^\theta_{t,T}]}.$$

Let $\rho^{\Xi_T, M^\theta_{t,T}}$ denote the Pearson correlation coefficient between Ξ_T and $M^\theta_{t,T}$ i.e.

$$\rho^{\Xi_T, M^\theta_{t,T}} = \frac{\mathbb{C}\text{ov}(\Xi_T, M^\theta_{t,T})}{\sqrt{\mathbb{V}\text{ar}[\Xi_T]\mathbb{V}\text{ar}[M^\theta_{t,T}]}}.$$

With the optimal λ^* we then have that the variance reduction obtained from the control variate is

$$\frac{\mathbb{V}\text{ar}[\mathcal{V}^{\pi,\theta,\lambda^*,N}_{t,T}]}{\mathbb{V}\text{ar}[\Xi_T]} = 1 - \left(\rho^{\Xi_T, M^\theta_{t,T}}\right)^2.$$

See [50, Ch. 4.1] for more details. Therefore we set the learning task as:

$$\theta^{*,cor} := \arg\min_\theta \left[1 - \left(\rho^{\Xi_T, M^\theta_{t,T}}\right)^2\right].$$

The implementable version requires the definition of $\mathcal{V}^{\beta,\pi,\theta,\lambda,N}_{t,T}$ in (2.9), where we set

$$\begin{aligned}\Xi^{\beta,\pi,i}_T &:= D^\pi(t,T))^i g(X^{\beta,\pi,i}_T)\\ M^{\beta,\pi,i,\theta}_{t,T} &:= \sum_{k=1}^{N_{\text{steps}}-1} (D^\pi(t,t_k))^i \mathcal{R}[\partial_x v]_\theta(t_k, X^{\beta,\pi,i}_{t_k})\sigma(t_k, X^{\beta,\pi,i}_{t_k})\left(W^i_{t_{k+1}} - W^i_{t_k}\right)\end{aligned}$$

The full method is stated as Algorithm 6.

Algorithm 6 Martingale control variates solver: Empirical correlation maximization

Initialisation: θ, N_{trn}
for $i : 1 : N_{\text{trn}}$ **do**
Generate training set:

- Sample different values of β from a fixed distribution (for example Uniform distribution).
- Generate $(x^{\beta,\pi,i}_t)_{t\in\pi}$ by using numerical SDE solver on (2.2)

end for
Find $\theta^{*,N_{\text{trn}}}$ using SGD where

$$\theta^{*,N_{\text{trn}}} := \arg\min_\theta \left[1 - \left(\frac{\mathbb{C}\text{ov}^{N_{\text{trn}}}(\Xi^{\beta,\pi}_T)}{\sqrt{\mathbb{V}\text{ar}^{N_{\text{trn}}}[\Xi^{\beta,\pi}_T]\mathbb{V}\text{ar}^{N_{\text{trn}}}[M^{\beta,\pi,\theta}_{t,T}]}}\right)^2\right],$$

where $\mathbb{V}\text{ar}^{N_{\text{trn}}}, \mathbb{C}\text{ov}^{N_{\text{trn}}}$ denote the empirical variance and covariance.
return $\theta^{*,N_{\text{trn}}}$.

2.C Artificial neural networks

We fix a locally Lipschitz function $\mathbf{a} : \mathbb{R} \to \mathbb{R}$ and for $d \in \mathbb{N}$ define $\mathbf{A}_d : \mathbb{R}^d \to \mathbb{R}^d$ as the function given, for $x = (x_1, \ldots, x_d)$ by $\mathbf{A}_d(x) = (\mathbf{a}(x_1), \ldots, \mathbf{a}(x_d))$. We fix $L \in \mathbb{N}$ (the number of layers),

$l_k \in \mathbb{N}$, $k = 0, 1, \dots L-1$ (the size of input to layer k) and $l_L \in \mathbb{N}$ (the size of the network output). A fully connected artificial neural network is then given by $\Phi = ((W_1, B_1), \dots, (W_L, B_L))$, where, for $k = 1, \dots, L$, we have real $l_{k-1} \times l_k$ matrices W_k and real l_k dimensional vectors B_k.

The artificial neural network defines a function $\mathcal{R}_\Phi : \mathbb{R}^{l_0} \to \mathbb{R}^{l_L}$ given recursively, for $x_0 \in \mathbb{R}^{l_0}$, by

$$\mathcal{R}_\Phi(x_0) = W_L x_{L-1} + B_L\,, \quad x_k = \mathbf{A}_{l_k}(W_k x_{k-1} + B_k)\,, k = 1, \dots, L-1\,.$$

We can also define the function $\mathcal{P}$ which counts the number of parameters as

$$\mathcal{P}(\Phi) = \sum_{i=1}^{L} (l_{k-1} l_k + l_k)\,.$$

We will call such class of fully connected artificial neural networks $\mathcal{DN}$. Note that since the activation functions and architecture are fixed the learning task entails finding the optimal $\Phi \in \mathbb{R}^{\mathcal{P}(\Phi)}$.

2.D Additional numerical results

Example 2.D.1 (Low dimensional basket option)**.** *We consider the basket options problem of pricing, using the example from [58, Sec 4.2.3]. The payoff function is*

$$g(S) := \max\left(0, \sum_{i=1}^{d} S^i - K\right)\,.$$

We first consider the basket options problem on two assets, with $d = 2$, $S_0^i = 70$, $r = 50\%$, $\sigma^i = 100\%$, $\Sigma^{ii} = 1$, $\Sigma^{ij} = 0$ for $i \neq j$, and constant strike $K = \sum_{i=1}^{d} S_0^i$. In line with the example from [58, Sec 4.2.3] for comparison purposes we organise the experiment as follows. The control variates on $20\,000$ batches of $5\,000$ samples each of $(s_{t_n}^i)_{n=0}^{N_{steps}}$ by simulating the SDE 2.12, where $N_{steps} = 50$. The assets' initial values s_{t_0} are always constant $S_{t_0}^i = 0.7$. We follow the evaluation framework to evaluate the model, simulating $N_{MC} \cdot N_{in}$ paths by simulating 2.12 with constant $S_0^i = 0.7$ for $i = 1, \dots, 100$. We have the following results:

i) Table 2.5 provides the empirical variances calculated over 10^6 generated Monte Carlo samples and their corresponding control variates. The variance reduction measure indicates the quality of each control variate method. Table 2.5 also provides the cost of training for each method, given by the number of optimiser iterations performed before hitting the stopping criteria, defined before with $\epsilon = 5 \times 10^{-6}$.

ii) Table 2.6 provides the confidence intervals for the variance of the Monte Carlo estimator, and the Monte Carlo estimator with control variate assuming these are calculated on 10^6 random paths.

iii) Figures 2.10 and 2.11 study the iterative training for the BSDE solver. We train the same model four times for different values of ϵ between 0.01 and 5×10^{-6} and we study the number of iterations necessary to meet the stopping criteria defined by ϵ, the variance reduction once the stopping criteria is met, and the relationship between the number of iterations and the variance reduction. Note that the variance reduction stabilises for $\epsilon < 10^{-5}$. Furthermore, the number of training iterations increases exponentially as ϵ decreases. We choose $\epsilon = 5 \times 10^{-6}$.

We note that in the example from [58, Sec 4.2.3], the control variate is trained with $S_0 = 0.7$ fixed. Using this setting, Algorithm 3 cannot be used to approximate the control variate in (2.9)*: since the*

network at $t = 0$, $\mathcal{R}[v]_{\eta_0}$, is trained only at $S_0 = 0.7$, then automatic differentiation to approximate $\partial_x \mathcal{R}[v]_{\eta_0}(0.7)$ will yield a bad approximation of $\partial_x v(0.7)$; indeed, during training $\mathcal{R}[v]_{\eta_0}$ is unable to capture how v changes around S_0 at $t = 0$. For this reason, Algorithm 3 is not included in the following results.

Method	Emp. Var.	Var. Red. Fact.	Train. Paths	Optimizer steps
Monte Carlo	1.39	-	-	-
Algorithm 4	1.13×10^{-3}	1219	8×10^7	16 129
Algorithm 5	1.13×10^{-3}	1228	3×10^7	6 601
Algorithm 6	1.29×10^{-3}	1076	4×10^7	8 035

Table 2.5: Results on basket options problem on two assets, Example 2.D.1. Models trained with S_0 fixed, non-random. Empirical Variance and variance reduction factor are presented.

Method	Confidence Interval Variance	Confidence Interval Estimator
Monte Carlo	$[4.49 \times 10^{-5}, 1.19 \times 10^{-4}]$	$[0.665, 0.671]$
Monte Carlo + antithetic paths	$[1.43 \times 10^{-5}, 2.51 \times 10^{-5}]$	$[0.667, 0.670]$
Algorithm 4	$[2.1329 \times 10^{-8}, 5.6610 \times 10^{-8}]$	$[0.6696, 0.6697]$
Algorithm 5	$[1.687 \times 10^{-8}, 4.47 \times 10^{-7}]$	$[0.6695, 0.6697]$
Algorithm 6	$[1.746 \times 10^{-8}, 4.63 \times 10^{-8}]$	$[0.6695, 0.6697]$

Table 2.6: Results on basket options problem on two assets, Example 2.D.1. Models trained with S_0 fixed, non-random.

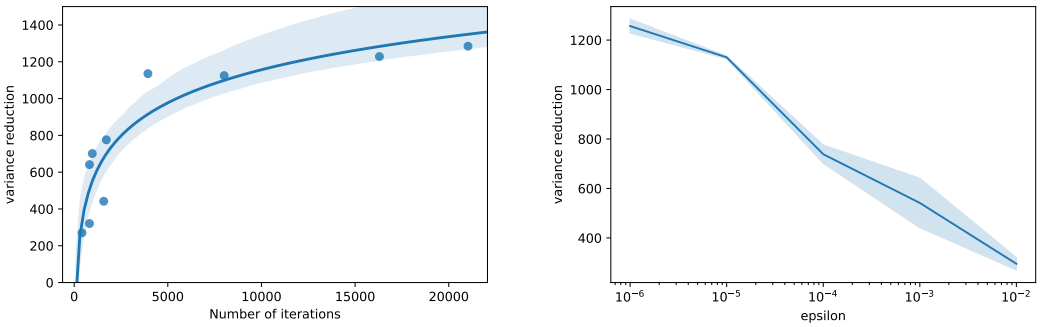


Figure 2.10: Left: Variance reduction in terms of number of optimiser iterations. Right: Variance reduction in terms of epsilon. Both refer to Algorithm 4 used in Example 2.D.1.

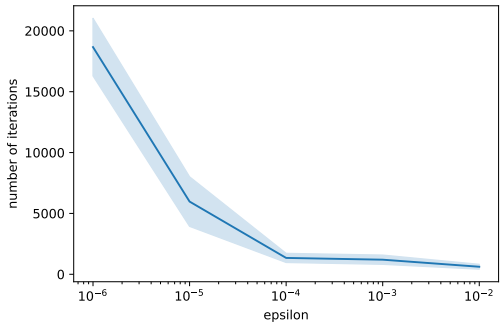


Figure 2.11: Number of optimiser iterations in terms of ϵ for Algorithm 4 used in Example 2.D.1.

Example 2.D.2 (basket option with random sigma)**.** *In this example, as in Example 2.4.2, we aim to show how our approach - where we build a control variate by approximating the process* $(Z_{t_k})_{k=0,\dots,N_{steps}}$ *- is more robust compared to directly approximating the price by a certain function in a high-dimensional setting.*

We use the methodology proposed in [31], where the authors present a deep learning-based calibration method proposing a two-steps approach: first the authors learn the model that approximates the pricing map using a artificial neural network in which the inputs are the parameters of the volatility model. Second the authors calibrate the learned model using available data by means of different optimisation methods.

For a fair comparison between our deep learning based control variate approach vs. the method proposed in [31], we make the following remarks:

i) We will only use the first step detailed in [31] where the input to the model that approximates the pricing map are the volatility model's parameters: $\sigma \in \mathbb{R}^d, r$*, and the initial price is considered constant for training purposes. We run the experiment for* $d = 5$*.*

ii) In [31] the authors build a training set, and then perform gradient descent-based optimisation on the training set for a number of epochs. This is somewhat a limiting factor in the current setting where one can have as much data as they want since it is generated from some given distributions. In line with our experiments, instead of building a training set, in each optimisation step we sample a batch from the given distributions.

iii) In [31], the price mapping function is learned for a grid of combinations of maturities and strikes. In this experiment, we reduce the grid to just one point considering $T = 0.5$*,* $K = \sum_i S_0^i$*, where* $S_0^i = 0.7 \ \forall i$*.*

iv) We will use Algorithm 4 to build the control variate with the difference that now $\sigma \in \mathbb{R}^d, r \in \mathbb{R}$ *will be passed as input to the each network at each time step* $\mathcal{R}[v]_{\eta_k}, \mathcal{R}[\partial_x v]_{\theta_k}$*.*

The experiment is organised as follows:

i) We train the network proposed in [31] approximating the price using Black-Scholes model and Basket options payoff. In each optimisation iteration a batch of size $1\,000$*, where the volatility model's parameters are sampled using* $\sigma \sim \mathcal{U}(0.9, 1.1)$ *and* $r \sim \mathcal{U}(0.4, 0.6)$*. We keep a test set of size 150,* $\mathcal{S} = \{[(\sigma^i, r^i); p(\sigma^i, r^i)], i = 1, \dots, 150\}$ *where* $p(\sigma^i, r^i)$ *denotes the price and is generated using* $50\,000$ *Monte Carlo samples.*

ii) We use Algorithm 4 to build the control variate, where σ *and* r *are sampled as above and are included as inputs to the network. We denote the trained model by* $\mathcal{R}[\partial_x v]_{\theta_k}$ *where* $k = 1, \dots, N_{steps}$*. In contrast with Algorithm 4.*

We present the following results:

i) Figure 2.1 displays the histogram of the squared error of the approximation of the PDE solution $\mathcal{R}[v]_\eta$ *for each instance in* $\mathcal{S}$*. In this sample, it spans from almost* 10^{-8} *to* 10^{-3}*, i.e. for almost five orders of magnitude.*

ii) We build the control variate for that instance in the test set for which $\mathcal{R}[v]_\eta$ *generalises the worst. For those particular* σ, r*, Table 2.7 provides its variance reduction factor.*

Method	Emp. Var.	Var. Red. Fact.
Monte Carlo	1.29	-
Algorithm 4	0.035	37

Table 2.7: Results on basket options problem on 5 assets. Model trained with non-random S_0, and random σ, r.

Example 2.D.3 (High dimensional basket option)**.** *We also consider the basket options problem on* $d = 100$ *assets but otherwise identical to the setting of Example 2.D.1. We compare our results against the same experiment in [58, Sec 4.2.3, Table 6 and Table 7].*

Method	Emp. Var.	Var. Red. Fact.	Train. Paths	Opt. Steps
Monte Carlo	79.83	-	-	-
Algorithm 4	4.72×10^{-4}	168 952	24×10^7	47369
Algorithm 5	1.79×10^{-4}	349 525	37×10^6	7383
Algorithm 6	1.54×10^{-4}	517 201	35×10^6	7097
Method ζ_a^1 in [58]	8.67×10^{-1}	97	-	-
Method ζ_a^2 in [58]	4.7×10^{-3}	17 876	-	-

Table 2.8: Results on basket options problem on 100 assets, Example 2.D.3. Models trained with non-random S_0 so that the results can be directly compared to [58].

Table 2.8 shows a significant improvement of the variance reduction factor (10x and 100x better) of all our Algorithms than the methods proposed in [58] and applied in the same example.

Method	Confidence Interval Variance	Confidence Interval Estimator
Monte Carlo	$[8.57 \times 10^{-4}, 2.27 \times 10^{-3}]$	$[27.351, 27.380]$
Monte Carlo + antithetic paths	$[5.34 \times 10^{-4}, 9.35 \times 10^{-4}]$	$[27.354, 27.371]$
Algorithm 4	$[7.001 \times 10^{-9}, 1.8583 \times 10^{-8}]$	$[27.3692, 27.3693]$
Algorithm 5	$[2.41 \times 10^{-9}, 6.39 \times 10^{-9}]$	$[27.36922, 27.36928]$
Algorithm 6	$[4.1672 \times 10^{-9}, 1.1060 \times 10^{-8}]$	$[27.36922, 27.36928]$

Table 2.9: Results on basket options problem on 100 assets, Example 2.D.3. Models trained with non-random S_0.

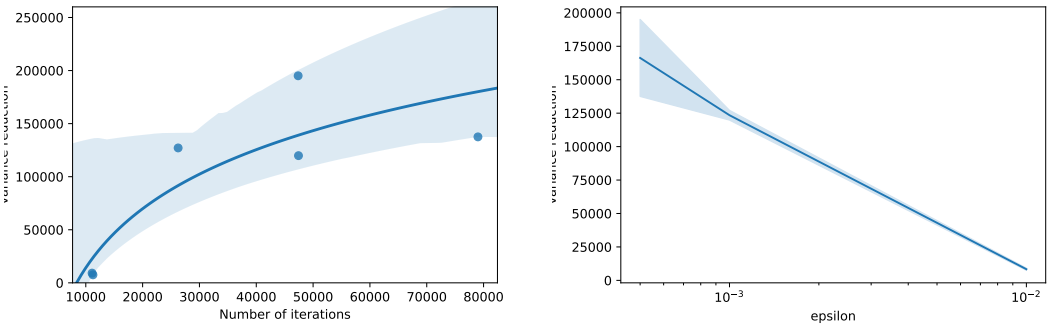


Figure 2.12: Left: Variance reduction in terms of number of optimiser iterations. Right: Variance reduction in terms of epsilon. Both are for Example 2.D.3 and Algorithm 4.

2.D.1 Empirical network diagnostics

In this subsection we consider the exchange options problem on two assets from Example 2.4.1, where the time horizon is one day. We consider different network architectures for the BSDE method described by Algorithm 4 in order to understand their impact on the final result and their ability to approximate the solution of the PDE and its gradient. We choose this problem given the existence of an explicit solution that can be used as a benchmark. The experiment is organised as follows:

i) Let $L-2$ be the number of hidden layers of $\mathcal{R}[\partial_x v]_{\theta_{t_0}} \in \mathcal{DN}$ and $\mathcal{R}[v]_{\theta_{t_0}} \in \mathcal{DN}$. Let l_k be the number of neurones per hidden layer k.

ii) We train four times all the possible combinations for $L-2 \in \{1,2,3\}$ and for $l_k \in \{2,4,6,\dots,20\}$ using $\epsilon = 5 \times 10^{-6}$ for the stopping criteria. The assets' initial values $s^i_{t_0}$ are sampled from a lognormal distribution

$$X \sim \exp((\mu - 0.5\sigma^2)\tau + \sigma\sqrt{\tau}\xi),$$

where $\xi \sim N(0,1), \mu = 0.08, \tau = 0.1$.

iii) We approximate the L^2-error of $\mathcal{R}[v]_{\theta_{t_0}}(x)$ and $\mathcal{R}[v]_{\theta_{t_0}}(x)$ with respect to the exact solution given by Margrabe's formula and its gradient.

Figure 2.13 displays the average of the L^2-errors and its confidence interval. We can conclude that for this particular problem, the accuracy of $\mathcal{R}[v]_{\theta_{t_0}}(x)$ does not strongly depend on the number of layers, and that there is no improvement beyond 8 nodes per hidden layer. The training resources (training sample size, the gradient descent algorithm together with the stopping criteria) becomes the limiting factor. The accuracy of $\mathcal{R}[v]_{\theta_{t_0}}(x)$ is clearly better with two or three hidden layers than with just one. Moreover it seems that there is benefit in taking as many as 10 nodes per hidden layer.

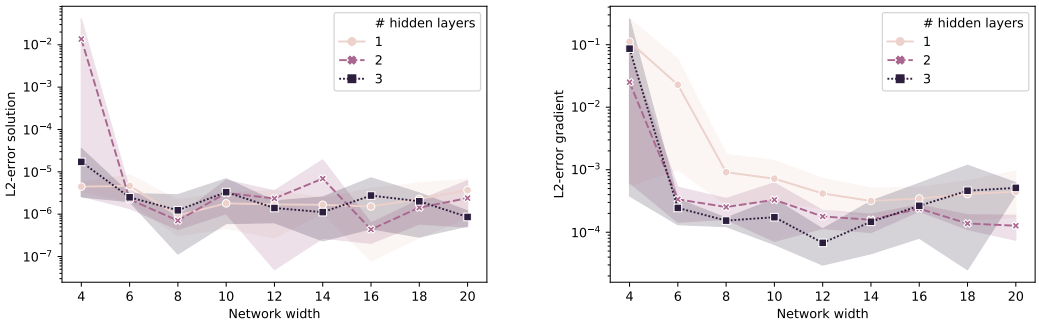


Figure 2.13: Average squared error of PDE solution approximation and its gradient and 95% confidence interval of different combination of # of layers and net width. Left: error model. Right: Error grad model.

Chapter 3

Solving path dependent PDEs with LSTM networks and path signatures

Using a combination of recurrent neural networks and signature methods from the rough path theory we design efficient algorithms for solving parametric families of path dependent partial differential equations (PPDEs) that arise in pricing and hedging of path-dependent derivatives or from use of non-Markovian model, such as rough volatility models [19]. The solutions of PPDEs are functions of time, a continuous path (the asset price history) and model parameters. As the domain of the solution is infinite dimensional many recently developed deep learning techniques for solving PDEs do not apply. Similarly as in [6], we identify the objective function used to learn the PPDE by using a martingale representation theorem. As a result we can de-bias and provide confidence intervals for the neural network-based algorithm. We validate our algorithm using classical models for pricing lookback and auto-callable options and report errors for approximating both prices and hedging strategies.

3.1 Introduction

Deep neural networks trained with stochastic gradient descent algorithms are extremely successful in a number of applications such as computer vision, natural language processing, generative models or reinforcement learning [61]. As these methods work extremely well in seemingly high-dimensional settings, it is natural to investigate their performance in solving high-dimensional PDEs. Starting with the pioneering work [1, 2], where probabilistic representation has been used to learn PDEs using neural networks, recent years have brought an influx of interest in deep PDE solvers. As a result there is now a number of efficient algorithms for solving linear and non-linear PDEs, employing probabilistic representations, that work in high dimensions [9, 10, 48, 21, 62].

The methods in [1, 2] approximate a solution to a single PDE at a single point in space. The first paper to extend the above techniques to families of parametric PDEs with approximations across the whole domain was [6]. In this chapter we extend this to parametric families of path-dependent PDEs (PPDEs). Let $B \subseteq \mathbb{R}^p, p \geq 1$ be a parameter space. Consider $F = F(t, \omega; \beta)$ satisfying

$$\begin{aligned} &\left[\partial_t F + b\nabla_\omega F + \frac{1}{2}\text{tr}\left[\nabla^2_\omega F \sigma^* \sigma\right] - rF\right](t, \omega; \beta) = 0\,, \\ &F(T, \omega; \beta) = g(\omega; \beta)\,, \;\; t \in [0, T]\,, \;\; \omega \in \mathcal{C}([0, T]; \mathbb{R}^d)\,, \;\; \beta \in B\,. \end{aligned} \tag{3.1}$$

Here $t \in [0,T]$, $\omega \in \mathcal{C}([0,T];\mathbb{R}^d)$ the space of continuous paths and $\beta \in B$ and b, σ, r and g are functions of $(t,\omega;\beta)$ which specify the problem. Notice that we are dealing with an equation in an infinite dimensional space. The derivatives ∇_ω and ∇^2_ω are derivatives on the path space $\mathcal{C}([0,T];\mathbb{R}^d)$, see Appendix 3.A, introduced in [63, 64, 60] in the context of functional Itô calculus.

The Feynman–Kac theorem provides a probabilistic representation for F and so Monte Carlo methods can be used to approximate F for a fixed $(t,\omega;\beta)$. Nevertheless, it is clear that approximating the PPDE solution F across the whole space $[0,T] \times \mathcal{C}([0,T];\mathbb{R}^d) \times B$ is an extremely challenging task. A key step is efficiently encoding the information in ω in some finite dimensional structure.

Equations of the form (3.1) arise in mathematical finance with F representing a price of some (path-dependent) derivative. In this context β would be model parameters, t the current time and ω a path "stopped at t" representing the price history of some assets. Moreover $\nabla_\omega F$ is an object which, in many models, gives access to a "hedging strategy" which is of key importance for risk management purposes. Having an approximation of F that can be quickly evaluated for any $\beta \in B$ is essential for model calibration (see e.g. [31]).

3.1.1 Main contributions

The main contributions in this paper are the following:

i) We provide two methods for efficiently encoding the paths ω using long-short-term-memory (LSTM)-based deep learning methods and path-signatures to approximate the solution of a parabolic PPDE on the whole domain and parameter space.

ii) We provide methods for removing bias in the approximation and a posteriori confidence intervals for the neural network-based approximation.

iii) The algorithms we develop are applicable to parametric families of solutions and hence can be used for efficient model calibration from data.

iv) The algorithms we develop provide approximation of $\nabla_\omega F$ thus providing access to the hedging strategies.

v) We test the algorithms for various models and study their relative performance. The code for our proposed methods and for the numerical experiments can be found at `https://github.com/msabvid/Deep-PPDE`.

3.1.2 Overview of existing methods

Let us now provide a brief overview of other methods available in the literature. As has already been mentioned, the probabilistic methods for deep PDE solvers were first explored in [1, 2] (providing solution for a single point in time and space). These methods were further extended in [16, 65, 53, 66] to build deep learning solvers for non-linear PDEs. In [6] the probabilistic methods were extended to families of parametric PDEs with approximations across the whole domain. A different (non-probabilistic) approach was adopted in [15]. There a deep neural network is directly trained to satisfy the differential operator, initial and boundary conditions. This relies on automatic differentiation to calculate the gradient of the network in terms of its input. Related algorithms are being developed in the so-called physics inspired machine learning [67] where one uses PDEs as regulariser of the neural network. See [68] for a survey of recent efforts to solve PDEs in high dimension using deep learning.

Deep Learning methods that approximate the solution of PPDEs have been studied in [69], where the authors extend the work from [15] and they use a LSTM network to include the path-dependency of the solution of the PPDE. A different approach is proposed in [19] where the authors first discretise the space to approximate a PPDE by high-dimensional classical PDE, and use deep BSDE solver approach to solve it.

Unlike the methods mentioned earlier, the algorithms in this chapter can be used for parametric families of solutions of PDEs (PPDEs). Having approximation of $F(\cdot;\beta)$ for any $\beta \in B$ allows for swift calibration of models to market data (e.g options prices) since we also have access to $\nabla_\beta F$ using automatic differentiation. This is particularly appealing for high dimensional problems or for models for which computation of the pricing operator is costly. This line of research has been recently studied in various settings and with various datasets [31, 70, 33, 34, 35, 36, 37]. Recent work in [71, 7] use Neural SDEs to perform a data-driven model calibration i.e. without using a prior assumption on the form of the dynamics of the price process.

3.1.3 Outline

In Section 3.2 we define the notion of Path Dependent PDE, relying on Functional Itô Calculus, and we introduce the Feynman–Kac formula extended for path-dependent functionals. Section 3.3 develops the martingale representation of the discounted price of a path-dependent option, and how it can be used to retrieve the hedging strategy. Section 3.4 develops the algorithms to approximate the solution of a linear PPDE, built on the probabilistic representation of the solution of the PPDE, and the properties of the conditional expectation and the martingale representation of the discounted price. We finally provide some numerical experiments in Section 3.5.

3.1.4 Notation

We will use the following notation

- $t \wedge s = \min(t, s)$
- Let $m, d, \kappa, p, N \in \mathbb{N}$, $B \subseteq \mathbb{R}^p$. Let $F : [0,T] \times \mathcal{C}([0,T], \mathbb{R}^d) \times B \to \mathbb{R}^m$ such that $F(\cdot, \cdot; \beta)$ is a non-anticipative functional for all $\beta \in B$ (see Section 3.2 and Appendix 3.A). We denote a neural network with weights $\theta \in \mathbb{R}^\kappa$ approximating F as:

  $$\mathcal{R}_\theta[F] : [0,T] \times \mathbb{R}^N \times B \to \mathbb{R}^m$$

  where a path $\omega \in \mathcal{C}([0,T], \mathbb{R}^d)$ is encoded by an element of $\mathbb{R}^N$ (see Sections 3.4.1, 3.4.2).
- $\nabla_\omega F(t, \omega; \beta)$ and $\nabla^2_\omega F(t, \omega; \beta)$ are the vector and matrix denoting the first and second order path-derivatives (see Appendix 3.A) of a non-anticipative functional. Furthermore, the space of non-anticipative functionals admitting time-derivative up to first order and path-derivative up to second order, in addition to satisfying the boundedness property of their time and path-derivatives (Appendix 3.A) is denoted by $\mathbb{C}^{1,2}$.
- $S_n(\cdot)_{t_i,t_j}$ denotes the path signature up to the n-th iterated integral of a path $(\omega_t)_{t\in[t_i,t_j]}$.

3.2 PPDE-SDE relationship

Appendix 3.A provides a brief review of the notion of non-anticipative functionals and their path derivatives. In short, a non-anticipative functional $F : [0,T] \times \mathcal{C}([0,T],\mathbb{R}^d) \to \mathbb{R}$ does not look into the future, i.e. given $t \in [0,T]$ and two different paths $\omega, \omega' \in \mathcal{C}([0,T],\mathbb{R}^d)$ such that $\omega_{s\wedge t} = \omega'_{s\wedge t}\ \forall s \in [0,T]$, then $F(t,\omega) = F(t,\omega')$.

The following result allows to represent the solution of a linear PPDE with terminal condition as the expected value of a random variable. Fix a probability space $(\Omega, \mathcal{F}, \mathbb{P})$, and consider a continuous process $(X_t)_{t\in[0,T]}$ given by

$$dX_t^\beta = b(t,(X^\beta_{t\wedge s})_{s\in[0,T]};\beta)dt + \sigma(t,(X^\beta_{t\wedge s})_{s\in[0,T]};\beta)dW_t \tag{3.2}$$

where b, σ are non-anticipative functionals and $(W_t)_{t\in[0,T]}$ is a Brownian motion, and $\beta \in B \subseteq \mathbb{R}^p$. Furthermore, assume that b, σ are such that the SDE admits a unique strong solution. On the other hand, let F satisfy the following conditions:

i) it is regular enough admitting time derivative up to first order and path-derivatives up to second order (as defined in Appendix 3.A).

ii) it is the solution of the following linear PPDE,

$$\begin{aligned} &\left[\partial_t F + b\nabla_\omega F + \frac{1}{2}\text{tr}\left[\nabla^2_\omega F \sigma^*\sigma\right] - rF\right](t,\omega;\beta) = 0\,, \\ &F(T,\omega;\beta) = g(\omega;\beta)\,,\ \ t\in[0,T]\,,\ \omega \in \mathcal{C}([0,T];\mathbb{R}^d)\,,\ \beta \in B\,. \end{aligned} \tag{3.3}$$

Then one can establish a probabilistic representation of $F(t,\omega;\beta)$ via the Feynman-Kac formula. In the following result, we assume β is fixed, and we abuse the notation to write $F(t,\omega) := F(t,\omega;\beta)$.

Theorem 3.2.1 (Feynman-Kac formula for path-dependent functionals, see Th. 8.1.13 in [72])**.** *Consider the functional $g : \mathcal{C}([0,T],\mathbb{R}^d) \to \mathbb{R}$, continuous with respect to the distance $d_\infty(\omega,\omega') := \sup_{t\in[0,T]}|\omega(t)-\omega'(t)|$. If for every $(t,\omega) \in \big([0,T],\mathcal{C}([0,T],\mathbb{R}^d)\big)$ the functional $F \in \mathbb{C}^{1,2}$ verifies* (3.3)*, then F has the probabilistic representation*

$$F(t,\omega) = e^{-r(T-t)}\mathbb{E}\left[g((X_t)_{t\in[0,T]})\middle|(X_{t\wedge s})_{s\in[0,T]} = (\omega_{t\wedge s})_{s\in[0,T]}\right]. \tag{3.4}$$

with $(X_t)_{t\in[0,T]}$ given by (3.2)*.*

A direct consequence of the Feynman–Kac formula for path-dependent functionals is that solving (3.3) is equivalent to pricing the path-dependent option with payoff at T given by $g((X_s)_{s\in[0,T]}) \in L^2(\mathcal{F}_T)$ where $(X_t)_{t\geq 0}$ is the solution of (3.2) and $(\mathcal{F}_t)_{t\geq 0}$ denotes the filtration generated by $(X_t)_{t\geq 0}$. We will build two algorithms based on two different approaches:

i) Option pricing using the martingale representation theorem (Algorithm 9).

ii) Considering the conditional expectation in (3.4) as the orthogonal projection of $g(X_T)$ on $\mathcal{L}^2(\mathcal{F}_t)$ where $(\mathcal{F}_t)_{t\geq 0}$ is the filtration generated by $(X_t)_{t\geq 0}$ (Algorithm 8).

3.3 Option pricing via Martingale Representation Theorem

In this section we assume constant interest rate and a complete market but the results readily extend to the case of stochastic interest rates and incomplete markets.

Let $(\Omega, \mathcal{F}, \mathbb{Q})$ be a probability space where $\mathbb{Q}$ is the risk-neutral measure. Consider an $\mathbb{R}^d$-valued Wiener process $W = (W^j)_{j=1}^d = ((W_t^j)_{t\geq 0})_{j=1}^d$. We will use $(\mathcal{F}_t^W)_{t\geq 0}$ to denote the filtration generated by W. Consider an $D \subseteq \mathbb{R}^d$-valued, continuous, stochastic process $X = (X^i)_{i=1}^d = ((X_t^i)_{t\geq 0})_{i=1}^d$ that is adapted to $(\mathcal{F}_t^W)_{t\geq 0}$. We will use $(\mathcal{F}_t)_{t\geq 0}$ to denote the filtration generated by X.

We recall that we denote by $\beta \in B \subseteq \mathbb{R}^p$ the family of parameters of the dynamics of the underlying asset (for instance, in the Black–Scholes model with fixed risk-free rate, β denotes the volatility). Let $g : \mathcal{C}([0,T], \mathbb{R}^d) \times B \to \mathbb{R}$ such that $g(\cdot, \beta)$ is a measurable function for each $\beta \in B \subseteq \mathbb{R}^p$. We shall consider contingent claims of the form $g((X_s^\beta)_{s\in[0,T]}; \beta)$. This means that we can consider path-dependent derivatives. Finally, let r be some risk-free rate, and consider, for each β, the SDE

$$dX_t^\beta = rX_t^\beta dt + \sigma(t, (X_{s\wedge t}^\beta)_{s\in[0,T]}; \beta)dW_t. \tag{3.5}$$

We immediately see that $\bar{X}_t^\beta = (e^{-rt}X_t^\beta)_{t\in[0,T]}$ is a (local) martingale.

In order to price an option at time t with payoff g, under appropriate assumptions on g and σ, the random variable

$$M_t^\beta := \mathbb{E}\left[e^{-rT} g((X_s^\beta)_{s\in[0,T]}; \beta)\middle| \mathcal{F}_t^\beta\right]$$

is square-integrable. $M_T^\beta = e^{-rT} g((X_s^\beta)_{s\in[0,T]})$ is the discounted payoff at T. Hence $F(t, (X_{t\wedge s}^\beta)_{s\in[0,T]}; \beta) = e^{rt}M_t^\beta$ is the fair price of the option with payoff g at time t. By the Martingale representation theorem, for each β there exists a unique $\mathcal{F}_t$-adapted process Z_s^β with $\mathbb{E}\left[\int_0^T (Z_s^\beta)^2 ds\right] < \infty$ such that

$$M_T^\beta = \mathbb{E}[M_T^\beta | \mathcal{F}_0] + \int_0^T Z_s^\beta dW_s. \tag{3.6}$$

In order to retrieve the real hedging strategy from the martingale representation, it is necessary to apply the Itô formula for non-anticipative functionals of a continuous semimartingale, see [63, 73]:

Proposition 3.3.1. . *Let X be a continuous semimartingale defined on $(\Omega, \mathcal{F}, \mathbb{Q})$. Then, for any non-anticipative functional $F \in \mathbb{C}^{(1,2)}$ (introduced in Section 3.1.4) and any $t \in [0,T]$, we have*

$$\begin{aligned} dF(t, (X_{s\wedge t})_{s\in[0,T]}) =& \partial_t F(t, (X_{s\wedge t})_{s\in[0,T]})dt + \nabla_\omega F(t, (X_{s\wedge t})_{s\in[0,T]})dX_t \\ &+ \frac{1}{2} tr\left((\nabla_\omega^2 F(t, (X_{s\wedge t})_{s\in[0,T]}))dX_t dX_t\right). \end{aligned} \tag{3.7}$$

Let $\bar{F}_t := e^{-rt}F(t, (X_{t\wedge s}^\beta)_{s\in[0,T]}; \beta)$ be the discounted price of the option with payoff g at time t. Then, using (3.7),

$$d\bar{F}_t = \left(-rF + \partial_t F + \frac{1}{2}\text{tr}(\nabla_\omega^2 F\sigma^*\sigma) + rX_t\nabla_\omega F\right)e^{-rt}dt + \nabla_\omega F \cdot e^{-rt}\sigma dW_t\,. \tag{3.8}$$

Moreover, $\bar{F}_t$ is the value of the discounted portfolio at t (since the market is complete) thus the coefficient of dt in (3.8) is 0. Noting that $d\bar{X}_t^\beta = e^{-rt}\sigma(t, (X_{t\wedge s}^\beta)_{s\in[0,T]}; \beta)dW_t$, we get

$$d\bar{F}_t = \nabla_\omega F d\bar{X}_t.$$

Hence after replacing in (3.6) one can retrieve the hedging strategy

$$M_t^\beta = \mathbb{E}[M_t^\beta|\mathcal{F}_0] + \int_0^t \nabla_\omega F \, d\bar{X}_s^\beta, \quad M_T^\beta = e^{-rT} g((X_s^\beta)_{s\in[0,T]}; \beta)\,. \tag{3.9}$$

Both M_t^β and the stochastic integral in (3.9) are martingales. This will be used in Section 3.4.4 to build a learning task to solve the BSDE (3.6) to jointly approximate the fair price of the option F, and the hedging strategy $\nabla_\omega F$.

3.4 Deep PPDE solver Methodology

In this section we present the PPDE solver methodology consisting on the data simulation scheme (Section 3.4.1). We briefly define the signature of a path, that we will use as a path feature extractor (Section 3.4.2). We describe two optimisation tasks to approximate the price of path-dependent derivatives, relying on conditional expectation properties (Section 3.4.3), and on the martingale representation of the discounted price (Section 3.4.4). We present the learning scheme leveraging the learning methods, the different deep network architectures considered, and the path signatures (Section 3.4.5). Finally, we present the evaluation metrics used in the numerical experiments (Section 3.4.6), as well as the numerical results in Section 3.5.

3.4.1 Data simulation – Forward discretisation scheme

We consider the SDE with path-dependent coefficients in equation (3.5). and the path-dependent payoff, $g : \mathcal{C}([0,T],\mathbb{R}^d) \times B \to \mathbb{R}$. We will consider two different time discretisations.

i) First, a fine time discretisation $\pi^f := \{0 = t_0^f < t_1^f < \ldots < t_N^f = T\}$ used by the numerical SDE solver to sample paths from (3.5).

ii) We consider a second, coarser, time discretisation $\pi^c := \{0 = t_0^c < t_1^c < \ldots < t_M^c = T\} \subseteq \pi^f$ on which we learn the deep learning approximation of the price of the option. We will see in the next section that the path dependence of the solution of the PDE at time t requires building an algorithm that takes as input the trajectories that we build by interpolating between the points of the finer time discretisation.

Furthermore, we fix the distribution of $\beta \in B \subseteq \mathbb{R}^p$. We will denote the discretisation of $(X_t^\beta)_{t\in[0,T]}$ in π^f using Euler scheme as $(X_t^{\beta,\pi^f})_{t\in\pi^f}$.

3.4.2 Signatures and elements from rough path theory

In this section we collect some objects and properties that will be used later. If not mentioned otherwise, the reader is referred to [74] and the references therein, as well as to [75], for the use of signatures in machine learning.

It is first necessary to introduce the space of formal series of tensors, which is the space signatures live in. For simplicity, we restrict ourselves to tensors over $\mathbb{R}^d$. We denote by $(\mathbb{R}^d)^{\otimes n}$ the usual space of tensors over $\mathbb{R}^d$ of order $n \geq 0$.

Definition 3.4.1. *(i) The space of formal series of tensors of $\mathbb{R}^d$, denoted by $T((\mathbb{R}^d))$, is defined as space of sequences,*

$$T((\mathbb{R}^d)) := \{\mathbf{a} = (a_0, a_1, a_2, \ldots) : a_n \in (\mathbb{R}^d)^{\otimes n}, n \in \mathbb{N}\}.$$

For two elements $\mathbf{a} = (a_0, a_1, \dots)$ *and* $\mathbf{b} = (b_0, b_1, \dots)$ *we can define an addition and a product by*

$$\mathbf{a} + \mathbf{b} = (a_0 + b_0, a_1 + b_1, \dots), \quad \mathbf{a} \otimes \mathbf{b} = (c_0, c_1, \dots),$$

where for each $n \in \mathbb{N}_0$*, with the usual (finite-dimensional) tensor product* $\otimes$*,* $c_n = \sum_{k=1}^{n} a_k \otimes b_{n-k}$*. (ii) Let* $N \in \mathbb{N}$ *and define* $B_N = \{\mathbf{a} \in T((\mathbb{R}^d)) : a_0 = \cdots = a_N = 0\}$*. Then the truncated tensor algebra of order* N *is the quotient algebra*

$$T^N(\mathbb{R}^d) = T((\mathbb{R}^d))/B_N,$$

with the canonical homomorphism $\pi_N : T((\mathbb{R}^d)) \to T^N(\mathbb{R}^d)$*.*

We can naturally identify $T^N(\mathbb{R}^d)$ with $\mathbb{R} \oplus \mathbb{R}^d \oplus \cdots \oplus (\mathbb{R}^d)^{\otimes N}$. Now we can introduce the (truncated) signature.

Definition 3.4.2. *Let* $X : [0, T] \to \mathbb{R}^d$ *be a path of finite variation and for* $s, t \in [0, T]$*,* $n \in \mathbb{N}$ *define the iterated integral*

$$X_{s,t}^{(n)} := \int \cdots \int_{s<s_1<\cdots<s_n<t} dX_{s_1} \otimes \cdots \otimes dX_{s_n}.$$

Then the signature of X *over* $(s, t) \subset [0, T]$ *is*

$$S(X)_{s,t} = (1, X_{s,t}^{(1)}, X_{s,t}^{(2)}, \dots) \in T((\mathbb{R}^d)).$$

Similarly, the truncated signature is

$$S_N(X)_{s,t} = (1, X_{s,t}^{(1)}, \dots, X_{s,t}^{(N)}) \in T^N(\mathbb{R}^d).$$

Though signature captures deep geometric properties of a path, it does not necessarily characterise the path completely. It was shown that for continuous paths of bounded variation, the signature determines the path up to tree like equivalence [76]. A sufficient result for the present case is the following, Theorem 2.29 in [74].

Theorem 3.4.3. *Among all paths with bounded variation sharing the same signature, there exists a path with minimal length, which is unique up to reparametrization.*

It is clear that for a basis $(e_1, \dots, e_d)$ of $\mathbb{R}^d$ and its dual basis $(e_1^*, \dots, e_d^*)$ of $(\mathbb{R}^d)^*$, the elements $(e_I = e_{i_1} \otimes \cdots \otimes e_{i_n})_{I=\{i_1,\dots,i_n\} \subset \{1,\dots,d\}^n}$ form a basis of $(\mathbb{R}^d)^{\otimes n}$, just as the elements $(e_I^* = e_{i_1}^* \otimes \cdots \otimes e_{i_n}^*)_{I=\{i_1,\dots,i_n\} \subset \{1,\dots,d\}^n}$ form a basis of $((\mathbb{R}^d)^*)^{\otimes n}$. Recall that we can canonically identify $((\mathbb{R}^d)^*)^{\otimes n}$ with $((\mathbb{R}^d)^{\otimes n})^*$. Thus we have a linear mapping $((\mathbb{R}^d)^*)^{\otimes n} \to (T((\mathbb{R}^d)))^*$ by the relation

$$e_I^*(\mathbf{a}) = e_I^*(\pi_n(\mathbf{a})) = a_{i_1,\dots,i_n},$$

which is the coefficient in front of the basis vector e_I in $\mathbf{a}$. In this way we get a linear mapping [74]

$$T((\mathbb{R}^d)^*) = \bigoplus_{n=0}^{\infty} ((\mathbb{R}^d)^*)^{\otimes n} \to (T((\mathbb{R}^d)))^*.$$

Thus also, for a path X and its signature $S(X)$, by linearity

$$e_I^*(\mathbf{X}) = \int\limits_{0<s_1<\cdots<s_n<t} \cdots \int e_{i_1}^*(dX_{s_1}) \otimes \cdots \otimes e_{i_n}^*(dX_{s_n}).$$

The following result motivates the usage of signatures as path feature extractors in our algorithms, [77].

Theorem 3.4.4. *Consider a compact set K in the space of continuous paths of bounded variation. Let $f : K \to \mathbb{R}$ be continuous. Then for every $\varepsilon > 0$ there exists an integer $M > 0$ and a linear functional $\mathbf{L} \in (T^M(\mathbb{R}^d))^*$ acting on the truncated signatures such that*

$$\sup_{X \in K} |f(X) - \langle \mathbf{L}, S_M(X) \rangle| \leq \varepsilon.$$

Remark 3.4.5. Theorem 3.4.4 refers to continuous paths of bounded variation. This constraint is enough in our setting, where the data trajectories are built by linearly interpolating between the points in $(X_t^{\beta,\pi^f})_{t\in\pi^f}$.

Parametrisation of the PPDE solution using LSTM networks and the path signature

We consider the augmented path $\mathbf{X} = (t, X_t)_{t\in[0,T]}$ including time as the first path coordinate. We use Theorem (3.4.4) to approximate $F((\mathbf{X}_{t\wedge s})_{s\in[0,T]})$ locally around some basis point $F((\mathbf{X}_{u\wedge s})_{s\in[0,T]})$ for $u < t$,

$$F((\mathbf{X}_{t\wedge s})_{s\in[0,T]}) \approx F((\mathbf{X}_{u\wedge s})_{s\in[0,T]}) + \langle \mathbf{L}, S_M(\mathbf{X})_{u,t} \rangle \tag{3.10}$$

for some appropriate $\mathbf{L} \in (T^M(\mathbb{R}^(d+1)))^*, M > 0$. We use recurrent networks with LSTM cells (see Appendix 3.B.2) to parameterise the recurrent form of (3.10).

Recall we have defined two time discretisations, π^f, π^c, and we use the low-capital notation $\mathbf{x}$ as the time-augmented path that is linear between points in π^f. Then

$$F((\mathbf{x}_{t_k\wedge s})_{s\in[0,T]}) \approx \mathcal{R}[F]_\theta \left(S_n \left((\mathbf{x}_t^{\pi^f})_{t\in\pi^f} \right)_{u,t_k} \right) \tag{3.11}$$

where $u := \max_{t\in\pi^c} t \leq t_k$, and $\mathcal{R}[F]_\theta$ is defined recursively as follows:

$$\begin{aligned} h_{t_k} &:= LSTM \left(h_{t_{k-1}}, S_M \left((\mathbf{x}_t^{\pi^f})_{t\in\pi^f} \right)_{u,t_k} ; \theta_h \right), \text{ where } h_{-1} = 0 \\ \mathcal{R}[F]_\theta &\left(S_n \left((\mathbf{x}_t^{\pi^f})_{t\in\pi^f} \right)_{u,t_k} \right) := \ell(h_{t_k}), \end{aligned} \tag{3.12}$$

with $\ell(\cdot)$ denoting an affine transformation, see Figure 3.1. Details of the operations performed in each LSTM cell are provided in Appendix 3.B.2. We add a linear layer to the output of each LSTM cell so that the network is able to approximate any value in $\mathbb{R}$.

3.4.3 Learning conditional expectation as orthogonal projection

The following theorem recalls a well known property of conditional expectations:

Theorem 3.4.6. *Let $X \in \mathcal{L}^2(\mathcal{F})$. Let $\mathcal{G} \subset \mathcal{F}$ be a sub σ-algebra. There exists a random variable $Y \in \mathcal{L}^2(\mathcal{G})$ such that*

$$\mathbb{E}[|X - Y|^2] = \inf_{\eta\in\mathcal{L}^2(\mathcal{G})} \mathbb{E}[|X - \eta|^2]. \tag{3.13}$$

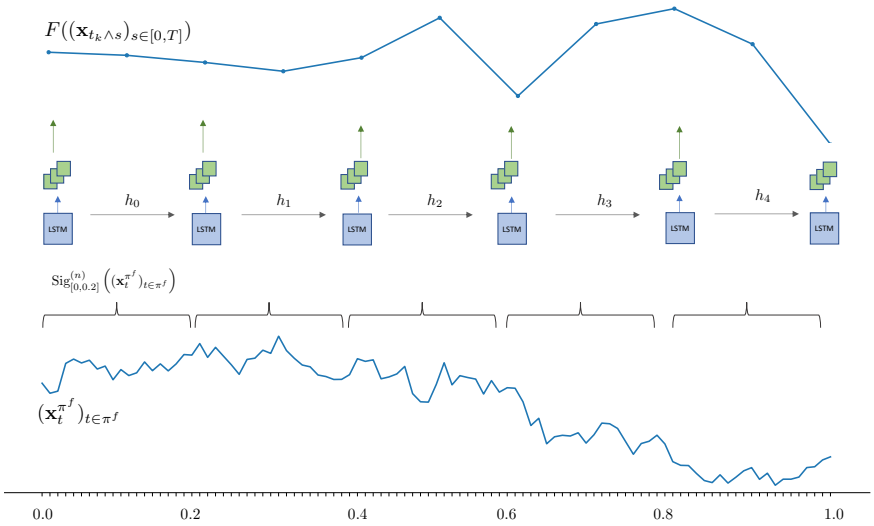


Figure 3.1: Diagram of LSTM network

The minimiser, Y, is unique and is given by $Y = \mathbb{E}[X|\mathcal{G}]$.

The theorem tells us that the conditional expectation is an orthogonal projection of a random variable X onto $\mathcal{L}^2(\mathcal{G})$. To formulate the learning task in our problem, we replace X in (3.13) by $e^{-r(T-t)}g((X^\beta_s)_{s\in[0,T]};\beta)$. We also replace $\mathcal{G}$ by $\mathcal{F}^\beta_t$, and $\mathcal{F}$ by $\mathcal{F}^\beta_T$. Then by Theorem 3.4.6

$$\begin{aligned} F(t,(X^\beta_s)_{s\in[0,T]};\beta) &= \mathbb{E}[e^{-r(T-t)}g((X^\beta_s)_{s\in[0,T]};\beta)|\mathcal{F}^\beta_t] \\ &= \arg\inf_{\eta\in\mathcal{L}^2(\mathcal{F}_t)} \mathbb{E}[|e^{-r(T-t)}g((X^\beta_s)_{s\in[0,T]};\beta)-\eta|^2] \end{aligned}$$

By the Doob–Dynkin Lemma [51, Th. 1.3.12] we know that every $\eta \in L^2(\mathcal{F}^\beta_t)$ can be expressed as $\eta = h_t((X^\beta_{s\wedge t})_{s\in[0,T]})$ for some appropriate measurable h_t. For the practical algorithm we restrict the search for the function h_t to the class that can be expressed as deep neural networks.

We then consider deep network approximations of the price of the path-dependent option at any time t^c_k in the time partition π^c, by using the stream of signatures of the time-augmented path:

$$(\mathcal{X}^{\beta,\pi^f}_t)_{t\in\pi^c} := \left(S_{(n)}((\mathbf{x}^{\beta,\pi^f}_t)_{t\in\pi^f})_{t_k,t_{k+1}}\right)_{t_k\in\pi^c}, \quad n\geq 1 \tag{3.14}$$

and set the learning task as

$$\theta^* = \arg\min_\theta \mathbb{E}_\beta\left[\mathbb{E}_{(X^{\beta,\pi^f}_t)_{t\in\pi^f}}\left[\sum_{k=0}^{N}\left(e^{-r(T-t_k)}g((X^{\beta,\pi^f}_t)_{t\in\pi^f};\beta)-\mathcal{R}_\theta[F]((\mathcal{X}^{\beta,\pi^f}_t)_{t\in\pi^c},\beta)\right)^2\right]\right]. \tag{3.15}$$

We observe that the inner expectation in (3.15) is taken across all paths generated using the numerical SDE solver on (3.5) on π^f for a fixed β and it allows to price an option for such β. The outer expectation is taken on β for which the distribution is fixed (as specified in the data simulation scheme), thus allowing to find the optimal neural network weights θ^* to price the parametric family of options.

Algorithm 7 Data simulation

Initialisation: $N_{\text{trn}} \in \mathbb{N}$ large, distribution of β.
for $i : 1 : N_{\text{trn}}$ **do**
Generate training set:

- Sample different values of β from a fixed distribution (for example Uniform distribution).
- Generate $(x_t^{\beta,\pi^f,i})_{t\in\pi}$ by using numerical SDE solver on (2.2)

end for
return $(x_t^{\beta,\pi^f,i})_{t\in\pi^c}, (\mathcal{X}_{t_k}^{\beta,\pi^f,i})_{t_k\in\pi^c}$ for $i = 1, \ldots, N_{\text{trn}}$.

Algorithm 8 Learning orthogonal projection

Initialisation: Weights θ of networks $\mathcal{R}[F]_\theta$, $N_{\text{trn}} \in \mathbb{N}$ large, distribution of β.
Use SGD to find θ^*, where in each iteration of SGD we generate a batch of paths (or a batch of stream of signatures) using Algorithm 7.

$$\theta^* = \arg\min_\theta \ \mathbb{E}^{\mathbb{Q}^{N_{\text{trn}}}}\left[\sum_{k=0}^{N_{\text{steps}}-1}\left(e^{-r(T-t_k)}g((X_t^{\beta,\pi^f})_{t\in\pi^f}) - \mathcal{R}[F]_\theta((\mathcal{X}_t^{\beta,\pi^f})_{t\in\pi^c},\beta)\right)^2\right]$$

where $\mathbb{E}^{\mathbb{Q}^{N_{\text{trn}}}}$ denotes the empirical mean.
return θ^*.

3.4.4 Learning martingale representation of the option price

From Section 3.3, for fixed β the discounted price of the option with payoff g is given by

$$M_T^\beta = \mathbb{E}[M_T^\beta|\mathcal{F}_0] + \int_0^T \nabla_\omega F \, d\bar{X}_s^\beta. \tag{3.16}$$

Since both $(M_t^\beta)_{t\in[0,T]}$ and the stochastic integral are martingales, after taking expectations conditioned on $\mathcal{F}_s^\beta, \mathcal{F}_t^\beta$ on both sides for $s < t \leq T$ one gets

$$M_t^\beta = M_s^\beta + \int_s^t \nabla_\omega F \, d\bar{X}_r^\beta. \tag{3.17}$$

After replacing the pair s, t in (3.17) by all possible consecutive times in π^c, one obtains a *backward* system of equations starting from the final condition $M_T^\beta = e^{-rT}g((X_t^\beta)_{t\in\pi^f};\beta)$

$$M_{t_k}^\beta = M_{t_{k-1}}^\beta + \int_{t_{k-1}}^{t_k} \nabla_\omega F \, d\bar{X}_r^\beta, \quad k = N, \ldots, 1. \tag{3.18}$$

Considering the deep learning approximations of the price of the option and the hedging strategy with two neural networks and replacing them in (3.18) then the sum of the L^2-errors that arise from (3.18)

yields the optimisation task to learn the weights of $\mathcal{R}[F]_\theta, \mathcal{R}[\nabla_\omega F]_\phi$:

$$
\begin{aligned}
(\theta^*, \phi^*) := \arg\min_{(\theta,\phi)} \mathbb{E}_\beta \Bigg[\mathbb{E}_{(X_t^{\beta,\pi^f})_{t\in\pi^f}} \Bigg[\Big(g((X_t^{\beta,\pi^f})_{t\in\pi_f}; \beta) - \mathcal{R}[F]_\theta((\mathcal{X}_t^{\beta,\pi^f})_{t\in\pi^c}; \beta) \Big)^2 + \\
\sum_{m=0}^{N-1} |\mathcal{E}_{m+1}^{(\theta,\phi)}|^2 \Bigg] \Bigg], \qquad (3.19) \\
\mathcal{E}_{m+1}^{(\eta,\theta)} := e^{-rt_{m+1}} \mathcal{R}[F]_\theta((\mathcal{X}_t^{\beta,\pi^f})_{t\in\pi^c}; \beta) - e^{-rt_m} \mathcal{R}[F]_\theta((\mathcal{X}_t^{\beta,\pi^f})_{t\in\pi^c}; \beta) \\
- e^{-rt_m} \mathcal{R}[\nabla_\omega F]_\phi((\mathcal{X}_t^{\beta,\pi^f})_{t\in\pi^c}; \beta) \sigma((\mathcal{X}_t^{\beta,\pi^f})_{t\in\pi^c}; \beta) \Delta W_{t_m} ,
\end{aligned}
$$

where as before $(\mathcal{X}_t^{\beta,\pi^f})_{t\in\pi^c}$ denotes the choice of the input used in the learning algorithm.

Learning task in pseudocode is provided in Algorithm 9.

Algorithm 9 Learning Martingale representation

Initialisation: Weights θ, ϕ of networks $\mathcal{R}[F]_\theta, \mathcal{R}[\nabla_\omega F]_\phi$, $N_{\text{trn}} \in \mathbb{N}$ large, distribution of β.
Use SGD to find (θ^*, ϕ^*), where in each iteration of SGD we generate a batch of paths (or a batch of stream of signatures) using Algorithm 7.

$$
\begin{aligned}
(\theta^*, \phi^*) := \arg\min_{(\theta,\phi)} \mathbb{E}^{\mathbb{Q}^{N_{\text{trn}}}} \Bigg[\Big(g((X_t^{\beta,\pi^f})_{t\in\pi_f}; \beta) - \mathcal{R}[F]_\theta((\mathcal{X}_t^{\beta,\pi^f})_{t\in\pi^c}; \beta) \Big)^2 \\
+ \sum_{m=0}^{N-1} |\mathcal{E}_{m+1}^{(\theta,\phi)}|^2 \Bigg] , \\
\mathcal{E}_{m+1}^{(\eta,\theta)} := e^{-rt_{m+1}} \mathcal{R}[F]_\theta((\mathcal{X}_t^{\beta,\pi^f})_{t\in\pi^c}; \beta) - e^{-rt_m^c} \mathcal{R}[F]_\theta((\mathcal{X}_t^{\beta,\pi^f})_{t\in\pi^c}; \beta) \\
- e^{-rt_m} \mathcal{R}[\nabla_\omega F]_\phi((\mathcal{X}_t^{\beta,\pi^f})_{t\in\pi^c}; \beta) \sigma((\mathcal{X}_t^{\beta,\pi^f})_{t\in\pi^c}; \beta) \Delta W_{t_m} ,
\end{aligned}
$$

where $\mathbb{E}^{\mathbb{Q}^{N_{\text{trn}}}}$ denotes the empirical mean.
return (θ^*, ϕ^*).

3.4.5 Unbiased PPDE solver

An unbiased estimator of the solution of the PPDE at any $t \in [0, T]$ can be obtained with a Monte Carlo estimator using the Feynman–Kac theorem. We will use notation from Section 3.3 and fix $\beta \in B$ as the parameters and $t \in [0, T]$ as the current time and $(x_{t\wedge s}^\beta)_{s\in[0,T]}$ as a stopped path at t. From (3.9) we see that with $M_t = e^{-rt} F(t, (x_{t\wedge s}^\beta)_{s\in[0,T]}; \beta)$ we have

$$
M_t = M_T - \int_t^T e^{-rs'} \big[(\nabla_\omega F)\sigma \big](s', (X_{s'\wedge s}^\beta)_{s\in[0,T]}; \beta) \, dW_{s'} .
$$

If we replace the exact gradient $\nabla_\omega F$ by $\mathcal{R}_\phi[\nabla_\omega F]$ we can use this as control variate. Let

$$
M_t^{\text{cv},\phi} := M_T - \int_t^T e^{-rs'} \big(\mathcal{R}_\phi[\nabla_\omega F]\sigma \big)(s', (X_{s'\wedge s}^\beta)_{s\in[0,T]}; \beta) \, dW_{s'} .
$$

While $\mathbb{V}\text{ar}[M_t] = 0$ for a good approximation $\mathcal{R}_\phi[\nabla_\omega F]$ to $\nabla_\omega F$ we will have $\mathbb{V}\text{ar}[M_t^{\text{cv},\phi}]$ small. Consider $(W^i)_{i=1}^N$, N i.i.d copies of W. Let $\mathbb{Q}^N := \frac{1}{N} \sum_{i=1}^N \delta_{X^i}$ be the empirical approximation of the risk neutral measure. Then

$$
F(t, (X_{t\wedge s}^\beta)_{s\in[0,T]}; \beta) \approx e^{rt} \mathbb{E}^{\mathbb{Q}^N} \big[M_t^{\text{cv},\phi} | \mathcal{F}_t \big] . \qquad (3.20)
$$

Central Limit Theorem tells us that

$$\mathbb{Q}\left(F(t,(X^{\beta}_{t\wedge s})_{s\in[0,T]};\beta)\in\left[e^{rt}\,\mathbb{E}^{\mathbb{Q}^N}\left[M^{\mathrm{cv},\phi}_t|\mathcal{F}_t\right]\pm z_{\alpha/2}\frac{\Sigma}{\sqrt{N}}\right]\right)\to 1\quad\text{as}\quad N\to\infty\,.$$

Here $\Sigma^2:=\mathbb{V}\mathrm{ar}[e^{rt}M^{\mathrm{cv},\phi}_t]$ is small by construction and $z_{\alpha/2}$ is such that $1-\mathrm{CDF}_Z(z_{\alpha/2})=\alpha/2$ with Z the standard normal distribution. Hence (3.20) is a very accurate approximation even for small values of N since $\Sigma^2=\mathbb{V}\mathrm{ar}[e^{rt}M^{\mathrm{cv},\phi}_t]$ is small by construction.

An unbiased approximation of $F(t,(X^{\beta}_{t\wedge s})_{s\in[0,T]};\beta)$ together with confidence intervals can be obtained using Algorithm 10.

Algorithm 10 Unbiased PPDE solver

Input: current time t, path history $(x^{\beta,\pi^f}_{t\wedge s})_{s\in\pi^f}$, model parameters β, confidence level α, $N_{MC}\in\mathbb{N}$, optimal weights ϕ^* for $\mathcal{R}[\nabla_\omega F]_{\phi^*}$.
Use Algorithm 7 to generate N_{MC} paths using the numerical SDE solver on (3.5) **starting from** (t,x^{β,π^f}_t), obtaining

$$(x^{\beta,\pi^f,i}_s)_{s\in\pi^c},(\mathcal{X}^{\beta,\pi^f,i}_s)_{s\in\pi^c}\text{ for }i=1,\dots,N_{MC}$$

such that for each i, $(x^{\beta,\pi^f,i}_{t\wedge s})_{s\in\pi^c}=(x^{\beta,\pi^f}_{t\wedge s})_{s\in\pi^c}$.
Use the generated N_{MC} paths to calculate

$$F^{\mathrm{MC}}(t,(x^{\beta,\pi^f}_{t\wedge s})_{s\in\pi^c};\beta)=\mathbb{E}^{\mathbb{Q}^{N_{MC}}}\left[e^{rt}M^{cv,\phi^*}_t\,\middle|\,(X^{\beta}_{t\wedge s})_{s\in[0,T]}=(x^{\beta,\pi^f}_{t\wedge s})_{s\in\pi^c}\right]$$
$$(\Sigma^{\mathrm{MC}})^2:=\mathbb{V}\mathrm{ar}^{\mathbb{Q}^{N_{MC}}}\left[e^{rt}M^{cv,\phi^*}_t\,\middle|\,(X^{\beta}_{t\wedge s})_{s\in[0,T]}=(x^{\beta,\pi^f}_{t\wedge s})_{s\in\pi^c}\right]$$

where

$$M^{cv,\phi^*}_t:=e^{-rT}g((X^{\beta,\pi^f}_s)_{s\in\pi^f},\beta)-\sum_{s\in\pi^c,s\geq t}e^{-rs}(\mathcal{R}[\nabla_\omega F]_{\phi^*}\sigma)(s,(\mathcal{X}^{\beta,\pi^f}_s)_{s\in\pi^c})\Delta W_s,$$

and $\mathbb{E}^{\mathbb{Q}^{N_{MC}}}$ and $\mathbb{V}\mathrm{ar}^{\mathbb{Q}^{N_{MC}}}$ denote the empirical mean and the empirical variance of the Monte Carlo estimator respectively.
Calculate the confidence interval of the unbiased estimator:

$$I:=\left(F^{\mathrm{MC}}(t,(x^{\beta,\pi^f}_{t\wedge s})_{s\in\pi^c};\beta)\pm z_{\alpha/2}\frac{\Sigma^{\mathrm{MC}}}{\sqrt{N_{MC}}}\right).$$

return de-biased estimate $F^{\mathrm{MC}}(t,(x^{\beta,\pi^f}_{t\wedge s})_{s\in\pi^c};\beta)$ and confidence interval I.

3.4.6 Evaluation scheme

In this section we provide the evaluation measures of the deep solvers introduced in Algorithms 8 and 9.

Integral error of the option price parametrisation

We provide the absolute integral error of the solution of the path-dependent PDE, calculated on a test set for a fixed β. Recall that the Feynman–Kac formula tells us that the solution of the PPDE has a stochastic representation

$$F(t,\omega;\beta)=e^{-r(T-t)}\mathbb{E}\left[g((X^{\beta}_t)_{t\in[0,T]};\beta)|(X^{\beta}_{t\wedge s})_{s\in[0,T]}=(\omega^{\beta}_{t\wedge s})_{s\in[0,T]}\right].$$

where $F : [0,T] \times \mathcal{C}([0,T],\mathbb{R}^d) \times B \to \mathbb{R}$ and the asset follows the SDE (3.2). We denote by $F^{MC} : [0,T] \times \mathbb{R}^N \times B \to \mathbb{R}$ as the approximation of the price of the option calculated on a discretised path using 10^6 Monte Carlo samples.

$$\mathcal{E}^{\beta}_{\text{integral}} = \mathbb{E}^{N_{\text{trn}}}\left[\sum_{t_k \in \pi^c} (t_{k+1} - t_k) \cdot \left| F^{MC}\left(t_k, (X^{\beta,\pi^f}_t)_{t\in\pi^c}; \beta\right) - \mathcal{R}[F]_\theta\left(t_k, (\mathcal{X}^{\beta,\pi^f}_t)_{t\in\pi^c}; \beta\right)\right|\right].$$

Stochastic integral of the hedging strategy as a control variate

In Section 3.4.5, the discounted price is approximated by the estimator with low variance

$$M^{\text{cv},\phi}_t := M_T - \int_t^T e^{-rs'}\big(\mathcal{R}_\phi[\nabla_\omega F]\sigma\big)(s', (X^\beta_{s'\wedge s})_{s\in[0,T]}; \beta)\, dW_{s'}\,.$$

The correlation between the stochastic integral and M_T should be close to 1 in order to yield a good control variate (see [50, Chapter 4.1]) and hence a good approximation of the hedging strategy. We provide the correlation between M_T and the stochastic integral with $t = 0$, and $x_0 = 1$.

$$\rho\left(M_T, \int_0^T e^{-rs'}\big(\mathcal{R}_\phi[\nabla_\omega F]\sigma\big)(s', (X^\beta_{s'\wedge s})_{s\in[0,T]}; \beta)\, dW_{s'}\right).$$

3.5 Numerical experiments

Our Deep Learning models are implemented using Pytorch [78]. We use [79] to calculate the path signatures.

3.5.1 Black Scholes model and lookback option

Take a d-dimensional Wiener process W. We assume that we are given a symmetric, positive-definite matrix (covariance matrix) Σ and a lower triangular matrix C s.t. $\Sigma = CC^*$.[1] The risky assets will have volatilities given by σ^i. We will (abusing notation) write $\sigma^{ij} := \sigma^i C^{ij}$, when we don't need to separate the volatility of a single asset from correlations. The risky assets under the risk-neutral measure are then given by

$$dS^i_t = rS^i_t\, dt + \sigma^i S^i_t \sum_j C^{ij} dW^j_t\,. \tag{3.21}$$

Consider the lookback path-dependent payoff given by:

$$g\left((S_t)_{t\in[0,T]}\right) = \left[\sum_i S^i_T - \min_{t\in[0,T]} \sum_i S^i_t\right]_+$$

For $d = 1$, let $m_t = \inf_{0\le u\le t} S(u)$, then the price of the option at t has the following analytical form

$$F(t, (S_{t\wedge s})_{s\in[0,T]}) = S(t)\Phi(a_1) - m_t e^{-r(T-t)\Phi(a_2)} - y_t \frac{\sigma^2}{2r}\left(\Phi(-a_1) - e^{-r(T-t)}\left(\frac{m_t}{y_t}\right)^{2r/\sigma^2}\Phi(-a_3)\right)$$

[1] For such Σ we can always use Cholesky decomposition to find C.

where

$$\begin{aligned} a_1 =& \frac{\log(S(t)/m_t) + (r + \sigma^2/2)(T-t)}{\sigma\sqrt{T-t}} \\ a_2 =& a_1 - \sigma\sqrt{T-t} \\ a_3 =& a_1 - \frac{2r}{\sigma}\sqrt{T-t}, \end{aligned}$$

and Φ is the distribution function of the standard normal distribution. We can then calculate the integral error in closed form without using a Monte Carlo approximation as the ground truth.

We perform two experiments with $d = 1, 2$, $r = 5\%$, $\Sigma^{ii} = 1$, $\Sigma^{ij} = 0$ for $i \neq j$. For $d = 1$ we compare our solution to the closed form, and also we benchmark it with the same example provided in [69].

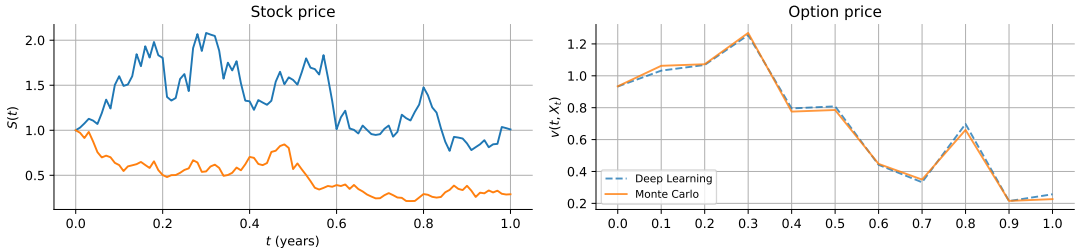
We take the maturity time $T = 1$. We divide the time interval $[0, T]$ in 100 equal time steps in the fine time discretisation, i.e. $\pi^f = \{0 = t_0 < \ldots < t_N = T, N = 1\,00\}$, and the following coarser time discretisation with 10 timesteps at which the network learns the prices: $\pi^c = \{0 = t_0 < \ldots < t_{10} = T\} \subset \pi^f$.

We train our models with batches of 200 random paths $(s^i_{t_k})_{n=0}^{N_{\text{steps}}}$ sampled from the SDE (3.21) using Euler scheme with $N_{\text{steps}} = 100$ using SGD for 50 000 iterations. The assets' initial values $s^i_{t_0}$ are sampled from a lognormal distribution $S_0 \sim \exp((\mu - 0.5\sigma^2)\tau + \sigma\sqrt{\tau}\xi)$, where $\xi \sim \mathcal{N}(0, 1)$, $\mu = 0.08, \tau = 0.1$.

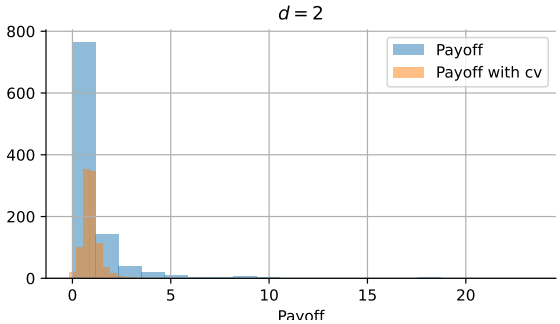
All the path signatures are calculated up to the first iterated integral (i.e. just providing the increments of the extended path), or up to the fourth iterated integral. Results are shown in Table 3.1, with algorithm 9 using the path signature truncated at depth 4 outperforming the other experiments and also the same experiment in [69]. We note that using first order path signatures is equivalent to feeding path increments to the Neural Network approximation between consecutive time steps in the coarse time discretisation. This implies that in this case we are just using the path samples taken at π^c, and in practice only the path (and not the signature) is used to calculate the solution of the PPDE. Using the signature with depth 4 implies encoding the path between consecutive points in the time discretisation.

Table 3.1: Numerical results of pricing Lookback options with Black-Scholes model

Algorithm	Sig depth	$\mathcal{E}_{\text{integral}}$	ρ
$d = 1$			
Orthogonal proj - algo. 8	1	9.52×10^{-4}	-
Martingale repr - algo. 9	1	8.06×10^{-4}	0.9706
Orthogonal proj - algo. 8	4	8.45×10^{-4}	-
Martingale repr - algo. 9	4	$\mathbf{6.19 \times 10^{-4}}$	**0.971**
PDGM [69]	-	8.8×10^{-4}	-
$d = 2$			
Orthogonal proj - algo. 8	1	3.9×10^{-3}	-
Martingale repr - algo. 9	1	3.9×10^{-3}	0.967
Orthogonal proj - algo. 8	4	2.3×10^{-3}	-
Martingale repr - algo. 9	4	$\mathbf{2.0 \times 10^{-3}}$	**0.9722**



(a) Path sample from 2-dimensional Black Scholes model (left) and Monte Carlo and Deep Learning solution using algorithm 9 and the depth-4 signature (right)



(b) Histogram of the payoff with and without the learned hedging strategy as a control variate (cv)

Figure 3.2

3.5.2 Parametric PPDE: Rough Stochastic Volatility model and European option.

Consider the following rough stochastic volatility model

$$\begin{aligned} S_t &= s_0 + \int_0^t r S_u du + \int_0^t \exp(V_u) S_u dW_u^S \\ V_t &= v_0 + \int_0^t \mathcal{K}(t-u)\left(\kappa(V_t - V_\infty)du + \eta dW_u^V\right), \quad \mathcal{K}(u) := \frac{u^{H-1/2}}{\Gamma(H+0.5)} \\ d\langle W^S, W^V\rangle_t &= \rho dt \end{aligned} \tag{3.22}$$

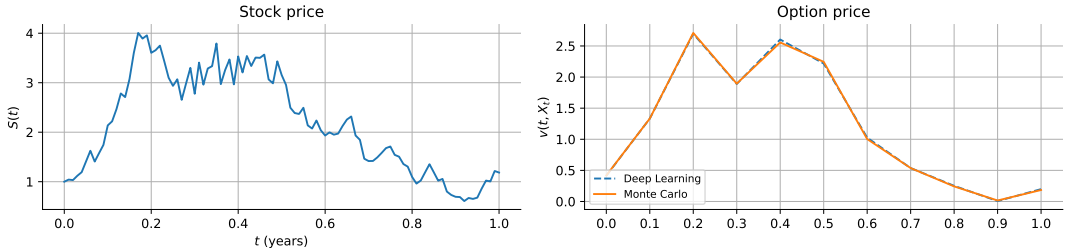
We consider the European call option $g(S_T) = (S_T - K)_+$ with maturity $T = 0.5$. We fix the parameters to $r = 0.05, s_0 = 1, v_0 = 0.04, V_\infty = 0.1, H = 0.25, \rho = 0$.

We solve the parametric PPDE with different values of the κ, η sampled from $[0.4, 0.6]$ and $[0.1, 0.3]$ respectively.

We train our models with batches of 200 random paths $(s^i_{t_k})_{n=0}^{N_{\text{steps}}}$ sampled from the SDE (3.22) using Euler scheme with $N_{\text{steps}} = 100$ using SGD for 50 000 iterations. Results are shown in Table 3.1.

Table 3.2: Numerical results of pricing European Call options under rough stochastic volatility model

Algorithm	Sig depth	$\mathcal{E}_{\text{integral}}$	ρ
$\kappa \in [0.4, 0.6], \eta \in [0.1, 0.3], K = 1$			
Orthogonal proj - algo. 8	1	1.6×10^{-3}	-
Martingale repr - algo. 9	1	1.4×10^{-3}	0.972
Orthogonal proj - algo. 8	4	9×10^{-4}	-
Martingale repr - algo. 9	4	$\mathbf{1.8 \times 10^{-4}}$	0.986



(a) Path sample from the Rough Volatility model model (left) and Monte Carlo and Deep Learning solution using algorithm 9 and the depth-4 signature (right) for $\kappa = 0.5, \eta = 0.2$

Figure 3.3

3.6 Conclusions

In this chapter we implement two numerical methods to approximate the solution of a parametric family of PPDEs given in (3.3), by using the probabilistic representation of $F(t, \omega; \beta)$ given by Feynman-Kac formula. This representation allows to tackle the problem of solving the PPDE, as pricing an option where the underlying asset follows a certain SDE in the risk neutral measure. This is the setting of our numerical experiments, where we price lookback and autocallable options using properties of the conditional expectation (algorithm 8) or the martingale representation of the price (algorithm 9). In the latter algorithm, we parametrise $\nabla_\omega F$ by a neural network, which in some models provides the hedging strategy. We combine path signatures to encode the information in ω in some finite dimensional structure together with LSTM networks to model non-anticipative functionals. We empirically show in our experiments that using tha path signature of higher order as an input to the Neural Network approximation helps in the accuracy of the algorithm, in contrast to just using the path or, equivalently, using the first order path signature.

3.A Functional Ito Calculus

In this section we define the notion of path-dependent PDE. and we review the theory that it relies on, Functional Ito calculus [63, 72]. We consider the space of continuous paths paths in $[0, T]$, $\mathcal{C}([0, T], \mathbb{R}^d)$. The space of stopped paths is the quotient space

$$\Lambda_T := \left([0, T] \times \mathcal{C}([0, T], \mathbb{R}^d)\right) / \sim$$

defined by the equivalence relationship

$$(t, \omega) \sim (t', \omega') \Leftrightarrow t = t' \text{ and } (\omega_{s\wedge t})_{s\in[0,T]} = (\omega'_{s\wedge t})_{s\in[0,T]}.$$

Consider a functional $F : \Lambda_T \to \mathbb{R}$. The continuity of F is defined with respect to the metric in Λ_T:

$$d_\infty((t, \omega), (t', \omega')) = \sup_{s\in[0,T]} |\omega_{s\wedge t} - \omega'_{s\wedge t'}| + |t - t'|$$

Furthermore, a functional F is *boundedness preserving* if for every compact set $K \subset \mathbb{R}^n$, and $\forall t_0 \in [0, T]$ there exists a constant $C > 0$ depending on K, t_0 such that

$$\forall t \in [0, t_0], \quad \forall (t, \omega) \in \Lambda_T, \quad (\omega_s)_{s\in[0,t]} \subset K \Rightarrow F(t, \omega) < C.$$

And finally we define, when the limit, exists the horizontal derivative of a functional F

$$\partial_t F(t,\omega) := \lim_{h\to 0^+} \frac{F(t+h,\omega) - F(t,\omega)}{h}$$

and the vertical derivative

$$\nabla_\omega F(t,\omega) := (\partial_i F(t,\omega), i = 1,\dots,d) \in \mathbb{R}^d \tag{3.23}$$

where

$$\partial_i F(t,\omega) := \lim_{h\to 0} \frac{F(t,(\omega_{t\wedge s})_{s\in[0,T]} + he_i \mathbf{1}_{[t,T]}) - F(t,(\omega_{t\wedge s})_{s\in[0,T]})}{h}$$

with e_i the canonical basis of $\mathbb{R}^d$. The functional $\nabla_\omega F : \Lambda_T \to \mathbb{R}^d$ is well defined in the quotioent space Λ_T, and therefore one can calculate higher order path derivatives by repeating the same operation when the limit exists.

A functional $F : \Lambda_T \to \mathbb{R}$ belongs to $\mathbb{C}^{1,2}$ if:

i) It is continuous.

ii) It is boundedness preserving.

iii) It has continuous, boundedness preserving derivatives $\partial_t F, \nabla_\omega F, \nabla^2_\omega F$.

3.B Deep Neural Networks for function approximation

3.B.1 Feedforward neural networks

We refer the reader to 2.C for a formulation of a feedforward Neural Networks as a composition of affine transformations and non-linear activation functions. More specifically, we fix $L \in \mathbb{N}$ (the number of layers), $l_k \in \mathbb{N}$, $k = 0, 1, \dots L-1$ (the size of input to layer k) and $l_L \in \mathbb{N}$ (the size of the network output). A fully connected artificial neural network is then given by $\theta = ((W_1, B_1), \dots, (W_L, B_L))$, where, for $k = 1, \dots, L$, we have real $l_{k-1} \times l_k$ matrices W_k and real l_k dimensional vectors B_k.

The artificial neural network defines a function $\mathcal{R}_\theta : \mathbb{R}^{l_0} \to \mathbb{R}^{l_L}$ given recursively, for $x_0 \in \mathbb{R}^{l_0}$, by

$$\mathcal{R}_\theta(x_0) = W_L x_{L-1} + B_L\,, \quad x_k = \mathbf{A}_{l_k}(W_k x_{k-1} + B_k)\,, k = 1,\dots,L-1\,.$$

3.B.2 Long Short Term Memory Networks

Long Short Term Memory (LSTM) networks [80] are a particular example of Recurrent Neural Networks with a specific architecture detailed below, which are useful when the input is a sequence of points

$$\{x_0, x_1, \dots, x_n\}.$$

Each element x_t of the input sequence is fed to the Recurrent Neural Network which in addition to returning an output y_t, also stores some information (or hidden state) a_t that is used to perform computations in the next step. More formally,

$$\mathcal{R}_\theta(x_t, a_{t-1}) = (y_t, a_t).$$

LSTM networks are designed to tackle the problem of exploding or vanishing gradients that plain RNN suffer from (see[81]). They do this by regulating the information carried forward by the hidden

state given each input of the sequence, using the so-called *gates*. Specifically, the operations performed for the t-th element of the sequence x_i, receiving the hidden state $a_{t-1} := (h_{t-1}, c_{t-1})$ are:

$$
\begin{aligned}
i_t &= \sigma(W_{xi}x_t + b_{xi} + W_{hi}h_{t-1} + b_{hi}) \\
f_t &= \sigma(W_{xf}x_t + b_{xf} + W_{hf}h_{t-1} + b_{hf}) \\
g_t &= \tanh(W_{xg}x_t + b_{xg} + W_{hg}h_{t-1} + b_{hg}) \\
o_t &= \sigma(W_{xo}x_t + b_{xo} + W_{ho}h_{t-1} + b_{ho}) \\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t \\
h_t &= o_t \odot \tanh(c_t)
\end{aligned}
$$

in addition, since $h_t \in (0, 1)$, we add a linear layer

$$y_t = W_{hy}h_t + b_{hy}.$$

Where $x_t \in \mathbb{R}^d, W_{x*} \in \mathbb{R}^{k \times d}, b_{x*} \in \mathbb{R}^k$, and $\odot$ is the element-wise multiplication of two vectors.

Chapter 4

Robust pricing and hedging via Neural SDEs

Modern data science techniques are opening the door to data-driven model selection mechanisms. However, most machine learning models are "black-boxes" as individual parameters do not have a meaningful interpretation. In contrast, classical risk models based on stochastic differential equations (SDEs) with fixed parametrisation are well understood. Unfortunately, the risk of using an inadequate model is hard to detect and quantify.

In this chapter, instead of choosing a fixed parametrisation for the model SDE, we aim to learn the drift and diffusion from data using overparametrised neural networks. The resulting model, called neural SDE, allows consistent calibration under both the risk-neutral and the real-world measures and is an instantiation of generative models closely linked with the theory of causal optimal transport. We demonstrate how neural SDEs allow one to find robust bounds for the prices of derivatives and the corresponding hedging strategies. Furthermore, the model can simulate market scenarios needed for assessing risk profiles and hedging strategies. We develop and analyse novel algorithms for efficient use of neural SDEs and we validate our approach with numerical experiments using both market and synthetic data.

The code used is available at `https://github.com/msabvid/robust_nsde`.

4.1 Introduction

Model uncertainty is an essential part of mathematical modelling, but is particularly acute in mathematical finance and economics where one cannot base models on well established physical laws. Until recently, these models were mostly conceived in a three step fashion: 1) gathering statistical properties of the underlying time-series or the so-called stylised facts; 2) handcrafting a parsimonious model, which would best capture the desired market characteristics without adding any needless complexity and 3) calibration and validation of the handcrafted model. Indeed, model complexity was undesirable, amongst other reasons, for increasing the computational effort required to perform in particular calibration but also pricing and risk calculations. With greater uptake of machine learning methods and greater computational power, more complex models can now be used. This is due to the fact that arguably the most complicated and computationally expensive step of calibration has been addressed. Indeed, the seminal paper [82] used

neural networks to learn the calibration map from market data directly to model parameters. Subsequently, many papers followed [32, 83, 83, 84, 85, 86, 87, 88, 33, 6]. However, these approaches focused on the calibration of a fixed parametric model, but did not address perhaps even more important issues – model selection and model uncertainty.

The approach taken in this chapter is fundamentally different in the sense that we let the data dictate the model, however we impose a strong prior on the model form. This is achieved by using SDEs for the model dynamics, but instead of choosing a fixed parametrisation for the model SDEs, we allow the drift and diffusion to be given by overparametrised neural networks. We will refer to these as neural SDEs. These are shown to not only provide a systematic framework for model selection, but also, quite remarkably, to produce robust estimates on the derivative prices. Here, the calibration and model selection are done simultaneously. In this sense, model selection is data-driven. Since the neural SDE model is overparametrised, there is a large pool of possible models and the training algorithm selects a model. Unlike in handcrafted models, individual parameters do not carry any meaning. This makes it hard to argue why one model is better than another. Hence, the ability to efficiently compute interval estimators, which algorithms in this paper provide, is critical.

4.1.1 Brief overview of prior non-parametric methods

Non-parametric calibration of SDE coefficients dates back to at least [89], where the functional form of the volatility parameter was obtained by minimising a cost functional given by the Kullback-Leibler divergence and involves solving a high-dimensional system of parabolic PDEs. Similar to our approach, their algorithm can be used to interpolate between implied volatilities of traded options in an arbitrage-free way. A more modern approach was considered in [90], where time continuous martingale optimal transport was applied in order to interpolate between initial and final asset price distributions, with an aim to calibrate the local volatility function in a non-parametric way. Similar idea was considered in [91], where semi-martingale optimal transport was used instead. The cost function given by the prices of European options was optimised to obtain the functional form of a LSV model. The resulting fully non-linear HJB equations stemming from the dual of the convex optimisation problem then had to be solved numerically. Following this, [92] consider calibration via semi-martingale optimal transport for path-dependent derivatives, such as barrier and lookback options, which accordingly translate to solving a path-dependent PDE.

In parallel to this work, in [71], the authors considered local stochastic volatility models with the leverage function approximated with a neural network. Their model can be seen as an example of a neural SDE.

4.1.2 Problem overview

Let us now consider a probability space $(\Omega, \mathcal{F}, (\mathcal{F}_t)_{t\in[0,T]}, \mathbb{P})$ and a random variable $\Psi \in L^2(\mathcal{F}_T)$ that represents the discounted payoff of a derivative (typically illiquid and path-dependent). The problem of calculating a market consistent price of a financial derivative can be seen as equivalent to finding a map that takes market data (e.g., prices of underlying assets, interest rates, prices of liquid options) and returns a no-arbitrage price of the derivative. Typically an Itô process $(X_t^\theta)_{t\in[0,T]}$, with parameters $\theta \in \mathbb{R}^q$ has been the main component used in constructing such a pricing function. Such parametric model induces a martingale probability measure, denoted by $\mathbb{Q}(\theta)$, which is then used to compute no-arbitrage prices of derivatives.

In our work, the market data (input data) is represented by payoffs $\{\Phi_i\}_{i=1}^M$ of liquid derivatives, and

their corresponding market prices $\{\mathfrak{p}(\Phi_i)\}_{i=1}^M$. We will assume throughout that this price set is free of arbitrage. To make the model $\mathbb{Q}(\theta)$ consistent with market prices, one seeks parameters θ^* such that the difference between $\mathfrak{p}(\Phi_i)$ and $\mathbb{E}^{\mathbb{Q}(\theta^*)}[\Phi_i]$ is minimised for all $i = 1, \ldots, M$ (w.r.t. some metric). If for all $i = 1, \ldots, M$ we have $\mathfrak{p}(\Phi_i) = \mathbb{E}^{\mathbb{Q}(\theta^*)}[\Phi_i]$, then we will say the model is consistent with the market data (perfectly calibrated). There may be infinitely many models that are consistent with the market. This is called Knightian uncertainty [93, 94].

Let $\mathcal{M}$ be the set of all martingale measures / models that are perfectly calibrated to market inputs. In the robust finance paradigm, see [95, 96], one takes a conservative approach and instead of computing a single price (that corresponds to a model from $\mathcal{M}$) one computes the price interval $(\inf_{\mathbb{Q}\in\mathcal{M}} \mathbb{E}^{\mathbb{Q}}[\Psi], \sup_{\mathbb{Q}\in\mathcal{M}} \mathbb{E}^{\mathbb{Q}}[\Psi])$. The bounds can be computed using tools from martingale optimal transport, which also, through the dual representation, yield the corresponding super- and sub- hedging strategies, [97]. Without imposing further constraints, the class of all calibrated models $\mathcal{M}$ might be too large and the corresponding bounds too wide to be of practical use [98]. See, however, an effort to incorporate further market information to tighten the pricing interval, [99, 100]. Another shortcoming of working with the entire class of calibrated models $\mathcal{M}$ is that, in general, it is not clear how to obtain a practical/explicit model out of the measures that yield the price bounds. For example, such explicit models are useful when one wants consistently calibrate under a pricing measure $\mathbb{Q}$ and the real-world measure $\mathbb{P}$ as needed for risk estimation and stress testing [101, 102] or learn hedging strategies in the presence of transaction costs and an illiquidity constraints [103].

4.1.3 Neural SDEs

Fix $T > 0$ and for simplicity assume constant interest rate $r \in \mathbb{R}$. Consider a parameter space $\Theta = \Theta^b \times \Theta^\sigma \subseteq \mathbb{R}^q$ and parametric functions $b : \mathbb{R}^{d+1} \times \Theta^b \to \mathbb{R}^d$ and $\sigma : \mathbb{R}^{d+1} \times \Theta^\sigma \to \mathbb{R}^{d\times n}$. Let $(W_t)_{t\in[0,T]}$ be an n-dimensional Brownian motion supported on $(\Omega, \mathcal{F}, (\mathcal{F}_t)_{t\in[0,T]}, \mathbb{Q})$ so that $\mathbb{Q}$ is the Wiener measure and $\Omega = C([0,T]; \mathbb{R}^n)$. We consider the following parametric SDE

$$dX_t^\theta = b(t, X_t^\theta, \theta)dt + \sigma(t, X_t^\theta, \theta)dW_t\,, \;\; X_0^\theta = x_0\,, \;\; t \in [0,T]\,. \tag{4.1}$$

By imposing suitable conditions on the coefficients (b, σ), we know that the unique solution to (4.1) exists, [104, Chapter 2]. These conditions can be satisfied by neural networks, e.g. by using suitably regular activation functions and by applying weight clipping or by taking regular and uniformly bounded activation functions. Throughout the paper we assume that (4.1) has a unique strong solution. We split X^θ, which is the entire stochastic model, into traded assets and non-tradable components. Let $X^\theta = (S^\theta, V^\theta)$, where S are the traded assets and V are the components that are not traded. We will assume that for all $t \in [0,T]$, $x = (s,v) \in \mathbb{R}^d$ and $\theta \in \Theta$ that

$$b(t,(s,v),\theta) = \big(rs, b^V(t,(s,v),\theta)\big) \in \mathbb{R}^d \;\text{ and }\; \sigma(t,(s,v),\theta) = \big(\sigma^S(t,(s,v),\theta), \sigma^V(t,(s,v),\theta)\big)\,.$$

Then we can write (4.1) as

$$\begin{aligned} dS_t^\theta &= rS_t^\theta\, dt + \sigma^S(t, X_t^\theta, \theta)\, dW_t\,, \\ dV_t^\theta &= b^V(t, X_t^\theta, \theta)\, dt + \sigma^V(t, X_t^\theta, \theta)\, dW_t\,, \\ X_t^\theta &= (S_t^\theta, V_t^\theta)\,. \end{aligned} \tag{4.2}$$

Observe that σ^S and σ^V, set to take values in the space of appropriately shaped matrices, encode arbitrary correlation structures between the traded assets and the non-tradable components. Moreover,

we immediately see that $(e^{-rt}S_t)_{t\in[0,T]}$ is a (local) martingale and thus the model is free of arbitrage.

In a situation when (b,σ) are defined to be neural networks (see Appendix 4.F), we call the SDE (4.1) a neural SDE and we denote by $\mathcal{M}^{\text{nsde}}(\theta)$ the class of all solutions to (4.1) for b,σ in the Neural Networks class defined in Section 2.C. Note that due to universal approximation property of neural networks, see [105, 106, 107], $\mathcal{M}^{\text{nsde}}(\theta)$ contains large class of SDEs solutions. Furthermore, neural networks can be efficiently trained with stochastic gradient descent methods and hence one can seek calibrated models in $\mathcal{M}^{\text{nsde}}(\theta)$. Finally, neural SDE integrate black-box neural network type models with the known and well studied SDE models. One consequence of that is that one can: a) consistently calibrate these under the risk neutral measure as well as the real-world measure; b) easily integrate additional market information, e.g constraints on realised variance; c) verify martingale property. We want to remark that for simplicity we work in Markovian setting, but one could consider neural SDEs with path-dependent coefficients and/or consider more general noise processes. We postpone the analysis of these cases to a follow up paper.

We will denote the law of X^θ on $C([0,T];\mathbb{R}^d)$ by $\mathbb{Q}(\theta) := \mathcal{L}((X^\theta_t)_{t\in[0,T]}) := \mathbb{Q}\circ(X^\theta)^{-1}$. Consider a payoff function for a path dependent derivative $\phi : C([0,T];\mathbb{R}^d) \to \mathbb{R}$. It is well known that the expectation under $\mathbb{Q}$ and under the push-forward measure $\mathbb{Q}\circ(X^\theta)^{-1} = \mathbb{Q}(\theta)$ satisfy

$$\mathbb{E}^{\mathbb{Q}}\Big[\phi(X^\theta)\Big] = \int_{C([0,T];\mathbb{R}^n)} \phi(X^\theta(\omega))\,\mathbb{Q}(d\omega) = \int_{C([0,T];\mathbb{R}^d)} \phi(\tilde\omega)\,\big(\mathbb{Q}\circ(X^\theta)^{-1}\big)(d\tilde\omega) = \mathbb{E}^{\mathbb{Q}(\theta)}[\phi]\,. \tag{4.3}$$

Thus, for $\phi : C([0,T],\mathbb{R}^d) \to \mathbb{R}$ we see that we can either work with $\mathbb{Q}$, with W as the canonical process and the random variable $\Phi = \phi(X^\theta) \in L^2(\mathcal{F}_T,\mathbb{Q})$ or equivalently with $\mathbb{Q}(\theta)$, X^θ as the canonical process and the random variable $\phi \in L^2(\mathcal{F}_T,\mathbb{Q}(\theta))$. In the rest of the paper, we will be abusing notation by writing Φ as both the payoff and the mapping $C([0,T],\mathbb{R}^d) \to \mathbb{R}$. Later, we will also consider an empirical approximation of $\mathbb{Q}(\theta)$, based on N i.i.d. copies of (4.1), which will be denoted by $\mathbb{Q}^N(\theta)$. See Section 4.2.1.

Given a loss function $\ell : \mathbb{R}\times\mathbb{R} \to \mathbb{R}^+$, the search for a calibrated model can be written as

$$\theta^* \in \arg\min_{\theta\in\Theta} \sum_{i=1}^M \ell(\mathbb{E}^{\mathbb{Q}(\theta)}[\Phi_i], \mathfrak{p}(\Phi_i))\,.$$

To consistently extend the calibration to the real world measure, assume that the risk-neutral calibration described above has been completed and assume that we are given some statistical facts, e.g. moments or other distributional properties, that the price process (or the non tradable components) should satisfy. Let $\zeta : [0,T]\times\mathbb{R}^d\times\mathbb{R}^q \to \mathbb{R}^n$ be another parametric function (e.g. a neural network). We extend the parameter space to $\Theta = \Theta^b\times\Theta^\sigma\times\Theta^\zeta \subseteq \mathbb{R}^q$ and let

$$\begin{aligned} b^{S,\mathbb{P}}(t,X^\theta_t,\theta) &:= rS^\theta_t + \sigma^S(t,X^\theta_t,\theta)\zeta(t,X^\theta_t,\theta)\,,\\ b^{V,\mathbb{P}}(t,X^\theta_t,\theta) &:= b^V(t,X^\theta_t,\theta) + \sigma^V(t,X^\theta_t,\theta)\zeta(t,X^\theta_t,\theta)\,. \end{aligned}$$

We now define a real-world measure $\mathbb{P}(\theta)$ via the Radon–Nikodym derivative

$$\frac{d\mathbb{P}(\theta)}{d\mathbb{Q}(\theta)} := \exp\left(-\int_0^T \zeta(t,X^\theta_t,\theta)\,dW_t - \frac{1}{2}\int_0^T |\zeta(t,X^\theta_t,\theta)|^2\,dt\right)\,.$$

Under appropriate assumptions on ζ (e.g. bounded) the measure $\mathbb{P}(\theta)$ is a probability measure and by

using the Girsanov theorem we can find a Brownian motion $(W_t^{\mathbb{P}(\theta)})_{t\in[0,T]}$ such that

$$\begin{aligned} dS_t^\theta &= b^{S,\mathbb{P}}(t, X_t^\theta, \theta)\, dt + \sigma^S(t, X_t^\theta, \theta)\, dW_t^{\mathbb{P}(\theta)}\,, \\ dV_t^\theta &= b^{V,\mathbb{P}}(t, X_t^\theta, \theta)\, dt + \sigma^V(t, X_t^\theta, \theta)\, dW_t^{\mathbb{P}(\theta)}\,. \end{aligned} \tag{4.4}$$

This is now the neural SDE model in the real-world measure $\mathbb{P}(\theta)$ and accordingly one would like use market data to seek ζ.

This overparametrised form of market price of risk is hard to interpret. This is undesirable from the point of application. However it is still a familiar object compared to what one would obtain by trying to build a black-box path generator based solely on neural network. To have a form that can be interpreted one could incorporate a prior real-world measure. For example, one could minimise some distance (e.g. total variation or KL divergence) between the prior and $\mathbb{P}(\theta)$. We have not tested calibrations to the real-world measure in this paper and leave this for future work.

Let $\mathbb{P}^{\text{market}}$ denote the empirical distribution of market data and $(\mathbb{E}^{\mathbb{P}^{\text{market}}}[\mathcal{S}_i])_{i=1}^{\tilde{M}}$ be a corresponding set of statistics one aims to match. These might be, e.g. the autocorrelation function, realised variance or moment generating functions. The calibration to the real-world measure, with $(b^V, \sigma^V, \sigma^S)$ fixed, consists of finding θ^* such that

$$\theta^* \in \arg\min_{\theta\in\Theta} \sum_{i=1}^{\tilde{M}} \ell(\mathbb{E}^{\mathbb{P}(\theta)}[\mathcal{S}_i], \mathbb{E}^{\mathbb{P}^{\text{market}}}[\mathcal{S}_i])\,.$$

But in fact we can write

$$\mathbb{E}^{\mathbb{P}(\theta)}[\mathcal{S}_i] = \mathbb{E}^{\mathbb{Q}(\theta)}\left[\mathcal{S}_i \frac{d\mathbb{P}(\theta)}{d\mathbb{Q}(\theta)}\right].$$

Thus we see that in this framework there needs to be no distinction between a derivative price Φ_i and a real-world statistic $\mathbb{E}^{\mathbb{P}^{\text{market}}}[\mathcal{S}_i]$. Hence, from now on we will write only about risk-neutral calibrations bearing in mind that methodologically this leads to no loss of generality. An alternative approach recently introduced in [108], avoids the calculation of the Radon-Nikodym derivative by first learning the diffusion coefficient under $\mathbb{Q}(\theta)$ and then, with $\sigma(t, X_t^\theta, \theta)$ fixed, the Neural SDE is viewed as a generative model and the drift coefficient is learned by fitting to $\mathbb{P}^{\text{market}}$.

Let us connect neural SDEs to the concept of generative modelling, see [109, 110]. Let $\mathbb{Q}^{\text{market}} \in \mathcal{M}$ be the true martingale measure (so by definition all liquid derivatives are perfectly calibrated under this measure, i.e., $\mathbb{E}^{\mathbb{Q}^{\text{market}}}[\Phi_i] = \mathfrak{p}(\Phi_i)$ for all $i = 1, \dots, M$). We know that when (4.1) admits a strong solution, then for any $\theta \in \Theta$ there exists a measurable map $G^\theta : \mathbb{R}^d \times C([0,T];\mathbb{R}^n) \to C([0,T];\mathbb{R}^d)$ such that $X^\theta = G^\theta(x_0, W)$, see [111, Corollary 3.23]. Hence, one can view (4.1) as a generative model that maps μ, the joint distribution of X_0 on $\mathbb{R}^d$ and the Wiener measure on $C([0,T];\mathbb{R}^n)$ into $\mathbb{Q}^\theta = (G_t^\theta)_{\#}\mu$. We see that by construction G is a causal transport map, i.e., a transport map that is adapted to the filtration $(\mathcal{F}_t)_{t\in[0,T]}$, see also [112, 113].

One then seeks θ^* such that $G^{\theta^*}_{\#}\mu$ is a good approximation of $\mathbb{Q}^{\text{market}}$ with respect to user specified metric. In this paper we work with

$$D(G^\theta_{\#}\mu, \mathbb{Q}^{\text{market}}) := \sum_{i=1}^{M} \ell\left(\int_{C([0,T],\mathbb{R}^d)} \Phi_i(\omega)(G^\theta_{\#}\mu)(d\omega), \int_{C([0,T],\mathbb{R}^d)} \Phi_i(\omega)\mathbb{Q}^{\text{market}}(d\omega)\right).$$

As we shall see in Sections 4.C.1 and 4.5.1 there are many neural SDE models that can be well calibrated to market data and produce significantly different prices for derivatives that were not part of the calibration data. In practice, these would be illiquid derivatives where we require a model to obtain prices. Therefore,

we compute price intervals for illiquid derivatives within the class of calibrated neural SDE models. To be more precise we compute

$$\inf_{\theta}\left\{\mathbb{E}^{\mathbb{Q}(\theta)}[\Psi]\ :\ D(G(\theta)_{\#}\mu_0,\mathbb{Q}^{\text{market}})=0\right\},\quad \sup_{\theta}\left\{\mathbb{E}^{\mathbb{Q}(\theta)}[\Psi]\ :\ D(G(\theta)_{\#}\mu_0,\mathbb{Q}^{\text{market}})=0\right\}.$$

We solve the above constrained optimisation problem by penalisation. See [114] for related ideas.

4.1.4 Key conclusions and methodological contributions

The results in this chapter presented in ensuing sections, lead to the following conclusions.

i) Neural SDEs provide a systematic framework for model selection and produce robust estimates on the derivative prices. The calibration and model selection are done simultaneously and thus the model selection is data-driven.

ii) With neural SDEs, the modelling choices one makes are: network architecture, structure of a neural SDE (e.g. traded and non-traded assets), training methods and data. For classical handcrafted models the choice of the algorithm for calibrating parameters has not been considered as a part of modelling choice, but for machine learning this is one of the key components. Moreover, in Section 4.5, where we show how the change in initialisation of stochastic gradient method used for training leads to different prices of illiquid options, thus providing one way of obtaining price bounds. Furthermore, even for a basic local volatility model that is unique for continuum of strikes and maturities, a neural SDE produces ranges of prices of illiquid derivatives when calibrated to finite data sets.

iii) The above optimisation problem is not convex. Nonetheless, empirical experiments in Sections 4.5.1 and 4.C.1 demonstrate that gradient descent methods, used to minimise the loss functional D, converge to a set of parameters for which the implied volatilities closely match market implied volatilities as most of the calibrated surface is within the bid-ask spread. Theoretical framework for analysing such algorithms is being developed in [115].

iv) By augmenting classical risk models with modern machine learning approaches we are able to benefit from expressibility of neural networks while staying within realm of classical models, well understood by traders, risk managers and regulators. This mitigates, to some extent, the concerns that regulators have around use of black-box solutions to manage financial risk. Finally, while our focus here is on SDE type models, the devised framework naturally extends to time-series type models.

The main methodological contributions of this work are as follows.

i) By leveraging martingale representation theorem we developed an efficient Monte Carlo based method that simultaneously learns the model and the corresponding hedging strategy.

ii) The calibration problem does not fit into the classical framework of stochastic gradient algorithms, as the mini-batches of the gradient of the cost function are biased. We provide an analysis of the bias and show how the inclusion of hedging strategies in training mitigates this bias.

iii) We devise a novel, memory efficient randomised training procedure. The algorithm allows us to keep memory requirements constant, independently of the number of neural networks in the neural SDE. This is critical to efficiently calibrate to path dependent contingent claims. We provide a theoretical analysis of our method in Section 4.4 and numerical experiment supporting the claims in Section 4.5.

The chapter is organised as follows. In Section 4.2 we outline the exact optimisation problem, introduce a deep neural network control variate (or hedging strategy), address the process of calibration to single/multiple option maturities and state the exact algorithms. In Section 4.3 we analyse the bias in Algorithms 11 and 12. In Section 4.4 we show that the novel, memory-efficient, drop-out-like training procedure for path-dependent derivatives does not introduce any bias to the new estimator. Finally, the performance of the neural Local Volatility and Local Stochastic Volatility models is presented in Section 4.5. Some of the proofs and more detailed results from numerical experiments are relegated to the Appendix.

The code used is available at `github.com/msabvid/robust_nsde`.

4.2 Robust pricing and hedging

Let $\ell : \mathbb{R} \times \mathbb{R} \to [0, \infty)$ be a convex loss function such that $\ell(x, y) = 0$ if and only if $x = y$. For example we can take $\ell(x, y) = |x - y|^2$. Given ℓ, our aim is to solve the following optimisation problems:

i) Find model parameters θ^* such that model prices match market prices:

$$\theta^* \in \arg\min_{\theta\in\Theta} \sum_{i=1}^{M} \ell(\mathbb{E}^{\mathbb{Q}(\theta)}[\Phi_i], \mathfrak{p}(\Phi_i)) \,. \tag{4.5}$$

This is equivalent to finding some θ^* such that $\sum_{i=1}^{M} \ell(\mathbb{E}^{\mathbb{Q}(\theta^*)}[\Phi_i], \mathfrak{p}(\Phi_i)) = 0$. This is due to inherent overparametrisation of neural SDEs and the fact that $\ell(x, y) = 0$ if and only if $x = y$.

ii) Find model parameters $\theta^{l,*}$ and $\theta^{u,*}$, which provide robust arbitrage-free price bounds for an illiquid derivative, subject to the available market data:

$$\begin{aligned} \theta^{l,*} &\in \arg\min_{\theta\in\Theta} \mathbb{E}^{\mathbb{Q}(\theta)}[\Psi] \,, \quad \text{subject to } \sum_{i=1}^{M} \ell(\mathbb{E}^{\mathbb{Q}(\theta)}[\Phi_i], \mathfrak{p}(\Phi_i)) = 0 \,, \\ \theta^{u,*} &\in \arg\max_{\theta\in\Theta} \mathbb{E}^{\mathbb{Q}(\theta)}[\Psi], \quad \text{subject to } \sum_{i=1}^{M} \ell(\mathbb{E}^{\mathbb{Q}(\theta)}[\Phi_i], \mathfrak{p}(\Phi_i)) = 0 \,. \end{aligned} \tag{4.6}$$

The no-arbitrage price of Ψ, computed over the class of neural SDEs, then lies in the interval $\left[\mathbb{E}^{\mathbb{Q}(\theta^{l,*})}, \mathbb{E}^{\mathbb{Q}(\theta^{u,*})}\right]$.

4.2.1 Learning hedging strategy as a control variate

A starting point in the derivation of the practical algorithm is to estimate $\mathbb{E}^{\mathbb{Q}(\theta)}[\Phi]$ using a Monte Carlo estimator. Consider $(X^{i,\theta})_{i=1}^{N}$, i.e., N i.i.d. copies of (4.1) and let $\mathbb{Q}^N(\theta) := \frac{1}{N}\sum_{i=1}^{N} \delta_{X^{i,\theta}}$ be the empirical approximation of $\mathbb{Q}(\theta)$. Due to the Law of Large Numbers, $\mathbb{E}^{\mathbb{Q}^N(\theta)}[\Phi]$ converges to $\mathbb{E}^{\mathbb{Q}(\theta)}[\Phi]$ in probability. Moreover, the asymptotic confidence interval implied by the Central Limit Theorem for real-valued random variables is given by

$$\mathbb{Q}(\theta)\left(\mathbb{E}^{\mathbb{Q}(\theta)}[\Phi] \in \left[\mathbb{E}^{\mathbb{Q}^N(\theta)}[\Phi] - z_{\alpha/2}\frac{\sigma}{\sqrt{N}}, \mathbb{E}^{\mathbb{Q}^N(\theta)}[\Phi] + z_{\alpha/2}\frac{\sigma}{\sqrt{N}}\right]\right) \to 1 \text{ as } N \to \infty \,,$$

where $\sigma = \sqrt{\mathbb{V}\mathrm{ar}[\Phi]}$ and $z_{\alpha/2}$ is such that $1 - \mathrm{CDF}_Z(z_{\alpha/2}) = \alpha/2$ with Z the standard normal distribution. We see that by increasing N, we reduce the width of the above confidence intervals, but this increases the overall computational cost. A better strategy is to find a good control variate, i.e., we seek a

random variable Φ^{cv} such that:

$$\mathbb{E}^{\mathbb{Q}(\theta)}[\Phi^{\text{cv}}] = \mathbb{E}^{\mathbb{Q}(\theta)}[\Phi] \quad \text{and} \quad \mathbb{V}\text{ar}^{\mathbb{Q}(\theta)}[\Phi^{\text{cv}}] < \mathbb{V}\text{ar}^{\mathbb{Q}(\theta)}[\Phi]\,. \tag{4.7}$$

In the following we construct Φ^{cv} using a hedging strategy. A similar approach was recently developed in [6] in the context of pricing and hedging with deep neural networks.

Martingale representation theorem (see for example Th. 14.5.1 in [51]) provides a general methodology for finding Monte Carlo estimators with the above stated properties (4.7).

Since we are assuming that $\mathbb{E}^{\mathbb{Q}(\theta)}[|\Phi|^2] < \infty$, there exists a unique process $Z = (Z_t)_{t\in[0,T]}$ adapted to the filtration $(\mathcal{F}_t)_{t\in[0,T]}$ with $\mathbb{E}^{\mathbb{Q}(\theta)}\left[\int_0^T |Z_s|^2\,ds\right] < \infty$ such that

$$\mathbb{E}^{\mathbb{Q}(\theta)}[\Phi|\mathcal{F}_0] = \Phi - \int_0^T Z_s\,dW_s\,.$$

Define

$$\Phi^{\text{cv}} := \Phi - \int_0^T Z_s\,dW_s\,,$$

and note that for any fixed $\theta \in \Theta$ we have

$$\mathbb{E}^{\mathbb{Q}(\theta)}[\Phi^{\text{cv}}|\mathcal{F}_0] = \mathbb{E}^{\mathbb{Q}(\theta)}[\Phi|\mathcal{F}_0] \quad \text{and} \quad \mathbb{V}\text{ar}^{\mathbb{Q}(\theta)}[\Phi^{\text{cv}}|\mathcal{F}_0] = 0\,.$$

The process Z has more explicit representations using the corresponding (possibly path dependent) backward Kolomogorov equation or Bismut–Elworthy–Li formula. Both approaches require further approximation, see [6] and [12]. Here, this approximation will be provided by an additional neural network.

Consider now a neural network $\mathfrak{h} : [0,T] \times C([0,T],\mathbb{R}^d) \times \mathbb{R}^q \to \mathbb{R}^d$ with parameters $\xi \in \mathbb{R}^{q'}$ with $q' \in \mathbb{N}$ and define the following learning task, in which θ (the parameters on the neural SDE model) is fixed:

Find

$$\xi^* \in \arg\min_{\xi} \mathbb{V}\text{ar}^{\mathbb{Q}}\left[\phi((X_t^\theta)_{t\in[0,T]}) - \int_0^T \mathfrak{h}(s,(X^\theta_{s\wedge t})_{t\in[0,T]},\xi)dW_s\middle|\mathcal{F}_0\right]\,. \tag{4.8}$$

Recall that using (4.3) and $\mathbb{Q}\circ(X^\theta)^{-1} = \mathbb{Q}(\theta)$, we know that for $\phi : C([0,T],\mathbb{R}^d) \to \mathbb{R}$ we can either work with $\mathbb{Q}$, with W as the canonical process and the random variable $\Phi = \phi(X^\theta) \in L^2(\mathcal{F}_T,\mathbb{Q})$ or equivalently with $\mathbb{Q}(\theta)$, X^θ as the canonical process and the random variable $\phi \in L^2(\mathcal{F}_T,\mathbb{Q}(\theta))$.

Note that ξ^* depends on θ, which means that the control variate we found is good for the particular value θ, but not necessarily for other choices of θ. In a similar manner one can derive Ψ^{cv} for the payoff of the illiquid derivative for which we seek the robust price bounds. Then (4.6) can be restated as

$$\begin{aligned}
\theta^{l,*} &\in \arg\min_{\theta\in\Theta} \mathbb{E}^{\mathbb{Q}(\theta)}[\Psi^{\text{cv}}]\,, \quad \text{subject to} \quad \sum_{i=1}^M \ell(\mathbb{E}^{\mathbb{Q}(\theta)}[\Phi_i^{\text{cv}}],\mathfrak{p}(\Phi_i)) = 0\,,\\
\theta^{u,*} &\in \arg\max_{\theta\in\Theta} \mathbb{E}^{\mathbb{Q}(\theta)}[\Psi^{\text{cv}}]\,, \quad \text{subject to} \quad \sum_{i=1}^M \ell(\mathbb{E}^{\mathbb{Q}(\theta)}[\Phi_i^{\text{cv}}],\mathfrak{p}(\Phi_i)) = 0\,.
\end{aligned} \tag{4.9}$$

The learning problem (4.9) is better than (4.6) from the point of view of algorithmic implementation, as it will enjoy lower Monte Carlo variance and hence require simulation of fewer paths of the neural SDE in each step of the stochastic gradient algorithm. Furthermore, when using (4.9) we learn a (possibly

abstract) hedging strategy for trading in the underlying asset to replicate the derivative payoff. Since the market may be incomplete, this abstract hedging strategy may not be usable in practice. More precisely, since the process X^θ will contain tradable as well as non-tradable assets, the control variate for the latter has to be adapted by either performing a projection or deriving a strategy for the corresponding tradable instrument.

To deduce a real hedging strategy recall that $X^\theta = (S^\theta, V^\theta)$ with S^θ being the tradable assets and V^θ the non-tradable components. Decompose the abstract hedging strategy as $\mathfrak{h} = (\mathfrak{h}^S, \mathfrak{h}^V)$. Let $\bar{S}^\theta_t := e^{-rt} S^\theta_t$ and note that due to (4.2) we have

$$d(\bar{S}^\theta_t) = e^{-rt}\sigma^S(t, X^\theta_t, \theta)\, dW_t\,.$$

An alternative approach to (4.8), possibly yielding a better hedge, but worse variance reduction would be to consider finding

$$\xi^* \in \arg\min_{\xi} \mathbb{V}\mathrm{ar}^{\mathbb{Q}}\left[\Phi((X^\theta_t)_{t\in[0,T]}) - \int_0^T \bar{\mathfrak{h}}(r, (X^\theta_{r\wedge t})_{t\in[0,T]}, \xi)\, d\bar{S}^\theta_r \middle| \mathcal{F}_0\right] \tag{4.10}$$

for some other neural network $\bar{\mathfrak{h}}$. This is the version we present in Algorithms 11 and 12.

4.2.2 Time discretisation

In order to implement (4.9), we define a partition π of $[0,T]$ as $\pi := \{t_0 = 0, t_1, \ldots, t_{N_{\text{steps}}} = T\}$. We first approximate the stochastic integral in (4.8) with the appropriate Riemann sum. Depending on the choice of the neural network architecture approximating b and σ in the neural SDE (4.1), we may have coefficients, which grow super-linearly as a function of x. In such a case, the moments of the classical Euler scheme are known to blow up in finite time, see [116], even if the moments of the solution to the SDE are finite. In order to have a generic algorithm that works for a wide class of activation functions avoiding blow ups of moments of the simulated paths during training, we apply a tamed Euler method, see [117, 118]. We found that the tamed Euler scheme works better even if the activation functions only have linear growth as the training gradient descent may choose large weights leading to potential instability of a classical Euler scheme. The tamed Euler scheme is given by

$$X^{\pi,\theta}_{t_{k+1}} = X^{\pi,\theta}_{t_k} + \frac{b(t_k, X^{\pi,\theta}_{t_k}, \theta)}{1 + |b(t_k, X^{\pi,\theta}_{t_k}, \theta)|\sqrt{\Delta t_k}}\Delta t_k + \frac{\sigma(t_k, X^{\pi,\theta}_{t_k}, \theta)}{1 + |\sigma(t_k, X^{\pi,\theta}_{t_k}, \theta)|\sqrt{\Delta t_k}}\Delta W_{t_{k+1}}\,, \tag{4.11}$$

with $\Delta t_k = t_{k+1} - t_k$ and $\Delta W_{t_{k+1}} = W_{t_{k+1}} - W_{t_k}$.

4.2.3 Algorithms

We now present the algorithm to calibrate the neural SDE (4.1) to market prices of derivatives (Algorithm 11) and the algorithm to find robust price bounds for an illiquid derivative (Algorithm 12). Note that during training we aim to calibrate the SDE (4.1) and at the same time adapt the abstract hedging strategy to minimise the variance (4.8). In instances where we say we "freeze parameters", what we do is detach the computational graph for the purposes of backward propagation so that the gradient is calculated only with respect to the "unfrozen" parameters. To be exact: we use the `detach()` method in PyTorch.

Algorithms 11, 12 use a frozen copy of θ, denoted by $\tilde{\theta}$, in the calculation of the control variate to perform the minimisation (4.12) for fixed $(\xi_j)_{j=1,\ldots,M}$. The reason for this is that the approximation of

the stochastic integral in (4.12) is solely used to reduce the variance of the gradient calculated in each gradient descent step:

$$\frac{\partial}{\partial\theta}\left(\mathbb{E}^{\mathbb{Q}^N}\left[\Phi_j\left(X^{\pi,\theta}\right)-\sum_{k=0}^{N_{\text{steps}}-1}\bar{\mathfrak{h}}(t_k,X_{t_k}^{\pi,\tilde{\theta}},\xi_j)\Delta\bar{S}_{t_k}^{\pi,\tilde{\theta}}\right]-\mathfrak{p}(\Phi_j)\right)^2$$
$$=2\left(\underbrace{\mathbb{E}^{\mathbb{Q}^N}\left[\Phi_j\left(X^{\pi,\theta}\right)-\sum_{k=0}^{N_{\text{steps}}-1}\bar{\mathfrak{h}}(t_k,X_{t_k}^{\pi,\tilde{\theta}},\xi_j)\Delta\bar{S}_{t_k}^{\pi,\tilde{\theta}}\right]}_{\text{Low variance}}-\mathfrak{p}(\Phi_j)\right)\mathbb{E}^{\mathbb{Q}^N}\left[\frac{\partial\Phi}{\partial x}(X^{\pi,\theta})\frac{\partial X^{\pi,\theta}}{\partial\theta}\right].$$

By excluding the calculation of the gradient of $\bar{\mathfrak{h}}$ from the algorithm we protect against the possibility of optimising θ using an inaccurate approximation of the hedging strategy, and we additionally speed up the backpropagation in each gradient descent step by making the computational graph smaller.

In both Algorithms 11 and 12 as well as in the numerical experiments, we use the squared error for the nested loss function: $\ell(x,y)=|x-y|^2$. Furthermore, the calibration of the neural SDE to market derivative prices with robust price bounds for an illiquid derivative in Algorithm 12 is done using a constrained optimisation using the method of Augmented Lagrangian [119], where the Lagrange multiplier parameters are updated every $N^{\text{al}}\in\mathbb{N}$ iterations with the update rule specified in Algorithm 13.

The augmented Lagrangian optimisation allows not only to fit the parameters θ of the Neural SDEs coefficients so that the illiquid option price is maximised (or minimised) but also to optimise the Lagrange multipliers so that the liquid option prices are perfectly fit.

Algorithm 11 Calibration to market European option prices for one maturity

Input:
$\pi\coloneqq\{t_0,t_1,\ldots,t_n\}$ time grid
$(\Phi)_{i=1}^M$ vector of option payoffs
$(\mathfrak{p}(\Phi_j))_{i=1}^M$ market option price vector
Initialisation:
$\theta\in\Theta$ for neural SDE parameters.
$\xi\in\Xi$ for control variate approximation.
for epoch $:1:N_{\text{epochs}}$ **do**
Generate N paths $(X_{t_n}^{\pi,\theta,i})_{n=0}^{N_{\text{steps}}}\coloneqq(S_{t_n}^{\pi,\theta,i},V_{t_n}^{\pi,\theta,i})_{n=0}^{N_{\text{steps}}}$, $i=1,\ldots,N$ using the tamed Euler scheme from (4.11). Let $\tilde{\theta}:=\theta$.
i) Freeze ξ and $\tilde{\theta}$, use Adam (see [120]) to update θ, where

$$\theta\leftarrow\arg\min_{\theta}\sum_{j=1}^{M}\left(\mathbb{E}^{\mathbb{Q}^N}\left[\Phi_j\left(X^{\pi,\theta}\right)-\sum_{k=0}^{N_{\text{steps}}-1}\bar{\mathfrak{h}}(t_k,X_{t_k}^{\pi,\tilde{\theta}},\xi_j)\Delta\bar{S}_{t_k}^{\pi,\tilde{\theta}}\right]-\mathfrak{p}(\Phi_j)\right)^2 \tag{4.12}$$

and where $\mathbb{E}^{\mathbb{Q}^N}$ denotes the empirical expected value calculated on the N paths.
ii) Freeze θ, use Adam to update ξ, by optimising the sample variance

$$\xi\leftarrow\arg\min_{\xi}\sum_{j=1}^{M}\mathbb{V}\text{ar}^{\mathbb{Q}^N}\left[\Phi_j\left(X^{\pi,\theta}\right)-\sum_{k=0}^{N_{\text{steps}}-1}\bar{\mathfrak{h}}(t_k,X_{t_k}^{\pi,\theta},\xi_j)\Delta\bar{S}_{t_k}^{\pi,\theta}\right]$$

end for
return θ,ξ.

4.2.4 Algorithm for multiple maturities

Algorithms 11 and 12 calibrate the SDE (4.16) to one set of derivatives. If the derivatives for which we have liquid market prices can be grouped by maturity, as is the case e.g. for call / put prices, we can use a more efficient algorithm to achieve the calibration.

This follows the natural approach used e.g. in [71] and [6] in the context of learning PDEs, where the networks for $b(t, X_t^\theta, \theta)$ and $\sigma(t, X_t^\theta, \theta)$ are split into different networks, one per maturity. Let $\theta = (\theta_1, \dots, \theta_{N_m})$, where N_m is the number of maturities. Let

$$\begin{aligned} b(t, X_t^\theta, \theta) &:= \mathbf{1}_{t\in[T_{i-1},T_i]}(t) b^i(t, X_t^\theta, \theta_i), \quad i \in \{1, \dots, N_m\}, \\ \sigma(t, X_t^\theta, \theta) &:= \mathbf{1}_{t\in[T_{i-1},T_i]}(t) \sigma^i(t, X_t^\theta, \theta_i), \quad i \in \{1, \dots, N_m\}, \end{aligned} \tag{4.13}$$

with each b^i and σ^i a feed forward neural network.

Regarding the SDE parametrisation, we fit feed-forward neural networks (see Appendix 4.F) to the diffusion of the SDE of the price process under some risk-neutral measure. In the particular case where we calibrate the neural SDE to market data without imposing any bounds on the resulting exotic option prices, one can then do an incremental learning as follows,

1. Consider the first maturity T_i with $i = 1$.
2. Calibrate the SDE using Algorithm 11 to the vanilla prices at maturity T_i.
3. Freeze the parameters of σ_i, set $i \leftarrow i + 1$, and go back to the previous step.

In the above algorithm we are assuming that the observed price at T_{i+1} does not affect the volatility coefficients before T_i. The above algorithm is memory efficient, as it only needs to backpropagate through that last maturity in each gradient descent step.

4.3 Stochastic approximation algorithm for the calibration problem

4.3.1 Classical stochastic gradient

First, let us review the basics about stochastic gradient algorithm. Let us for a moment consider a general probability space $(\Omega, \mathcal{F}, \mathbb{P})$ and let $H : \Omega \times \Theta \to \mathbb{R}^d$. Consider the following optimisation problem

$$\min_{\theta\in\Theta} h(\theta)\,, \quad \text{where} \quad h(\theta) := \mathbb{E}[H(\theta)]\,.$$

Notice that the minimisation task (4.5) does not fit this pattern as in our case the expectation is inside ℓ. Nevertheless, we know that the classical gradient algorithm, with the learning rates $(\eta_k)_{k=1}^\infty$, $\eta_k > 0$ for all k, applied to this optimisation problem is given by

$$\theta_{k+1} = \theta_k - \eta_k \partial_\theta(\mathbb{E}[H(\theta_k)])\,.$$

Under suitable conditions on H and η_k, it is known that θ_k converges to a minimiser of h, see [121]. As $\mathbb{E}[H(\theta_k)]$ can rarely be computed explicitly, the above algorithm is not practical and is replaced with stochastic gradient descent (SGD) given by

$$\theta_{k+1} = \theta_k - \eta_k \frac{1}{N}\sum_{i=1}^N \partial_\theta H^i(\theta_k)\,,$$

where $(H^i(\theta))_{i=1}^N$ are independent samples from the distribution of $H(\theta)$ and $N \in \mathbb{N}$. The choice of a "good" estimator for $\mathbb{E}[H(\theta)]$ in the context of stochastic gradient algorithms is an active research area research, see e.g. [122]. When the estimator of $\mathbb{E}[H(\theta)]$ is unbiased, the SGD can be shown to converge to a minimum of h, [121].

4.3.2 Stochastic algorithm for the calibration problem

Let us return to the probability space introduced in Section 4.1.2 and recall that our overall objective in the calibration is to minimise $J = J(\theta)$ given by

$$J(\theta) = \sum_{i=1}^{M} \ell\Big(\mathbb{E}^{\mathbb{Q}(\theta)}[\Phi_i^{\mathrm{cv}}], \mathfrak{p}(\Phi_i)\Big) .$$

We write $X^\theta := (X_t^\theta)_{t\in[0,T]}$ and note again that since $\mathbb{Q}(\theta) = \mathbb{Q} \circ (X^\theta)^{-1}$, we then have that $\mathbb{E}^{\mathbb{Q}(\theta)}\big[\Phi_i\big] = \mathbb{E}^{\mathbb{Q}}\big[\Phi_i(X^\theta)\big]$. Recall also that in Algorithms 11 and 12 we do not optimise over θ in the term $\mathfrak{h}(s, (X^{\bar\theta}_{s\wedge t})_{t\in[0,T]}, \xi)$. Hence $\mathbb{E}^{\mathbb{Q}}[\partial_\theta \Phi_i^{\mathrm{cv}}(X^\theta)] = \mathbb{E}^{\mathbb{Q}}[\partial_\theta \Phi_i(X^\theta)]$.

Since the summation plays effectively no role in further analysis we will assume, without loss of generality, that $M = 1$. We differentiate $J = J(\theta)$ and work with the pathwise representation of this derivative (using language from [50]). For that we impose the following assumption.

Assumption 4.3.1. *We assume that payoffs $G := (\Psi, \Phi)$, $G : C([0,T], \mathbb{R}^d) \to \mathbb{R}$ are such that*

$$\partial_\theta \mathbb{E}^{\mathbb{Q}}\left[G(X^\theta)\right] = \mathbb{E}^{\mathbb{Q}}\left[\partial_\theta G(X^\theta)\right] .$$

We refer reader to [50, chapter 7] for exact conditions when exchanging integration and differentiation is possible. We also remark that for the payoffs for which the Assumption 4.3.1 does not hold, one can use likelihood method and more generally Malliavin weights approach for computing greeks, [123]. We don't pursue this here for simplicity.

Writing $\ell = \ell(x, y)$, applying Assumption 4.3.1 and noting that $\mathbb{E}^{\mathbb{Q}(\theta)}\big[\Phi_i\big] = \mathbb{E}^{\mathbb{Q}}\big[\Phi(X^\theta)\big]$ we see that

$$\begin{aligned}\partial_\theta J(\theta) &= (\partial_x \ell)\Big(\mathbb{E}^{\mathbb{Q}(\theta)}[\Phi^{\mathrm{cv}}], \mathfrak{p}(\Phi)\Big)\partial_\theta \mathbb{E}^{\mathbb{Q}(\theta)}\big[\Phi^{\mathrm{cv}}\big] \\ &= (\partial_x \ell)\Big(\mathbb{E}^{\mathbb{Q}}[\Phi^{\mathrm{cv}}(X^\theta)], \mathfrak{p}(\Phi)\Big)\mathbb{E}^{\mathbb{Q}}\big[\partial_\theta \Phi(X^\theta)\big] .\end{aligned}$$

Hence, if we wish to update θ to some $\tilde\theta$ in such a way that J is decreased, then we need to take (for some $\gamma > 0$)

$$\tilde\theta = \theta - \gamma(\partial_x \ell)\Big(\mathbb{E}^{\mathbb{Q}}[\Phi^{\mathrm{cv}}(X^\theta)], \mathfrak{p}(\Phi)\Big)\mathbb{E}^{\mathbb{Q}}\big[\partial_\theta \Phi(X^\theta)\big]$$

so that

$$\begin{aligned}\frac{d}{d\varepsilon} J(\theta + \varepsilon(\tilde\theta - \theta))\Big|_{\varepsilon=0} &= (\partial_x \ell)\Big(\mathbb{E}^{\mathbb{Q}}[\Phi^{\mathrm{cv}}(X^\theta)], \mathfrak{p}(\Phi)\Big)\mathbb{E}^{\mathbb{Q}}\big[\partial_\theta \Phi(X^\theta)\big](\tilde\theta - \theta) \\ &= -\gamma\Big|(\partial_x \ell)\Big(\mathbb{E}^{\mathbb{Q}}[\Phi^{\mathrm{cv}}(X^\theta)], \mathfrak{p}(\Phi)\Big)\mathbb{E}^{\mathbb{Q}}\big[\partial_\theta \Phi(X^\theta)\big]\Big|^2 \le 0 .\end{aligned}$$

If we had one network for each time step, leading to some resnet-like-network architecture for the time discretisation, then it may be more efficient to use a backward equation representation in the training. This representation can be derived using similar analysis as in [124] (see also [115]).

We will work with the following optimisation objective

$$h(\theta) := \ell\Big(\mathbb{E}^{\mathbb{Q}(\theta)}[\Phi^{\mathrm{cv}}], \mathfrak{p}(\Phi)\Big)\,.$$

Then in the gradient step update we have

$$\partial_\theta h(\theta) = \partial_x \ell\left(\mathbb{E}^{\mathbb{Q}}[\Phi^{\mathrm{cv}}(X^\theta)], \mathfrak{p}(\Phi)\right) \mathbb{E}^{\mathbb{Q}}[\partial_\theta \Phi(X^\theta)]\,,$$

Since ℓ is typically not an identity function, a mini-batch estimator of $\partial_\theta h(\theta)$, obtained by replacing $\mathbb{Q}$ with $\mathbb{Q}^N$, is given by

$$\partial_\theta h^N(\theta) := \partial_x \ell\left(\mathbb{E}^{\mathbb{Q}^N}[\Phi^{\mathrm{cv}}(X^\theta)], \mathfrak{p}(\Phi)\right) \mathbb{E}^{\mathbb{Q}^N}[\partial_\theta \Phi(X^\theta)]\,,$$

is a biased estimator of $\partial_\theta h$. Nonetheless, the bias can be estimated in terms of number of samples N and the variance. The fact the bias is controlled by the variance, further justifies why it is important to reduce the variance when calibrating the models with stochastic gradient algorithm. An alternative perspective is to view $\partial_\theta h^N(\theta)$ as non-linear function of $\mathbb{Q}^N$. It turns out that that there is a general theory studying smoothness and corresponding expansions of such functions of measures and we refer reader to [125] for more details.

In Appendix 4.A we provide a result on the bias for a general loss function. For the square loss function the bias is given below.

Theorem 4.3.2. *Let Assumption 4.3.1 hold. Consider the family of neural SDEs* (4.1)*. For* $\ell(x, y) = |x - y|^2$*, we have*

$$\left|\mathbb{E}^{\mathbb{Q}}\left[\partial_\theta h^N(\theta)\right] - \partial_\theta h(\theta)\right| \leq \frac{2}{N}\left(\mathbb{V}\mathrm{ar}^{\mathbb{Q}}[\Phi^{\mathrm{cv}}(X^\theta)]\right)^{1/2}\left(\mathbb{V}\mathrm{ar}^{\mathbb{Q}}[\partial_\theta \Phi(X^\theta)]\right)^{1/2}\,.$$

Proof. This is an immediate consequence of Theorem 4.A.1. □

Hence, we see that by reducing the variance of the first term we are also reducing the bias of the gradient. This justifies superiority of learning task (4.9) over (4.6).

4.4 Analysis of the randomised training

4.4.1 Case of general cost function ℓ

While the idea of calibrating to one maturity at the time described in Section 4.2.4 works well if our aim is only to calibrate to vanilla options, it cannot be directly applied to learn robust bounds for path dependent derivatives, see (4.6). This is because the payoff of path dependent derivatives, in general, is not an affine function of maturity. On the other hand, training all neural networks at every maturity all at once, makes every step of the gradient algorithm used for training computationally heavy.

In what follows, we introduce a randomisation of the gradient so that at each step of the gradient algorithm the derivatives with respect to the network parameters are computed at only one maturity at the time while keeping parameters at all other maturities unchanged. This is similar to the popular dropout method, see [126], that is known to help with overfitting when training deep neural networks, but for us

the main aim is computational efficiency. Recall how we split the networks for drift and diffusion

$$\begin{aligned} b(t, X_t^\theta, \theta) &:= \mathbf{1}_{t\in[T_{i-1},T_i]}(t) b^i(t, X_t^\theta, \theta_i), \quad i \in \{1, \dots, N_m\}, \\ \sigma(t, X_t^\theta, \theta) &:= \mathbf{1}_{t\in[T_{i-1},T_i]}(t) \sigma^i(t, X_t^\theta, \theta_i), \quad i \in \{1, \dots, N_m\}, \end{aligned} \tag{4.13'}$$

where $N_m \in \mathbb{N}$ is the number of subintervals in the partition of the time horizon $[0, T]$ and can coincide with the number of maturities. Let $U \sim \mathbb{U}[1, \dots, N_m]$ be a uniform random variable over the set $\{1, \dots, N_m\}$ defined on a new probability space $(\Omega^{\mathbb{U}}, \mathcal{F}^{\mathbb{U}}, \mathbb{P}^{\mathbb{U}})$. Let Z be given by

$$\begin{aligned} dZ_t^\theta(U) =& \left(\sum_{i=1}^{N_m} \mathbf{1}_{[T_{i-1},T_i]}(t) \partial_x b^i(t, X_t^\theta, \theta_i) Z_t^\theta(U) + N_m \mathbf{1}_{[T_{U-1},T_U]}(t) \partial_{\theta_U} b^U(t, X_t^\theta, \theta_U) \right) dt \\ &+ \left(\sum_{i=1}^{N_m} \mathbf{1}_{[T_{i-1},T_i]}(t) \partial_x \sigma^i(t, X_t^\theta, \theta_i) Z_t^\theta(U) + N_m \mathbf{1}_{[T_{U-1},T_U]}(t) \partial_{\theta_U} \sigma^U(t, X_t^\theta, \theta_U) \right) dW_t \,, \end{aligned} \tag{4.14}$$

where b^U, σ^U are simply neural networks sampled from the random index U.

Theorem 4.4.1. *Assume $\partial_x[b, \sigma](t, \cdot, \theta)$ exists, is bounded with (t, θ) fixed and that $\partial_\theta[b, \sigma](t, x, \cdot)$ exists and is bounded with (t, x) fixed. Assume $\phi : \mathbb{R}^d \to \mathbb{R}$ is differentiable with derivatives of at most polynomial growth. Let $h(\theta) = \ell(\mathbb{E}^{\mathbb{Q}}[\phi(X_T^\theta)], \mathfrak{p}(\Phi))$ and let its randomised gradient be*

$$(\partial_\theta h)(\theta, U) = \partial_x \ell \left(\mathbb{E}^{\mathbb{Q}}[\phi(X_T^\theta)], \mathfrak{p}(\Phi) \right) \mathbb{E}^{\mathbb{Q}}[(\partial_x \phi)(X_T^\theta) Z_T^\theta(U)] \,,$$

where $Z^\theta(U)$ is given by (4.14)*. Then $\mathbb{E}^{\mathbb{U}}[(\partial_\theta h)(\theta, U)] = (\partial_\theta h)(\theta)$. In other words the randomised gradient is an unbiased estimator of $(\partial_\theta h)(\theta)$.*

Remark 4.4.2. Less stringent assumption on derivatives of b and σ are possible, but we do not want to overburden the present chapter with technical details. Moreover Theorem 4.4.1 is stated for simple payoffs but a similar result can be stated and proved for path dependent payoffs ϕ by replacing the gradient $\partial_x \phi$ with an appropriate path dependent derivative.

Proof. It is well known, e.g [42, 127], that

$$d(\partial_\theta X_t^\theta) = \sum_{i=1}^{N_m} \mathbf{1}_{[T_{i-1},T_i]}(t) \left[\left(\alpha_t^i(\theta) \partial_\theta X_t^\theta + \beta_t^i(\theta) \right) dt + \left(\gamma_t^i(\theta) \partial_\theta X_t^\theta + \eta_t^i(\theta) \right) dW_t \right],$$

where $\alpha_t^i(\theta) := \partial_x b^i(t, X_t^\theta, \theta_i)$, $\beta_t^i(\theta) := \partial_{\theta_i} b^i(t, X_t^\theta, \theta_i)$, $\gamma_t^i(\theta) := \partial_x \sigma^i(t, X_t^\theta, \theta_i)$ and $\eta_t^i(\theta) := \partial_{\theta_i} \sigma^i(t, X_t^\theta, \theta_i)$.

Let $U \sim \mathbb{U}[1, \dots, N_m]$ be a uniform random variable over set $\{1, \dots, N_m\}$ defined on a new probability space $(\Omega^{\mathbb{U}}, \mathcal{F}^{\mathbb{U}}, \mathbb{P}^{\mathbb{U}})$. We introduce the process Z as follows

$$\begin{aligned} dZ_t^\theta(U) =& \left(\sum_{i=1}^{N_m} \mathbf{1}_{[T_{i-1},T_i]}(t) \alpha_t^i(\theta) Z_t^\theta(U) + N_m \mathbf{1}_{[T_{U-1},T_U]}(t) \beta_t^U(\theta) \right) dt \\ &+ \left(\sum_{i=1}^{N_m} \mathbf{1}_{[T_{i-1},T_i]}(t) \gamma_t^i(\theta) Z_t^\theta(U) + N_m \mathbf{1}_{[T_{U-1},T_U]}(t) \eta_t^U(\theta) \right) dW_t \,. \end{aligned}$$

Note that

$$\mathbb{E}^{\mathbb{U}}[\mathbf{1}_{[T_{U-1},T_U]}(t) \beta_t^U(\theta)] = \sum_{i=1}^{N_m} \frac{1}{N_m} \mathbf{1}_{[T_{i-1},T_i]}(t) \beta_t^i(\theta) \,,$$

$$\mathbb{E}^{\mathbb{U}}[\mathbf{1}_{[T_{U-1},T_U]}(t)\eta_t^U(\theta)] = \sum_{i=1}^{N_m} \frac{1}{N_m}\mathbf{1}_{[T_{i-1},T_i]}(t)\eta_t^i(\theta)\,.$$

Now using Fubini-type Theorem for Conditional Expectation, [128, Lemma A5], we have

$$\begin{aligned} d\mathbb{E}^{\mathbb{U}}\big[Z_t^\theta(U)\big] =& \Big(\sum_{i=1}^{N_m}\mathbf{1}_{t\in[T_{i-1},T_i]}(t)\alpha_t^i(\theta)\mathbb{E}^{\mathbb{U}}\big[Z_t^\theta(U)\big] + \sum_{i=1}^{N_m}\mathbf{1}_{t\in[T_{i-1},T_i]}(t)\beta_t^i(\theta)\Big)\,dt \\ &+ \Big(\sum_{i=1}^{N_m}\mathbf{1}_{t\in[T_{i-1},T_i]}(t)\gamma_t^i(\theta)\mathbb{E}^{\mathbb{U}}\big[Z_t^\theta(U)\big] + \sum_{i=1}^{N_m}\mathbf{1}_{t\in[T_{i-1},T_i]}(t)\eta_t^i(\theta)\Big)\,dW_t\,. \end{aligned}$$

Hence the process $\mathbb{E}^{\mathbb{U}}\big[Z_t^\theta(U)\big]$ solves the same linear equation as $\partial_\theta X^\theta$. As the equation has unique solution we conclude that

$$\mathbb{E}^{\mathbb{U}}[Z_t^\theta(U)] = \partial_\theta X_t^\theta \quad \forall t\in[0,T]\,. \tag{4.15}$$

Recall that $h(\theta) = \ell\left(\mathbb{E}^{\mathbb{Q}}[\phi(X_T^\theta)], \mathfrak{p}(\Phi)\right)$, and so

$$(\partial_\theta h)(\theta) = \partial_x\ell\left(\mathbb{E}^{\mathbb{Q}}[\phi(X_T^\theta)], \mathfrak{p}(\Phi)\right)\,\mathbb{E}^{\mathbb{Q}}[(\partial_x\phi)(X_T^\theta)\partial_\theta X_T^\theta]\,.$$

Recall the randomised gradient

$$(\partial_\theta h)(\theta, U) = \partial_x\ell\left(\mathbb{E}^{\mathbb{Q}}[\phi(X_T^\theta)], \mathfrak{p}(\Phi)\right)\,\mathbb{E}^{\mathbb{Q}}[(\partial_x\phi)(X_T^\theta)Z_T^\theta(U)]\,.$$

Note that due to (4.15)

$$\mathbb{E}^{\mathbb{U}}[\mathbb{E}^{\mathbb{Q}}[(\partial_x\phi)(X_T^\theta)Z_T^\theta(U)]] = \mathbb{E}^{\mathbb{Q}}[(\partial_x\phi)(X_T^\theta)\partial_\theta X_T^\theta]\,.$$

which implies that $\mathbb{E}^{\mathbb{U}}[(\partial_\theta h)(\theta, U)] = (\partial_\theta h)(\theta)$. □

4.4.2 Case of square loss function ℓ

Here we show that, in the special case when $\ell(x,y) = |x-y|^2$, randomised gradient as described in Section 4.4 is an unbiased estimator of the full gradient even in the case when $\mathbb{Q}$ is replaced by its empirical measure $\mathbb{Q}^N$. Consequently, standard theory on stochastic approximation applies. Let $(\bar\Omega, \bar{\mathcal{F}}, (\bar{\mathcal{F}}_t)_{t\in[0,T]}, \bar{\mathbb{P}})$ be a copy of $(\Omega, \mathcal{F}, (\mathcal{F}_t)_{t\in[0,T]}, \mathbb{P})$. Then for $\phi : \mathbb{R}^d \to \mathbb{R}$, which is differentiable with derivatives of at most polynomial growth, we can write

$$h^N(\theta) := (\mathbb{E}^{\bar{\mathbb{Q}}^N}[\phi(\bar X_T^\theta)] - \mathfrak{p}(\Phi))(\mathbb{E}^{\mathbb{Q}^N}[\phi(X_T^\theta)] - \mathfrak{p}(\Phi))\,.$$

See also [71] for the same observation. The gradient of h is given by

$$\begin{aligned} (\partial_\theta h^N)(\theta) =& (\mathbb{E}^{\bar{\mathbb{Q}}^N}[\phi(\bar X_T^\theta)] - \mathfrak{p}(\Phi))\mathbb{E}^{\mathbb{Q}^N}[(\partial_x\phi)(X_T^\theta)\partial_\theta X_T^\theta] \\ &+ \mathbb{E}^{\bar{\mathbb{Q}}^N}[(\partial_x\phi)(\bar X_T^\theta)\partial_\theta \bar X_T^\theta](\mathbb{E}^{\mathbb{Q}^N}[\phi(X_T^\theta)] - \mathfrak{p}(\Phi)) \\ =& 2\mathbb{E}^{\mathbb{Q}^N}[(\partial_x\phi)(X_T^\theta)\partial_\theta X_T^\theta](\mathbb{E}^{\bar{\mathbb{Q}}^N}[\phi(\bar X_T^\theta)] - \mathfrak{p}(\Phi))\,. \end{aligned}$$

Note that by introducing $\bar{\mathbb{Q}}^N$, $h^N(\theta)$ and $(\partial_\theta h^N)(\theta)$ are unbiased estimators of $h(\theta)$ and $\partial_\theta h(\theta)$ respectively, as they are the product of two independent random variables. Furthermore, due to Theorem 4.4.1,

we have that

$$(\partial_\theta h^N)(\theta, U) = 2(\mathbb{E}^{\bar{\mathbb{Q}}^N}[\phi(\bar{X}_T^\theta)] - \mathfrak{p}(\Phi))\mathbb{E}^{\mathbb{Q}^N}[(\partial_x \phi)(X_T^\theta) Z_T^\theta(U)]\,,$$

is an unbiased estimator of $(\partial_\theta h^N)(\theta)$, since we are also multiplying two independent random variables. To implement the above algorithm one simply needs to generate two independent sets of samples.

4.5 Testing neural SDE calibrations

All the algorithms were implemented using PyTorch v1.4, see [129] and [130]. The code used is available at `github.com/msabvid/robust_nsde`. We showcase the neural SDE calibration on two datasets: first, with the market data extracted from OptionMetrics using SPX data from 20-Dec-2019. The data is included in Appendix 4.D. Results of this calibration are included in this section. We also test our model against synthetic target data generated using the Heston model, see Appendix 4.C.

We calibrate to European option prices, with the initial price of $s_0 = 3\,221$. The model price is calculated as

$$\mathfrak{p}(\Phi) := \mathbb{E}^{\mathbb{Q}(\theta)}[\Phi] = e^{-rT}\mathbb{E}^{\mathbb{Q}}\left[\left(S_T^\theta - K\right)_+ \middle| S_0 = s_0\right]$$

for maturities of $1, 2, \ldots, 6$ months and 21 different uniformly spaced strikes between in $[2\,800, 3\,500]$. As an example of an illiquid derivative for which we wish to find robust bounds we take the lookback option

$$\mathfrak{p}(\Psi) := \mathbb{E}^{\mathbb{Q}(\theta)}[\Psi] = e^{-rT}\mathbb{E}^{\mathbb{Q}}\left[\max_{t\in[0,T]} S_t^\theta - S_T \middle| S_0 = s_0\right].$$

4.5.1 Local stochastic volatility neural SDE model

In this section we consider, as a proof of concept, a Local Stochastic Volatility (LSV) neural SDE model. See, for example, [131]. The risky asset price price process is $(S_t)_{t\in[0,T]}$, where the drift is equal to the risk-free bond rate r. The volatility function in the LSV neural SDE model depends on (t, S_t, V_t), where $(V_t)_{t\in[0,T]}$ is stochastic volatility process. Here $(V_t)_{t\in[0,T]}$ is not a traded asset. The model is thus given by

$$\begin{aligned} dS_t &= rS_t dt + \sigma^S(t, S_t, V_t, \nu) S_t\, dB_t^S, \quad S_0 = s_0, \\ dV_t &= b^V(V_t, \varphi^{(b)})\, dt + \sigma^V(V_t, \varphi^{(\sigma)})\, dB_t^V, \quad V_0 = v_0, \\ d\langle B^S, B^V\rangle_t &= \rho dt\,, \end{aligned} \tag{4.16}$$

where $\theta := \{\nu, \varphi^{(b)}, \varphi^{(\sigma)}, v_0, \rho\}$, $\rho, v_0 \in \mathbb{R}$, as the set of (multi-dimensional) parameters that we aim to optimise so that the model is calibrated to the observed market data. In the implementation we set $r = 0$ for simplicity.

Furthermore, from (4.16) we obtain the log-price process, which we use since it is more stable in our numerical experiments:

$$\begin{aligned} S_t &:= \exp(Y_t), \quad S_0 = s_0 \\ dY_t &= (r - \frac{1}{2}\sigma^S(t, S_t, V_t, \nu))dt + \sigma^S(t, S_t, V_t, \nu)\, dB_t^Y, \quad Y_0 = \log(s_0), \\ dV_t &= b^V(V_t, \varphi^{(b)})\, dt + \sigma^V(V_t, \varphi^{(\sigma)})\, dB_t^V, \quad V_0 = v_0, \\ d\langle B^Y, B^V\rangle_t &= \rho dt\,. \end{aligned} \tag{4.17}$$

We calibrate the neural LSV model only to synthetic data (see Appendix 4.C), since we only consider it as a proof of concept.

4.5.2 Deep learning setting for the LSV neural SDE model

In the SDE (4.16) the functions σ^S, b^V and σ^V are parametrised by one feed-forward neural network per maturity (see Section 4.2.4 and Appendix 4.F) with 2 hidden layers and 40 neurons in each layer. The non-linear activation function used in each of the hidden layers is the linear rectifier `relu`. In addition, in σ^S and σ^V after the output layer we apply the non-linear rectifier $\texttt{softplus}(x) = \log(1 + \exp(x))$ to ensure a smooth positive output.

The parameterisation of the hedging strategy for the vanilla option prices is also a feed-forward linear network with 2 hidden layers, 20 neurons per hidden layer and `relu` activation functions. In order to get one hedging strategy per vanilla option considered in the market data, the output of $\mathfrak{h}(t_k, s^i_{t_k,\theta}, \xi_{K_j})$ has as many neurons as strikes and maturities.

Finally, the parameterisation of the hedging strategy for the exotic options price is an LSTM network [132] with 20 neurons in the hidden layer with a linear transformation in the last layer to ensure the output is in $\mathbb{R}$. The choice of this architecture is motivated by the fact that the recurrent neural networks (RNNs) are path dependent by construction, which is necessary for the parametrisation of the hedging strategy of an exotic option. The log-price process (4.17), was discretised using the tamed Euler scheme (4.11) with 8 uniform time steps per month. As already mentioned, our empirical experiments show higher numerical stability if we employ the discretisation at log-price level.

The number of Monte Carlo trajectories in each stochastic gradient descent iteration was $N = 4 \times 10^4$ and the abstract hedging strategy was used as a control variate. Finally, in the evaluation of the calibrated neural SDE, the option prices are calculated using $N = 4 \times 10^5$ trajectories of the calibrated neural SDEs, which we generated with 2×10^5 Brownian paths and their antithetic paths; in addition we also used the learned hedging strategies to calculate the Monte Carlo estimators with lower variance of $\mathbb{E}^{\mathbb{Q}(\theta)}[\Phi_i^{\text{cv}}]$ and $\mathbb{E}^{\mathbb{Q}(\theta)}[\Psi^{\text{cv}}]$.

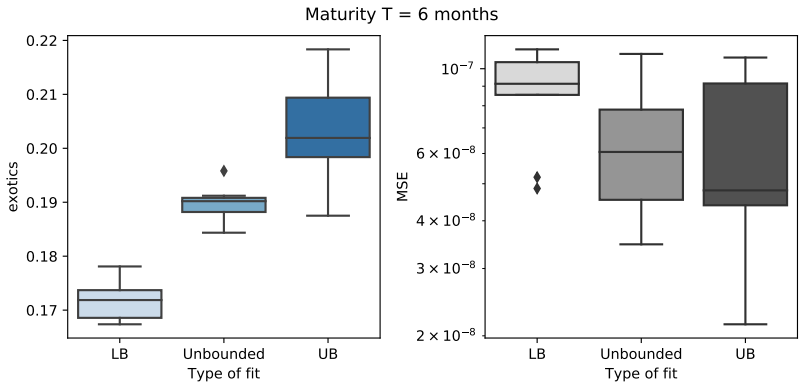


Figure 4.1: Box plots for the Local Stochastic Volatility model (4.16). Exotic option price quantiles are in blue in the left-hand box-plot groups. The MSE quantiles of market data calibration is in grey, in the right-hand box-plot groups. Each box plot comes from 10 different runs of neural SDE calibration, where in each run the parameters of the neural SDE are initialised with a different seed. The three box-plots in each group arise respectively from aiming for a lower bound of the illiquid derivative (left), only calibrating to market and then pricing the illiquid derivative (middle) and aiming for a upper bound of the illiquid derivative (right).

4.5.3 Results from calibrating for LSV neural SDE

Each calibration is run ten times with different initialisations of the network parameters, with the goal to check the robustness of the exotic option price $\mathbb{E}^{\mathbb{Q}(\theta)}[\Psi]$ for each calibrated neural SDE. Each run takes on average 51 minutes on an NVIDIA Tesla V100 SXM2 GPU, meaning that in order to produce each boxplot in Figure 4.1 the experiment takes $51\text{mins} \times 10 = 8.5$ hours. This is repeated to calculate the bounds of the exotic option price. Thus, in total the experiment takes 8.5×3 hours. The blue boxplots in Figure 4.1 provide different quantiles for the exotic option price $\mathbb{E}^{\mathbb{Q}(\theta)}[\Psi]$ and the obtained bounds after running all the experiments 10 times. We make the following observations from calibrating the LSV neural SDE:

i) We note that our methods achieve high calibration accuracy to the market data (measured in terms of the MSE) with consistent bounds on the exotic option prices. See MSE in Figure 4.1.

ii) The calibration is accurate not only in terms of the MSE on prices (Figure 4.1), but also on the corresponding implied volatility curves, see Figure 4.2 and 4.3 for calibration to SPX option prices, Figures 4.5 and 4.6 for calibration to SPX option prices minimising the exotic price, Figures 4.7 and 4.8 for calibration to SPX option prices maximising exotic price. The implied volatility curves are calculated using the Python library `pyvollib`.

iii) The LSV neural SDE produces noticeable ranges for prices of illiquid derivatives, see again Figure 4.1.

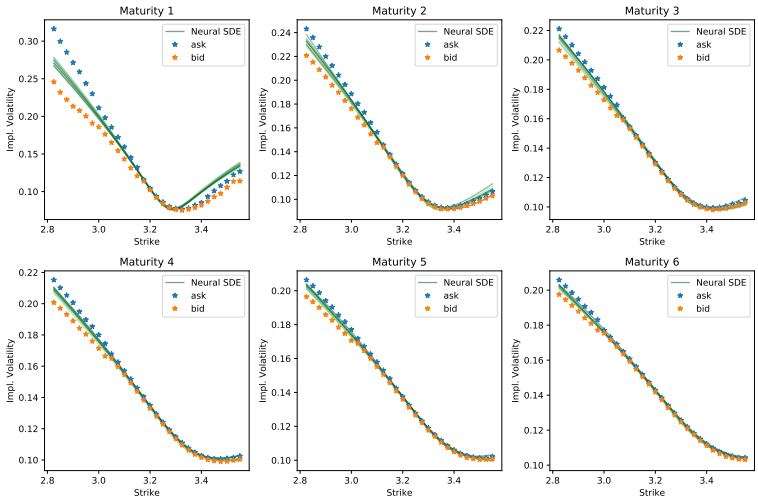


Figure 4.2: Comparing market implied volatility to the model implied volatility for the neural SDE LSV model (4.16) when targeting only the *market data*. Strikes from the market data are scaled by a factor of 10^{-3}. Implied volatility curves of 10 different calibrated neural SDEs are presented in comparison to the market bid-ask spread for different maturities.

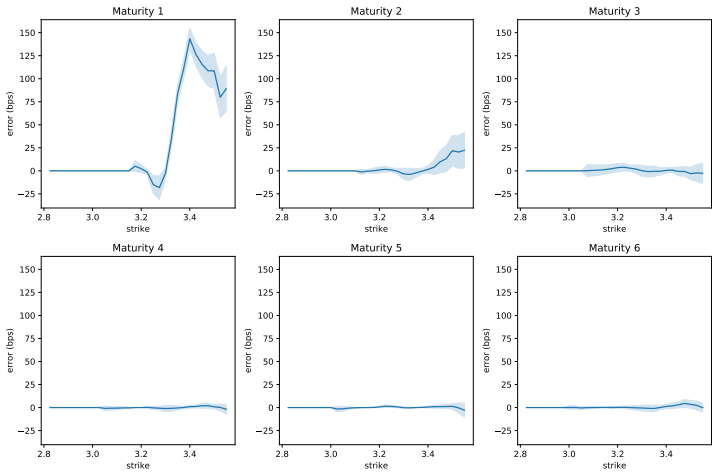


Figure 4.3: The average error and standard deviation in basis points (bps) of the implied volatility curves from Figure 4.2. The error is considered to be zero if the implied volatility falls into the bid-ask spread. If the implied volatility falls outside the spread, the error is calculated as the difference between the predicted implied volatility and the closest bid / ask implied volatility. Strikes from the market data are scaled by a factor of 10^{-3}.

4.5.4 Hedging strategy evaluation

We calculate the mean squared error of a learned portfolio hedging strategy of a lookback option at maturity $T = 6$ months, given by the empirical variance for a trained set of parameters $\tilde{\theta} \in \Theta$:

$$\mathbb{V}\mathrm{ar}^{\mathbb{Q}^N}\left[\Psi\left(X^{\pi,\tilde{\theta}}\right) - \sum_{k=0}^{N_{\text{steps}}-1} \bar{\mathfrak{h}}(t_k, (X^{\pi,\tilde{\theta}}_{t_k\wedge t_j})_{j=0}^{N_{\text{steps}}}, \xi_\Psi)\Delta\bar{S}^{\pi,\tilde{\theta}}_{t_k}\right].$$

The histogram in Figure 4.4 is calculated on $N = 400\,000$ different paths and provides the values of s and s^{cv}, as well as their variances. We can observe a significant reduction of the variance.

$$s := \Psi\left(X^{\pi,\tilde{\theta}}\right), \qquad s^{\text{cv}} := \Psi\left(X^{\pi,\tilde{\theta}}\right) - \sum_{k=0}^{N_{\text{steps}}-1} \bar{\mathfrak{h}}(t_k, (X^{\pi,\tilde{\theta}}_{t_k\wedge t_j})_{j=0}^{N_{\text{steps}}}, \xi_\Psi)\Delta\bar{S}^{\pi,\tilde{\theta}}_{t_k},$$

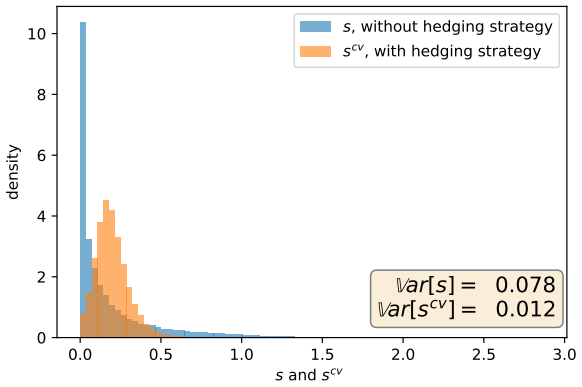


Figure 4.4: Histograms of s, s^{cv} with Ψ the lookback option

4.A Bound on bias in gradient descent

We complete the analysis from Section 4.3.2 for a general loss function here.

Theorem 4.A.1. *Let Assumption 4.3.1 hold. Consider the family of neural SDEs* (4.1)*. We have*

$$|\mathbb{E}[\partial_\theta h^N(\theta)] - \partial_\theta h(\theta)| \leq \left(\mathbb{E}\left[\left(\partial_x \ell\left(\mathbb{E}^{\mathbb{Q}^N(\theta)}[\Phi^{\mathrm{cv}}(X^\theta)], \mathfrak{p}(\Phi)\right) - \partial_x \ell\left(\mathbb{E}^{\mathbb{Q}}[\Phi^{\mathrm{cv}}(X^\theta)], \mathfrak{p}(\Phi)\right)\right)^2\right]\right)^{1/2} \times \left(\mathbb{E}\left[\left(\mathbb{E}^{\mathbb{Q}^N(\theta)}[\partial_\theta \Phi(X^\theta)]\right)^2\right]\right)^{1/2}. \tag{4.18}$$

If in addition we assume that the loss function ℓ is three times differentiable in the first variable with all derivatives of second and third order bounded, then

$$\begin{aligned}
&\left|\mathbb{E}^{\mathbb{Q}}\left[\partial_\theta h^N(\theta)\right] - \partial_\theta h(\theta)\right| \\
&\leq \frac{1}{2}\Big\{ \|\partial_x^3 \ell\|_\infty |\mathbb{E}^{\mathbb{Q}}[\partial_\theta \Phi(X^\theta)]| \frac{1}{N}\mathbb{V}\mathrm{ar}^{\mathbb{Q}}[\Phi^{\mathrm{cv}}(X^\theta)] \\
&+ \|\partial_x^3 \ell\|_\infty \left(\frac{1}{N}\mathbb{V}\mathrm{ar}^{\mathbb{Q}}[\partial_\theta \Phi(X^\theta)]\right)^{1/2} \left(\frac{1}{N^3}\mathbb{E}[(\Phi^{\mathrm{cv}}(X^\theta) - \mathbb{E}^{\mathbb{Q}}[\Phi^{\mathrm{cv}}(X^\theta)])^4] + \frac{3}{N^2}(\mathbb{V}\mathrm{ar}^{\mathbb{Q}}[\Phi^{\mathrm{cv}}(X^\theta)])^2\right)^{1/2} \\
&+ 2\|\partial_x^2 \ell\|_\infty \left(\frac{1}{N}\mathbb{V}\mathrm{ar}^{\mathbb{Q}}[\partial_\theta \Phi(X^\theta)]\right)^{1/2} \left(\frac{1}{N}\mathbb{V}\mathrm{ar}^{\mathbb{Q}}[\Phi^{\mathrm{cv}}(X^\theta)]\right)^{1/2} \Big\}.
\end{aligned} \tag{4.19}$$

Proof. Observe that

$$\mathbb{E}\left[\mathbb{E}^{\mathbb{Q}^N}[\Phi^{\mathrm{cv}}(X^\theta)]\right] = \mathbb{E}^{\mathbb{Q}}[\Phi^{\mathrm{cv}}(X^\theta)] \quad \text{and} \quad \mathbb{E}\left[\mathbb{E}^{\mathbb{Q}^N}[\partial_\theta \Phi(X^\theta)]\right] = \mathbb{E}^{\mathbb{Q}}[\partial_\theta \Phi(X^\theta)]\,.$$

The second equality implies that

$$\mathbb{E}\left[\partial_x \ell\left(\mathbb{E}^{\mathbb{Q}}[\Phi^{\mathrm{cv}}(X^\theta)], \mathfrak{p}(\Phi)\right) \mathbb{E}^{\mathbb{Q}^N}\left[\partial_\theta \Phi(X^\theta)\right]\right] = \partial_\theta h(\theta)\,.$$

Next, by adding and subtracting $\partial_x \ell(\mathbb{E}^{\mathbb{Q}}[\Phi^{\mathrm{cv}}(X^\theta)], \mathfrak{p}(\Phi))$ and using above observation along with the Cauchy–Schwarz inequality we have

$$\begin{aligned}
&|\mathbb{E}[\partial_\theta h^N(\theta)] - \partial_\theta h(\theta)| \\
&= \Big|\mathbb{E}\Big[\Big(\partial_x \ell\left(\mathbb{E}^{\mathbb{Q}^N}[\Phi^{\mathrm{cv}}(X^\theta)], \mathfrak{p}(\Phi)\right) + \partial_x \ell\left(\mathbb{E}^{\mathbb{Q}}[\Phi^{\mathrm{cv}}(X^\theta)], \mathfrak{p}(\Phi)\right) - \partial_x \ell\left(\mathbb{E}^{\mathbb{Q}}[\Phi^{\mathrm{cv}}(X^\theta)], \mathfrak{p}(\Phi)\right)\Big) \\
&\qquad \times \mathbb{E}^{\mathbb{Q}^N}\left[\partial_\theta \Phi(X^\theta)\right]\Big] - \partial_\theta h(\theta)\Big| \\
&= \left|\mathbb{E}\left[\left(\partial_x \ell\left(\mathbb{E}^{\mathbb{Q}^N}[\Phi^{\mathrm{cv}}(X^\theta)], \mathfrak{p}(\Phi)\right) - \partial_x \ell\left(\mathbb{E}^{\mathbb{Q}}[\Phi^{\mathrm{cv}}(X^\theta)], \mathfrak{p}(\Phi)\right)\right) \mathbb{E}^{\mathbb{Q}^N}\left[\partial_\theta \Phi(X^\theta)\right]\right]\right|.
\end{aligned}$$

Hence

$$\begin{aligned}
&|\mathbb{E}[\partial_\theta h^N(\theta)] - \partial_\theta h(\theta)| \\
&\leq \left(\mathbb{E}\left[\left(\partial_x \ell\left(\mathbb{E}^{\mathbb{Q}^N}[\Phi^{\mathrm{cv}}(X^\theta)], \mathfrak{p}(\Phi)\right) - \partial_x \ell\left(\mathbb{E}^{\mathbb{Q}}[\Phi^{\mathrm{cv}}(X^\theta)], \mathfrak{p}(\Phi)\right)\right)^2\right]\right)^{1/2} \left(\mathbb{E}\left[\mathbb{E}^{\mathbb{Q}^N}[\partial_\theta \Phi(X^\theta)]\right]^2\right)^{1/2}.
\end{aligned}$$

This concludes the proof of (4.18). To prove (4.19), we view $\partial_\theta h^N(\theta)$ as function of $(\mathbb{E}^{\mathbb{Q}^N}[\Phi^{\mathrm{cv}}(X^\theta)],$

$\mathbb{E}^{\mathbb{Q}^N}[\partial_\theta \Phi(X^\theta)]$) and expand into its Taylor series around $(\mathbb{E}^{\mathbb{Q}}[\Phi^{\mathrm{cv}}(X^\theta)], \mathbb{E}^{\mathbb{Q}}[\partial_\theta \Phi(X^\theta)])$, i.e.,

$$
\begin{aligned}
&\partial_\theta h^N(\theta) = \partial_\theta h(\theta) \\
&+ \partial_x^2 \ell \left(\mathbb{E}^{\mathbb{Q}}[\Phi^{\mathrm{cv}}(X^\theta)], \mathfrak{p}(\Phi) \right) \mathbb{E}^{\mathbb{Q}}[\partial_\theta \Phi(X^\theta)] \left(\mathbb{E}^{\mathbb{Q}^N}[\Phi^{\mathrm{cv}}(X^\theta)] - \mathbb{E}^{\mathbb{Q}}[\Phi^{\mathrm{cv}}(X^\theta)] \right) \\
&+ \partial_x \ell \left(\mathbb{E}^{\mathbb{Q}}[\Phi^{\mathrm{cv}}(X^\theta)], \mathfrak{p}(\Phi) \right) \left(\mathbb{E}^{\mathbb{Q}^N}[\partial_\theta \Phi(X^\theta)] - \mathbb{E}^{\mathbb{Q}}[\partial_\theta \Phi(X^\theta)] \right) \\
&+ \frac{1}{2} \int_0^1 \Big\{ \partial_x^3 \ell \left(\xi_1^\alpha, \mathfrak{p}(\Phi) \right) \xi_2^\alpha \left(\mathbb{E}^{\mathbb{Q}^N}[\Phi^{\mathrm{cv}}(X^\theta)] - \mathbb{E}^{\mathbb{Q}}[\Phi^{\mathrm{cv}}(X^\theta)] \right)^2 \\
&\quad + 2 \partial_x^2 \ell \left(\xi_1^\alpha, \mathfrak{p}(\Phi) \right) \left(\mathbb{E}^{\mathbb{Q}^N}[\partial_\theta \Phi(X^\theta)] - \mathbb{E}^{\mathbb{Q}}[\partial_\theta \Phi(X^\theta)] \right) \left(\mathbb{E}^{\mathbb{Q}^N}[\Phi^{\mathrm{cv}}(X^\theta)] - \mathbb{E}^{\mathbb{Q}}[\Phi^{\mathrm{cv}}(X^\theta)] \right) \Big\} d\alpha \,,
\end{aligned}
$$

where for $\alpha \in (0,1)$

$$
\begin{aligned}
\xi_1^\alpha =& \mathbb{E}^{\mathbb{Q}}[\Phi^{\mathrm{cv}}(X^\theta)] + \alpha \left(\mathbb{E}^{\mathbb{Q}^N}[\Phi^{\mathrm{cv}}(X^\theta)] - \mathbb{E}^{\mathbb{Q}}[\Phi^{\mathrm{cv}}(X^\theta)] \right) , \\
\xi_2^\alpha =& \mathbb{E}^{\mathbb{Q}}[\partial_\theta \Phi(X^\theta)] + \alpha \left(\mathbb{E}^{\mathbb{Q}^N}[\partial_\theta \Phi(X^\theta)] - \mathbb{E}^{\mathbb{Q}}[\partial_\theta \Phi(X^\theta)] \right) .
\end{aligned}
$$

Hence, using Cauchy-Schwarz inequality

$$
\begin{aligned}
&\left| \mathbb{E}^{\mathbb{Q}} \left[\partial_\theta h^N(\theta) \right] - \partial_\theta h(\theta) \right| \\
&\leq \frac{1}{2} \int_0^1 \mathbb{E} \Big[\Big\{ \|\partial_x^3 \ell\|_\infty \left| \mathbb{E}^{\mathbb{Q}}[\partial_\theta \Phi(X^\theta)] \right| \left(\mathbb{E}^{\mathbb{Q}^N}[\Phi^{\mathrm{cv}}(X^\theta)] - \mathbb{E}^{\mathbb{Q}}[\Phi^{\mathrm{cv}}(X^\theta)] \right)^2 \\
&+ \alpha \|\partial_x^3 \ell\|_\infty \left| \mathbb{E}^{\mathbb{Q}^N}[\partial_\theta \Phi(X^\theta)] - \mathbb{E}^{\mathbb{Q}}[\partial_\theta \Phi(X^\theta)] \right| \left(\mathbb{E}^{\mathbb{Q}^N}[\Phi^{\mathrm{cv}}(X^\theta)] - \mathbb{E}^{\mathbb{Q}}[\Phi^{\mathrm{cv}}(X^\theta)] \right)^2 \\
&+ 2 \|\partial_x^2 \ell\|_\infty \left| \mathbb{E}^{\mathbb{Q}^N}[\partial_\theta \Phi(X^\theta)] - \mathbb{E}^{\mathbb{Q}}[\partial_\theta \Phi(X^\theta)] \right| \left| \mathbb{E}^{\mathbb{Q}^N}[\Phi^{\mathrm{cv}}(X^\theta)] - \mathbb{E}^{\mathbb{Q}}[\Phi^{\mathrm{cv}}(X^\theta)] \right| \Big\} \Big] d\alpha
\end{aligned}
$$

and

$$
\begin{aligned}
&\left| \mathbb{E}^{\mathbb{Q}} \left[\partial_\theta h^N(\theta) \right] - \partial_\theta h(\theta) \right| \leq \frac{1}{2} \Big\{ \|\partial_x^3 \ell\|_\infty \left| \mathbb{E}^{\mathbb{Q}}[\partial_\theta \Phi(X^\theta)] \right| \frac{1}{N} \mathbb{V}\mathrm{ar}^{\mathbb{Q}}[\Phi^{\mathrm{cv}}(X^\theta)] \\
&+ \|\partial_x^3 \ell\|_\infty \left(\frac{1}{N} \mathbb{V}\mathrm{ar}^{\mathbb{Q}}[\partial_\theta \Phi(X^\theta)] \right)^{1/2} \left(\mathbb{E} \left[\left(\mathbb{E}^{\mathbb{Q}^N}[\Phi^{\mathrm{cv}}(X^\theta)] - \mathbb{E}^{\mathbb{Q}}[\Phi^{\mathrm{cv}}(X^\theta)] \right)^4 \right] \right)^{1/2} \\
&+ 2 \|\partial_x^2 \ell\|_\infty \left(\frac{1}{N} \mathbb{V}\mathrm{ar}^{\mathbb{Q}}[\partial_\theta \Phi(X^\theta)] \right)^{1/2} \left(\frac{1}{N} \mathbb{V}\mathrm{ar}^{\mathbb{Q}}[\Phi^{\mathrm{cv}}(X^\theta)] \right)^{1/2} \Big\} .
\end{aligned}
$$

Now let $\lambda^i := \Phi^{\mathrm{cv}}(X^{\theta,i}) - \mathbb{E}^{\mathbb{Q}}[\Phi^{\mathrm{cv}}(X^\theta)]$, and note that

$$
\begin{aligned}
\left(\sum_{i=1}^N \lambda^i \right)^4 =& \sum_{i=1}^N (\lambda^i)^4 + 3 \sum_{i_1 \neq i_2}^N (\lambda^{i_1})^2 (\lambda^{i_2})^2 + 4 \sum_{i_1 \neq i_2}^N (\lambda^{i_1})^1 (\lambda^{i_2})^3 \\
&+ 6 \sum_{i_1, i_2, i_3 \text{ distinct}}^N \lambda^{i_1} \lambda^{i_2} (\lambda^{i_3})^2 + \sum_{i_1, i_2, i_3, i_4 \text{ distinct}}^N \lambda^{i_1} \lambda^{i_2} \lambda^{i_3} \lambda^{i_4} .
\end{aligned}
$$

Hence

$$
\mathbb{E} \left[\left(\mathbb{E}^{\mathbb{Q}^N}[\Phi^{\mathrm{cv}}(X^\theta)] - \mathbb{E}^{\mathbb{Q}}[\Phi^{\mathrm{cv}}(X^\theta)] \right)^4 \right] = \frac{1}{N^3} \mathbb{E}[(\Phi^{\mathrm{cv}}(X^\theta) - \mathbb{E}^{\mathbb{Q}}[\Phi^{\mathrm{cv}}(X^\theta)])^4] + \frac{3}{N^2} (\mathbb{V}\mathrm{ar}^{\mathbb{Q}}[\Phi^{\mathrm{cv}}(X^\theta)])^2 .
$$

The proof is complete. □

4.B Additional results of calibration to market data

In this section we include additional figures displaying the neural SDE fit to market data when targeting the SPX option prices and minimising (Figures 4.5, 4.6) or maximising the exotic option price (Figures 4.7, 4.8).

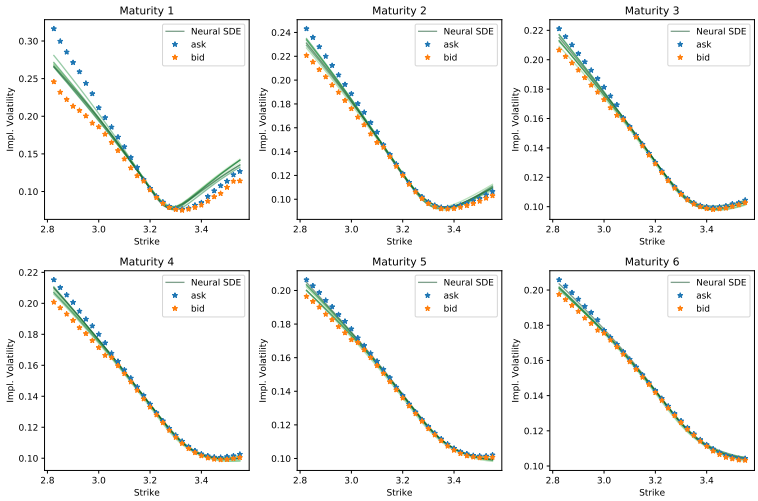


Figure 4.5: Comparing market implied volatlity and model implied volatility for the neural SDE LSV model (4.16) when targeting the *market data* and minimising the exotic option price. We see implied volatility curves of the 10 calibrated neural SDEs vs. the market bid-ask spread for different maturities.

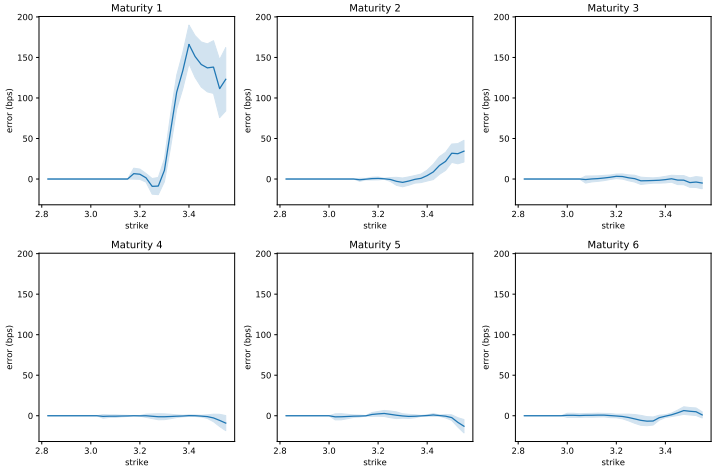


Figure 4.6: Average error and standard deviation in basis points (bps) of the implied volatility curves from Figure 4.5. The error is considered to be 0 if the implied volatility falls into the bid-ask spread.

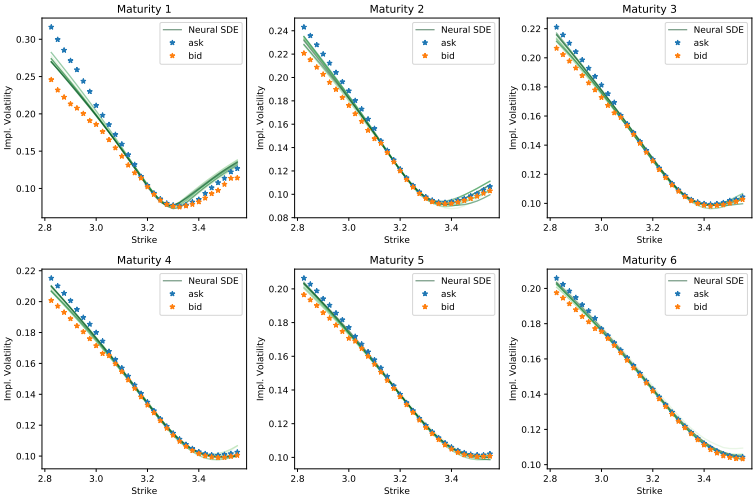


Figure 4.7: Comparing market implied volatlity and model implied volatility for the neural SDE LSV model (4.16) when targeting the *market data* and maximising the exotic option price. We see implied volatility curves of the 10 calibrated neural SDEs vs. the market bid-ask spread for different maturities.

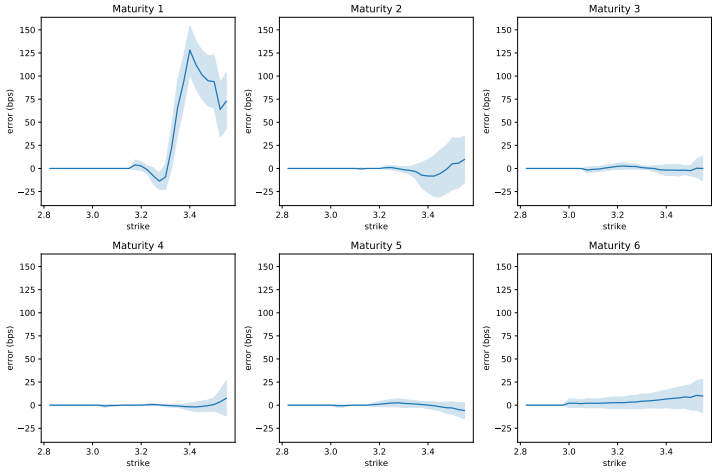


Figure 4.8: Average error and standard deviation in basis points (bps) of the implied volatility curves from Figure 4.7. The error is considered to be 0 if the implied volatility falls into the bid-ask spread.

4.C Calibration to Synthetic data

In this experiment, we calibrate to European option prices

$$\mathfrak{p}(\Phi) := \mathbb{E}^{\mathbb{Q}(\theta)}[\Phi] = e^{-rT}\mathbb{E}^{\mathbb{Q}}\left[\left(S_T^\theta - K\right)_+ \,|\, S_0 = 1\right]$$

for maturities of $2, 4, \ldots, 12$ months and 21 uniformly spaced strikes between in $[0.8, 1.2]$. As an example of an illiquid derivative for which we wish to find robust bounds we take the lookback option

$$\mathfrak{p}(\Psi) := \mathbb{E}^{\mathbb{Q}(\theta)}[\Psi] = e^{-rT}\mathbb{E}^{\mathbb{Q}}\left[\max_{t\in[0,T]} S_t^\theta - S_T \,|\, S_0 = 1\right].$$

The asset price S_t is generated using the Heston model described in Appendix 4.E. We consider the same setting as in Section 4.5 and additionally also fit a Local volatility neural SDE model.

4.C.1 Local volatility neural SDE model

In this section we consider a Local Volatility (LV) neural SDE model. It has been shown by [133] (see also [134]) that if the data consisted of a continuum of call / put prices for all strikes and maturities, then there is a unique function σ such that with the price process

$$dS_t = rS_t\,dt + S_t\,\sigma(t, S_t)dW_t\,, \quad S_0 = 1 \tag{4.20}$$

the model prices and market prices match exactly. In practice only some calls / put prices are liquid in the market and so to apply [133] one has to interpolate, in an arbitrage free way, the missing data. The choice of interpolation method is a further modelling choice on top of the one already made by postulating that the risky asset evolution is governed by (4.20).

We will use a neural SDE instead of directly interpolating the missing data. Let our LV neural SDE model be given by

$$dS_t^\theta = rS_t^\theta dt + \sigma(t, S_t^\theta, \theta)S_t^\theta dW_t^{\mathbb{Q}}, \tag{4.21}$$

where $S_t^\theta \geq 0$, $S_0^\theta = 1$ and $\sigma : [0, T] \times \mathbb{R} \times \mathbb{R}^q \to \mathbb{R}^+$ allows us to calibrate the model to observed market prices.

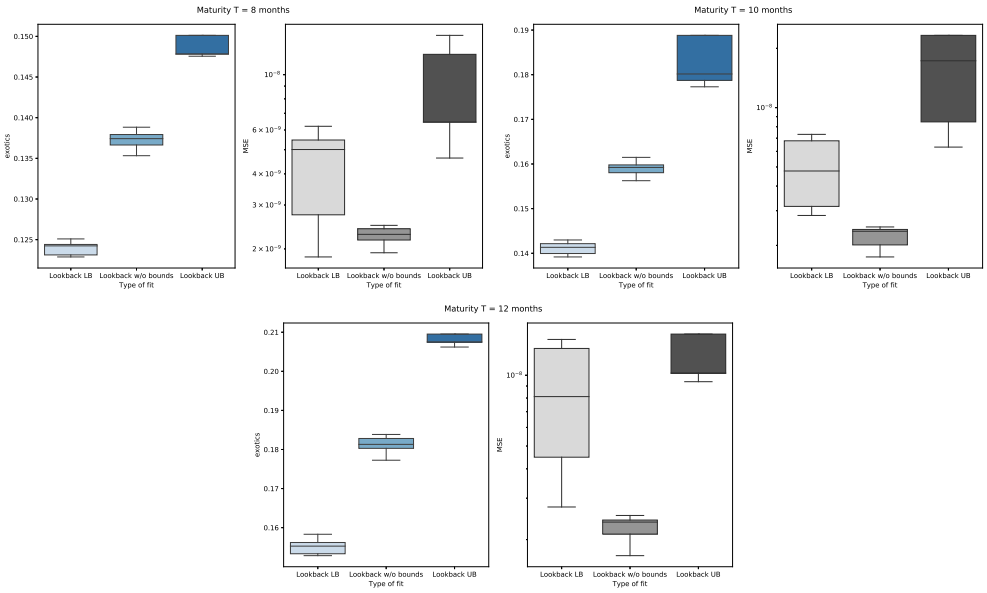


Figure 4.9: Box plots for the Local Volatility model (4.21). Exotic option price quantiles are in blue in the left-hand box-plot groups. The MSE quantiles of market data calibration is in grey, in the right-hand box-plot groups. Each box plot comes from 10 different runs of neural SDE calibration. The three box-plots in each group arise respectively from aiming for a lower bound of the illiquid derivative (left), only calibrating to the market and then pricing the illiquid derivative (middle) and aiming for a upper bound of the illiquid derivative (right).

4.C.2 Conclusions from calibrating for LV neural SDE

Each calibration is run 10 times with different initialisations of the network parameters, with the goal to check the robustness of the exotic option price $\mathbb{E}^{\mathbb{Q}(\theta)}[\Psi]$ for each calibrated neural SDE. The blue boxplots in Figure 4.9 provide different quantiles for the exotic option price $\mathbb{E}^{\mathbb{Q}(\theta)}[\Psi]$ and the obtained bounds after running. We make the following observations from calibrating LV neural SDE:

i) It is possible to obtain high accuracy of calibration with MSE of about 10^{-9} for 6 month maturity, about 10^{-8} for 12 month maturity when the only target is to fit market data. If we are minimizing / maximizing the illiquid derivative price at the same time, then the MSE increases somewhat so that it is about 10^{-8} for both 6 and 12 month maturities. See Figure 4.9. The calibration has been performed using $K = 21$ strikes.

ii) The calibration is accurate not only in terms of MSE on prices but also on individual implied volatility curves, see Figure 4.10 and others in Appendix 4.G.

iii) As we increase the number of strikes per maturity the range of possible values for the illiquid derivative narrows. See Figure 4.22 and Tables 4.2, 4.3, 4.4 and 4.5. One expects, assuming sufficient expressiveness of the neural network, that as the number of strikes (and maturities) would increase to infinity we would recover the unique σ given by the Dupire formula that fits the continuum of European option prices.

iv) With limited amount of market data (which is closer to practical applications) even the LV neural SDE produces noticeable ranges for prices of illiquid derivatives, see again Figure 4.9.

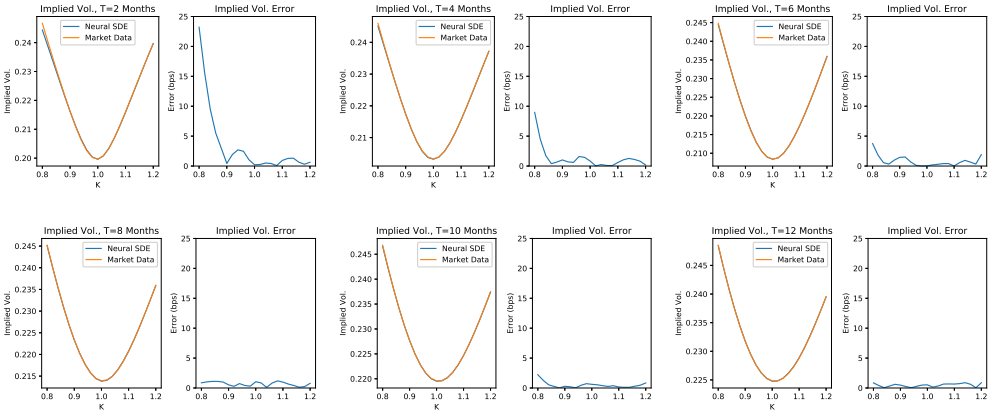


Figure 4.10: Calibrated neural SDE LV model and target market implied volatility comparison.

In Appendix 4.I we provide more details on how different random seeds, different constrained optimization algorithms and different number of strikes used in the market data input affect the illiquid derivative price.

In Appendix 4.G we present market price and implied volatility fit for constrained and unconstrained calibrations. High level of accuracy in all calibrations is achieved due to the hedging neural network incorporated into model training.

4.C.3 Conclusions from calibrating for LSV neural SDE

Each calibration is run ten times with different initialisations of the network parameters, with the goal to check the robustness of the exotic option price $\mathbb{E}^{\mathbb{Q}(\theta)}[\Psi]$ for each calibrated neural SDE. The blue

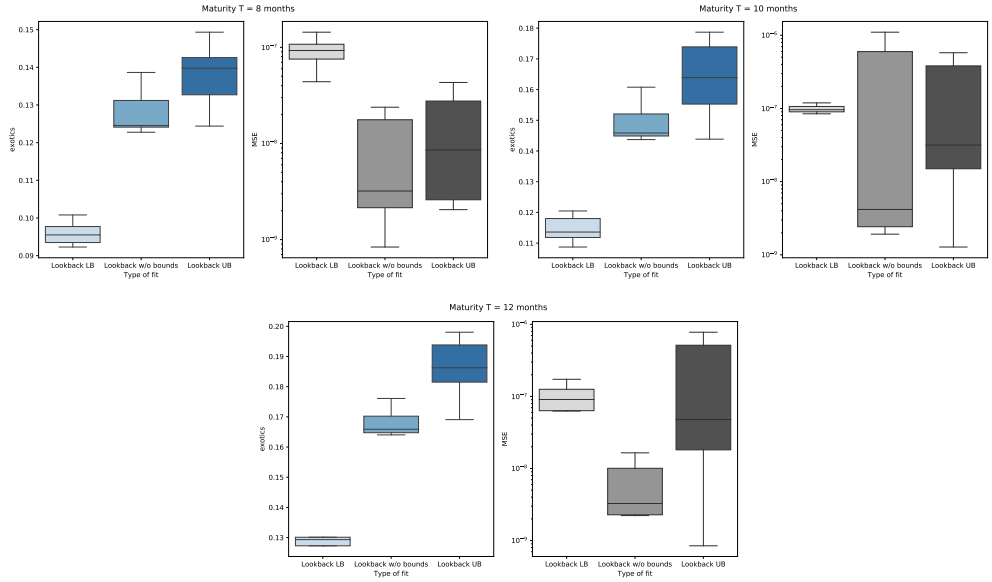


Figure 4.11: Box plots for the Local Stochastic Volatility model (4.16). Exotic option price quantiles are in blue in the left-hand box-plot groups. The MSE quantiles of market data calibration is in grey, in the right-hand box-plot groups. Each box plot comes from 10 different runs of neural SDE calibration. The three box-plots in each group arise respectively from aiming for a lower bound of the illiquid derivative(left), only calibrating to market and then pricing the illiquid derivative (middle) and aiming for a upper bound of the illiquid derivative (right).

boxplots in Figure 4.11 provide different quantiles for the exotic option price $\mathbb{E}^{\mathbb{Q}(\theta)}[\Psi]$ and the obtained bounds after running all the experiments 10 times. We make the following observations from calibrating LSV neural SDE:

i) We note that our methods achieve high calibration accuracy to the market data (measured by MSE) with consistent bounds on the exotic option prices. See Figure 4.11.

ii) The calibration is accurate not only in MSE on prices but also on individual implied volatility curves, see Figure 4.12 and others in Appendix 4.H.

iii) The LSV neural SDE produces noticeable ranges for prices of illiquid derivatives, see again Figure 4.11.

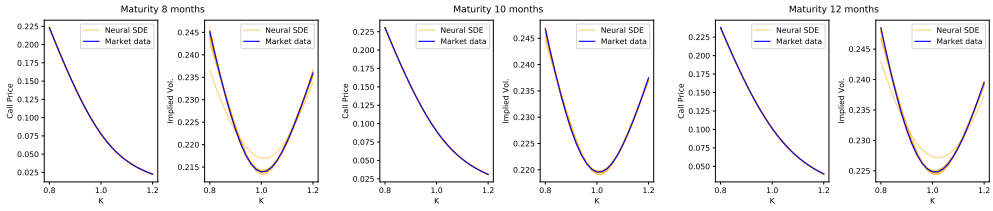


Figure 4.12: Comparing market and model data fit for the neural SDE LSV model (4.16) when targeting only the *market data*. We see vanilla option prices and implied volatility curves of the 10 calibrated neural SDEs vs. the market data for different maturities.

Finally, we study the effect of the control variate parametrisation on the learning speed in Algorithm 11. Figure 4.13 displays the evolution of the Root Mean Squared Error of two runs of calibration to market vanilla option prices for two-months maturity: the blue line using Algorithm 11 with simultaneous

learning of the hedging strategy, and the orange line without the hedging strategy. We recall that from Section 4.2.4, the Monte Carlo estimator $\partial_\theta h^N(\theta)$ is a biased estimator of $\partial_\theta h(\theta)$ An upper bound of the bias is given by Corollary 4.3.2, that shows that by reducing the variance of Monte Carlo estimator of the option price then the bias of $\partial_\theta h^N(\theta)$ is also reduced, yielding better convergence behaviour of the stochastic approximation algorithm. This can be observed in Figure 4.13.

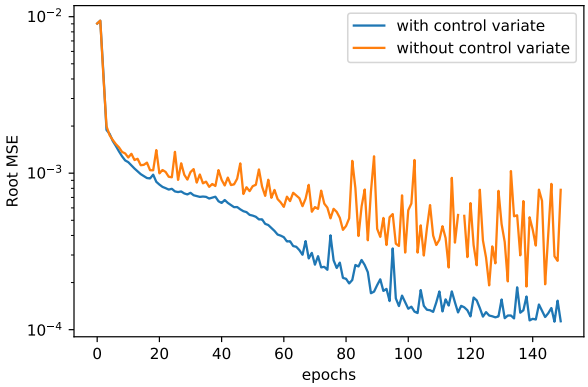


Figure 4.13: Root Mean Squared Error of calibration to Vanilla option prices with and without hedging strategy parametrisation

4.D Data used in calibration to real data

Table 4.1: SPX call prices. Maturities T are measured in months. For training purposes, data is scaled by a factor of 10^{-3}.

Strike	T = 1	T = 2	T = 3	T = 4	T = 5	T = 6
K = 2825	402.0	406.9	413.15	421.25	428.1	437.6
K = 2850	377.0	383.0	389.85	398.45	406.0	415.85
K = 2875	352.3	359.1	366.7	375.85	383.95	394.2
K = 2900	327.7	335.35	343.65	353.45	361.75	372.8
K = 2925	303.45	311.7	320.8	330.95	340.3	351.4
K = 2950	279.0	288.3	298.2	309.05	318.65	330.75
K = 2975	254.65	265.0	275.95	287.5	297.65	309.9
K = 3000	230.3	242.0	253.9	266.0	276.7	289.5
K = 3025	206.3	219.15	232.1	244.65	256.65	269.4
K = 3050	182.45	196.95	210.8	224.45	236.4	249.7
K = 3075	158.8	174.7	189.75	204.0	216.45	230.2
K = 3100	135.5	153.15	169.25	184.0	196.9	211.1
K = 3125	112.4	132.35	149.35	164.55	177.85	192.4
K = 3150	90.3	111.95	130.05	145.55	159.25	174.2
K = 3175	69.25	92.45	111.5	127.3	141.05	156.65
K = 3200	49.55	74.2	93.95	109.9	123.8	139.6
K = 3225	32.5	57.5	77.55	93.5	107.4	123.3
K = 3250	19.3	42.95	62.65	78.25	92.05	107.8
K = 3275	10.2	30.85	49.4	64.35	77.8	93.3
K = 3300	5.05	21.4	38.15	51.95	64.8	79.85
K = 3325	2.35	14.45	28.75	41.25	53.3	67.6
K = 3350	1.15	9.6	21.3	32.2	43.2	56.6
K = 3375	0.62	6.45	15.7	24.85	34.5	46.75
K = 3400	0.35	4.35	11.5	19.0	27.35	38.25
K = 3425	0.28	2.98	8.4	14.5	21.5	31.0
K = 3450	0.22	2.05	6.25	11.05	16.9	25.05
K = 3475	0.18	1.45	4.65	8.45	13.3	20.1
K = 3500	0.15	1.02	3.55	6.55	10.5	16.25
K = 3525	0.15	0.77	2.65	5.1	8.35	13.1
K = 3550	0.1	0.57	2.05	4.0	6.7	10.7
K = 3575	0.08	0.45	1.55	3.15	5.4	8.75
K = 3600	0.08	0.35	1.2	2.5	4.35	7.1
K = 3650	0.08	0.25	0.75	1.6	2.9	4.85

4.E Data used in calibration to synthetic data

We used Heston model to generate prices of calls and puts. The model is

$$\begin{aligned} dX_t &= rX_t dt + X_t\sqrt{V_t}dW_t, \quad X_0 = x_0 && (4.22)\\ dV_t &= \kappa(\mu - V_t)dt + \eta\sqrt{V_t}dB_t, \quad V_0 = v_0 && (4.23)\\ d\langle B, W\rangle_t &= \rho dt\,. && (4.24) \end{aligned}$$

It is well know that for this model a semi-analytic formula can be used to calculate option prices, see [135] but also [136]. The parameters below were used to generate target model calibration prices and were chosen so that the model produces a large skew at the short end of the smile:

$$x_0 = 1,\ r = 0.025\,,\ \ \kappa = 0.78\,,\ \ \mu = 0.11\,,\ \ \eta = 0.68\,,\ \ V_0 = 0.04\,,\ \ \rho = 0.044, \tag{4.25}$$

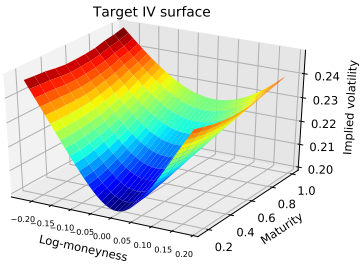


Figure 4.14: The "market" data used in calibration of the neural SDE models. In fact the implied volatility surface comes from (4.22) and (4.25).

Options with bi-monthly maturities up to one year with varying range of strikes were used as market data for neural SDE calibration. The call / put option prices were obtained from the Heston model using Monte Carlo simulation with 10^7 Brownian trajectories. We use bi-monthly maturities up to one year for considered calibrations. Varying range of strikes is used among different calibrations. See Figure 4.14 for the resulting "market" data.

4.F Feed-forward neural networks

We refer the reader to 2.C for a formulation of a feedforward Neural Networks as a composition of affine transformations and non-linear activation functions.

4.G LV neural SDEs calibration accuracy

Figures 4.10, 4.16, 4.18 present implied volatility fit of the local volatility neural SDE model (4.21) calibrated to: market vanilla data only; market vanilla data with lower bound constraint on lookback option payoff; market vanilla data with upper bound constraint on lookback option payoff respectively. Figures 4.15, 4.17, 4.19 present target option price fit of local volatility neural SDE model (4.21) calibrated to: market vanilla data only; market vanilla data with lower bound constraint on lookback option payoff; market vanilla data with upper bound constraint on lookback option payoff respectively.

High level of accuracy in all calibrations is achieved due to the hedging neural network incorporated into model training.

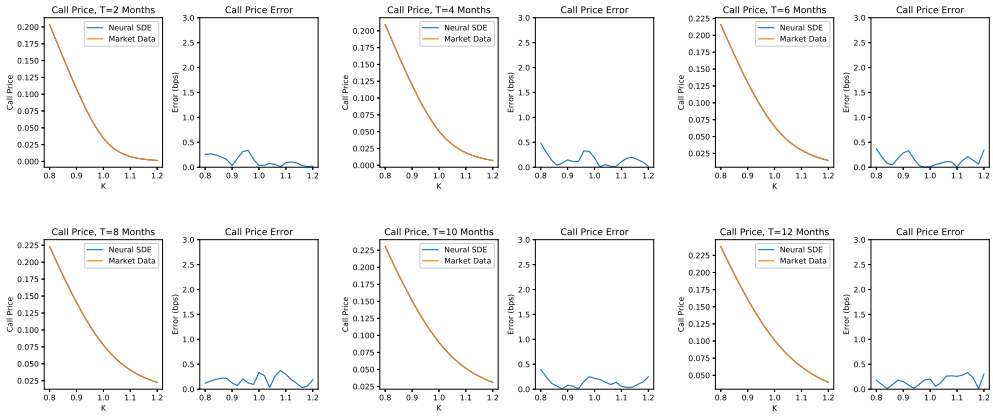


Figure 4.15: Calibrated neural SDE LV model and market target prices comparison.

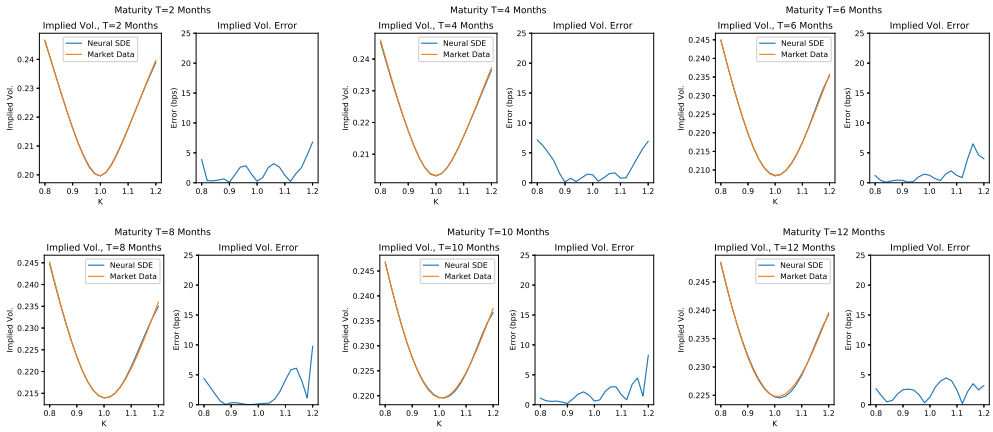


Figure 4.16: Calibrated neural SDE LV model (with lower bound minimization on exotic payoff) and target market data implied volatility comparison.

4.H LSV neural SDEs calibration accuracy

Figures 4.20, 4.12 and 4.21 provide the Vanilla call option price and the implied volatility curve for the calibrated models. In each plot, the blue line corresponds to the target data (generated using Heston model), and each orange line corresponds to one run of the neural SDE calibration. We note again in this plots how the absolute error of the calibration to the vanilla prices is consistently of $\mathcal{O}(10^{-4})$.

4.I Exotic price in LV neural SDEs

Below we see how different random seeds, constrained optimization algorithms and number of strikes used in the market data input affect the illiquid derivative price in the Local Volatility neural SDE model.

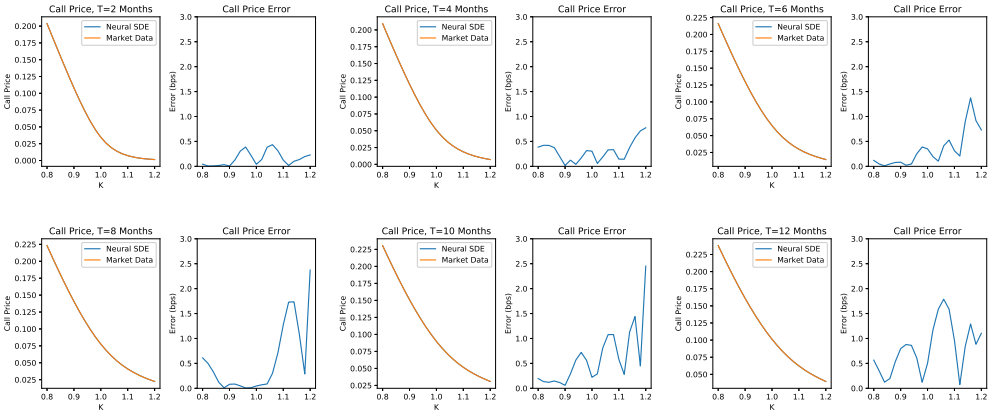


Figure 4.17: Calibrated neural SDE LV model (with lower bound minimization on exotic payoff) and target market prices comparison.

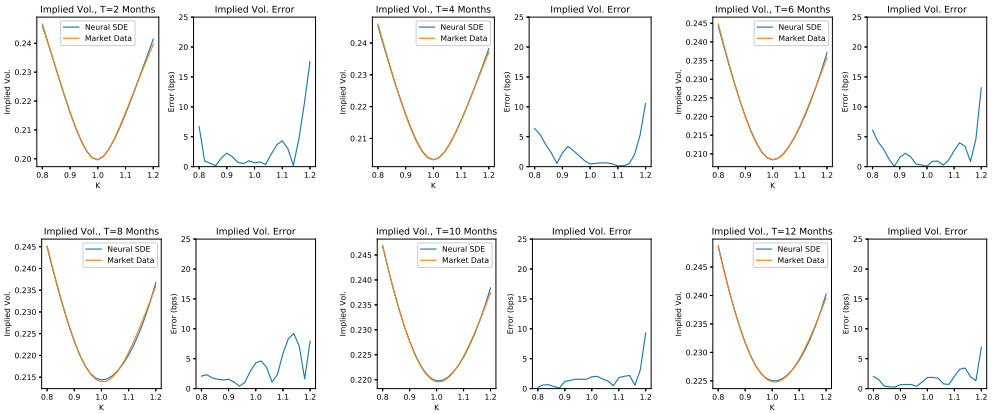


Figure 4.18: Calibrated neural LV model (with upper bound maximization on exotic payoff) and target market data implied volatility comparison.

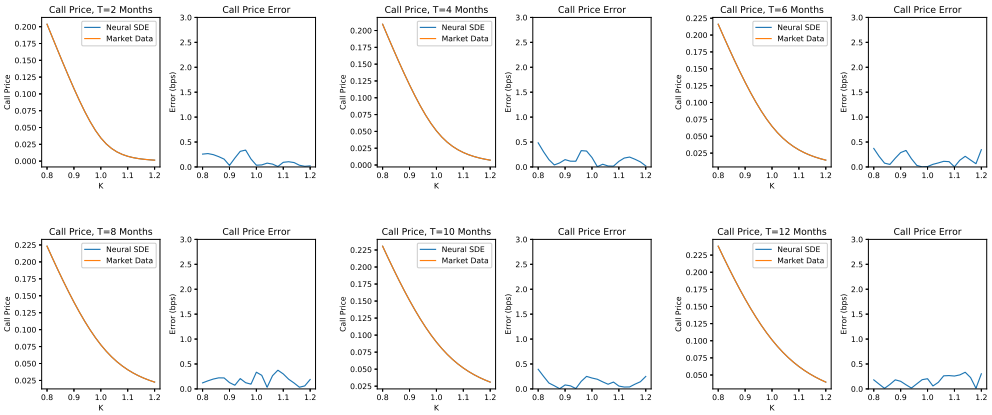


Figure 4.19: Calibrated LV neural model (with upper bound maximization on exotic payoff) and target market prices comparison.

Algorithm 12 Calibration to vanilla prices for one maturity with lower bound for exotic price

Input:

$\pi \coloneqq \{t_0, t_1, \dots, t_n\}$ time grid

$(\Phi)_{i=1}^M$ vector of vanilla option payoffs

$(\Psi)_{i=1}^M$ vector of exotic option payoffs

$(\mathfrak{p}(\Phi_j))_{i=1}^M$ market option price vector

N^{la} update frequency of Augmented Lagrangian parameters

Initialisation:

$\theta \in \Theta$ for neural SDE parameters.

$\xi \in \Xi$ for control variate approximation.

λ, c for Augmented Lagrangian algorithm for constrained optimisation.

for epoch $: 1 : N_{\text{epochs}}$ **do**

Generate N paths $(X_{t_n}^{\pi,\theta,i})_{n=0}^{N_{\text{steps}}} \coloneqq (S_{t_n}^{\pi,\theta,i}, V_{t_n}^{\pi,\theta,i})_{n=0}^{N_{\text{steps}}}$, $i = 1, \dots, N$ using the tamed Euler scheme from (4.11). Let $\tilde{\theta} := \theta$.

i) Freeze ξ and $\tilde{\theta}$, use Adam to find θ, where

$$f(\theta) \coloneqq \mathbb{E}^{\mathbb{Q}^N}\left[\Psi(X^{\pi,\theta}) - \sum_{k=0}^{N_{\text{steps}}-1} \bar{\mathfrak{h}}(t_k, (X_{t_k \wedge t_j}^{\pi,\tilde{\theta}})_{j=0}^{N_{\text{steps}}}, \xi_\Psi)\Delta \bar{S}_{t_k}^{\pi,\tilde{\theta}}\right]$$

$$h(\theta) \coloneqq \sum_{j=1}^{M}\left(\mathbb{E}^{\mathbb{Q}^N}\left[\Phi_j\left(X^{\pi,\theta}\right) - \sum_{k=0}^{N_{\text{steps}}-1} \bar{\mathfrak{h}}(t_k, X_{t_k}^{\pi,\tilde{\theta}}, \xi_j)\Delta \bar{S}_{t_k}^{\pi,\theta}\right] - \mathfrak{p}(\Phi_j)\right)^2$$

$$\theta \leftarrow \arg\min_{\theta} \underbrace{f(\theta) + \lambda h(\theta) + c\{h(\theta)\}^2}_{\text{Augmented Lagrangian}}$$

and where $\mathbb{E}^{\mathbb{Q}^N}$ denotes the empirical expected value calculated on the N paths.

ii) Freeze θ, use Adam to update ξ,

$$\xi \leftarrow \arg\min_{\xi} \sum_{j=0}^{M} \mathbb{V}\text{ar}^{\mathbb{Q}^N}\left[\Phi_j\left(X^{\pi,\theta}\right) - \sum_{k=0}^{N_{\text{steps}}-1} \bar{\mathfrak{h}}(t_k, X_{t_k}^{\pi,\theta}, \xi_j)\Delta \bar{S}_{t_k}^{\pi,\theta}\right] +$$

$$+ \mathbb{V}\text{ar}^{\mathbb{Q}^N}\left[\Psi\left(X^{\pi,\theta}\right) - \sum_{k=0}^{N_{\text{steps}}-1} \bar{\mathfrak{h}}(t_k, (X_{t_k \wedge t_j}^{\pi,\theta})_{j=0}^{N_{\text{steps}}}, \xi_\Psi)\Delta \bar{S}_{t_k}^{\pi,\theta}\right]$$

Every N^{al} **updates of** θ: Update λ, c using Algorithm 13

end for

return θ, ξ

Algorithm 13 Augmented Lagrangian parameters update [119]

Input:

$\lambda, c > 0$

$\tilde{\theta} \in \Theta$ neural network parameters

Update:

Evaluate the MSE of calibration to vanilla prices $h(\cdot)$ at $\tilde{\theta}$

$\lambda \leftarrow \lambda + c\, h(\tilde{\theta})$

$c \leftarrow 2c$

return c, λ

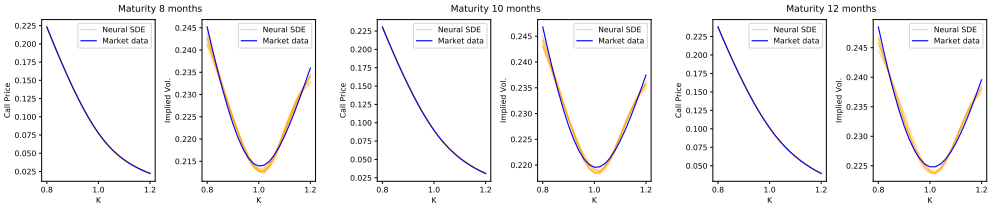


Figure 4.20: Comparing market and model data fit for the neural SDE LSV model (4.16) when targeting the *lower bound* on the illiquid derivative. We see vanilla option prices and implied volatility curves of the 10 calibrated neural SDEs vs. the market data for different maturities.

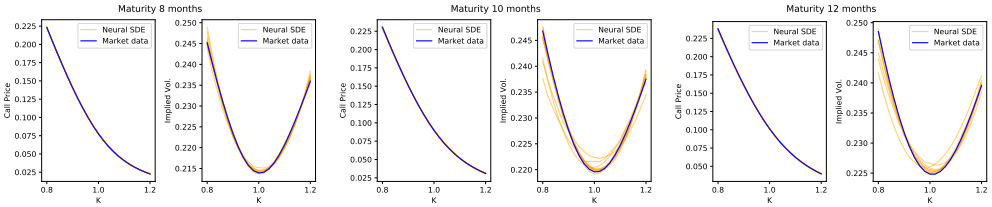


Figure 4.21: Comparing market and model data fit for the neural SDE LSV model (4.16) when targeting the *upper bound* on the illiquid derivative. We see vanilla option prices and implied volatility curves of the 10 calibrated neural SDEs vs. the market data for different maturities.

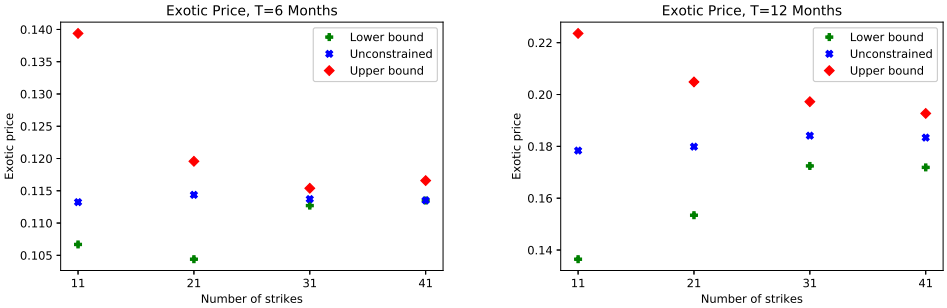


Figure 4.22: Lookback exotic option price in lower, upper and unconstrained implied by perfectly calibrated LV neural SDE calibrated to varying number of market option quotes.

Initialisation	Calibration type	t=2/12	t=4/12	t=6/12	t=8/12	t=10/12	t=1
1	Unconstrained	.055	.088	.113	.134	.159	.178
2	Unconstrained	.056	.086	.113	.132	.158	.175
1	LB Lag. mult.	.055	.086	.107	.127	.143	.154
2	LB Lag. mult.	.055	.084	.098	.113	.125	.139
1	UB Lag. mult.	.056	.099	.119	.142	.163	.214
2	UB Lag. mult.	.059	.101	.131	.156	.208	.220
1	LB Augmented	.055	.077	.107	.113	.127	.136
2	LB Augmented	.056	.085	.109	.123	.139	.158
1	UB Augmented	.058	.102	.139	.156	.188	.224
2	UB Augmented	.057	.088	.128	.151	.167	.184
-	Heston 400k paths	.058	.087	.111	.133	.154	.174

Table 4.2: Impact of initialisation and constrained optimization algorithms on prices of an illiquid derivative (lookback call) implied by LV neural SDE calibrated to vanilla prices with $K = 11$ strikes: $k_1 = 0.9, k_2 = 0.92, ..., k_{11} = 1.1$ for each maturity.

Initialisation	Calibration type	t=2/12	t=4/12	t=6/12	t=8/12	t=10/12	t=1
1	Unconstrained	.056	.087	.114	.140	.161	.182
2	Unconstrained	.056	.087	.114	.136	.161	.180
1	LB Lag. mult.	.056	.086	.110	.123	.141	.153
2	LB Lag. mult.	.056	.087	.108	.125	.150	.155
1	UB Lag. mult.	.056	.088	.120	.156	.179	.205
2	UB Lag. mult.	.056	.088	.118	.153	.187	.208
1	LB Augmented	.056	.087	.108	.128	.143	.164
2	LB Augmented	.056	.087	.108	.125	.150	.155
1	UB Augmented	.056	.091	.124	.155	.173	.194
2	UB Augmented	.056	.088	.125	.146	.167	.189
-	Heston 400k paths	.058	.087	.111	.133	.154	.174
-	Heston 10mil paths	.058	.087	.111	.133	.154	.174

Table 4.3: Impact of initialisation on Prices of ATM lookback call implied by LV neural SDE calibrated to vanilla prices with $K = 21$ strikes: $k_1 = 0.8, k_2 = 0.82, ..., k_{21} = 1.2$ for each maturity.

Initialisation	Calibration type	t=2/12	t=4/12	t=6/12	t=8/12	t=10/12	t=1
1	Unconstrained	.056	.087	.114	.138	.162	.184
2	Unconstrained	.056	.087	.114	.138	.160	.183
1	LB Lag. mult.	.056	.087	.113	.137	.149	.172
2	LB Lag. mult.	.056	.087	.113	.136	.155	.165
1	UB Lag. mult.	.056	.088	.115	.148	.170	.197
2	UB Lag. mult.	.056	.087	.114	.144	.170	.198
1	LB Augmented	.056	.087	.114	.138	.161	.183
2	LB Augmented	.056	.087	.112	.130	.154	.166
1	UB Augmented	.056	.087	.114	.138	.162	.183
2	UB Augmented	.056	.087	.114	.141	.164	.190
-	Heston 400k paths	.058	.087	.111	.133	.154	.174
-	Heston 10mil paths	.058	.087	.111	.133	.154	.174

Table 4.4: Impact of initialisation on Prices of ATM lookback call implied by LV neural SDE calibrated to vanilla prices with $K = 31$ strikes: $k_1 = 0.7, k_2 = 0.72, ..., k_{31} = 1.3$ for each maturity.

Initialisation	Calibration type	t=2/12	t=4/12	t=6/12	t=8/12	t=10/12	t=1
1	Unconstrained	.056	.087	.114	.138	.160	.183
2	Unconstrained	.056	.087	.113	.138	.162	.184
1	LB Lag. mult.	.056	.087	.113	.137	.158	.172
2	LB Lag. mult.	.056	.087	.113	.137	.153	.171
1	UB Lag. mult.	.056	.088	.117	.141	.166	.193
2	UB Lag. mult.	.056	.087	.116	.140	.166	.192
1	LB Augmented	.056	.087	.113	.136	.153	.172
2	LB Augmented	.056	.087	.113	.136	.152	.169
1	UB Augmented	.056	.087	.114	.138	.160	.182
2	UB Augmented	.056	.087	.116	.140	.164	.191
-	Heston 400k paths	.058	.087	.111	.133	.154	.174
-	Heston 10mil paths	.058	.087	.111	.133	.154	.174

Table 4.5: Impact of initialisation on Prices of ATM lookback call implied by LV neural SDE calibrated to vanilla prices with $K = 41$ strikes: $k_1 = 0.6, k_2 = 0.62, ..., k_{41} = 1.4$ for each maturity.

Chapter 5

Sig-Wasserstein GANs for time series generation

Synthetic data is an emerging technology that can significantly accelerate the development and deployment of AI machine learning pipelines. In this work, we develop high-fidelity time-series generators, the SigWGAN, by combining continuous-time stochastic models with the newly proposed signature W_1 metric. The former are the Logsig-RNN models based on the stochastic differential equations, whereas the latter originates from the universal and principled mathematical features to characterize the measure induced by time series. SigWGAN allows turning computationally challenging GAN min-max problem into supervised learning while generating high fidelity samples. We validate the proposed model on both synthetic data generated by popular quantitative risk models and empirical financial data. Codes are available at https://github.com/SigCGANs/Sig-Wasserstein-GANs.git

5.1 Introduction

The ability to model time-series data accurately is critical for numerous applications in the finance industry. In particular, synthetically generated time-series datasets can facilitate training and validation of data-driven risk models and enable data sharing by respecting the demand for privacy constraints. We refer the reader to [137, 138] for the overview of the applications and challenges for synthetic data generation and to [71, 139, 140, 7, 141, 142] for generative modelling perspective of some of the classical problems in quantitative risk management. While generative modelling has been highly successful in generating samples from seemingly high dimensional probability measures, off-the-shelf techniques, such as generative adversarial network's (GAN) [109], struggle to capture the temporal dependence of joint probability distributions induced by time-series data. Furthermore, the min-max objective function of classical GANs make them notoriously difficult to tune. In this paper, we use mathematically principled feature extraction machinery that emerged from the theory of rough paths to reduce the min-max formulation of GANs to an optimization problem. The proposed method, called Sig-Wasserstein GAN, can handle irregularly spaced data streams of variable lengths and is poetically efficient for data sampled at high frequency.

Related work It is by now well documented that popular machine learning frameworks enhanced with path signatures achieve the state of the art performance across many time series learning tasks. For example, the combination of rough path theory and variational autoencoder showed the strength

in simulating financial time series in small data environment [139]. In [143] authors showed that by combining signatures with classical quantitative finance models provide novel perspective on neural SDEs and lead to efficient calibration procedure. Related results have been obtained in [144]. The most closely related to this work is [145]. There authors developed the conditional Sig-Wasserstein GAN to simulate time series that mimics the conditional law of the future time series given the past. This work focuses on the unconditional case instead. Our Logsig-RNN generator extends the work of [146], as we do not require the equal time dimension of input data and output data.

The chapter is structured as follows. In Section 2 we overview the key elements of rough path theory. In Section 3 we present classical GAN framework which we then extend to Sig-Wasserstein GAN in Section 4. Section 5 contains numerical examples.

5.2 Rough path theory

When working with time-series data, especially sampled at high frequency and/or with irregular time stamps, it is useful to take a continuous-time perspective and view data as unknown continuous-time dynamics samples. In particular, rough path theory offers a mathematically principled and universal way of describing the continuous-time data trajectories (paths), which in turn allows designing computationally efficient algorithms. We briefly summarise these ideas below.

5.2.1 Signature of a path

Let $E := \mathbb{R}^d$ and J be a compact time interval. Let $X : J \to E$ denote a continuous path endowed with a norm denoted by $|\cdot|$. We first introduce the *p-variation* as a measure of the roughness of a path. The larger p-variation is, the rougher a path is.

Definition 5.2.1 (p-variation)**.** *Let $p \geq 1$ be a real number. Let $X : J \to E$ be a continuous path. The p-variation of X on the interval J is defined by $\|X\|_{p,J} = \sup_{\mathcal{D}\in J} \left[\sum_{j=0}^{r-1} |X_{t_{j+1}} - X_{t_j}|^p\right]^{\frac{1}{p}}$, where the supremum is taken over any finite time partition of J, i.e. $\mathcal{D} = (t_1, t_2, \cdots, t_r)$.*

We reintroduce some of the concepts in Section 3.4.2 so that this chapter is self-contained. We first introduce the space of formal series of tensors, which is the space signatures live in.

Definition 5.2.2. *(i) The space of formal series of tensors of E, denoted by $T((E))$, is defined as space of sequences,*

$$T((E)) := \{\mathbf{a} = (a_0, a_1, a_2, \dots) : a_n \in (E)^{\otimes n}, n \in \mathbb{N}\}.$$

For two elements $\mathbf{a} = (a_0, a_1, \dots)$ and $\mathbf{b} = (b_0, b_1, \dots)$ we can define an addition and a product by

$$\mathbf{a} + \mathbf{b} = (a_0 + b_0, a_1 + b_1, \dots), \quad \mathbf{a} \otimes \mathbf{b} = (c_0, c_1, \dots),$$

where for each $n \in \mathbb{N}_0$, with the usual (finite-dimensional) tensor product $\otimes$, $c_n = \sum_{k=1}^{n} a_k \otimes b_{n-k}$.
(ii) Let $N \in \mathbb{N}$ and define $B_N = \{\mathbf{a} \in T((E)) : a_0 = \cdots = a_N = 0\}$. Then the truncated tensor algebra of order N is the quotient algebra

$$T^N(E) = T((E))/B_N,$$

with the canonical homomorphism $\pi_N : T((E)) \to T^N(E)$.

We can naturally identify $T^N(E)$ with $\mathbb{R} \oplus \mathbb{R}^d \oplus \cdots \oplus (\mathbb{R}^d)^{\otimes N}$.

We also introduce the function $\|\cdot\|_p : T((E)) \to [0,\infty]$ for some $p \geq 1$ and any element $\mathbf{a} = (a_0, a_1, ...) \in T((E))$ as

$$\|\mathbf{a}\|_p = \left(\sum_{n \in \mathbb{N}} | a_n |_p^p \right)^p,$$

and we define $\mathbf{T}^p(E) := \{\mathbf{a} \in T((E)),\ \|\mathbf{a}\|_p < \infty\}$.

Now we can introduce the (truncated) signature.

Definition 5.2.3 (The signature of a path)**.** *Let $X : J \to E$ be a continuous path of finite p-variation such that the following integration makes sense. The signature of X denoted by $S(X)$ is defined as an infinite series of X_J^k, i.e. $S(X)_J = (1, X_J^1, \cdots, X_J^k, \cdots)$, where*

$$X_J^k = \int \cdots \int_{u_1 < \cdots < u_k; u_1, \cdots, u_k \in J} dX_{u_1} \otimes \cdots \otimes dX_{u_k}, \forall k \geq 1.$$

Let $S_k(X)_J$ denote the truncated signature of X of degree k, i.e. $S_k(X)_J = (1, X_J^1, ..., X_J^k)$.

The following Lemma is useful to consider a norm on the Signature space, see [74]

Lemma 5.2.4. *Fix some $p \geq 1$. The signature of any continuous path $X : J \to E$ of finite p-variation is an element of $\mathbf{T}^p(E)$.*

Path augumentations There are a few commonly used path augumentations methods to accompany with the signature feature to retain good properties of the signature. In our work, we use three augumentation methods (a) Time augumentation, (b) Visiability transformation, and (c) Lead-lag transformation, which add the extra feature dimension to encode the information on the time stamps, the starting point of the path and the lagged process respectively. We refer to [147] for the precise definition of the above path augmentations. For ease of notation, we denote the space of the time-augmented and visibility transformed paths of finite p-variation by $\Omega_0^p(J, E)$. Such augumented path has the path dimension $2d+1$ where d is the dimension of the original path.

Intuitively the signature of a path plays a role of a non-commutative polynomial on the path space. With appropriate path augmentations, the signature of a path has the *uniqueness* and *universality*, which make the signature a useful candidate for the feature set of a path:

Uniqueness: The signature of a path determines the path up to tree-like equivalence [76, 148]. More specifically, when restricted the path space to $\Omega_0^1(J, \mathbb{R}^d)$, the signature map is bijective. In other words, the signature of an augumented path by time augmentation and visibility transformation determines the path completely.

Universality: Any continuous functional on the paths in $\Omega_0^1(J, \mathbb{R}^d)$ can be arbitrarily well approximated by a linear functional of truncated signatures when the degree of the signature is high enough. To state it more precisely we have:

Theorem 5.2.5. *[See Theorem 3.1 of [77]] Consider a compact set $K \subset \Omega_0^1(J, \mathbb{R}^d)$. Let $f : K \to \mathbb{R}$ be any continuous function. Then, for any $\epsilon > 0$, there exists an integer $M > 0$, and a linear functional $l \in T^{(M)}(\mathbb{R}^{2d+1})^*$ acting on the truncated signature of degree M such that*

$$\sup_{X \in K} |f(X) - \langle l, S_M(X) \rangle| < \epsilon \tag{5.1}$$

The space $T^{(M)}(\mathbb{R}^{2d+1})^*$ used in the previous theorem can be defined as follows. It is clear that for a basis $(e_1, \ldots, e_d)$ of $\mathbb{R}^d$ and its dual basis $(e_1^*, \ldots, e_d^*)$ of $(\mathbb{R}^d)^*$, the elements $(e_I =$

$e_{i_1} \otimes \cdots \otimes e_{i_n})_{I=\{i_1,\ldots,i_n\} \subset \{1,\ldots,d\}^n}$ form a basis of $(\mathbb{R}^d)^{\otimes n}$, just as the elements $(e_I^* = e_{i_1}^* \otimes \cdots \otimes e_{i_n}^*)_{I=\{i_1,\ldots,i_n\} \subset \{1,\ldots,d\}^n}$ form a basis of $((\mathbb{R}^d)^*)^{\otimes n}$. Recall that we can canonically identify $((\mathbb{R}^d)^*)^{\otimes n}$ with $((\mathbb{R}^d)^{\otimes n})^*$. Thus we have a linear mapping $((\mathbb{R}^d)^*)^{\otimes n} \to (T((\mathbb{R}^d)))^*$ by the relation

$$e_I^*(\mathbf{a}) = e_I^*(\pi_n(\mathbf{a})) = a_{i_1,\ldots,i_n},$$

which is the coefficient in front of the basis vector e_I in $\mathbf{a}$. In this way we get a linear mapping [74]

$$T((\mathbb{R}^d)^*) = \bigoplus_{n=0}^{\infty}((\mathbb{R}^d)^*)^{\otimes n} \to (T((\mathbb{R}^d)))^*.$$

5.2.2 Log-signature of a path

The log-signature is a parsimonious representation for the signature feature. To define the log-signature, we introduce the logarithm of an element in the tensor algebra space, i.e. $T((E)) = \oplus E^{\otimes n}$. Let $a = (a_0, a_1, \cdots, a_n) \in T((E))$ be such that $a_0 = 1$ and $t = a - 1$. Then the logarithm map is defined as follows:

$$\log(a) = \log(1+t) = \sum_{n=1}^{\infty} \frac{(-1)^{n-1}}{n} t^{\otimes n}, \forall a \in T((E)). \tag{5.2}$$

Definition 5.2.6. *The log signature of path X is the logarithm of the signature of the path X, denoted by $LogSig(X)$. Let $LogSig_k(X)$ denote the truncated log signature of a path X of degree k.*

Uniqueness: Like the signature, the log-signature has the uniqueness as there is one-to-one correspondence between the signature and the log-signature.

Dimension reduction When truncated by the same degree, the log-signature is of lower dimension compared with the signature feature in general. We refer the reader to [149] for more details on the log-signature. It will be used for significant dimension reduction when the path dimension d and the truncated degree of the (log)-signature k is large. See the comparison figure of the dimension of the signature and log-signature. The dimension of the truncated signature of degree k corresponds to the dimension of $T^k(\mathbb{R}^d)$ which is $1 + d + d^2 + \ldots + d^k$. A formula for the dimension of the truncated log-signature of degree k is provided by the Necklace polynomials [149].

k \ d	2		3		4		5		6		7	
1	3	**2**	4	**3**	5	**4**	6	**5**	7	**6**	8	**7**
2	7	**3**	13	**6**	21	**10**	31	**15**	43	**21**	57	**28**
3	15	**5**	40	**14**	85	**30**	156	**55**	259	**91**	400	**140**
4	31	**8**	121	**32**	341	**90**	781	**205**	1555	**406**	2801	**728**
5	63	**14**	364	**80**	1365	**294**	3906	**829**	9331	**1960**	19608	**4088**

Table 5.1: The left integer is the signature dimension whereas the right integer (in bold) is the log-signature dimension.

5.2.3 Expected signature of a stochastic process

Let us consider a E-valued stochastic process X under the probability space. Assume that the signature of X is well defined almost surely, and $S(X)$ has finite expectation. We call $\mathbb{E}[S(X)] := (\mathbb{E}[X_J^0], \mathbb{E}[X_J^1], ...)$ the expected signature of X. $\mathbb{E}[S(X)]$ has infinite radius of convergence if and only if for every $\lambda \geq 0$, $\sum_{k \geq 0} \lambda^n E[\| X_J^k \|] < \infty$.

Intuitively the expected signatures serves the moment generating function, which can characterize the law induced by a stochastic process under some regularity condition. More concretely, an immediate

consequence of Proposition 6.1 in [150] on the uniqueness of the expected signature is summarized in the below theorem:

Theorem 5.2.7. *Let X and Y be two $\Omega_0^1(J,E)$-valued random variables. If $\mathbb{E}[S(X)] = \mathbb{E}[S(Y)]$ and $\mathbb{E}[S(X)]$ has the infinite radius of convergence, then $X = Y$ in the distribution sense.*

5.3 Wasserstein Generative Adversarial Network (WGAN)

Let $(\Omega, \mathcal{F}, \mathbf{P})$ be a probability space, under which μ and ν are two distributions induced by $\mathcal{X}$-valued stochastic process. The Kantorovich-Rubinstein dual representation of Wasserstein-1 (W_1) metric defines a distance between two measures μ and ν, denoted by $W_1(\mu,\nu)$ as follows:

$$W_1(\mu,\nu) = \sup_{\|\cdot\|_{\text{Lip}}\leq 1} \mathbb{E}_{X\sim\mu}[f(X)] - \mathbb{E}_{X\sim\nu}[f(X)], \tag{5.3}$$

where the supremum is over all the 1-Lipschitz functions $f : \mathcal{X} \to \mathbb{R}$ with its Lipschitz norm smaller than 1, i.e. for all $x_1, x_2 \in \mathcal{X}$, $|f(x_1) - f(x_2)| \leq |x_1 - x_2|$.

In the context of generative modelling, let $\mathcal{Z}$ denote a latent space and $Z \in \mathcal{Z}$ denote a variable with the known distribution $\mu_Z \in \mathcal{P}(\mathcal{Z})$. Let $\mu \in \mathcal{P}(\mathcal{X})$ denote a target distribution of observed data. The aim of Wasserstein Generative Adversarial Network (WGAN) is to train a model that induces distribution ν, so that $W_1(\mu,\nu)$ is small. The WGAN is composed of the generator G and the discriminator D. The generator $G_\theta : \mathcal{Z} \to \mathcal{X}$ is a parameterized map transporting the latent distribution μ_Z to the model distribution ν, i.e. the distribution induced by $G_\theta(Z)$, where $\theta \in \Theta^g$ is a parameter set. To discriminate between real and synthetic samples, one parameterises the test function f in definition of W_1 metric (Eqn. (5.3)) by a network f_η with the parameter set $\eta \in \Theta^d$. Training the generator entails solving min-max problem. Indeed, to find optimal $(\theta^\star, \eta^\star)$, one needs to solve

$$\min_\theta \max_{\|f_\eta\|_{\text{Lip}}\leq 1} \mathbb{E}[f_\eta(X)] - \mathbb{E}[f_\eta(G_\theta(Z))],$$

where for $f : \mathcal{X} \to \mathbb{R}$ we define

$$\|f\|_{Lip} := \sup_{x\neq y, x,y\in\mathcal{X}} \frac{|\, f(x) - f(y)\,|}{D(x,y)},$$

for some distance $D(\cdot,\cdot)$ defined in $\mathcal{X}$.

In practice, the min-max problem is solved by iterating gradient descent-ascent algorithms and it convergence can be studied using tools from game theory [151, 152]. It is well known that first order methods that are typically used in practice to solve the min-max problem might not converge, even if the convex-concave case, [153, 154, 155]. Consequently the adversarial training is notoriously difficult to tune, [156, 151], and generalisation error is very sensitive to the choice of discriminator and hype-parameters as it was demonstrated in large scale study in [157].

5.4 Sig-Wasserstein Generative Adversarial Network (Sig-WGAN)

In this work we design algorithms for generating time-series data. Let $x_{1:T} := (x_1, \cdots, x_T) \in \mathcal{X} \subseteq \mathbb{R}^{d\times T}$ denotes a sample, or trajectory, of a time series data and assume that $x_{1:T}$ is distributed according to some unknown target distribution $\nu \in \mathcal{P}(\mathcal{X})$. Unlike majority of work in the literature, we don't assume that the data points are equidistant from each other. Given N independent trajectories $(x^i_{1:T})_{i=1}^N$

sampled from $\nu \in \mathcal{P}(\mathcal{X})$ or given one long trajectory $(x_{iT,(i+1)T})_{i=0}^{N}$ of stationary data, the aim of generative modelling is to learn a model capable of producing high fidelity data samples from $\nu \in \mathcal{P}(\mathcal{X})$, without explicitly modelling the target distribution.

5.4.1 Signature Wasserstein-1 (Sig-W_1) metric

We propose a new Signature Wasserstein-1 (Sig-W_1) metric on the measures on the path space $\mathcal{X} = \Omega_0^1(J, \mathbb{R}^d)$ by combining the signature feature and the W_1 metric to achieve better computation efficiency. Let μ and ν be two measures on the path space $\mathcal{X}$. When using W_1 with discrete time series, one needs to fix the time dimension of time series a prior. Here we (linearly) interpolate the data and compute its signature, possibly using appropriate path augmentation methods without information loss. The signature, as a universal and principled feature, encodes the temporal information regardless of the sampling frequency and variable length. Hence one may consider using the W_1 metric on the signature space to define a distance between the measure induced by two measures on the path space μ and ν.

$$W_1^{\text{Sig space}}(\mu, \nu) = \sup_{\|f\|_{\text{Lip}} \leq 1} \mathbb{E}_{\mathbf{X}\sim\mu}[f(S(\mathbf{X}))] - \mathbb{E}_{\mathbf{X}\sim\nu}[f(S(\mathbf{X}))], \tag{5.4}$$

where for $f : S(\Omega_0^1(J, \mathbb{R}^d)) \to \mathbb{R}$ we define

$$\|f\|_{Lip} := \sup_{x\neq y, x, y \in S(\Omega_0^1(J,\mathbb{R}^2))} \frac{|\, f(x) - f(y)\,|}{\|x - y\|_2},$$

and the norm $\|\cdot\|_2$ on the signature space is well defined by Lemma 5.2.4.

While the use of signature features space $W_1^{\text{Sig space}}$ significantly reduces the time dimension of the problem, practical challenges of solving min-max problem remains. By working with the signature feature, one can reduce the computation of $W_1^{\text{Sig space}}$ distance over the class of Lipschitz functionals to the linear functionals on the signature space thanks to the universality property of the signature. This motivates us to consider the proposed Signature Wasserstein-1 metric between μ and ν defined by

$$\text{Sig-W}_1(\mu, \nu) := \sup_{L \text{ is linear, } \|L\| \leq 1} \mathbb{E}_{\mathbf{X}\sim\mu}[L(S(X))] - \mathbb{E}_{\mathbf{X}\sim\nu}[L(S(X))]\,. \tag{5.5}$$

Hence, by using Sig-W_1 we reduced the nonlinear optimisation task to the linear problem with constraints.

In practice, one needs to truncate the infinite dimensional signature to a finite degree for numerical computation of $W_1^{\text{Sig space}}$ and Sig-$W_1(\mu, \nu)$. The factorial decay of the signature enables us to approximate the signature in Eqn. (5.5) by its truncated signature up to degree M for a sufficiently large M. Therefore we propose to define the truncated Sig-$W_1(\mu, \nu)$ metric up to a degree M as follows:

$$\text{Sig-W}_1^{(M)}(\mu, \nu) := \sup_{\|L\|\leq 1, L \text{ is linear}} L(\mathbb{E}_{\mathbf{X}\sim\mu}[S_M(X)] - \mathbb{E}_{\mathbf{X}\sim\nu}[S_M(X)]), \tag{5.6}$$

When the norm of L is chosen as the l_2 norm of the linear coefficients of L, this reduced optimization problem admits the analytic solution

$$\text{Sig-W}_1^{(M)}(\mu, \nu) := |\mathbb{E}_\mu[S_M(X)] - \mathbb{E}_\nu[S_M(X)]| \tag{5.7}$$

where $|.|$ is l_2 norm. In [158], if one chooses the truncated signature up to degree M as the feature map, then the corresponding Maximum Mean Discrepancy (Sig-MMD) is the square of Sig-$\text{W}^{(M)}(\mu, \nu)$.

The following toy example illustrates the relationship between the Sig-W_1 distance and the $W1$

distance between two path distributions. Let $X = (X_t)_{t\in[0,T]}, \hat{X} = (\hat{X}_t)_{t\in[0,T]}$ two 1-dimensional GBMs given by,

$$dX_t = \theta_1 X_t dt + \sigma X_t dW_t, \;\; X_0 = 1;$$
$$d\hat{X}_t = \theta_2 \hat{X}_t dt + \sigma \hat{X}_t d\hat{W}_t, \;\; \hat{X}_0 = 1,$$

with the same volatility but with possibly different drifts θ_1, θ_2. Let μ, ν be the laws of $X, \hat{X}$. We fix $\sigma = 0.1, \theta_1 = 0.02$, and for $\theta_2 = 0.02 + j0.025, j = 0, \ldots, 4$. When θ_2 is increasing, the discrepancy between X and $\hat{X}$ is increasing. We calculate three distances, i.e. $W_1^{\text{path space}}$, $W_1^{\text{Sig space}}$ and Sig-W_1 to quantify the distance between X and $\hat{X}$ for different θ_2, which all increase when enlarging θ_2 as expected. Since Sig-$\mathrm{W}_1^{(M)}(\mu, \nu)$ admits an analytic solution, it is cheaper to calculate than $W_1^{\text{path space}}(\mu, \nu)$ and $W_1^{\text{Sig space}}(\mu, \nu)$, where one needs to parametrise f by a neural network and optimise its weights. Lipschitzness of the the resulting neural network is ensured by adding a gradient penalty in the objective function of the otpimisation, see [159]. We observe in Figure 5.1 how these three values increase with similar rate as θ_2 increases.

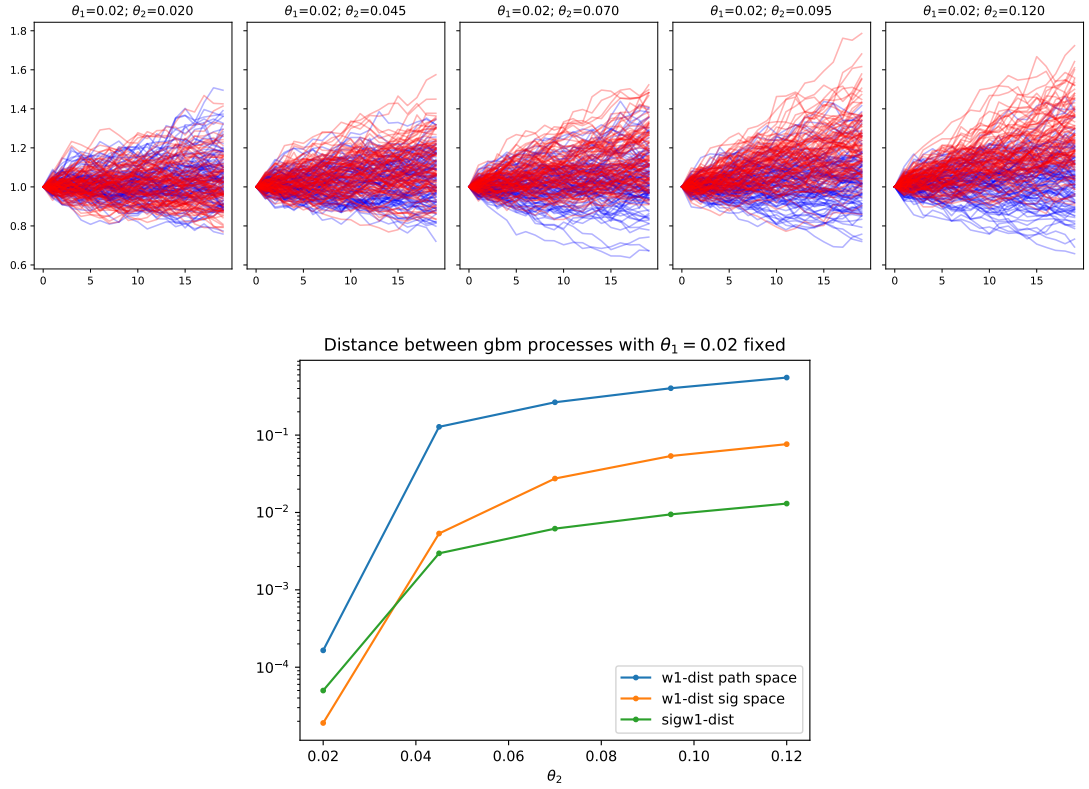


Figure 5.1: The top row displays blue and red samples from $\mathbf{X}, \hat{\mathbf{X}}$ respectively for fixed θ_1, and different values of θ_2

5.4.2 LogSig-RNN Generator

In this section, we introduce two generators of continuous type, which both are motivated by the numerical approximation of stochastic differential equations (SDEs). The first generator is the Neural SDE, which approximates the vector field by a feed-forward neural network in the Euler scheme, while the second generator – the Logsig-RNN resembles the high-order Taylor approximation of SDEs by

combining the recurrent neural network with the log-signature.

Fix a filtered probability space $(\Omega, \mathcal{F}, \{\mathcal{F}_t\}_{t\geq 0}, \mathbf{P})$, under which $W = (W_t)_{t\in[0,T]}$ is a d-dimensional Brownian motion. Let $\mathbf{W} = (\mathbf{W}_t)_{t\in[0,T]}$ denote the time-augmented Brownian motion, where $\mathbf{W}_t = (t, W_t)$ for ease of the notation. Consider a $\mathbb{R}^e$-valued process $X = (X_t)_{t\in[0,T]}$, which satisfies the following time-homogeneous SDEs driven by the Brownian motion W with the drift term μ and the volatility term σ, i.e.

$$dX_t = \mu(X_t)dt + \sigma(X_t)dW_t := f(X_t)d\mathbf{W}_t, \tag{5.8}$$

where the stochastic integral is taken in the Stratonovich sense and the vector field $f : \mathbb{R}^e \to L(\mathbb{R}^{d+1}, \mathbb{R}^e)$ such that $f(x)(s,\omega) = \mu(x)s + \sigma(x)\omega$. [1]

One can approximate the solution X defined by (5.8) via the step-M Taylor expansion locally; when s and t are close,

$$X_t - X_s \approx \sum_{m=1}^{M} f^{\circ k}(X_s) \int_{s<s_1<\cdots<s_m<t} d\mathbf{W}_{s_1} \otimes \cdots \otimes d\mathbf{W}_{s_m}, \tag{5.9}$$

where $0 < s < t < T$, $f^{\circ m} : \mathbb{R}^e \to L((\mathbb{R}^{d+1})^{\otimes m}, \mathbb{R}^e)$ is defined inductively by

$$f^{\circ 1} = f; f^{\circ m+1} = D(f^{\circ m})f$$

with $D(g)$ denoting the differential of the function g. This leads to the numerical approximation scheme by pasting the local step-M Taylor approximation together. Indeed, fix time partitions $\Pi_h = (u_j)_{j=0}^{N_1}$ and $\Pi_X = (t_k)_{k=0}^{N_2}$ of $[0,T]$, which are time partition of Taylor expansion and the time discretization of the solution X respectively. Without loss of generality, $\Pi_h \subset \Pi_X$. We define $\hat{X}$ evaluated at the time partition Π_X inductively; let $\hat{X}_0 = x_0$. For any $t_k \in \Pi_X$ find j such that $t_k \in (u_{j-1}, u_j]$, we apply the Taylor approximation of X_{t_k} around the reference time point $t = u_{j-1} \in \Pi_h$, and hence obtain

$$\begin{aligned} \hat{X}_{t_k} &= \hat{X}_{u_{j-1}} + \sum_{m=1}^{M} f^{\circ m}(\hat{X}_{u_{j-1}}) \int_{u_{j-1}<s_1<\cdots<s_m<t_k} d\mathbf{W}_{s_1} \otimes \cdots \otimes d\mathbf{W}_{s_k} \\ &:= F(\hat{X}_{u_{j-1}}, S^M(\mathbf{W}_{u_{j-1},t_k})) \approx \tilde{F}(\hat{X}_{u_{j-1}}, \text{LogSig}^M(\mathbf{W}_{u_{j-1},t_k})), \end{aligned} \tag{5.10}$$

where we approximate $F(\hat{X}_{u_{j-1}}, S^M(\mathbf{W}_{u_{j-1},t_k}))$ by a the function $\tilde{F}(\hat{X}_{u_{j-1}}, \text{LogSig}^M(\mathbf{W}_{u_{j-1},t_k}))$ that takes $\text{LogSig}^M(\mathbf{W}_{u_{j-1},t_k})$ as an input. Motivated by [160], we approximate $\tilde{F}$ in (5.10) by using a recurrent neural network. This leads to generalized Logsig-RNN model, which maps a d-dimensional Brownian motion $\mathbf{W}_{[0,T]}$ to $(o_{t_k})_{k=1}^{N_2} \in \mathbb{R}^{N_2\times e}$ as the generator of $\mathbb{R}^e$-valued stochastic process: $\forall j \in \{1, \cdots, N_1\}$ and $\forall k \in \{1, \cdots, N_2\}$,

$$\begin{aligned} h_{t_k} &= \sigma_1(\theta_1 h_{u_{j-1}} + \theta_2 \text{LogSig}^M(\mathbf{W}_{u_{j-1},t_k})); \\ o_{t_k} &= \sigma_2(\theta_3 h_{t_k}). \end{aligned} \tag{5.11}$$

where σ_1, σ_2 are two activation functions and $\Theta = \{\theta_1, \theta_2, \theta_3\}$ is the learnable parameter set. When $\Pi_h = \Pi_X$, this is exactly the Logsig-RNN model in [160]

[1] The time-inhomogeneous SDE can be viewed as the projection of the solution to time-homogeneous SDE by lifting X to its time-augmented process.

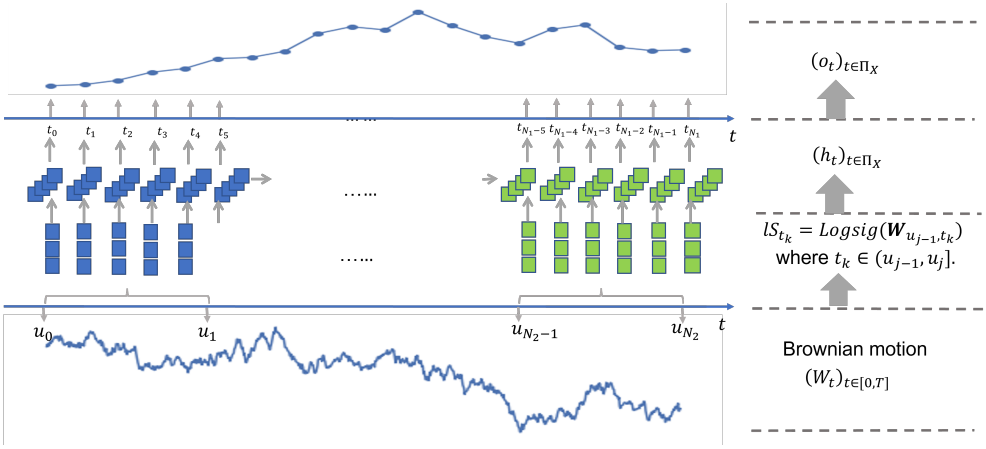


Figure 5.2: The pictorial illustration of the generalized Logsig-RNN model.

5.5 Numerical results

To validate the performance of the proposed Sig-Wasserstein GANs (SigWGANs), we consider three datasets, i.e. (1) synthetic data generated by multi-dimensional Geometric Brownian motion (GBM); (2) synthetic data generated by the rough volatility model; (3) stock price and realized volatility data. The former two datasets are representatives of commonly-used Markovian and non-Markovian model for the underlying price process. For each dataset, we compare the proposed SigWGANs with WGANs to demonstrate the advantages of Sig-Wasserstein metric in terms of the accuracy and efficiency. Besides for the SigWGANs, we benchmark the proposed Logsig-RNN generators of continuous type against the Long short-term memory (LSTM) model. To assess the quality of generated data, we consider the following test metrics:

1. **Sig-W_1 metric**.

2. **Marginal distribution metric**. To assess the fitting of the marginal distribution, we compute the average of Wasserstein distance (also called earth mover's distance, EMD for short) between each marginal distribution $X_t^{(i)}$ of the real time series and fake time series over all time t and feature channel i as the marginal distribution metric. In one dimensions, the EMD can be efficiently calculated numerically using [161].

3. **Correlation metric**. To quantify the fitting of spatial and temporal dependence, we consider how close the correlation of $X_t^{(i)}$ and $X_s^{(j)}$ for any feature coordinate i and j and any time s and t. The correlation metric of X and $\hat{X}$ is defined as $\mathrm{cor}(X, \hat{X}) = \sum_{s,t=1}^{T} \sum_{i,j=1}^{d} \left| \rho(X_s^{(i)}, X_t^{(j)}) - \rho(\hat{X}_s^{(i)}, \hat{X}_t^{(j)}) \right|$, where $\rho(X, Y)$ denotes the correlation of two real-valued random variables X and Y.

The smaller test metrics indicates better performance.

Moreover, on the synthetic dataset, we test the predictive performance of the generative model, which is trained on the dataset of one time frequency and is tested on another time frequency. The better fitting performance on the test time frequency shows the robustness and generalization of the trained generative model against variation in sampling time.

5.5.1 Multi-dimensional Geometric Brownian motion (GBM)

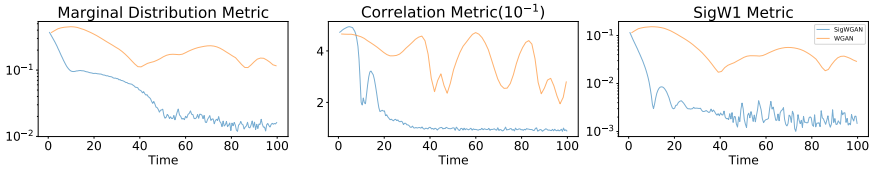
As a motivating example, we consider a d-dimensional GBM $X = (X_t)_{t\in[0,T]}$ satisfying the following SDE: for $i \in \{1, \cdots, d\}$,

$$dX_t^{(i)} = \mu_i X_t^{(i)} dt + \sigma_i X_t^{(i)} d\tilde{W}_t^{(i)}$$

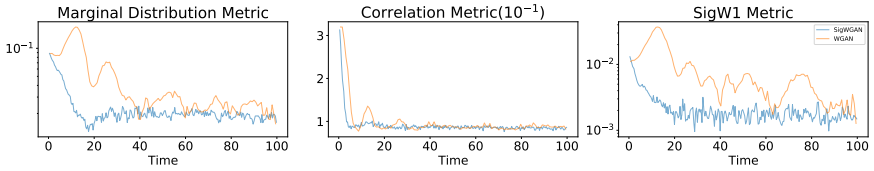
where $X_t \in \mathbb{R}^d$, $\mu \in \mathbb{R}^d$, $\sigma \in \mathbb{R}^d$ and $\tilde{W}$ is a d-dimensional correlated Brownian motions with correlation matrix $(\rho_{ij})_{i,j\in\{1,\cdots,d\}}$. In this example, we set $\mu^i = 0.1$, $\sigma^i = 0.2$, $\forall i \in \{1, ..., d\}$ and $\rho_{ij} = 0.5$, for $i \neq j$. We use the equally spaced time partition of the time interval $[0, 1]$ with step size $\Delta t = 10^{-3}$, and simulate 4000 samples of the solution X to the SDE (5.5.1) using the analytic formula: $X_{t_n}^{(i)} = X_{t_{n-1}}^{(i)} \exp[(\mu_i - \frac{1}{2}\sigma_i^2)\Delta t + \sigma_i(\tilde{W}_{t_n}^{(i)} - \tilde{W}_{t_{n-1}}^{(i)})]$ where $t_n = n\Delta t$ and $i \in \{1, \cdots, d\}$. Here we specify $d = 3$.

Performance comparison of SigWGAN v.s WGAN

Figure 5.3 shows that by replacing the W1 discriminator by Sig-W1 discriminator, three test metrics drop more quickly at the beginning of training period, which indicates that the Sig-WGAN is more efficient to train. Besides, regardless of the generator, the Sig-WGAN outperforms the WGAN in terms of the three test metrics during the training.



(a) LogsigRNN



(b) LSTM

Figure 5.3: The upper/lower panel are the evolution of the test metrics of the Logsig-RNN/LSTM generator during training for the first 100 seconds. From left to right, the test metrics are Marginal distribution metric (Left), Correlation metric (Middle) and SigW1 metric (Right). The results are trained on 3-dimensional Geometric Brownian motion.

Sensitively analysis in terms of number of time steps

Table 5.2. As shown in Table 5.2, the Sig-W1 GAN with the LSTM generator works best when N_T is small (e.g. 10 or 20). However, when increasing the number of time step N_T, it gets more challenging for the LSTM generator to learn the joint path distribution in terms of correlation metric. In contrast to the LSTM, the performance of the LogsigRNN is much robust in terms of high frequency sampling of the data. In particular, combined with Sig-W1 metric, the Logsig-RNN generator achieves the correlation metric and marginal distribution of narrow range $[0.045, 0.050]$ and $[0.919, 1.032]$ for each $N_T \in \{10, 20, 50, 100\}$ respectively. For $N_T = 100$, when using the SigWGAN, the test metrics of the LSTM generator is about four times of that of the Logsig-generator. For a fixed generator, we observe the consistent performance improvement of using Sig-W_1 over the W_1 metric as the discriminator.

	W1				Sig-W1			
N_T	10	20	50	100	10	20	50	100
SigW1 Metric(1e-1)								
LSTM	0.140	0.125	0.677	0.651	0.096	0.064	**0.020**	0.196
LogsigRNN	0.264	0.128	0.162	0.160	**0.047**	**0.040**	0.042	**0.037**
Correlation Metrics (1e-3)								
LSTM	1.057	0.862	5.512	2.658	**0.478**	**0.852**	0.9698	3.391
LogsigRNN	2.125	1.811	9.277	2.219	0.962	1.032	**0.967**	**0.919**
Marginal Distribution Metric								
LSTM	0.052	0.065	0.132	0.082	**0.026**	**0.043**	**0.036**	0.140
LogsigRNN	0.122	0.059	0.317	0.067	0.050	0.047	0.050	**0.045**
Training time (seconds)								
LSTM	376.55	374.78	375.44	375.53	134.33	134.66	134.71	135.89
LogsigRNN	682.21	718.21	721.21	781.21	261.23	322.33	371.21	472.21

Table 5.2: Evaluation for 3-dimensional GBM with various numbers of timesteps $N_T \in \{10, 20, 50, 100\}$.

5.5.2 Rough Volatility model

We consider a rough stochastic volatility model for an asset price process S_t, which satisfies the below SDE:

$$\begin{aligned} dS_t &= \sqrt{v_t} S_t dZ_t \\ v_t &:= \xi(t) \exp\left(\eta W_t^H - \frac{1}{2}\eta^2 t^{2H}\right) \end{aligned} \tag{5.12}$$

where $\xi(t)$ denotes the forward variance and W_t^H denotes the fBM given by

$$W_t^H := \int_0^t K(t-s) dW_s, \quad K(r) := \sqrt{2H} r^{H-1/2}$$

where Z_t, W_t are (possibly correlated) Brownian motions. In our experiments, the synthetic dataset is sampled from (5.12) with $t \in [0, 1], H = 0.25, \xi(t) = 0.25, \eta = 0.5, \rho = 0$ where $\rho dt = d\langle W, Z \rangle_t$. Each sampled path is a stream of data of 20 points sampled uniformly between $[0, 1]$.

For the training details, we train the generators to learn

- the log price distribution,
- the log price and the log volatility joint distributions

using WGAN and Sig-W1 GAN. In both learning algorithms we scale the log-price paths by 2 and the log-volatiliy by 0.5 so that they have similar variance. In addition, in the SigWGAN we augment the paths by adding time dimension and visibility transformation before computing the expected signature up to degree 4. For a fair comparison, we train the Wasserstein GAN and the Sig-W1 GAN for $2\,500$ iterations of the generator. The Sig-W1 GAN provides an explicit form for the Sig-W1 distance between two path distributions, which implies that the overall number of gradient steps is $2\,500$. However, training the Wasserstein GAN involves training the generator and the discriminator; in our settings, for every gradient step of the generator, we take three gradient steps on the discriminator, thus the overall number of gradient steps is $10\,000$.

Table 5.3 shows the evaluation metrics of the trained generators.

Sig-W1 GANs outperform GANs in all evaluation metrics, except for the training time in the case of the LSTM generator. The LSTM and the LogSigRNN generators have similar performances in terms of correlation metric, SigW1 metric and Marginal Distribution metric.

	W1		Sig-W1	
Data $(X_t)_t$	(S_t)	(S_t, v_t)	(S_t)	(S_t, v_t)
SigW1 Metric				
LSTM	0.29	0.42	**0.07**	**0.17**
LogsigRNN	0.20	1.43	0.11	0.32
Correlation Metric(1e-3)				
LSTM	4.76	5.21	1.58	8.23
LogsigRNN	2.19	8.19	**1.09**	**4.82**
Marginal Distribution Metric(1e-2)				
LSTM	2.37	1.95	1.62	4.83
LogsigRNN	1.17	9.25	**1.38**	**4.03**
Training time(seconds)				
LSTM	**158**	**205**	164	256
LogsigRNN	814	845	352	452

Table 5.3: The test metrics of the trained models on a one dimensional price data $(S_t)_{t\in[0,T]}$ and two dimensional price and volatility data $(S_t, v_t)_{t\in[0,T]}$ respectively.

Regarding the training time, the LSTM generator is marginally cheaper to train with the W1-GAN algorithm, even if the training of the W1-GAN involves 4x more gradient steps than the Sig-W1 GAN. This is due to the fact that calculating the signature of the augmented path in the Sig-W1 GAN algorithm accounts for the additional iterations taken to train the W1-GAN. However, for a more complex generator as is the case of the LogSigRNN, the Sig-W1 GAN is 2x cheaper than the W1-GAN in terms of training time.

Correlation metric comparison

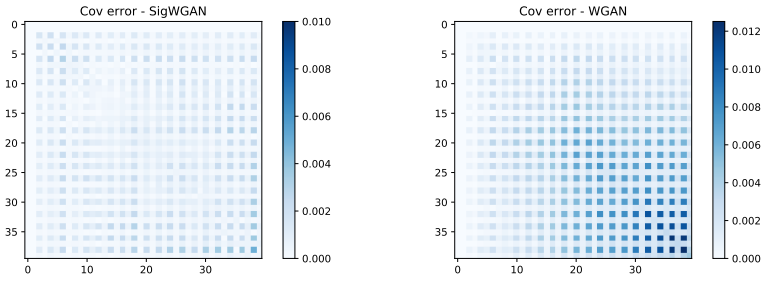
The error plot of the covariance matrix $\left(\text{cov}(X_s^{(i)}, X_t^{(j)})\right)_{i,j\in\{1,2\},s,t\in\{1,\cdots,N_T\}}$ in Figure 5.4 clearly shows that the combination of the Logsig-RNN and SigW1 metric are able to capture the temporal and spatial dependency of the rough volatility data best. The WGAN struggles to the temporal covariance of the volatility data.

Robustness to the variable sampling frequency

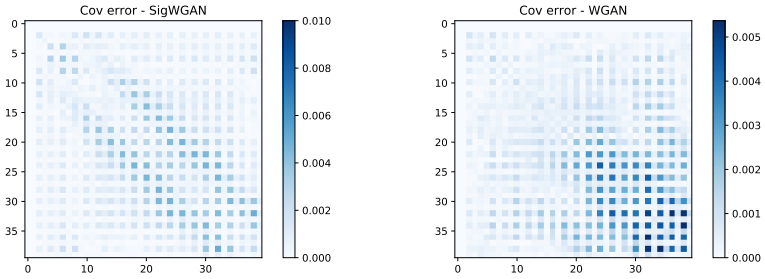
We test the robustness of the trained generator by generating data streams with a different frequency than the frequency used for training and we test them on synthetic data generated from (5.12) with the same new frequency. Namely, we generate data streams of 30 points on $[0, 1]$, whilst training was performed on data streams of 20 points on $[0, 1]$. Table 5.4 shows how in this new setting Sig-W1 GANs outperform GANs by two orders of magnitude. It is also verified qualitatively and visually by comparing the sample trajectories of the real paths and the synthetic path generated by the SigWGAN and WGAN using the same LogsigRNN generator in Figure 5.5.

	W1	Sig-W1
Data $(X_t)_t$	(S_t, v_t)	(S_t, v_t)
Correlation Metric(1e-4)		
LSTM	358.84	8.32
LogsigRNN	411.39	5.02

Table 5.4: The test metrics of the trained models on two dimensional price and volatility data $(S_t, v_t)_{t\in[0,T]}$. Models are trained on data streams sampled every $\frac{T}{20}$ units of time, and evaluated on data streams sampled every $\frac{T}{30}$ units of time.

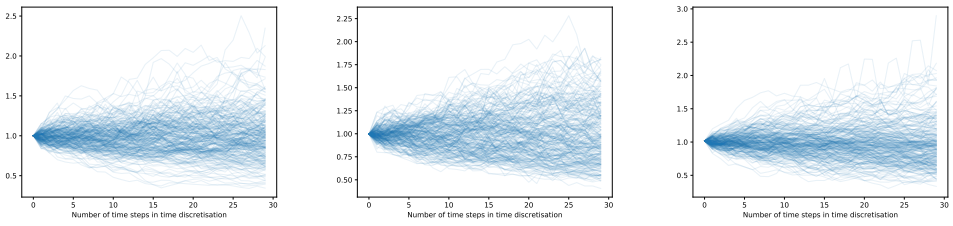


(a) LogsigRNN



(b) LSTM

Figure 5.4: The upper/lower figures are the covariance error plots of the Logsig-RNN/LSTM generators for each dimension (S_t, v_t) and each timestep. From Left to Right, each heatmap displays the covariance error of the Sig-WGAN and the WGAN respectively.



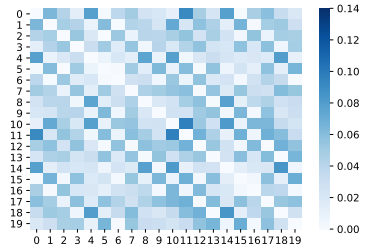
(a) Real paths of new frequency (b) SigWGAN+LogsigRNN (c) WGAN+LogSigRNN

Figure 5.5: From left to right, generated paths with new frequency (i.e. 30 time steps), generated paths using the trained LogSigRNN with Sig-W1 GAN on the old frequency (20 time steps), generated paths using the trained LogSigRNN with Wasserstein-GAN on the old frequency.

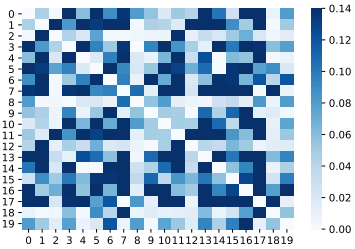
5.5.3 S&P 500 and DJI Market Data

We validate our proposed SigWGAN model on the empirical financial data. Here we choose daily spot-prices of the S&P 500 and DJI from January 1st 2005 to June 1st 2020. This dataset is from the Oxford MAN Realized Library. We compute the log return series from the close price of both S&P 500 and DJI, then apply rolling window to get 3856 samples of log return paths of 10 time steps. We use 80% samples as the training set and the rest 20% as the test set.

To train the SigWGAN, we use the basepoint path augmentation, cumulative sum and the lead-lag transformation. According to Table 5.5, for a fixed generator (LSTM/LogsigRNN), the SigWGAN improves the WGAN in terms of both the correlation metric and marginal distribution metrics. Besides, SigWGAN is more efficient than WGAN, which is illustrated by the half reduced training time when using the Logsig-RNN generator. Although the SigWGAN with the Logsig-RNN generator takes a bit



(a) SigWGAN + LogsigRNN



(b) WGAN + LogsigRNN

Figure 5.6: The correlation error of synthetic data generated by LogsigRNN using SigWGAN and WGAN resp. for stock data.

longer to train than the other models, it can significantly reduce the correlation metric from 0.5982 to 0.5031 and the marginal distribution metric from 0.1752 to 0.1304 compared with the second best model.

Type	W1	Sig-W1
SigW1 Metric(1e-4)		
LSTM	5.0632	**2.7510**
LogsigRNN	5.6269	2.8161
Correlation Metric(1e-1)		
LSTM	0.6422	0.5161
LogsigRNN	0.5982	**0.5031**
Marginal Distribution Metric(1e-2)		
LSTM	0.2001	0.1513
LogsigRNN	0.1752	**0.1304**
Training time(seconds)		
LSTM	0.138882	**0.103918**
LogsigRNN	0.472448	0.210008

Table 5.5: The test metrics of the trained models on the log price of S&P 500 index and S&P 500 and DJI index.

Correlation metric comparison

The error plot of the correlation matrix in Figure 5.6 clearly shows that the combination of the Logsig-RNN and SigW1 metric are able to capture the temporal and spatial dependency of the stock data best. However, the error plot of correlation matrix of Logsig-RNN and W1 metric shows more dark areas. Hence the combination of Logsig-RNN and W1 metric struggles to the temporal correlation of the volatility data.

Chapter 6

Concluding remarks

In this thesis we studied the application of Machine Learning to provide a new perspective on classical questions that arise in quantitative finance. We presented two main pieces of work, each divided in two chapters (a) Unbiased approximation of (path-dependent) linear PDEs with an application to option pricing and option hedging, and (b) Time series generative methods, with an application to data-driven market generators.

The common denominator in our methodology is that we employ deep neural networks as a powerful function approximation tool, but we rely on known results from Probability and Stochastic Analysis so that our results are robust. In Chapters 2 and 3 we create an unbiased (path-dependent) PDE solver by leveraging the probabilistic representation of a (P)PDE and deep neural networks. In Chapter 4 we create a data-driven market generator that learns the market dynamics under the risk-neutral measure and that in addition provides robust bounds for illiquid option prices not present in the training data. In Chapter 5 we introduce a generative model that learns the distribution of price trajectories, using the path signature as a central feature of our algorithm, an object that arises from Rough Path theory that keeps a surprising amount of information from a path. Extensive numerical experiments show the accuracy of our results.

During these last years there has been huge progress in the application of Machine Learning to quantitative finance. [162] incorporates some of the latest Machine Learning and Data Science research with application to finance into a book. These are just some of the directions in which the works of this thesis can be extended. For example, our unbiased linear (path dependent) PDE solvers could be extended to include semi-linear or non linear PDEs with applications to solve Stochastic Control problems. Another example is the application of market generative models in model-based Reinforcement Learning, so that so that one can train trading agents pursuing different tailored goals.

Bibliography

[1] Jiequn Han, Arnulf Jentzen, et al. Solving high-dimensional partial differential equations using deep learning. *arXiv:1707.02568*, 2017.

[2] E Weinan, Jiequn Han, and Arnulf Jentzen. Deep Learning-based numerical methods for high-dimensional parabolic partial differential equations and Backward Stochastic Differential equations. *Communications in Mathematics and Statistics*, 5(4):349–380, 2017.

[3] Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*, 2020.

[4] Valentin De Bortoli, James Thornton, Jeremy Heng, and Arnaud Doucet. Diffusion schrödinger bridge with applications to score-based generative modeling. *Advances in Neural Information Processing Systems*, 34:17695–17709, 2021.

[5] Freddy Delbaen and Walter Schachermayer. A general version of the fundamental theorem of asset pricing. *Mathematische annalen*, 300(1):463–520, 1994.

[6] Marc Sabate Vidales, David Šiška, and Lukasz Szpruch. Unbiased deep solvers for linear parametric PDEs. *Applied Mathematical Finance*, 28(4):299–329, 2021.

[7] Patryk Gierjatowicz, Marc Sabate-Vidales, David Siska, Lukasz Szpruch, and Zan Zuric. Robust pricing and hedging via Neural Stochastic Differential Equations. *Journal of Computational Finance*, 26(3), 2022.

[8] Hao Ni, Lukasz Szpruch, Marc Sabate-Vidales, Baoren Xiao, Magnus Wiese, and Shujian Liao. Sig-Wasserstein GANs for time series generation. In *Proceedings of the Second ACM International Conference on AI in Finance*, pages 1–8, 2021.

[9] Christian Beck, Lukas Gonon, and Arnulf Jentzen. Overcoming the curse of dimensionality in the numerical approximation of high-dimensional semilinear elliptic Partial Differential Equations, 2020.

[10] Julius Berner, Philipp Grohs, and Arnulf Jentzen. Analysis of the generalization error: empirical risk minimization over deep artificial neural networks overcomes the curse of dimensionality in the numerical approximation of Black–Scholes Partial Differential Equations. *SIAM Journal on Mathematics of Data Science*, 2(3):631–657, Jan 2020.

[11] Jaksa Cvitanic, Jianfeng Zhang, et al. The steepest descent method for forward-backward SDEs. *Electronic Journal of Probability*, 10:1468–1495, 2005.

[12] Marc Sabate-Vidales, David Šiška, and Lukasz Szpruch. Solving path dependent PDEs with LSTM networks and path signatures. *arXiv:2011.10630*, 2020.

[13] Ian J. Goodfellow, Jonathan Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv:1412.6572*, 2014.

[14] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436, 2015.

[15] Justin Sirignano and Konstantinos Spiliopoulos. DGM: A deep learning algorithm for solving Partial Differential Equations. *arXiv:1708.07469*, 2017.

[16] Quentin Chan-Wai-Nam, Joseph Mikael, and Xavier Warin. Machine Learning for semi linear PDEs. *Journal of Scientific Computing*, 79(3):1667–1712, 2019.

[17] Côme Huré, Huyên Pham, and Xavier Warin. Some Machine Learning schemes for high-dimensional nonlinear PDEs. *arXiv:1902.01599*, 2019.

[18] Christian Beck, Sebastian Becker, Philipp Grohs, Nor Jaafari, and Arnulf Jentzen. Solving Stochastic Differential Equations and Kolmogorov equations by means of deep learning. *arXiv preprint arXiv:1806.00421*, 2018.

[19] Antoine Jacquier and Mugad Oumgari. Deep PPDEs for rough local stochastic volatility. *arXiv:1906.02551*, 2019.

[20] Pierre Henry-Labordere. Deep Primal-Dual Algorithm for BSDEs: Applications of Machine Learning to CVA and IM. 2017.

[21] Philipp Grohs, Fabian Hornung, Arnulf Jentzen, and Philippe von Wurstemberger. A proof that artificial neural networks overcome the curse of dimensionality in the numerical approximation of Black-Scholes partial differential equations. *arXiv:1809.02362*, 2018.

[22] Arnulf Jentzen, Diyora Salimova, and Timo Welti. A proof that deep artificial neural networks overcome the curse of dimensionality in the numerical approximation of Kolmogorov partial differential equations with constant diffusion and nonlinear drift coefficients. *arXiv:1809.07321*, 2018.

[23] Kaitong Hu, Zhenjie Ren, David Šiška, and Lukasz Szpruch. Mean-Field Langevin dynamics and energy landscape of neural networks. *Annales de l'Institute Henry Poincaré Proability and Statistics*, 57(4):2043–2065, 2021.

[24] Song Mei, Andrea Montanari, and Phan-Minh Nguyen. A Mean Field view of the landscape of two-layer neural networks. *Proceedings of the National Academy of Sciences*, 115(33):E7665–E7671, 2018.

[25] Simon S Du, Jason D Lee, Haochuan Li, Liwei Wang, and Xiyu Zhai. Gradient descent finds global minima of deep neural networks. *arXiv preprint arXiv:1811.03804*, 2018.

[26] Lenaic Chizat and Francis Bach. On the global convergence of gradient descent for over-parameterized models using optimal transport. In *Advances in neural information processing systems*, pages 3040–3050, 2018.

[27] Grant M Rotskoff and Eric Vanden-Eijnden. Neural networks as interacting particle systems: Asymptotic convexity of the loss landscape and universal scaling of the approximation error. *arXiv:1805.00915*, 2018.

[28] Justin Sirignano and Konstantinos Spiliopoulos. Mean field analysis of neural networks: A central limit theorem. *Stochastic Processes and their Applications*, 2019.

[29] Zixuan Wang and Shanjian Tang. Gradient convergence of deep learning-based numerical methods for bsdes. *Chinese Annals of Mathematics, Series B*, 42(2):199–216, 2021.

[30] Jiequn Han and Jihao Long. Convergence of the deep BSDE method for coupled FBSDEs. *Probability, Uncertainty and Quantitative Risk*, 5(1):1–33, 2020.

[31] Blanka Horvath, Aitor Muguruza, and Mehdi Tomas. Deep Learning volatility. *arXiv preprint arXiv:1901.09647*, 2019.

[32] Shuaiqiang Liu, Anastasia Borovykh, Lech A Grzelak, and Cornelis W Oosterlee. A neural network-based framework for financial model calibration. *Journal of Mathematics in Industry*, 9(1):9, 2019.

[33] Christian Bayer and Benjamin Stemper. Deep calibration of rough stochastic volatility models. *arXiv:1810.03399*, 2018.

[34] Henry Stone. Calibrating rough volatility models: a convolutional neural network approach. *arXiv preprint arXiv:1812.05315*, 2019.

[35] Andres Hernandez. Model calibration with neural networks. *Available at SSRN 2812140*, 2016.

[36] A Itkin. Deep learning calibration of option pricing models: some pitfalls and solutions. *arXiv:1906.03507*, 2019.

[37] William A McGhee. An artificial neural network representation of the SABR stochastic volatility model. *Available at SSRN 3288882*, 2018.

[38] Nigel J Newton. Variance reduction for simulated diffusions. *SIAM Journal on Applied Mathematics*, 54(6):1780–1805, 1994.

[39] G Milstein and M Tretyakov. Solving parabolic stochastic partial differential equations via averaging over characteristics. *Mathematics of Computation*, 78(268):2075–2106, 2009.

[40] Denis Belomestny, Stefan Hafner, Tigran Nagapetyan, and Mikhail Urusov. Variance reduction for discretised diffusions via regression. *Journal of Mathematical Analysis and Applications*, 458(1):393–418, 2018.

[41] Geoffrey Grimmett and David Stirzaker. *Probability and random processes*. Oxford university press, 2020.

[42] NV Krylov. On Kolmogorov's equations for finite dimensional diffusions. In *Stochastic PDE's and Kolmogorov Equations in Infinite Dimensions*, pages 1–63. Springer, 1999.

[43] Bernt Øksendal. Stochastic differential equations. In *Stochastic differential equations*, pages 65–84. Springer, 2003.

[44] Dennis Elbrächter, Philipp Grohs, Arnulf Jentzen, and Christoph Schwab. DNN expression rate analysis of high-dimensional PDEs: Application to option pricing. *Constructive Approximation*, May 2021.

[45] Philipp Grohs, Arnulf Jentzen, and Diyora Salimova. Deep neural network approximations for Monte Carlo algorithms. *arXiv preprint arXiv:1908.10828*, 2019.

[46] Martin Hutzenthaler, Arnulf Jentzen, Thomas Kruse, and Tuan Anh Nguyen. A proof that rectified deep neural networks overcome the curse of dimensionality in the numerical approximation of semilinear heat equations. *SN Partial Differential Equations and Applications*, 1(2), Apr 2020.

[47] Gitta Kutyniok, Philipp Petersen, Mones Raslan, and Reinhold Schneider. A theoretical analysis of deep neural networks and parametric PDEs. *arXiv:1904.00377*, 2020.

[48] Lukas Gonon, Philip Grohs, Arnulf Jentzen, David Kofler, and David Šiška. Uniform error estimates for artificial neural network approximations for the heat equation. *IMA Journal Numerical Analysis*, 2021.

[49] Christoph Reisinger and Yufei Zhang. Rectified deep neural networks overcome the curse of dimensionality for nonsmooth value functions in zero-sum games of nonlinear stiff systems. *arXiv:1903.06652*, 2020.

[50] Paul Glasserman. *Monte Carlo methods in financial engineering*. Springer, 2013.

[51] Samuel N Cohen and Robert J Elliott. *Stochastic calculus and applications*. Springer, 2015.

[52] N. V. Krylov. *Introduction to the theory of random processes*. American Mathematical Society, 2002.

[53] Côme Huré, Huyên Pham, and Xavier Warin. Deep backward schemes for high-dimensional nonlinear PDEs. *arXiv:1902.01599*, 2020.

[54] Christian Szegedy Sergey Ioffe. Batch normalization: accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[55] Jimmy Ba Diederik P. Kingma. Adam: a method for stochastic optimization. *arXiv:1412.6980*, 2017.

[56] A Pranklin. *Introduction to the Theory of Statistics*. 1974.

[57] William Margrabe. The value of an option to exchange one asset for another. *The journal of finance*, 33(1):177–186, 1978.

[58] D Belomestny, L Iosipoi, and N Zhivotovskiy. Variance reduction via empirical variance minimization: convergence and complexity. *arXiv:1712.04667*, 2017.

[59] Mark Broadie, Yiping Du, and Ciamac C Moallemi. Risk estimation via regression. *Operations Research*, 63(5):1077–1097, 2015.

[60] Rama Cont and Yi Lu. Weak approximation of Martingale representations. *Stochastic Processes and their Applications*, 126(3):857–882, Mar 2016.

[61] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep Learning*. MIT press, 2016.

[62] Christian Beck, Sebastian Becker, Patrick Cheridito, Arnulf Jentzen, and Ariel Neufeld. Deep splitting method for parabolic PDEs. *arXiv:1907.08452*, 2019.

[63] Bruno Dupire. Functional Itô calculus. *Quantitative Finance*, 19(5):721–729, 2019.

[64] Rama Cont and David Fournie. A functional extension of the Ito formula. *Comptes Rendus Mathematique*, 348(1-2):57–61, 2010.

[65] Huyen Pham, Xavier Warin, and Maximilien Germain. Neural networks-based backward scheme for fully nonlinear PDEs. *arXiv preprint arXiv:1908.00412*, 2019.

[66] Christian Beck, Weinan E, and Arnulf Jentzen. Machine learning approximation algorithms for high-dimensional fully nonlinear partial differential equations and second-order backward stochastic differential equations. *Journal of Nonlinear Science*, 29:1563–1619, 2019.

[67] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

[68] Weinan E, Jiequn Han, and Arnulf Jentzen. Algorithms for solving high dimensional PDEs: from nonlinear Monte Carlo to Machine Learning. *arXiv:2008.13333*, 2020.

[69] Yuri F. Saporito and Zhaoyu Zhang. PDGM: a neural network approach to solve path-dependent partial differential equations. *arXiv:2003.02035*, 2020.

[70] Shuaiqiang Liu, Anastasia Borovykh, Lech A. Grzelak, and Cornelis W. Oosterlee. A neural network-based framework for financial model calibration. *Journal of Mathematics in Industry*, 9(1), Sep 2019.

[71] Christa Cuchiero, Wahid Khosrawi, and Josef Teichmann. A generative adversarial network approach to calibration of local stochastic volatility models. *Risks*, 8(4):101, 2020.

[72] Vlad Bally, Lucia Caramellino, Rama Cont, Frederic Utzet, and Josep Vives. *Stochastic integration by parts and functional Itô calculus*. Springer, 2016.

[73] Rama Cont and David-Antoine Fournié. Functional Itô calculus and stochastic integral representation of martingales. *The Annals of Probability*, 41(1):109–133, Jan 2013.

[74] Terry J Lyons, Michael Caruana, and Thierry Lévy. *Differential equations driven by rough paths*. Springer, 2007.

[75] Ilya Chevyrev and Andrey Kormilitzin. A primer on the signature method in machine learning. *arXiv preprint arXiv:1603.03788*, 2016.

[76] Ben Hambly and Terry Lyons. Uniqueness for the signature of a path of bounded variation and the reduced path group. *Annals of Mathematics*, pages 109–167, 2010.

[77] Daniel Levin, Terry Lyons, and Hao Ni. Learning from the past, predicting the statistics for the future, learning an evolving system. *arXiv:1309.0260*, 2013.

[78] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An Imperative Style, High-Performance Deep Learning Library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[79] Patrick Kidger and Terry Lyons. Signatory: differentiable computations of the signature and logsignature transforms, on both CPU and GPU. *arXiv:2001.00706*, 2021.

[80] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[81] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[82] Andres Hernandez. Model calibration with neural networks. *Risk*, 2016.

[83] Johannes Ruf and Weiguan Wang. Neural networks for option pricing and hedging: a literature review. *Available at SSRN 3486363*, 2019.

[84] Fred Espen Benth, Nils Detering, and Silvia Lavagnini. Accuracy of deep learning in calibrating HJM forward curves. *arXiv preprint arXiv:2006.01911*, 2020.

[85] Matteo Gambara and Josef Teichmann. Consistent recalibration models and deep calibration. *arXiv preprint arXiv:2006.09455*, 2020.

[86] Wahid Khosrawi Sardroudi. Polynomial semimartingales and a deep learning approach to local stochastic volatility calibration, 2019.

[87] Blanka Horvath, Aitor Muguruza, and Mehdi Tomas. Deep learning volatility: a deep neural network perspective on pricing and calibration in (rough) volatility models. *Quantitative Finance*, 21(1):11–27, 2021.

[88] Christian Bayer, Blanka Horvath, Aitor Muguruza, Benjamin Stemper, and Mehdi Tomas. On deep calibration of (rough) stochastic volatility models. *arXiv:1908.08806*, 2019.

[89] Marco Avellaneda, Craig Friedman, Richard Holmes, and Dominick Samperi. Calibrating volatility surfaces via relative-entropy minimization. *Applied Mathematical Finance*, 4(1):37–64, mar 1997.

[90] Ivan Guo, Grégoire Loeper, and Shiyi Wang. Local volatility calibration by optimal transport. *2017 MATRIX Annals, Vol. 2, 2019, 51-64*, September 2017.

[91] Ivan Guo, Gregoire Loeper, and Shiyi Wang. Calibration of local-stochastic volatility models by optimal transport. *arXiv:1906.06478*, June 2019.

[92] Ivan Guo and Gregoire Loeper. Path dependent optimal transport and model calibration on exotic derivatives. *arXiv:1812.03526*, December 2018.

[93] Frank H Knight. Risk, uncertainty and profit, 1921. *Library of Economics and Liberty*, 1971.

[94] Samuel N Cohen et al. Data and uncertainty in extreme risks-a nonlinear expectations approach. *World Scientific Book Chapters*, pages 135–162, 2018.

[95] David G Hobson. Robust hedging of the lookback option. *Finance and Stochastics*, 2(4):329–347, 1998.

[96] Alexander MG Cox and Jan Obloj. Robust hedging of double touch barrier options. *SIAM Journal on Financial Mathematics*, 2(1):141–182, 2011.

[97] Mathias Beiglböck, Pierre Henry-Labordère, and Friedrich Penkner. Model-independent bounds for option prices—a mass transport approach. *Finance and Stochastics*, 17(3):477–501, 2013.

[98] Stephan Eckstein, Gaoyue Guo, Tongseok Lim, and Jan Obloj. Robust pricing and hedging of options on multiple assets and its numerics. *arXiv preprint arXiv:1909.03870*, 2019.

[99] Sergey Nadtochiy and Jan Obloj. Robust trading of implied skew. *International Journal of Theoretical and Applied Finance*, 20(02):1750008, 2017.

[100] Anna Aksamit, Zhaoxu Hou, and Jan Obloj. Robust framework for quantifying the value of information in pricing and hedging. *SIAM Journal on Financial Mathematics*, 11(1):27–59, 2020.

[101] Mark Broadie, Yiping Du, and Ciamac C Moallemi. Efficient risk estimation via nested sequential simulation. *Management Science*, 57(6):1172–1194, 2011.

[102] Antoon Pelsser and Janina Schweizer. The difference between LSMC and replicating portfolio in insurance liability modeling. *European Actuarial Journal*, 6(2):441–494, 2016.

[103] Hans Buehler, Lukas Gonon, Josef Teichmann, and Ben Wood. Deep hedging. *Quantitative Finance*, pages 1–21, 2019.

[104] N. V. Krylov. *Controlled diffusion processes*, volume 14 of *Applications of Mathematics*. Springer-Verlag, New York-Berlin, 1980. Translated from the Russian by A. B. Aries.

[105] K. Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, 1991.

[106] Eduardo Sontag and Héctor Sussmann. Complete controllability of continuous-time recurrent neural networks. *Systems Control Lett.*, 30(4):177–183, 1997.

[107] Christa Cuchiero, Martin Larsson, and Joseph Teichmann. Deep neural networks, generic universal interpolation, and controlled ODEs. *arXiv preprint arXiv:1908.07838*, 2019.

[108] Samuel N Cohen, Christoph Reisinger, and Sheng Wang. Arbitrage-free neural-sde market models. *arXiv preprint arXiv:2105.11053*, 2021.

[109] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.

[110] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[111] Ioannis Karatzas and Steven Shreve. *Brownian motion and stochastic calculus*. Springer, 2012.

[112] Beatrice Acciaio, Julio Backhoff-Veraguas, and Anastasiia Zalashko. Causal optimal transport and its links to enlargement of filtrations and continuous-time stochastic optimization. *Stochastic Processes and their Applications*, 2019.

[113] Rémi Lassalle. Causal transport plans and their Monge–Kantorovich problems. *Stochastic Analysis and Applications*, 36(3):452–484, jan 2018.

[114] Stephan Eckstein and Michael Kupper. Computation of optimal transport and related hedging problems via penalization and neural networks. *Applied Mathematics & Optimization*, pages 1–29, 2019.

[115] David Šiška and Łukasz Szpruch. Gradient flows for regularized stochastic control problems. *arxiv preprint arXiv:2006.05956*, 2020.

[116] Martin Hutzenthaler, Arnulf Jentzen, and Peter E Kloeden. Strong and weak divergence in finite time of euler's method for stochastic differential equations with non-globally lipschitz continuous coefficients. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 467(2130):1563–1576, 2011.

[117] Martin Hutzenthaler, Arnulf Jentzen, Peter E Kloeden, et al. Strong convergence of an explicit numerical method for SDEs with nonglobally Lipschitz continuous coefficients. *The Annals of Applied Probability*, 22(4):1611–1641, 2012.

[118] Łukasz Szpruch and Xiling Zhāng. V-integrability, asymptotic stability and comparison property of explicit numerical schemes for non-linear SDEs. *Math. Comp.*, 87(310):755–783, 2018.

[119] Magnus R. Hestenes. Multiplier and gradient methods. *Journal of Optimization Theory and Applications*, 4(5):303–320, nov 1969.

[120] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv:1412.6980*, 2014.

[121] Albert Benveniste, Michel Métivier, and Pierre Priouret. *Adaptive algorithms and stochastic approximations*, volume 22. Springer, 2012.

[122] Mateusz B Majka, Marc Sabate-Vidales, and Łukasz Szpruch. Multi-index antithetic stochastic gradient algorithm. *arXiv preprint arXiv:2006.06102*, 2020.

[123] Eric Fournié, Jean-Michel Lasry, Jérôme Lebuchoux, Pierre-Louis Lions, and Nizar Touzi. Applications of Malliavin calculus to Monte Carlo methods in finance. *Finance and Stochastics*, 3(4):391–412, 1999.

[124] Jean-François Jabir, David Šiška, and Łukasz Szpruch. Mean-field neural ODEs via relaxed optimal control. *arXiv:1912.05475*, 2019.

[125] Jean-François Chassagneux, Lukasz Szpruch, and Alvin Tse. Weak quantitative propagation of chaos via differential calculus on the space of measures. *arXiv:1901.02556*, 2019.

[126] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[127] Hiroshi Kunita. *Stochastic flows and stochastic differential equations*. Cambridge University Press, 1997.

[128] William RP Hammersley, David Šiška, and Łukasz Szpruch. Weak existence and uniqueness for mckean-vlasov sdes with common noise. *arXiv:1908.00955*, 2019.

[129] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[130] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pages 8024–8035, 2019.

[131] Yu Tian, Zili Zhu, Geoffrey Lee, Fima Klebaner, and Kais Hamza. Calibrating and pricing with a stochastic-local volatility model. *The Journal of Derivatives*, 22(3):21–39, 2015.

[132] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, nov 1997.

[133] Bruno Dupire et al. Pricing with a smile. *Risk*, 7(1):18–20, 1994.

[134] István Gyöngy. Mimicking the one-dimensional marginal distributions of processes having an Itô differential. *Probability theory and related fields*, 71(4):501–516, 1986.

[135] S. L. Heston. A closed-form solution for options with stochastic volatility and applications to bond and currency options. *The Review of Financial Studies*, 6:327–343, 1997.

[136] Hansjoerg Albrecher, Philipp Mayer, Wim Schoutens, and Jurgen Tistaert. The little Heston trap. *Wilmott*, pages 83–92, 03 2007.

[137] Samuel A Assefa, Danial Dervovic, Mahmoud Mahfouz, Robert E Tillman, Prashant Reddy, and Manuela Veloso. Generating synthetic data in finance: opportunities, challenges and pitfalls. In *Proceedings of the First ACM International Conference on AI in Finance*, pages 1–8, 2020.

[138] Steven M Bellovin, Preetam K Dutta, and Nathan Reitinger. Privacy and synthetic datasets. *Stan. Tech. L. Rev.*, 22:1, 2019.

[139] Hans Buehler, Blanka Horvath, Terry Lyons, Imanol Perez Arribas, and Ben Wood. A data-driven market simulator for small data environments. *Available at SSRN 3632431*, 2020.

[140] Adriano Koshiyama, Nick Firoozye, and Philip Treleaven. Generative adversarial networks for financial trading strategies fine-tuning and combination. *Quantitative Finance*, 0(0):1–17, 2020.

[141] Magnus Wiese, Lianjun Bai, Ben Wood, and Hans Buehler. Deep hedging: learning to simulate equity option markets. *Available at SSRN 3470756*, 2019.

[142] Magnus Wiese, Robert Knobloch, Ralf Korn, and Peter Kretschmer. Quant GANs: deep generation of financial time series. *Quantitative Finance*, 20(9):1419–1440, 4 2020.

[143] Imanol Perez Arribas, Cristopher Salvi, and Lukasz Szpruch. Sig-SDEs model for quantitative finance. *ICAIF 2020*, 2020.

[144] Patrick Kidger, James Morrill, James Foster, and Terry Lyons. Neural controlled differential equations for irregular time series. *Conference on Neural Information Processing Systems*, 2020.

[145] Hao Ni, Lukasz Szpruch, Magnus Wiese, Shujian Liao, and Baoren Xiao. Conditional Sig-Wasserstein GANs for time series generation. *arXiv preprint arXiv:2006.05421*, 2020.

[146] Shujian Liao, Terry Lyons, Weixin Yang, Kevin Schlegel, and Hao Ni. Logsig-RNN: a novel network for robust and efficient skeleton-based action recognition. *Accepted by British Machine Vision Conference*, 2021.

[147] James Morrill, Adeline Fermanian, Patrick Kidger, and Terry Lyons. A generalised signature method for multivariate time series feature extraction. *arXiv preprint arXiv:2006.00873*, 2020.

[148] Horatio Boedihardjo and Xi Geng. The uniqueness of signature problem in the non-markov setting. *Stochastic Processes and their Applications*, 125(12):4674–4701, 2015.

[149] Jeremy Reizenstein. Calculation of iterated-integral signatures and log signatures. *arXiv preprint arXiv:1712.02757*, 2017.

[150] Ilya Chevyrev and Terry Lyons. Characteristic functions of measures on geometric rough paths. *Annals of Probability*, 44(6):4049–4082, 2016.

[151] Eric V Mazumdar, Michael I Jordan, and S Shankar Sastry. On finding local nash equilibria (and only local nash equilibria) in zero-sum games. *arXiv preprint arXiv:1901.00838*, 2019.

[152] Tianyi Lin, Chi Jin, and Michael Jordan. On gradient descent ascent for nonconvex-concave minimax problems. In *International Conference on Machine Learning*, pages 6083–6093. PMLR, 2020.

[153] Constantinos Daskalakis and Ioannis Panageas. The limit points of (optimistic) gradient descent in min-max optimization. *arXiv preprint arXiv:1807.03907*, 2018.

[154] Constantinos Daskalakis, Andrew Ilyas, Vasilis Syrgkanis, and Haoyang Zeng. Training GANs with optimism. *arXiv preprint arXiv:1711.00141*, 2017.

[155] Panayotis Mertikopoulos, Christos Papadimitriou, and Georgios Piliouras. Cycles in adversarial regularized learning. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2703–2717. SIAM, 2018.

[156] Farzan Farnia and Asuman Ozdaglar. GANs may have no Nash equilibria. *arXiv preprint arXiv:2002.09124*, 2020.

[157] Mario Lucic, Karol Kurach, Marcin Michalski, Sylvain Gelly, and Olivier Bousquet. Are GANs created equal? a large-scale study. *arXiv preprint arXiv:1711.10337*, 2017.

[158] Ilya Chevyrev and Harald Oberhauser. Signature moments to characterize laws of stochastic processes. *arXiv preprint arXiv:1810.10971*, 2018.

[159] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[160] Shujian Liao, Terry Lyons, Weixin Yang, and Hao Ni. Learning stochastic differential equations using RNN with log signature features. *arXiv preprint arXiv:1908.08286*, 2019.

[161] Aaditya Ramdas, Nicolás García Trillos, and Marco Cuturi. On wasserstein two-sample testing and related families of nonparametric tests. *Entropy*, 19(2):47, 2017.

[162] *Machine Learning and Data Sciences for Financial Markets: A Guide to Contemporary Practices*. Cambridge University Press, 2023.