Accuracy of Deep Learning in Calibrating HJM Forward Curves

Fred Espen Benth[*] Nils Detering[†] Silvia Lavagnini[‡]

June 4, 2020

Abstract

We price European-style options written on forward contracts in a commodity market, which we model with a state-dependent infinite-dimensional Heath-Jarrow-Morton (HJM) approach. We introduce a new class of volatility operators which map the square integrable noise into the Filipović space of forward curves, and we specify a deterministic parametrized version of it. For calibration purposes, we train a neural network to approximate the option price as a function of the model parameters. We then use it to calibrate the HJM parameters starting from (simulated) option market data. Finally we introduce a new loss function that takes into account bid and ask prices and offers a solution to calibration in illiquid markets. A key issue discovered is that the trained neural network might be non-injective, which could potentially lead to poor accuracy in calibrating the forward curve parameters, even when showing a high degree of accuracy in recovering the prices. This reveals that the original meaning of the parameters gets somehow lost in the approximation.

Keywords: Neural networks; Model calibration; Heath-Jarrow-Morton approach; Volatility operator; Forward curve; Option pricing; Bid-ask price; Stochastic partial differential equation.

1 Introduction

We price European-style options written on forward contracts in a commodity market. We follow the Heath-Jarrow-Morton (HJM) approach and model the forward curve by a stochastic partial differential equation with state-dependent volatility, and having values in the Filipović space. In our setting, the Hilbert valued Wiener process driving the noise of the forward curve takes values in $L^2(\mathcal{O})$, $\mathcal{O}$ being some Borel subset of $\mathbb{R}$ (possibly $\mathbb{R}$ itself). This requires that the volatility operator must smoothen elements in $L^2(\mathcal{O})$ into elements of the Filipović space. We achieve this by constructing the volatility as an integral operator with respect to some kernel function, and derive the conditions needed on the kernel function such that the volatility operator is well defined. We then focus on the pricing of forward contracts with delivery period, also called swaps. Typical examples are forward contracts in the electricity market, such as the ones traded at Nord Pool AS and the European Energy Exchange (EEX). For a deterministic

The authors would like to thank Vegard Antun for precious coding support and related advice, and Christian Bayer for useful discussions. Part of the project has been carried out during Silvia Lavagnini's three months visit at UCSB, funded by the Kristine Bonnevie travel stipend 2019 from the Faculty of Mathematics and Natural Sciences (University of Oslo).

[*] Department of Mathematics, University of Oslo, 0316 Blindern, Norway. Email: fredb@math.uio.no.

[†] Department of Statistics and Applied Probability, University of California, Santa Barbara, CA 93106, USA. Email: detering@pstat.ucsb.edu.

[‡] Department of Mathematics, University of Oslo, 0316 Blindern, Norway. Email: silval@math.uio.no.

volatility structure, we derive analytic pricing formulas based on a representation theorem for swaps presented in [11]. For a state-dependent stochastic volatility operator one needs instead to resort to simulation schemes for stochastic partial differential equations, as for example the Multilevel Monte Carlo method [3, 4].

Our fully parametrized model allows for pricing and calibration based on deep neural networks. Therefore we adopt the approach presented in [7] to our setting, and approximate the pricing functional with a neural network in a (possibly) costly off-line step. After training, the neural network is used to recover the model parameters in an optimization routine. The calibrated parameters can then be used together with the trained neural network for pricing options that are not traded on the exchange. Resorting to neural networks for both, the calibration and the pricing step, offers critical computational advantages for those models requiring more expensive techniques, such as Monte Carlo techniques. In fact, in this neural network approach, the computationally expensive simulation is only required to generate the training set in the learning process of the network, and is not needed for intraday calibration and pricing.

We perform a comprehensive case study of our framework and analyse the accuracy of neural networks for pricing and calibration in the infinite dimensional HJM setting. To avoid a training set based on large scale Monte Carlo simulation, which introduces additional sources of error, we restrict our focus on a deterministic but time dependent volatility operator. To our knowledge, this is indeed the first application of deep neural networks in an infinite dimensional HJM setup for the purpose of model calibration. We consider two different approaches both presented in [29], the pointwise and the grid-based learning approach, and we compare dense and convolutional neural network architectures. We then extend the framework to allow for calibration in markets with a wide bid and ask spread, where using a mid price is not feasible. The problem of wide bid-ask spreads is particularly pronounced in energy markets, since only the front end swaps are traded liquidly.

In the approximation step, we observe a high degree of accuracy, with an average relative error for the test set in the range 0.3%-3%. The picture is different when it comes to calibration. Here the trained neural network might fail to recover the true parameters, and we do indeed observe average relative errors reaching almost 50% in some cases. On the other hand, the prices estimated after calibration have an average accuracy around 5%. This failure in recovering the parameters is the result of two effects, one specific to the model and one to the network. In the specified model, several parameter vectors lead to similar prices for the training set, making it difficult to recover the true parameters. However, we also show that the trained neural network can be non-injective in the input parameters to a degree not justified by the original model. For instance, keeping all but the volatility scaling fixed, the call price should be an increasing function of volatility. This in fact is not always the case. Given that no-arbitrage conditions are not imposed on the neural network, this is actually not surprising, but contributes to the problem of calibration. It may cause the original meaning of the parameters to get lost in the approximation step, and shows that careful benchmarking is required when using the neural network approach for calibration, in particular for pricing more complicated options.

For the calibration in market environments with large bid-ask spread, the simple loss function proposed here seems to work well. We test it with respect to different bid-ask spread sizes, and in fact, after calibration, almost all prices lie within the bid-ask range and only few are outside, but still very close to either the bid or the ask price. In particular, whenever the bid-ask spread becomes more narrow, one can simply increase the number of iterations for the optimization routine to obtain good results. The optimization is fast because it does not require any simulation. Moreover, the observed errors in recovering the parameters are not increasing dramatically as compared to the calibration based on a zero bid-ask spread.

The rest of the article is organized as follows. In Section 1.1 we give some background and motivations on the model considered here. In Section 2 we define the HJM forward curve dynamics and our volatility operators, and we specify a deterministic version of it. In Section 3 we introduce forward contracts with delivery period and options written on them, and we derive the pricing formulas used in our case study. In Section 4 we define neural networks and introduce the two steps approach for model calibration, together with the newly proposed bid-ask loss function. Finally, in Section 5 we specify the parametric setting for the experiments, and in Section 6 we show our findings. Appendix A contains the proofs to all results.

1.1 Background and motivations

In energy markets, forwards and futures on power and gas deliver the underlying commodity over a period of time, rather than at a fixed delivery time. While one usually derives the arbitrage-free forward price in a model free way from the buy-and-hold strategy in the underlying, in energy markets this strategy can not be applied because storing electricity is costly. This implies that the forward price as derived from the spot price model is not backed by a replication strategy. Instead, what is often adopted as alternative in energy markets is the direct modelling of the tradable forward price. This is referred to as the Heath-Jarrow-Morton (HJM) approach, as it has first been introduced by [25] for interest rate markets. Later this idea has been transferred to other markets. [15] and [30], for example, model the whole call option price surface using the HJM methodology; [16] and [9] transferred the approach to commodity forward markets, the latter one, in particular, in the context of power markets.

Another important characteristic of energy forward markets is the high degree of idiosyncratic risk across different maturities, which has been observed by several studies, such as [1, 31, 22]. In [10], for example, the authors performed a Principal Component Analysis on the Nord Pool AS forward contracts, revealing that more than ten factors are needed to explain 95% of the volatility. This points out the necessity of modelling the time dynamics of the forward curve by a high dimensional, possibly infinite dimensional, noise process. In [12] the authors show that a reasonable state space for the forward curve is the so-called Filipović space, which is a separable Hilbert space first introduced by [21]. One can indeed realize energy forward prices as linear operators in this space, as done in [11] and [12].

In particular, when considering stochastic volatility operators, as for example in our state-dependent model, it is in general not possible to have closed price formulas for options written on the forward curve. Hence one has to resort to time consuming numerical methods, such as Monte Carlo techniques for SPDEs (see [3, 4]) or Finite Elements methods (see [2, 33]). In particular, such costly pricing procedures render calibration almost impossible as the pricing function has to be evaluated in each step of the optimization routine. As a result, more accurate models are often not used in practice.

To overcome the computational burden, machine learning may be useful. Machine learning is in fact replacing standard techniques in many different fields within scientific computing, and, in particular, within finance. In [20], for example, machine learning is used for the evaluation of derivatives, and in [14] for the problem of hedging a portfolio of derivatives with market frictions; in [19] and [32] machine learning tools are employed to learn the volatility surface or the term structure of forward crude oil prices, respectively; [38] and [26] propose approaches for solving Backward Stochastic Differential Equations in high dimension with deep neural networks. In the context of model calibration, a first application is proposed in [27], who calibrates stochastic models by training a neural network which, given market price observations, returns directly the optimal model parameters. Recently, [17] suggested to employ tools from generative adversarial networks to solve the calibration problem in the context of local stochastic volatility models.

Yet another strategy which is also employed here can be found in [6, 29, 7], where the authors introduce the following two steps approach. They propose to approximate the implied volatility surface, respective the price functional by a neural network in an off-line training step. Then, they use the trained neural network to calibrate the stochastic model to fit market observations. The training step is a priori cost demanding, especially if the underlying stochastic model is complex, such as for a stochastic volatility model, since it requires many costly simulations. But, even if generating the artificial data as well as training the neural network are expensive tasks, the advantage is that these are performed one-off. To ensure that the model reflects the current market situation, it is sufficient to run the calibration step regularly, say daily or even intra-daily. This step requires only evaluation of the neural network and is therefore fast.

2 The forward curve dynamics

Let $(\Omega, \mathcal{F}, \mathcal{F}_t, \mathbb{Q})$ be a filtered probability space, with $\mathbb{Q}$ the risk-neutral probability. We shall work directly under risk-neutrality. We consider the Filipović space on $\mathbb{R}_+$ denoted by $\mathcal{H}_\alpha := \mathcal{H}_\alpha(\mathbb{R}_+)$: for a given continuous and non-decreasing function $\alpha : \mathbb{R}_+ \to [1, \infty)$ with $\alpha(0) = 1$ this is the Hilbert space of all absolutely continuous functions $f : \mathbb{R}_+ \to \mathbb{R}$ for which

$$\int_{\mathbb{R}_+} f'(x)^2 \alpha(x) dx < \infty.$$

It turns out that $\mathcal{H}_\alpha$ is a separable Hilbert space with inner product defined by

$$\langle f_1, f_2 \rangle_\alpha := f_1(0) f_2(0) + \int_{\mathbb{R}_+} f_1'(x) f_2'(x) \alpha(x) dx,$$

for $f_1, f_2 \in \mathcal{H}_\alpha$ and norm $\|f\|_\alpha^2 := \langle f, f \rangle_\alpha$. We assume $\int_{\mathbb{R}_+} \alpha^{-1}(x) dx < \infty$. A typical example is $\alpha(x) = e^{\alpha x}$, for $\alpha > 0$. We refer to [21] for more properties of $\mathcal{H}_\alpha$. In [12, Section 2] it is shown that the Filipović space is an appropriate state space for the forward curve in energy markets. This motivates our choice of considering $\mathcal{H}_\alpha$ as the space for the forward curves.

For a Borel set $\mathcal{O} \subseteq \mathbb{R}$, we define $\mathcal{H} := L^2(\mathcal{O})$ as the Hilbert space of all square integrable functions. We further denote by $\lambda_{\mathcal{O}}$ the Lebesgue measure induced on $\mathcal{O}$, and by $\|\cdot\|$ and $\langle \cdot, \cdot \rangle$ respectively the norm and scalar product on $\mathcal{H}$. We assume that $\mathcal{O}$ is such that $\mathcal{H}$ is a separable Hilbert space, and we shall consider $\mathcal{H}$ as the noise space, like, e.g., in [5, 12, 13]. For a square integrable $\mathcal{H}$-valued random variable X (i.e. $\mathbb{E}[\|X\|^2] < \infty$), a linear operator $\mathcal{Q} \in \mathcal{L}(\mathcal{H}, \mathcal{H})$ is the unique covariance operator of X if $\mathbb{E}[\langle X, h_1 \rangle \langle X, h_2 \rangle] = \langle \mathcal{Q} h_1, h_2 \rangle$ for any $h_1, h_2 \in \mathcal{H}$. We then define an $\mathcal{H}$-valued Wiener process $\mathbb{W} := \{\mathbb{W}(t)\}_{t \geq 0}$ with zero mean and covariance operator $\mathcal{Q}$, as the $\mathcal{H}$-valued stochastic process with continuous trajectories, stationary increments with law $\mathbb{W}(t) - \mathbb{W}(s) \sim \mathcal{N}(0, (t-s)\mathcal{Q})$, and $\mathbb{W}(0) = 0$, see [35] or [18].

Let now $f(t, T)$ be the forward price in the commodity market at time $t \leq T$ for a contract with time of delivery T. Following the HJM approach in the Musiela parametrization, for $x := T - t$, the time to delivery, we define $g(t, x) := f(t, t + x)$ following the dynamics

$$dg(t, x) = (\partial_x g(t, x) + \beta(t, x)) \, dt + \sigma(t, x) d\mathbb{W}(t, x), \tag{2.1}$$

∂_x being the generator for the semigroup $\{\mathcal{U}_t\}_{t \geq 0}$ given by $\mathcal{U}_t g(x) = g(t + x)$, for any $t, x \in \mathbb{R}_+$ and $g \in \mathcal{H}_\alpha$. Here $\beta(t, \cdot) \in \mathcal{H}_\alpha$ and $\sigma(t, \cdot) \in \mathcal{L}(\mathcal{H}, \mathcal{H}_\alpha)$ is a linear and bounded operator from $\mathcal{H}$ to $\mathcal{H}_\alpha$. In order for the model (2.1) to be under the risk neutral measure directly, we must impose the condition $\beta \equiv 0$ (see [11, Section 3.2]). We also want to rewrite equation (2.1) as a model for the entire forward curve $(g(t, x))_{x \geq 0}$ for any time $t \geq 0$, and to allow for the volatility function σ to be stochastic itself. Without introducing any other external noise source, for instance with a second dynamics for the volatility, we consider σ to be state-dependent, namely depending on the current level of the forward curve g. We finally obtain the following partial stochastic differential equation

$$dg_t = \partial_x g_t dt + \sigma_t(g_t) d\mathbb{W}_t, \tag{2.2}$$

where $g_t := g(t, \cdot)$, $\sigma_t(g_t) := \sigma(t, g_t)$ and $\mathbb{W}_t := \mathbb{W}(t, \cdot)$. The following Theorem states conditions to ensure that a mild solution to equation (2.2) exists.

Theorem 2.1. *Let us assume that the mapping*

$$\sigma : \mathbb{R}_+ \times \mathcal{H}_\alpha \to \mathcal{L}(\mathcal{H}, \mathcal{H}_\alpha), \quad (t, g_t) \mapsto \sigma(t, g_t) = \sigma_t(g_t)$$

is measurable and that there exists an increasing function $C : \mathbb{R}_+ \to \mathbb{R}_+$ such that for all $f_1, f_2 \in \mathcal{H}_\alpha$ and $t \in \mathbb{R}_+$ we have

$$\begin{aligned} &\|\sigma_t(f_1) - \sigma_t(f_2)\|_{\mathcal{L}(\mathcal{H},\mathcal{H}_\alpha)} \leq C(t) \|f_1 - f_2\|_\alpha, \\ &\|\sigma_t(f_1)\|_{\mathcal{L}(\mathcal{H},\mathcal{H}_\alpha)} \leq C(t)(1 + \|f_1\|_\alpha). \end{aligned}$$

Then for every $s \geq t$ there exists a unique mild solution to equation (2.2) *of the form*

$$g_s = \mathcal{U}_{s-t} g_t + \int_t^s \mathcal{U}_{s-u} \sigma_u(g_u) d\mathbb{W}_u,$$

$g_t = g(t, \cdot) \in \mathcal{H}_\alpha$ being the initial condition.

Proof. We refer to [37].

By means of Theorem 2.1 we have the solution to the SPDE in equation (2.2) if the volatility operator satisfies the Lipschitz and linear growth conditions. Let us point out that in equation (2.2) the noise, namely the Wiener process $\mathbb{W}$, is an element of $\mathcal{H}$, while the forward curve g_t belongs to the space $\mathcal{H}_\alpha$. This means that the volatility operator $\sigma_t(g_t)$ must turn elements of $\mathcal{H}$ into elements of $\mathcal{H}_\alpha$. We shall study the volatility operator in the next Section.

2.1 The volatility operator

We focus on possible specifications for the volatility operator $\sigma : \mathbb{R}_+ \times \mathcal{H}_\alpha \to \mathcal{L}(\mathcal{H}, \mathcal{H}_\alpha)$. In particular, we need conditions to ensure that actually $\sigma_t(f) \in \mathcal{L}(\mathcal{H}, \mathcal{H}_\alpha)$ for every $f \in \mathcal{H}_\alpha$. The volatility operator $\sigma_t(g_t)$ must turn elements of $\mathcal{H}$ into elements of $\mathcal{H}_\alpha$. It thus has to smoothen the noise, and one way to do that is by integrating it over a suitably chosen kernel function. Additionally, $\sigma_t(f)$ has to fulfill the Lipchitz and growth conditions required for Theorem 2.1. We start with the following conditions on the kernel function.

Theorem 2.2. *For $t \geq 0$, let $\kappa_t : \mathbb{R}_+ \times \mathcal{O} \times \mathcal{H}_\alpha \to \mathbb{R}_+$ be a kernel function satisfying the following assumptions:*

1. *The map $\kappa_t(x, \cdot, f) \in \mathcal{H}$ for every $x \in \mathbb{R}_+, f \in \mathcal{H}_\alpha$.*

2. *For every $x \in \mathbb{R}_+, f \in \mathcal{H}_\alpha$, the derivative $\frac{\partial \kappa_t(x,y,f)}{\partial x}$ exists for $\lambda_\mathcal{O}$ almost all $y \in \mathcal{O}$. Moreover there exists a neighbourhood I_x of x and a function $\bar{\kappa}_x \in \mathcal{H}$ such that $\left|\frac{\partial \kappa_t(x,y,f)}{\partial x}\right| \leq \bar{\kappa}_x(y)$ for $\lambda_\mathcal{O}$ almost all y on I_x.*

3. $\int_{\mathbb{R}_+} \left\| \frac{\partial \kappa_t(x,\cdot,f)}{\partial x} \right\|^2 \alpha(x)dx < \infty.$

Then

$$\sigma_t(f) : \mathcal{H} \to \mathcal{H}_\alpha, \quad h \mapsto \sigma_t(f)h := \int_{\mathcal{O}} \kappa_t(\cdot, y, f)h(y)dy \tag{2.3}$$

is a linear and bounded operator from $\mathcal{H}$ *to* $\mathcal{H}_\alpha$*, namely* $\sigma_t(f) \in \mathcal{L}(\mathcal{H}, \mathcal{H}_\alpha)$*. In particular, for every* $x \in \mathbb{R}_+$*, we can also write the equality* $\sigma_t(f)h(x) = \langle \kappa_t(x,\cdot,f), h\rangle$*.*

Proof. See Appendix A.1.

Given the operator $\sigma_t(f)$ in equation (2.3), it will be necessary to find the corresponding adjoint operator, namely the operator $\sigma_t(f)^* \in \mathcal{L}(\mathcal{H}_\alpha, \mathcal{H})$ that for every $f_1 \in \mathcal{H}_\alpha$ and every $h \in \mathcal{H}$ satisfies $\langle \sigma_t(f)h, f_1\rangle_\alpha = \langle h, \sigma_t(f)^* f_1\rangle$. By [36, Theorem 6.1] any operator $\sigma_t(f)^*$ satisfying this equality is automatically bounded and is thus the unique adjoint operator of $\sigma_t(f)$.

Theorem 2.3. *With the assumptions of Theorem 2.2, the adjoint operator* $\sigma_t(f)^*$ *is given by*

$$\sigma_t(f)^* : \mathcal{H}_\alpha \to \mathcal{H}, \quad f_1 \mapsto \sigma_t(f)^* f_1 := \kappa_t(0,\cdot,f)f_1(0) + \int_{\mathbb{R}_+} \frac{\partial \kappa_t(x,\cdot,f)}{\partial x} f_1'(x)\alpha(x)dx.$$

In particular, for every $y \in \mathbb{R}_+$*, we can also write the equality* $\sigma_t(f)^* f_1(y) = \langle \kappa_t(\cdot, y, f), f_1\rangle_\alpha$*.*

Proof. See Appendix A.2.

In Theorem 2.2 we considered some assumptions that the kernel κ_t must satisfy to ensure that $\sigma_t(f) \in \mathcal{L}(\mathcal{H}, \mathcal{H}_\alpha)$ for every $f \in \mathcal{H}_\alpha$. We shall now look at conditions which also ensure that $\sigma_t(f)$ fulfils the assumptions of Lipschitz continuity and linear growth of Theorem 2.1.

Theorem 2.4. *Let* $\kappa_t : \mathbb{R}_+ \times \mathcal{O} \times \mathcal{H}_\alpha \to \mathbb{R}_+$ *be a kernel function satisfying the assumptions of Theorem 2.2. If there exists an increasing function* $C : \mathbb{R}_+ \to \mathbb{R}_+$ *such that, for every* $f_1, f_2 \in \mathcal{H}_\alpha$*, it holds:*

1. $\|\kappa_t(0,\cdot,f_1) - \kappa_t(0,\cdot,f_2)\| \le C(t)\,|f_1(0) - f_2(0)|\,,$

$\left\| \frac{\partial \kappa_t(x,\cdot,f_1)}{\partial x} - \frac{\partial \kappa_t(x,\cdot,f_2)}{\partial x} \right\| \le C(t)\,|f_1'(x) - f_2'(x)|\,,$

2. $\|\kappa_t(0,\cdot,f_1)\| \le C(t)(1 + |f_1(0)|)\,,$

$\left\| \frac{\partial \kappa_t(x,\cdot,f_1)}{\partial x} \right\| \le C(t)|f_1'(x)|$

then σ_t *defined in equation* (2.3) *satisfies the Lipschitz and growth conditions of Theorem 2.1.*

Proof. See Appendix A.3.

If the kernel function κ_t satisfies the assumptions of Theorem 2.2 and Theorem 2.4, then there exists a mild solution to equation (2.2), which models the dynamics of the forward curve.

2.2 A deterministic specification

In the previous Section we defined the volatility as an integral operator with respect to a kernel function κ_t. We now specify κ_t in order to reflect some properties that we believe to be crucial for the volatility operator. For instance, we shall include time dependency to account for seasonality effects. This was for example observed in electricity markets by [9], when analysing the volatility structure in a log normal model for the forward curve. The same authors also model a maturity effect, namely a monotone decay in the volatility when the time to maturity increases, also known as the Samuelson effect. This can be easily achieved with some decay function. Finally, we want to include that contracts with a certain maturity should be mainly influenced by the randomness of the noise in a neighbourhood of that maturity.

We shall restrict to a deterministic, time-dependent diffusion term. We therefore drop the state-dependency and define the kernel function κ_t as the product of two parts, one to incorporate the seasonal and the Samuelson effect, and a second one to smooth the noise in a neighbourhood of the time to maturity, namely

$$\kappa_t(x,y) := a(t)e^{-bx}\,\omega(x-y), \tag{2.4}$$

$$a(t) := a + \sum_{j=1}^{J}\left(s_j\sin(2\pi jt) - c_j\cos(2\pi jt)\right), \tag{2.5}$$

where $\omega : \mathbb{R} \to \mathbb{R}_+$ is a continuous weight function, while the term $e^{-bx}, b \geq 0$, captures the Samuelson effect, and $a(t)$ is the seasonal function defined for $a \geq 0$, s_j and c_j real constants, and t measured in years.

With the following Proposition, we state some assumptions on the weight function ω to ensure that $\kappa_t(x,y)$ as defined above fulfils the assumptions of Theorem 2.2 for every $t \geq 0$.

Proposition 2.5. *Let $\omega : \mathbb{R} \to \mathbb{R}_+$ be such that:*

1. *For every $x \in \mathbb{R}_+$, $\omega(x-\cdot)\,|_{\mathcal{O}} \in \mathcal{H}$.*
2. *The derivative $\omega'(x)$ exists for almost all $x \in \mathbb{R}$ and whenever it exists, there exists a neighbourhood I_x of x and a function $\bar{\omega}_x \in \mathcal{H}$ with $\|\bar{\omega}_x\| \leq C_1$ for some C_1 independent of x, and such that $|(\omega'(x-y) - b\omega(x-y))\,|_{\mathcal{O}}\,| \leq \bar{\omega}_x(y)$ on I_x.*

Let further $\int_{\mathbb{R}_+} e^{-2bx}\alpha(x)dx < \infty$. Then for every $t \geq 0$ the volatility operator σ_t given by

$$\sigma_t : \mathcal{H} \to \mathcal{H}_\alpha, \quad h \mapsto \sigma_t h := \int_{\mathcal{O}} \kappa_t(\cdot, y)h(y)dy$$

is well defined, and satisfies the Lipschitz and linear growth conditions of Theorem 2.1.

Proof. See Appendix A.4.

For the function $\alpha(x) = e^{\alpha x}$ for instance, the integrability assumption of Proposition 2.5 is satisfied if $0 < \alpha < 2b$.

3 Forward contracts with delivery period

We consider now energy forward contracts with a delivery period, which we refer to as swaps, in order to not confuse them with the contracts discussed in the previous Section. For $0 \leq t \leq$

$T_1 \leq T_2$, we denote by $F(t, T_1, T_2)$ the price at time t of a swap contract on energy delivering over the interval $[T_1, T_2]$. From [10, Proposition 4.1], this price can be expressed by

$$F(t, T_1, T_2) = \int_{T_1}^{T_2} w(T; T_1, T_2) f(t, T) dT, \tag{3.1}$$

$f(t, T)$ being the forward curve introduced above, and $w(T; T_1, T_2)$ a deterministic weight function. Focusing on forward style swaps in the electricity markets as traded, for example, at Nord Pool AS and the European Energy Exchange (EEX), the weight function takes the form

$$w(T; T_1, T_2) = \frac{1}{T_2 - T_1}. \tag{3.2}$$

According to [12], we introduce the Musiela representation for $F(t, T_1, T_2)$. For $x := T_1 - t$ the time until start of delivery, and $\ell := T_2 - T_1 > 0$ the length of delivery of the swap, we define the weight function $w_\ell(t, x, y) := w(t + y; t + x, t + x + \ell)$. Motivated by practical examples (see [12, Section 2]), we shall consider only time-independent and stationary weight functions. With abuse of notation, let then $w_\ell : \mathbb{R}_+ \to \mathbb{R}_+$ be bounded and measurable, such that

$$G_\ell^w(t, x) := F(t, t + x, t + x + \ell) = \int_x^{x+\ell} w_\ell(y - x) g_t(y) dy,$$

for g_t in equation (2.2). For $w(T; T_1, T_2)$ as in equation (3.2), one simply gets $w_\ell(y - x) = \frac{1}{\ell}$.

Following [11, Section 4], for any $g_t \in \mathcal{H}_\alpha$, we represent G_ℓ^w as the following linear operator:

$$\mathcal{D}_\ell^w(g_t) := \mathcal{W}_\ell(\ell)\mathrm{Id}(g_t) + \mathcal{I}_\ell^w(g_t), \tag{3.3}$$

namely, $G_\ell^w(t, \cdot) = \mathcal{D}_\ell^w(g_t)(\cdot)$. In equation (3.3), Id is the identity operator, while

$$\mathcal{W}_\ell(u) := \int_0^u w_\ell(v) dv, \quad u \geq 0, \tag{3.4}$$

$$\mathcal{I}_\ell^w(g_t)(\cdot) := \int_0^\infty q_\ell^w(\cdot, y) g_t'(y) dy, \tag{3.5}$$

$$q_\ell^w(x, y) := (\mathcal{W}_\ell(\ell) - \mathcal{W}_\ell(y - x)) \mathbb{I}_{[0,\ell]}(y - x). \tag{3.6}$$

For a swap contract of forward type with weight function in equation (3.2), it turns out that the operator $\mathcal{D}_\ell^w$ has an easier representation as provided in the following Lemma.

Lemma 3.1. *For a forward-style swap contract, the operator $\mathcal{D}_\ell^w$ can be represented as*

$$\mathcal{D}_\ell^w(g_t)(x) = \int_0^\infty d_\ell(x, y) g_t(y) dy,$$

where $d_\ell : \mathbb{R}_+ \times \mathbb{R}_+ \to \mathbb{R}_+, d_\ell(x, y) := \frac{1}{\ell}\mathbb{I}_{[x,x+\ell]}(y)$ is called the delivery period function.

Proof. See Appendix A.5.

With the notation just introduced, thanks to [12, Lemma 3.3], for every $0 \leq t \leq \tau \leq T_1$ the price of the swap contract at time τ with delivery over the interval $[T_1, T_2]$ can be expressed as a linear functional acting on the forward curve g_t:

$$F(\tau, T_1, T_2) = \delta_{T_1 - t}\mathcal{D}_\ell^w g_t + \int_t^\tau \delta_{T_1 - s}\mathcal{D}_\ell^w \sigma_s(g_s) d\mathbb{W}_s,$$

for $\ell = T_2 - T_1$. Further, [11, Theorem 2.1] allows us to derive a real-valued stochastic process for the swap dynamics. More precisely, for every $0 \leq t \leq \tau \leq T_1$, we can write that

$$F(\tau, T_1, T_2) = \delta_{T_1-t}\mathcal{D}_\ell^w g_t + \int_t^\tau \Sigma(s)dW(s), \tag{3.7}$$

$$\Sigma^2(s) := \left(\delta_{T_1-s}\mathcal{D}_\ell^w \sigma_s(g_s)\mathcal{Q}\,\sigma_s(g_s)^*\left(\delta_{T_1-s}\mathcal{D}_\ell^w\right)^*\right)(1), \tag{3.8}$$

where W is a standard Wiener process with values in $\mathbb{R}$. In particular, equation (3.7) tells us that the swap price $F(\tau, T_1, T_2)$ which is driven by the $\mathcal{H}$-valued Wiener process $\mathbb{W}$ with covariance operator $\mathcal{Q}$ and volatility operator σ_t, can in fact be represented as driven by a one-dimensional Wiener process with diffusion term given in equation (3.8).

For a swap contract of forward type, considering the covariance operator $\mathcal{Q}$ to be an integral operator of the form

$$\mathcal{Q}h(x) = \langle h, q(x, \cdot)\rangle = \int_{\mathcal{O}} q(x,y)h(y)dy, \quad h \in \mathcal{H},$$

with kernel $q(x,y)$ such that $\mathcal{Q}$ is well defined, and given the volatility operator in equation (2.3), we can rewrite the univariate volatility in equation (3.8) more explicitly.

Proposition 3.2. *For a swap contract of forward type with weight function in equation* (3.2)*, the volatility* $\Sigma^2(s)$*,* $t \leq s \leq \tau$*, defined in equation* (3.8) *is equivalent to the fourth integral*

$$\Sigma^2(s) = \int_{\mathbb{R}_+}\int_{\mathbb{R}_+}\int_{\mathcal{O}}\int_{\mathcal{O}} d_\ell(T_1 - s, u)d_\ell(T_1 - s, v)\kappa_s(v, z, g_s)q(z,y)\kappa_s(u, y, g_s)dydzdudv,$$

where d_ℓ *is the delivery period function defined in Lemma 3.1.*

Proof. See Appendix A.6.

In the deterministic setting introduced in Section 2.2, the univariate volatility formula of Proposition 3.2 can be further simplified.

Corollary 3.3. *With a volatility kernel factorized as in equation* (2.4)*, the formula for* $\Sigma^2(s)$*,* $t \leq s \leq \tau$*, in Proposition 3.2 is equivalent to*

$$\Sigma^2(s) = a(s)^2\int_{\mathbb{R}_+}\int_{\mathbb{R}_+}\int_{\mathcal{O}}\int_{\mathcal{O}} e^{-bu}e^{-bv}d_\ell(T_1 - s, u)d_\ell(T_1 - s, v)\omega(v-z)q(z,y)\omega(u-y)dydzdudv.$$

Proof. This is a direct consequence of Proposition 3.2. □

3.1 European options on the energy forwards

We focus on European-style options written on energy swap contracts. These kinds of derivatives are traded, for example, at Nord Pool AS. With the price of the swap at time t being $F(t, T_1, T_2)$, we consider an option with payoff function $\pi : \mathbb{R} \to \mathbb{R}$ and exercise time $0 \leq \tau \leq T_1$. Classical examples are standard call and put options with strike $K \geq 0$, for which the payoff function is defined by $\pi(x) = \max(x - K, 0)$, respectively $\pi(x) = \max(K - x, 0)$.

From equation (3.7), the price at time $0 \leq t \leq \tau$ of the option with payoff $\pi(F(\tau, T_1, T_2))$ at time $0 \leq \tau \leq T_1$ is given by

$$\Pi(t) = e^{-r(\tau-t)}\mathbb{E}\left[\pi\left(\delta_{T_1-t}\mathcal{D}_\ell^w g_t + \int_t^\tau \Sigma(s)dW(s)\right)\middle|\,\mathcal{F}_t\right], \tag{3.9}$$

for $r > 0$ the risk-free interest rate, here considered to be constant. Assuming π to be measurable and of at most linear growth, from [12, Proposition 3.1] we get that for every $\ell > 0$ and for every $g_t \in \mathcal{H}_\alpha$ such that $\mathbb{E}\left[\|g_t\|_\alpha\right] < \infty$, the expectation (3.9) is well defined. The same holds, in particular, also for the expectation of the payoff of an option on a forward with fixed time to delivery T, namely $\pi\left(f(\tau, T)\right)$. We refer to [12] for more details.

We end the Section with a result from [12], which allows to rewrite the price functional in equation (3.9) for a deterministic volatility operator, like the one we introduced in Section 2.2.

Proposition 3.4. *For σ_t deterministic, the price functional in equation* (3.9) *becomes*

$$\Pi(t) = e^{-r(\tau-t)}\mathbb{E}\left[\pi\left(\mu(g_t) + \xi X\right) \middle| \mathcal{F}_t\right], \tag{3.10}$$

$$\xi^2 := \int_t^\tau \Sigma^2(s)ds, \tag{3.11}$$

$$\mu(g_t) := \delta_{T_1-t}\mathcal{D}_\ell^w g_t, \tag{3.12}$$

with X standard normal distributed random variable and Σ^2 in equation (3.8).

Proof. We refer to [12, Proposition 3.7].

With Proposition 3.4 the price of an option on the forward curve can be calculated in closed form if the volatility operator is deterministic. In order to do that, by equations (3.11) and (3.12) one needs to define the volatility operator, as well as the covariance operator and the initial forward curve. Before giving full specification for those, we define in the next Section the two steps approach to calibrate the HJM model with neural networks.

4 The neural networks approach

For the purpose of calibration, we shall specify a fully parametric model, depending on a parameter vector θ taking values in a set $\Theta \subset \mathbb{R}^n$. In the framework described in Section 2, θ is a vector of parameters defining the volatility operator, the covariance operator and the initial forward curve. Moreover, the option price function depends on some features of the contract, such as time to delivery, strike, etc. We denote the vector of these contract parameters by $\lambda \in \Lambda \subset \mathbb{R}^m$. Then, the price function (3.9) is $\Pi(t) = \Pi(t; \lambda, \theta)$.

As the contract features λ are given by the market, to get a fully specified price functional we need to calibrate the chosen model and determine the vector θ that best matches the observed prices of liquidly traded options. This will give us the best coefficient functions (in terms of calibration) for the HJM model defined in Section 2. We do that by the two steps approach with neural networks presented in [7]. In what follows, we first define feedforward neural networks, and then the calibration problem, together with the two steps approach.

4.1 Feedforward neural networks

We define an L-layer feedforward neural network as a function $\mathcal{N} : \mathbb{R}^d \to \mathbb{R}^p$ of the form

$$\mathcal{N}(x) := H_L(\rho(H_{L-1}(\rho(\dots\rho(H_1(x)))))), \tag{4.1}$$

where each $H_i : \mathbb{R}^{n_{i-1}} \to \mathbb{R}^{n_i}$ is an affine map of the form $H_i(x) = V_i x + v_i$, for $V_i \in \mathbb{R}^{n_i \times n_{i-1}}$ and $v_i \in \mathbb{R}^{n_i}$. Here the V_i's are referred to as the weights and the v_i's as the biases. In particular, $n_0 = d$ equals the input dimension, and $n_L = p$ equals the output dimension. We call L the depth of the network and n_i represents the number of nodes of the i-th layer. We set $\mathbf{n} := (n_0, \dots, n_L)$. The map $\rho : \mathbb{R} \to \mathbb{R}$ is the so-called activation function, which is typically non-linear and applies component wise on the output of the affine maps H_i. Typical choices are the Rectified Linear Unit (ReLU) given by $\rho(x) = \max(x, 0)$, the Exponential Linear Unit (ELU) given by $\rho(x) = \max(e^x - 1, x)$ or the Sigmoid $\rho(x) = \frac{1}{1+e^{-x}}$. We point out that in the definition (4.1), the activation function is the same between all the layers, and we do not consider any activation function for the L-th layer. This is important to allow for expressivity. With the sigmoid function in the last layer, for example, all values would lie between 0 and 1.

We shall denote by $\mathcal{V}$ the set of all parameters involved, namely $\mathcal{V} := \{V_i, v_i\}_{i=1}^L$, and we use the notation $\mathcal{N} = \mathcal{N}(x; \mathcal{V})$. The cardinality of $\mathcal{V}$ is then given by $M := |\mathcal{V}| = \sum_{i=1}^L n_i(n_{i-1}+1)$. As we can see, the number of parameters involved in the neural network, namely, the number of parameters which must be optimized, grows quite fast. A particular subclass of feedforward neural network is given by the convolutional networks. A convolution is a special kind of linear operation, which is represented by a sparse matrix. This is cost efficient, both in terms of computing time and required memory. We call a network convolutional when at least one of the V_i's is a convolution operator. We shall refer to dense networks when the weights V_i's are full matrices instead. The architecture of the neural network, namely the width L, the number of nodes per layer, $\mathbf{n} = (n_1, \ldots, n_{L-1})$, and the activation function are the hyper-parameters which must be chosen in accordance to each specific problem to solve.

The idea behind neural networks is to approximate a map $\varphi : \mathbb{R}^d \to \mathbb{R}^p$ starting from the set of input-outputs $\{(x_i, \varphi(x_i))\}_{i=1}^N$ of size N. This translates into an optimization problem, called the training of the neural network, which is solved by finding the best set of parameters $\mathcal{V} \in \mathbb{R}^M$ so that the neural network output $\mathcal{N}(x; \mathcal{V})$ best approximates the observations $\varphi(x)$, for $x \in \{x_i\}_{i=1}^N$, with respect to some loss function R to be chosen. A common loss function is the mean squared error defined by

$$R\left(\{x_i\}_{i=1}^N, \{y_i\}_{i=1}^N\right) := \frac{1}{N}\sum_{i=1}^N (x_i - y_i)^2 .$$

Training the neural network means to find a $\hat{\mathcal{V}} \in \mathbb{R}^M$ that solves the optimization problem

$$\underset{\mathcal{V}\in\mathbb{R}^M}{\text{minimize}}\, \frac{1}{N}\sum_{i=1}^N \left(\mathcal{N}(x_i; \mathcal{V}) - \varphi(x_i)\right)^2 .$$

Given the optimal weights $\hat{\mathcal{V}}$, we denote by $\hat{\mathcal{N}}(x) := \mathcal{N}(x; \hat{\mathcal{V}})$ the trained neural network. We point out that this is a non-convex optimization problem and one typically only finds an approximation to a local solution. For more details on feedforward neural networks, activation functions and training of the network, we refer the reader to [28, 23].

4.2 The calibration problem and the two steps approach

We focus on the option price $\Pi = \Pi(\lambda, \theta)$, omitting the time dependency to simplify the notation, and we consider N option contracts with features $\{\lambda_i\}_{i=1}^N \in \Lambda$, whose price can be observed in the market. This means we have a set of market observed prices $\{\Pi_i\}_{i=1}^N$, where Π_i is the price of the contract corresponding to λ_i. Calibrating the HJM model for the forward curve means to find vector of model parameters $\hat{\theta} \in \Theta$ which minimizes the distance between the prices observed in the market and the corresponding prices given by the stochastic model. This is done with respect to a certain cost functional. With the mean squared error functional, the calibration corresponds to find $\hat{\theta} \in \Theta$ which solves the optimization problem

$$\underset{\theta \in \Theta}{\text{minimize}} \, \frac{1}{N} \sum_{i=1}^{N} \left(\Pi(\lambda_i, \theta) - \Pi_i\right)^2 . \tag{4.2}$$

Often, it is not possible to obtain a closed formula for the price functional Π, due to the complexity of the underlying model. In these cases, the calibration problem presented in equation (4.2) must be slightly adjusted by substituting Π with $\tilde{\Pi}$, which is an approximated price functional, obtained by, for example, Monte Carlo simulation. This makes the procedure costly. In particular, the more complex the underlying stochastic model is, the greater the time needed for simulation and hence for calibration. This has the consequence that more accurate models are often left out from practical applications because of their complexity in calibration.

An alternative solution proposed in [7] is to divide the calibration problem in equation (4.2) into two steps. We start by approximating the price functional Π (or its approximation $\tilde{\Pi}$) with a neural network, $\mathcal{N}(\lambda, \theta; \mathcal{V}) \approx \Pi(\lambda, \theta)$. This is done by generating a training set by considering (many) different vectors of parameters θ's and option features λ's and the corresponding price values. The training set is then used to train the neural network. This step is computationally demanding for two reasons. First of all, we need to generate a large training set of input-output data by (potentially) costly simulations. Moreover, the training of the neural network is an $M = |\mathcal{V}|$ dimensional optimization problem, with M usually large, and with respect to a large training set and is therefore time consuming. However, the training has the advantage to be off-line, namely, it does not use any market information. Thus, it can potentially be run only once and does not require frequent updating when new market information arises.

Once the neural network is trained, the second step is calibration. The advantage is that we replace in equation (4.2) the function Π (or $\tilde{\Pi}$) with the trained neural network $\hat{\mathcal{N}}$. Evaluation of the latter one is fast and does not require any further simulation, thus to perform a calibration becomes an easier and much faster task, also due to the fact that most machine learning frameworks have very efficient implementations. In [7], for example, the calibration for the full implied volatility surface in the rough Bergomi model is obtained in less than 40 milliseconds. Price market data are used in the second step, meaning that the model should be re-calibrated every time that new market data is available. However, with this fast implementation via neural networks, getting the results is possible in short time, and the model can be updated with low computational cost.

We present now two alternative approaches to this procedure. In the first case, we train a neural network as function of both λ and θ and with output a single real number, being the price of the option with features λ and given by the model with parameters θ. In the second case, we require a lower dimensional functional: the neural network is trained as function of only the model parameters θ. Then, the output is a (discrete) surface, namely, a (potentially multi-dimensional) grid, of prices corresponding to different contracts, namely different λ's.

4.2.1 The pointwise learning approach

Let $(\lambda, \theta) \in \Lambda \times \Theta \subset \mathbb{R}^d$, for $d = n + m$, such that $\Pi = \Pi(\lambda, \theta)$. In the pointwise learning approach, we approximate the pricing map Π (or $\tilde{\Pi}$) by a neural network that maps the vector (λ, θ) into prices. Given the training set $\{((\lambda_i, \theta_i), \Pi(\lambda_i, \theta_i))\}_{i=1}^{N_{train}}$ for $(\lambda_i, \theta_i) \in \Lambda \times \Theta$ and N_{train} the size of the training set, we train a neural network $\mathcal{N} : \Lambda \times \Theta \to \mathbb{R}_+$ by computing

$$\hat{\mathcal{V}} \in \underset{\mathcal{V} \in \mathbb{R}^M}{\text{argmin}} \, \frac{1}{N_{train}} \sum_{i=1}^{N_{train}} \left(\mathcal{N}(\lambda_i, \theta_i; \mathcal{V}) - \Pi(\lambda_i, \theta_i)\right)^2 .$$

Once we have $\hat{\mathcal{N}}(\lambda, \theta) = \mathcal{N}(\lambda, \theta; \hat{\mathcal{V}})$, we use it for the calibration step with respect to the model parameters, that is, we look for an approximate solution $\hat{\theta} \in \Theta$ to the following problem:

$$\underset{\theta \in \Theta}{\text{minimize}} \frac{1}{N_{cal}} \sum_{i=1}^{N_{cal}} \left(\hat{\mathcal{N}}(\lambda_i, \theta) - \Pi_i \right)^2,$$

where $\{\Pi_i\}_{i=1}^{N_{cal}}$ are market observed prices with respect to some contract features $\{\lambda_i\}_{i=1}^{N_{cal}}$, and N_{cal} is the size of the calibration set.

4.2.2 The grid-based learning approach

The idea of the grid-based learning approach is to train a neural network which is a function only of the model parameters θ. In this case, the output is no longer a single price value, but a discrete grid of values corresponding to different option specifications λ, which are decided in the first step. These should reflect the options traded in the market. Let us suppose $m = 2$ and $\lambda = (\lambda^1, \lambda^2)$. Then for $m_1, m_2 \in \mathbb{N}$, we create a grid of values $\{(\lambda_j^1, \lambda_k^2)\}_{j=1,k=1}^{m_1,m_2}$ and the training set $\{(\theta_i, \{\Pi(\theta_i, (\lambda_j^1, \lambda_k^2))\}_{j=1,k=1}^{m_1,m_2})\}_{i=1}^{N_{train}}$. We then train a neural network $\mathcal{N} : \Theta \to \mathbb{R}_+^{m_1 \times m_2}$ by solving the following optimization problem

$$\hat{\mathcal{V}} \in \underset{\mathcal{V} \in \mathbb{R}^M}{\text{argmin}} \frac{1}{N_{train}} \frac{1}{m_1 m_2} \sum_{i=1}^{N_{train}} \sum_{j,k=1}^{m_1,m_2} \left(\mathcal{N}(\theta_i; \mathcal{V})_{j,k} - \Pi(\theta_i, (\lambda_j^1, \lambda_k^2)) \right)^2.$$

Once trained $\hat{\mathcal{N}}(\theta) = \mathcal{N}(\theta; \hat{\mathcal{V}})$, we use the neural network in the calibration step in order to find the optimal model parameters $\hat{\theta} \in \Theta$ for fitting the market observations $\{\Pi_{j,k}\}_{j=1,k=1}^{m_1,m_2}$:

$$\underset{\theta \in \Theta}{\text{minimize}} \frac{1}{m_1 m_2} \sum_{j,k=1}^{m_1,m_2} \left(\hat{\mathcal{N}}(\theta)_{j,k} - \Pi_{j,k} \right)^2.$$

The main difference of the grid-based approach compared with the pointwise one, is that the neural network is trained to price only specific options, namely options related to the grid $\{(\lambda_j^1, \lambda_k^2)\}_{j=1,k=1}^{m_1,m_2}$ defined in the approximation step. This might be seen as a weakness, however, every price for a contract not included in this grid can be obtained by interpolation (which is done automatically by the network in the pointwise approach). Moreover, the grid in the first step can be chosen as fine as wished. The advantage is that the dimension of the input vector is smaller, making it easier for the network to approximate the price functional.

4.3 The bid-ask constraint

In order book based markets there is no single price, but the so-called bid and ask price corresponding to the cheapest sell and the most expensive buy order. Depending on market liquidity, the spread between bid and ask price (bid-ask spead) can be significant. Thus the calibration problem described in equation (4.2) breaks down as we do not have exact prices to aim at in a mean squared loss sense. We need a new loss function taking the bid-ask spread into account. To penalize prices lying outside the bid-ask range, we introduce the loss R^{bid}_{ask} as

$$R^{bid}_{ask}\left(\{x_i\}_{i=1}^N, \{y_i^{bid}\}_{i=1}^N, \{y_i^{ask}\}_{i=1}^N\right) \\ := \frac{1}{N}\sum_{i=1}^N \left\{\left(x_i - y_i^{bid}\right)^2 \mathbb{I}_{\{x_i<y_i^{bid}\}} + \left(x_i - y_i^{ask}\right)^2 \mathbb{I}_{\{x_i>y_i^{ask}\}}\right\},$$

which equals zero for those prices within the bid-ask interval, and it is a quadratic function of the distance to the boundary outside the interval, as it is shown in Figure 1.

In the grid-based learning framework described in Section 4.2.2 with $m = 2$ and $\lambda = (\lambda^1, \lambda^2) \in \{(\lambda^1_j, \lambda^2_k)\}_{j=1,k=1}^{m_1,m_2}$, we consider the market observations $\{\Pi^{bid}_{j,k}, \Pi^{ask}_{j,k}\}_{j=1,k=1}^{m_1,m_2}$, where $\Pi^{bid}_{j,k}$ and $\Pi^{ask}_{j,k}$ are, respectively, the bid price and the ask price of the contract with features $(\lambda^1_j, \lambda^2_k)$. Given the trained neural network $\hat{\mathcal{N}} : \Theta \to \mathbb{R}_+^{m_1\times m_2}$, we then look for the optimal model parameters $\hat{\theta} \in \Theta$ to fit the market observations:

$$\underset{\theta\in\Theta}{\text{minimize}} \frac{1}{m_1 m_2}\sum_{j,k=1}^{m_1,m_2} \left\{\left(\hat{\mathcal{N}}(\theta)_{j,k} - \Pi^{bid}_{j,k}\right)^2 \mathbb{I}_{\{\hat{\mathcal{N}}(\theta)_{j,k}<\Pi^{bid}_{j,k}\}} + \right. \\ \left. + \left(\hat{\mathcal{N}}(\theta)_{j,k} - \Pi^{ask}_{j,k}\right)^2 \mathbb{I}_{\{\hat{\mathcal{N}}(\theta)_{j,k}>\Pi^{ask}_{j,k}\}}\right\}.$$

The loss function R^{bid}_{ask} can be also applied to the pointwise learning in a similar manner.

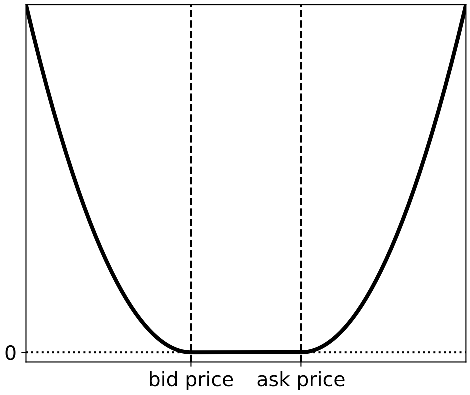


Figure 1: The bid-ask loss function.

5 The setting for the experiments

We shall fully specify the setting. Let $\alpha(x) = e^{\alpha x}$, $x \in \mathbb{R}_+$, be the weight function for the Filipović space $\mathcal{H}_\alpha$. By Proposition 2.5 the real constant α must satisfy $0 < \alpha < 2b$. Moreover, we deal with swap contracts of forward type on energy with delivery over an interval $[T_1, T_2]$. Thus we take $w(T; T_1, T_2) = \frac{1}{T_2-T_1}$ (see equation (3.2)) and, consequently, $w_\ell(y-x) = \frac{1}{\ell}$. Then $\mathcal{W}_\ell(\ell) = 1$ (see equation (3.4)) and $q^w_\ell(x, y) = \frac{1}{\ell}(x + \ell - y))\,\mathbb{I}_{[x,x+\ell]}(y)$ (see equation (3.6)). Finally, we select European-style call options with payoff function $\pi(x) = \max(x - K, 0)$, for $K > 0$ the strike price. The ultimate goal would be of course to price all kind of options written on forward contracts with or without delivery periods. This in fact is possible with the framework but beyond the scope of the paper.

In order to benchmark the two-steps approach, as explained in Section 1, we focus our attention on deterministic volatility operators, such as the one introduced in Section 2.2, which will be further specified here. For σ_t deterministic, we have seen in Proposition 3.4 that the price $\Pi(t)$ at time $0 \leq t \leq \tau$ of the option with payoff $\pi(F(\tau, T_1, T_2))$ at time $0 \leq \tau \leq T_1$, can be expressed in terms of a standard Gaussian random variable X, a variance $\xi^2 = \int_t^\tau \Sigma^2(s)ds$ and a drift $\mu(g_t) = \delta_{T_1-t}\mathcal{D}_\ell^w g_t$. In the case of an European-style call option, $\Pi(t)$ has a closed form solution, by means of a type of Black-76 formula. This allows for exact price values to be used for both the training and the calibration step, with the advantage to avoid external sources of error, such as resulting, for example, from a Monte Carlo simulation approach. As direct consequence of Proposition 3.4, we find explicitly the price functional.

Proposition 5.1. *The price of an European-style call option with strike price $K > 0$ and maturity time $\tau \leq T_1$ is given by*

$$\Pi(t) = e^{-r(\tau-t)} \left\{ \xi\phi\left(\frac{\mu(g_t) - K}{\xi}\right) + (\mu(g_t) - K)\,\Phi\left(\frac{\mu(g_t) - K}{\xi}\right)\right\}, \tag{5.1}$$

ϕ and Φ being, respectively, the density function and the cumulative distribution function of a standard Gaussian random variable.

Proof. The proof follows by direct calculation, starting from equation (3.10) for $\pi(x) = \max(x-K, 0)$, and using standard techniques for the expected value of a Gaussian random variable. □

To compute the price in equation (5.1), we need the variance ξ and the shift $\mu(g_t)$, hence we need to specify the volatility operator σ_t and the covariance operator $\mathcal{Q}$. Last, we must define an appropriate initial forward curve, $g_t \in \mathcal{H}_\alpha$. In view of the neural network approach, we need to define a fully parametric model, that is, a model fully specified by a vector of parameters θ.

5.1 The covariance operator

We need to introduce a suitable covariance operator $\mathcal{Q}$ that depends only on a finite number of parameters. The analysis conducted by [13], reveals a covariance structure that is well approximated by an exponential function, i.e. $\mathrm{Cov}(\mathbb{W}_t(x), \mathbb{W}_t(y)) \approx e^{-k|x-y|}$. Despite the operation $\mathbb{W}_t(x) = \delta_x(\mathbb{W}_t)$ not being well defined on our space $\mathcal{H}$, we can approximate it by the scalar product $\delta_x(\mathbb{W}_t) \approx \langle \eta_x, \mathbb{W}_t\rangle$, with some "bell-shaped" function η_x centred in x, such as a Gaussian density function. For every $x, y \in \mathcal{O}$, denoting by $c(x, y)$ the empirical covariance function between $\mathbb{W}_t(x)$ and $\mathbb{W}_t(y)$, we then get:

$$\begin{aligned} c(x,y) &= \mathbb{E}[\mathbb{W}_t(x)\mathbb{W}_t(y)] = \mathbb{E}[\delta_x(\mathbb{W}_t)\delta_y(\mathbb{W}_t)] \\ &\approx \mathbb{E}[\langle \eta_x, \mathbb{W}_t\rangle\langle \eta_y, \mathbb{W}_t\rangle] = \langle \mathcal{Q}\eta_y, \eta_x\rangle \\ &\approx \mathcal{Q}\eta_y(x) = \int_{\mathcal{O}} e^{-k|x-z|}\eta_y(z)dz \\ &\approx e^{-k|x-y|}, \end{aligned}$$

which shows that in $\mathcal{H}$ a covariance operator based on an exponential kernel indeed approximates the empirically observed covariance structure of the Wiener process $\mathbb{W}$ across different maturities. We thus define a one parameter covariance operator by

$$\mathcal{Q}h(x) = \int_{\mathcal{O}} e^{-k|x-y|}h(y)dy, \quad h \in \mathcal{H}. \tag{5.2}$$

Because $e^{-k|\cdot|}$ is the characteristic function of a Cauchy distributed random variable with location parameter 0 and scale parameter k, it follows by Bochner's Theorem that it is positive-definite. Since it is also symmetric and continuous, for $\mathcal{O}$ compact it follows from [35, Theorem

A.8] that $\mathcal{Q}$ is in fact a covariance operator. In the following, we therefore choose $\mathcal{O} := [-\gamma, \gamma]$ for some large γ which ensures that all maturities of interest are covered.

5.2 The volatility operator

We consider a specification of the volatility operator that does not depend on time and space, which we denote by σ instead of σ_t to simplify the notation. For the seasonal component in equation (2.5), we consider $a(t) = a \geq 0$ for every $t \in \mathbb{R}_+$. This means that we do not account for seasonality and the level a corresponds to the implied spot price volatility, as pointed out in [9]. Moreover, we define the following weight function

$$\omega(x) := (1 - |x|)\,\mathbb{I}_{\{|x|\leq 1\}} = \begin{cases} (1 - |x|) & \text{if } |x| \leq 1 \\ 0 & \text{otherwise} \end{cases}. \tag{5.3}$$

Let us notice that this parameter-free specification of ω fulfils the assumptions of Proposition 2.5. The kernel κ_t in equation (2.4) becomes then $\kappa_t(x, y) = \kappa(x, y) = ae^{-bx}\omega(x - y)$, where $b \geq 0$ determines the strength of the maturity effect. The volatility operator is given by

$$\sigma h(x) = ae^{-bx} \int_{\mathcal{O}} (1 - |x - y|)\,\mathbb{I}_{\{|x-y|\leq 1\}} h(y)dy, \tag{5.4}$$

and is well defined by Theorem 2.2. Let us remember that the role of σ is to smoothen the noise from the space $\mathcal{H}$ to $\mathcal{H}_\alpha$, and we have achieved this by considering an integral operator. The weight function ω has then a double role. First of all, it functions as (a part of) the kernel for the integral operator which smoothen the noise. On the other hand, it weights the randomness coming from the Wiener process $\mathbb{W}_t$ so that a contract with time to maturity x is only influenced by $\mathbb{W}_t(y)$, for y in a neighbourhood of x. Other weight functions ω could be considered to obtain a similar weighting effect.

We have to calculate the volatility $\Sigma^2(s)$ using the formula provided in Corollary 3.3. However, the expression turns out cumbersome when integrating over $\mathcal{O} = [-\gamma, \gamma]$. For this reason, we integrate over $\mathbb{R}$ instead and calculate $\Sigma^2(s)$ according to the formula

$$\Sigma^2(s) := a^2 \int_{\mathbb{R}_+} \int_{\mathbb{R}_+} \int_{\mathbb{R}} \int_{\mathbb{R}} e^{-bu} e^{-bv} d_\ell(T_1 - s, u) d_\ell(T_1 - s, v) \omega(v - z) q(z, y) \omega(u - y) dydzdudv.$$

Since all terms are positive and $\Sigma^2(s) < \infty$, it is then easy to see that, for

$$\Sigma^2_\gamma(s) := a^2 \int_{\mathbb{R}_+} \int_{\mathbb{R}_+} \int_{-\gamma}^{\gamma} \int_{-\gamma}^{\gamma} e^{-bu} e^{-bv} d_\ell(T_1 - s, u) d_\ell(T_1 - s, v) \omega(v - z) q(z, y) \omega(u - y) dydzdudv,$$

the limit $\lim_{\gamma\to\infty} \Sigma^2_\gamma(s) = \Sigma^2(s)$ holds. We then calculate $\Sigma^2(s)$ explicitly.

Proposition 5.2. *In the setting described above, the volatility $\Sigma^2(s)$, $t \leq s \leq \tau$, is given by*

$$\Sigma^2(s) = \frac{2a^2}{kb^4\ell^2} \left\{ \frac{2}{3} \left(b^2 + 3\right) \left(1 + e^{-2b\ell}\right) - 2e^{-b\ell} \left(\frac{b^2}{6} \left(3(\ell - 2)\ell^2 + 4\right) - 3\ell + 2 \right) \right\} e^{-2b(T_1 - s)}.$$

Proof. See Appendix A.7.

And, consequently, we can calculate ξ^2 in equation (3.11).

Proposition 5.3. *In the setting described above, we get*

$$\xi^2 = \frac{a^2}{kb^5\ell^2}\left(e^{-2b(T_1-\tau)} - e^{-2b(T_1-t)}\right) \cdot \\ \cdot \left\{\frac{2}{3}\left(b^2+3\right)\left(1+e^{-2b\ell}\right) - 2e^{-b\ell}\left(\frac{b^2}{6}\left(3(\ell-2)\ell^2+4\right) - 3\ell+2\right)\right\}.$$

Proof. The results follows by integrating $\Sigma^2(s)$ from Proposition 5.2 over the interval $[t,\tau]$. □

5.3 The initial forward curve

Finally, we need to introduce a suitably parametrized initial forward curve, $g_t = g(t,\cdot) \in \mathcal{H}_\alpha$. We choose the Nelson-Siegel curve, which is defined for $x \geq 0$ by

$$g_t(x) = g_{NS}(x) := \alpha_0 + (\alpha_1 + \alpha_2\alpha_3 x)\, e^{-\alpha_3 x}, \tag{5.5}$$

for $\alpha_0, \alpha_1, \alpha_2, \alpha_3 \in \mathbb{R}$, $\alpha_3 > 0$. Let us notice that it does not depend on time t. This curve has been first introduced for modelling the forward rates by [34], and has been already applied in the context of energy markets by [8]. We need however to check that $g_{NS} \in \mathcal{H}_\alpha$. This is ensured in the following Lemma.

Lemma 5.4. *The Nelson-Siegel curve in equation* (5.5) *belongs to* $\mathcal{H}_\alpha$ *if and only if* $\alpha < 2\alpha_3$.

Proof. The condition is found by calculating the $\mathcal{H}_\alpha$-norm of g_{NS}, namely

$$\begin{aligned}\|g_{NS}\|_\alpha^2 &= g_{NS}^2(0) + \int_{\mathbb{R}_+} g_{NS}'^2(x)\alpha(x)dx \\ &= (\alpha_0+\alpha_1)^2 + \int_{\mathbb{R}_+} \alpha_3^2\left(\alpha_2 - \alpha_1 - \alpha_2\alpha_3 x\right)^2 e^{(\alpha-2\alpha_3)x}dx,\end{aligned}$$

where the integral converges if and only if $\alpha - 2\alpha_3 < 0$. □

With the explicit representation for g_t, we compute $\mu(g_t)$: from Lemma 3.1 we get that

$$\mu(g_t) = \frac{1}{\ell}\int_{T_1-t}^{T_1+\ell-t} g_{NS}(y)dy,$$

and by direct computation, we then calculate the drift

$$\mu(g_t) = \frac{1}{\ell}\left\{\alpha_0\ell + \frac{1}{\alpha_3}e^{-\alpha_3(T_1-t)}\left(\alpha_1+\alpha_2+\alpha_2\alpha_3(T_1-t)\right) + \\ -\frac{1}{\alpha_3}e^{-\alpha_3(T_1+\ell-t)}\left(\alpha_1+\alpha_2+\alpha_2\alpha_3(T_1+\ell-t)\right)\right\}, \tag{5.6}$$

which is the last component we need in order to get a fully specified option price functional.

6 Implementation and results

In this Section, we describe implementation details and report our findings. From Section 5, we obtain that the vector of model parameter is $\theta = (a, b, k, \alpha_0, \alpha_1, \alpha_2, \alpha_3) \in \mathbb{R}^7$, i.e., $n = 7$. In this vector $a, b \geq 0$ are parameters of the volatility operator introduced in Section 5.2, and

$k \geq 0$ is the parameter of the covariance operator, see Section 5.1. Finally $\alpha_0, \alpha_1, \alpha_2, \alpha_3 \in \mathbb{R}$ with $\alpha_3 > 0$ are the parameters for the Nelson-Siegel curve as introduced in Section 5.3.

Since we are considering European-style call options written on forward-style swaps with delivery period, the vector of contract features λ is given by $\lambda = (K, \tau, T_1, \ell) \in \mathbb{R}^4$, hence $m = 4$, where $K > 0$ is the strike price, τ is the time to maturity, $T_1 \geq \tau$ is the starting of delivery of the swap, and, finally, $\ell > 0$ is the length of the delivery period. Let us remember that, in view of the grid-based learning approach, we shall create a grid of values for λ. In our framework, this would be to create a four-dimensional grid. We then decide to set $T_1 = \tau$, namely the time to maturity for the option coincides with the start of delivery of the swap, and $\ell = 1/12$, namely we consider only contracts with one month of delivery as the time unit is one year. Then $\lambda = (\tau, K) \in \mathbb{R}^2$ and $m = 2$, so that we will have to create a two-dimensional grid, as introduced in Section 4.2.2. Finally, for all the experiments we fix $t = 0$ as evaluation time, and $r = 0$ as risk-free interest rate. In particular, we shall consider three different experiments: pointwise learning, grid-based learning and grid-based learning with convolutional networks.[1]

In all the experiments, we use the Adam optimizer both for training the neural network and in the calibration step, and we consider the mean squared error loss function. We set the number of epochs to 200 for the approximation step, since considering a higher number does not improve the accuracy of the network. We set the number of epochs for the calibration step to 1000 instead, since more computational effort is necessary for fitting the prices in this second step. The batch size is fixed to 30 for all the experiments.

6.1 Grid-based learning

Fo the grid-based learning approach, we have $\lambda = (\tau, K)$, hence we create a grid of values of times to maturity and strike prices which we believe to be reasonable for a case study in the electricity markets. Let $\Lambda^{grid} := \Lambda^{grid}_\tau \times \Lambda^{grid}_K$ where

$$\Lambda^{grid}_\tau = \{1/12, 2/12, 3/12, 4/12, 5/12, 6/12, 1\},$$
$$\Lambda^{grid}_K = \{31.6, 31.8, 32.0, 32.2, 32.4, 32.6, 32.8, 33.0, 33.2\}.$$

The grid has thus dimension $m_1 \times m_2 = 7 \times 9$. We point out that the range in Λ^{grid}_K is relatively narrow if compared to the strike prices available, for example, at EEX. The reason for this choice is that by selecting a wider range for K, some options are far out of the money, and thus very cheap for parameter choices that result in low overall volatility. As considering options with value less than a cent is not very interesting and poses also numerical challenges, we restrict the setting of our experiments to this narrow range. To widen the range, one may consider options with low strikes only in combination with model parameters a, b and k that lead to large overall volatility ξ. From a practical perspective, this is reasonable since options far out of the money are typically only in demand in large volatility environments. While the network architecture would not actually change significantly, choosing the grid Λ^{grid}_K specified above for all possible model parameters is convenient and thus preferred for this case study.

We define a neural network $\mathcal{N} : \Theta \to \mathbb{R}^{7\times 9}_+$, with $\Theta \subset \mathbb{R}^7$ so that $d = n = 7$ is the input dimension and $p = 7 \times 9$ is the output dimension. In particular, we consider a four-layer neural network ($L = 4$) with number of nodes $\mathbf{n} = (30, 30, 30, 63)$, and the ReLU activation function. The final output is reshaped into a grid of dimension 7×9. The number of total parameters to calibrate is $M = |\mathcal{V}| = 4053$. We consider a training set of size $N_{train} = 40000$ and a test set of size $N_{test} = 4000$, which is also used for the calibration step. We consider

[1] The code is implemented in TensorFlow 2.1.0 and is available at GitHub: HJM_calibration_with_NN.

$\theta = (a, b, k, \alpha_0, \alpha_1, \alpha_2, \alpha_3) \in \Theta$ with Θ defined by

$$\Theta := [0.2, 0.5] \times [0.5, 0.8] \times [8.0, 9.0] \times [34.2, 34.7] \times [-1.5, -1.0] \times [0.2, 1.2] \times [4.5, 5.0].$$

Thus the annual implied spot price volatility is between 20% and 50%, while the half-life for the process, that is the time taken for the price to revert half-way back to its long-term level, is approximately between 4.5 and 7 months, see [16] for details.

Since Θ has dimension $n = 7$, if one wants to consider for each dimension a number of ν different values for the training set, and then combine every value in the first dimension with every value in the second dimension, and so on, then the size of the training set would be ν^7, which easily gets very large. This approach would allow for training the network with all possible combinations of the parameters and possibly ensure a good approximation also of the 1-dimensional restrictions of the pricing function. However, for a reasonable size of the training set, as for instance $N_{train} + N_{test} = 44000$ chosen here, one would get only $44000^{1/7} \approx 5$ different values for each of the parameter. We therefore consider a uniform grid of size $N_{train} + N_{test} = 44000$ for each of the $n = 7$ dimensions composing Θ, and we randomly match the values in each dimension to form a training set of size $N_{train} = 40000$ and a test set of size $N_{test} = 4000$. The corresponding prices have then been calculated with the formula (5.1). We point out that each of the N_{train} (or N_{test}) samples are a grid of size $63 = 7 \times 9$ with each dimension corresponding to the different values for K's and τ's. Each vector calibration is thus performed starting from a sample of 63 prices, as defined in Section 4.2.2.

In Figure 2 we report the average relative error and the maximum relative error in the training step, both for the training and test set. In particular, the errors have been clustered in order to obtain a grid corresponding to the different contracts $(\tau, K) \in \Lambda_\tau^{grid} \times \Lambda_K^{grid}$. We notice that the average relative error is quite low, showing a good performance of the neural network in approximating the price functional. The worst accuracy is for the contracts with higher strike price, probably due to the fact that the price for these contracts is small.

In Figure 3 we report the relative error for the components of $\hat{\theta}$ in the calibration step. For some of the parameters, such as a and α_2 we notice a certain pattern, namely, for higher values of the parameter the error is smaller and vice versa. However, the performance in calibration is not particularly good: a, b, and α_2 have a mean relative error which is more than 20%. On the other hand, even if the model parameters are not accurately estimated, by substituting $\hat{\theta}$ in the neural network, we get a price approximation $\hat{\Pi}$ which is quite good. This can be seen in Figure 4, where we report the average and the maximum relative error after calibration. For mid-maturity contracts we observe the best accuracy.

6.2 Grid-based learning with convolutional networks

With an output dimension of $p = 63$ in the grid-based learning approach, the number of neural network parameters grows fast, since in the last layer the weight matrix V_L must have $n_L = 63$ rows. In imaging, a solution adopted to reduce the total number of parameters is to consider convolutional networks as described in Section 4. We consider a four-layer neural network ($L = 4$). The first two layers are dense with node size $n_1 = 30$ and $n_2 = 63$. It follows a reshaping into $\mathbb{R}^{7\times 9}$, and two convolutional layers with filter size, respectively, 63 and 1. Both the convolutional layers have kernel size equal to 3. We use the ReLU activation function and the number of total parameters to calibrate is $M = |\mathcal{V}| = 2764$. The training and test sets are the same ones we used for the dense network case above, hence each of the $N_{train} = 40000$ and $N_{test} = 4000$ sample is a grid of 63 values corresponding to the different K's and τ's, and each vector calibration is performed with a sample of 63 prices.

In Figure 5 we report the average relative error and the maximum relative error in the

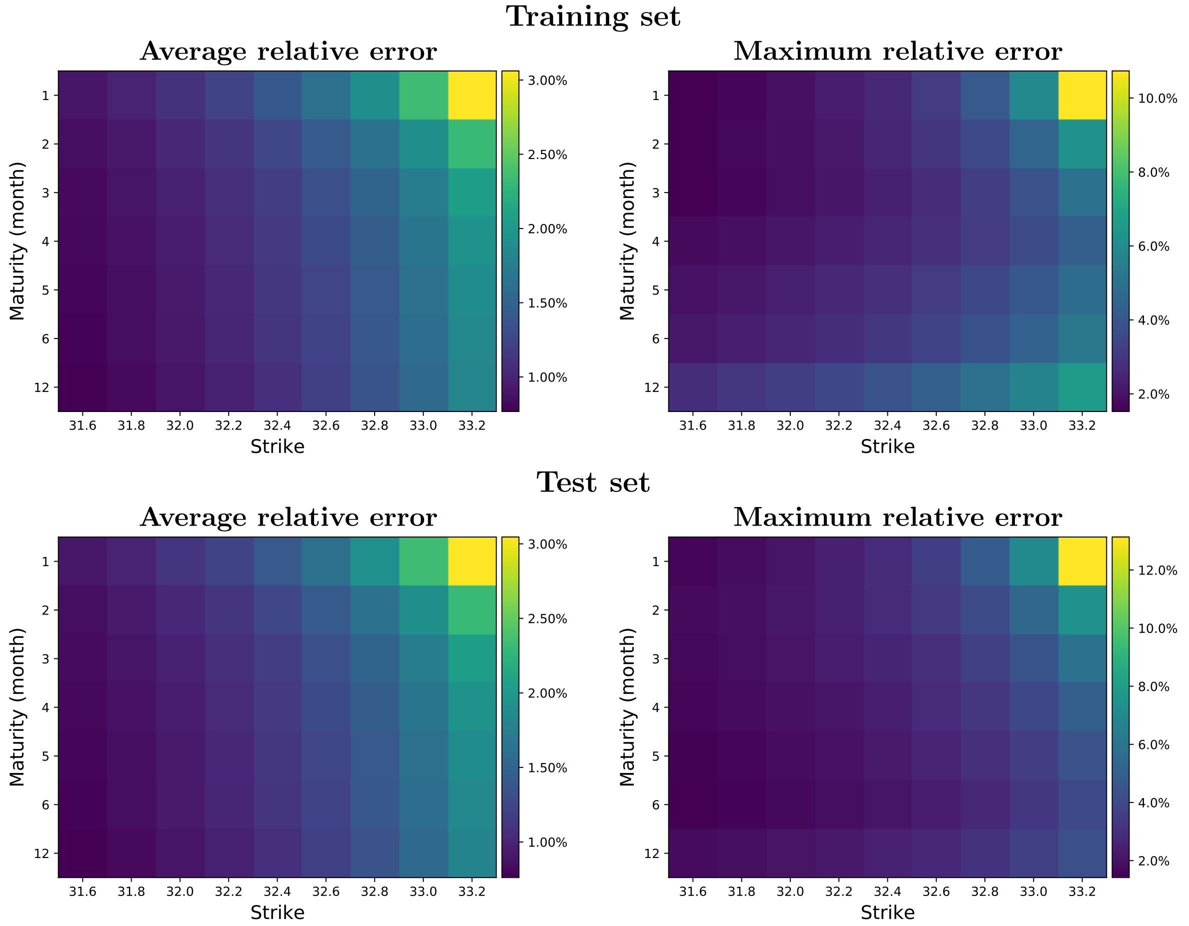
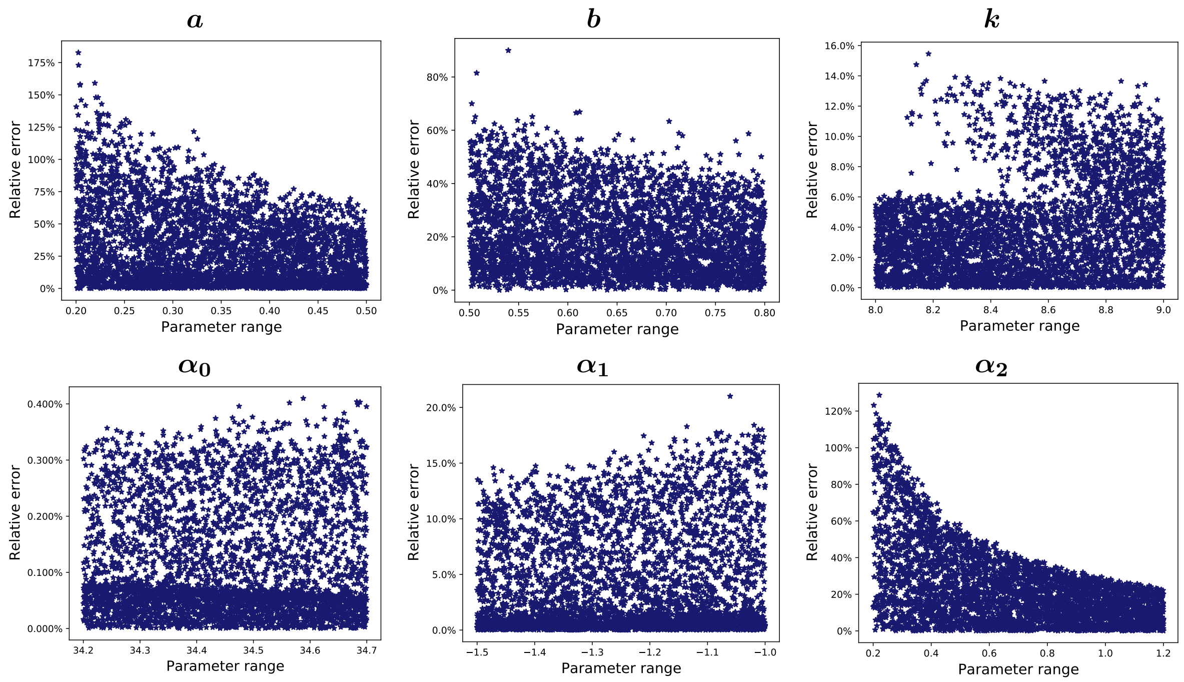
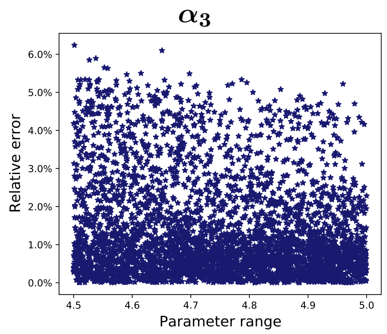


Figure 2: Average relative error and Maximum relative error in the approximation step with the grid-based learning approach and a dense neural network.





	Mean	Median
$\boldsymbol{a}$:	32.4%	24.6%
$\boldsymbol{b}$:	20.7%	17.9%
$\boldsymbol{k}$:	3.94%	3.24%
$\boldsymbol{\alpha_0}$:	0.12%	0.07%
$\boldsymbol{\alpha_1}$:	4.29%	1.91%
$\boldsymbol{\alpha_2}$:	22.3%	17.2%
$\boldsymbol{\alpha_3}$:	1.34%	0.86%

Figure 3: Relative error for the model parameters calibrated with the grid-based learning approach and a dense neural network.

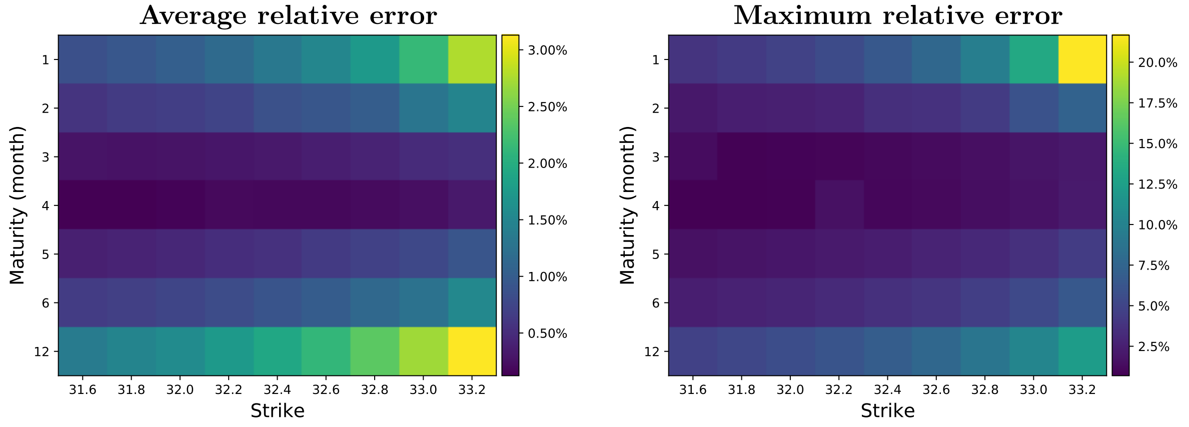


Figure 4: Average relative error and Maximum relative error after calibration with the grid-based learning approach and a dense neural network.

approximation step clustered with respect to the different contracts. Looking at the average relative error, we notice a better accuracy compared to the grid-based approach with a dense network. However, for the maximum relative error, we get in the upper-right corner (corresponding to the contract with $(\tau, K) = (1/12, 33.2)$) a value which is almost three times its corresponding value with the dense network. Also, the rest of the grid for the maximum relative error is approximately around 5%, while for the dense network it is around 2% for most of the cells. We conclude that the convolutional neural network considered has better average relative errors, but worse maximum relative errors.

In Figure 6 we report the relative error for the components of $\hat{\theta}$ in the calibration step. We notice the same pattern for a and α_2 previously found. We also notice worse accuracy for a, but much better accuracy for b, so that overall this is not really different from the dense network experiment. In Figure 7 a similar pattern is visible as in the price approximation after calibration with dense network, where the mid-maturity contracts have the lowest relative error. However, the two networks considered have approximately the same overall level of accuracy.

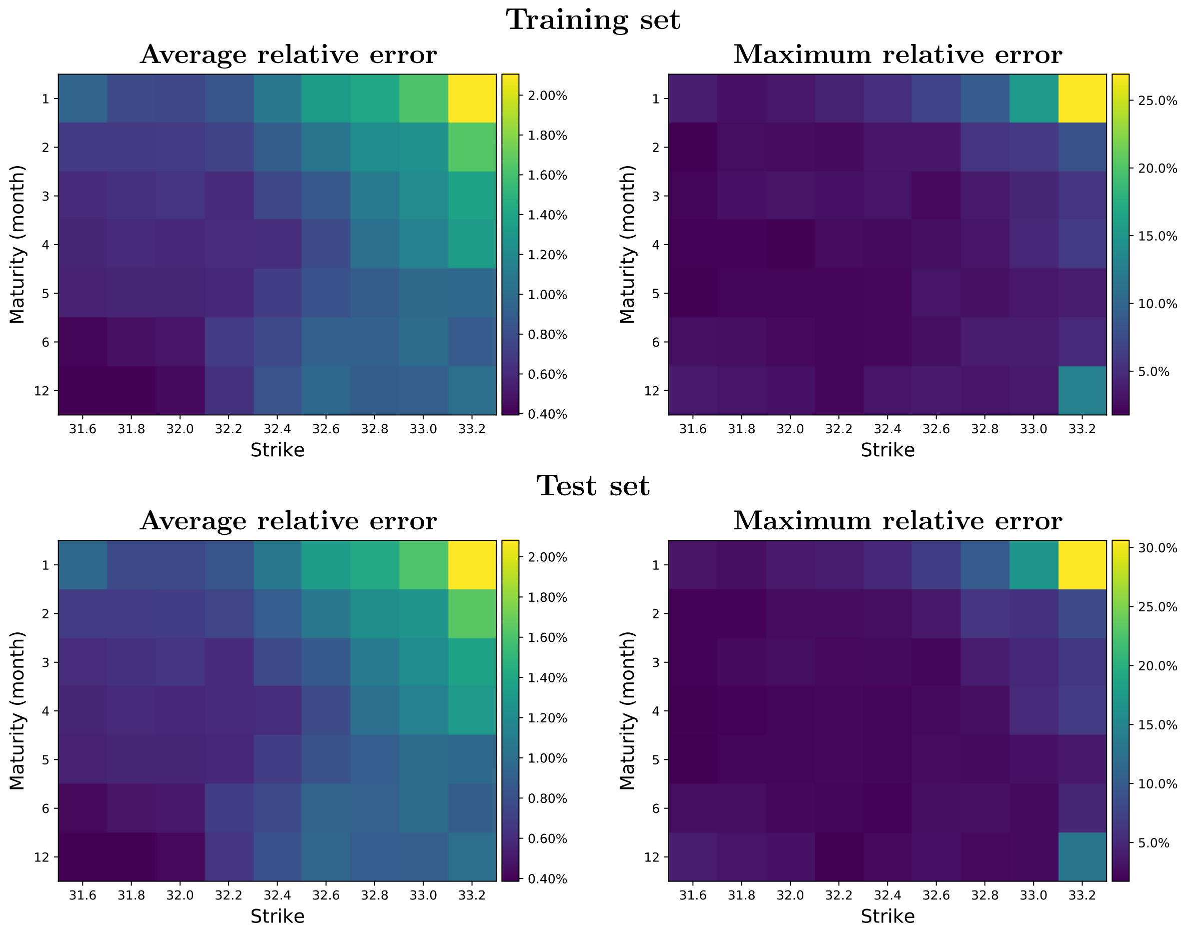
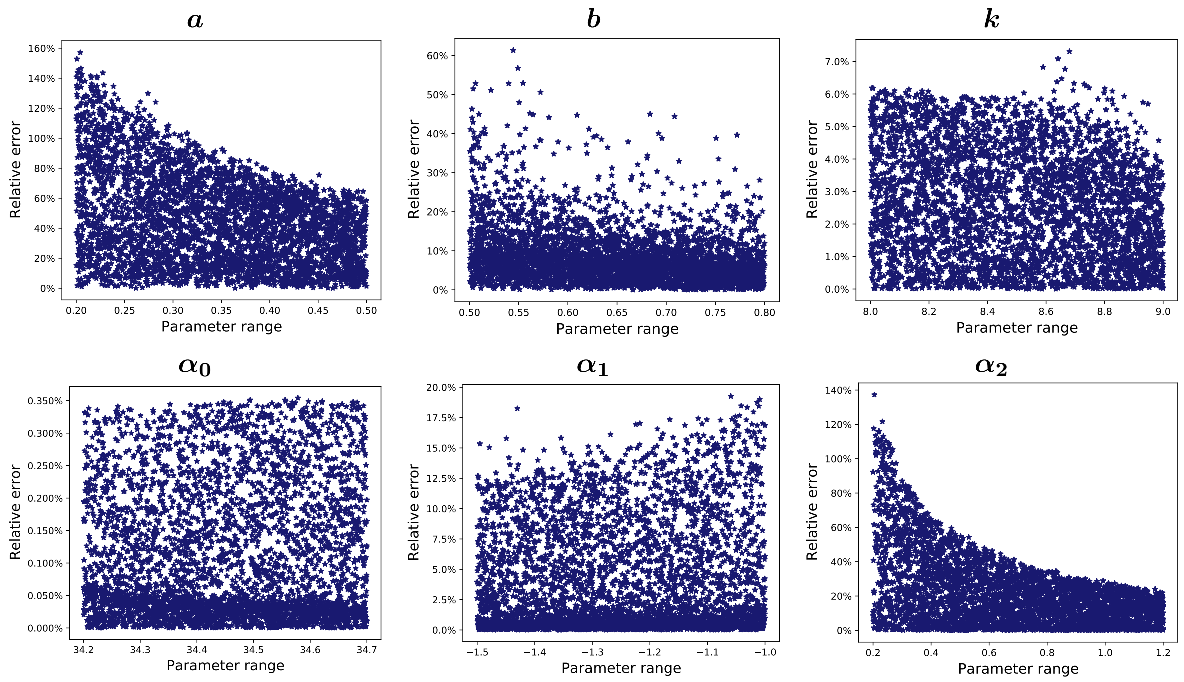
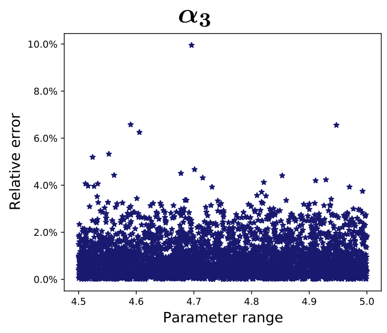


Figure 5: Average relative error and Maximum relative error in the approximation step with the grid-based learning approach and a convolutional neural network.

6.3 Pointwise learning

We implement the pointwise learning approach. We define a neural network $\mathcal{N} : \Lambda \times \Theta \to \mathbb{R}_+$, with $\Lambda \subset \mathbb{R}^2$ and $\Theta \subset \mathbb{R}^7$, then $d = 9$ is the input dimension and $p = 1$ is the output dimension.





	Mean	Median
$\boldsymbol{a}$:	44.9%	41.1%
$\boldsymbol{b}$:	8.43%	6.61%
$\boldsymbol{k}$:	2.62%	2.52%
$\boldsymbol{\alpha_0}$:	0.11%	0.07%
$\boldsymbol{\alpha_1}$:	4.35%	2.31%
$\boldsymbol{\alpha_2}$:	21.4%	16.2%
$\boldsymbol{\alpha_3}$:	0.86%	0.65%

Figure 6: Relative error for the model parameters calibrated with the grid-based learning approach and a convolutional neural network.

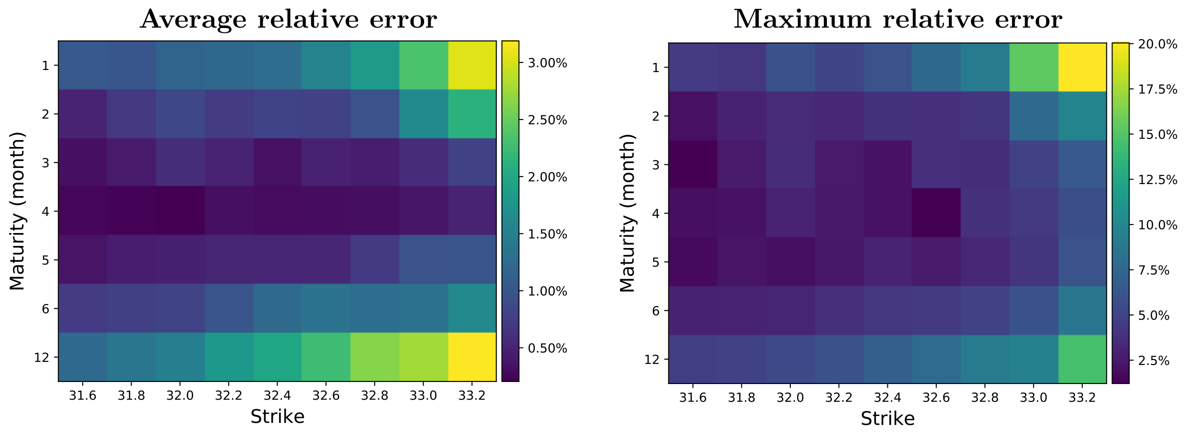


Figure 7: Average relative error and Maximum relative error after calibration with the grid-based learning approach and a convolutional neural network.

We consider a four-layer neural network ($L = 4$) with nodes size $\mathbf{n} = (30, 30, 30, 1)$, and the ELU activation function. The number of total parameters to calibrate is $M = |\mathcal{V}| = 2191$. We consider a training set of size $N_{train} = 60000$ and a test set of size $N_{test} = 6000$. These are bigger than the sets considered in the grid-based approach to reflect that each sample corresponds to a single value, while for the grid-based approach each sample corresponds to 63 values, hence in this latter case the training sample is actually bigger. We take $\theta = (a, b, k, \alpha_0, \alpha_1, \alpha_2, \alpha_3) \in \Theta$ with Θ as above, while $\lambda = (\tau, K) \in \Lambda^{point}$ for $\Lambda^{point} = \Lambda_\tau^{point} \times \Lambda_K^{point} = [1/12, 1] \times [31.6, 33.2]$. We notice that $\Lambda^{grid} \subset \Lambda^{point}$: this allows us to compare the pointwise learning approach with the grid-based learning approach. Training and test sets have been generated starting from $\Theta \times \Lambda$ in the same manner as described in the grid-based approach, calculating the corresponding prices with the formula in equation (5.1).

In order to produce error plots comparable to the ones in the grid-based learning, we cluster the prices in the following way. We introduce a new grid $\hat{\Lambda} := \hat{\Lambda}_\tau \times \hat{\Lambda}_K$, where

$$\hat{\Lambda}_\tau := \{1/12, 1/12 + 1/24, 2/12 + 1/24, 3/12 + 1/24, 4/12 + 1/24, 5/12 + 1/24, 6/12 + 1/24, 1\},$$
$$\hat{\Lambda}_K := \{31.6, 31.7, 31.9, 32.1, 32.3, 32.5, 32.7, 32.9, 33.1, 33.2\},$$

and we label with $K = 31.6$ the prices corresponding to a strike in the interval $[31.6, 31.7]$, while we label with $K = 31.8$ the prices corresponding to a strike in the interval $[31.7, 31.9]$, and so on for each of the strike price in Λ_K^{grid} defined in the grid-based learning approach. We do the same for the time to maturity τ, in order to obtain a grid of clustered values corresponding to the grid Λ^{grid}. Moreover, since each of the $N_{test} = 6000$ samples corresponds to a different parameter vector, we can not perform calibration starting from the test set, as we have done previously. Instead, we calibrate by using the test set from the grid-based approach, after shape adjustment. In this manner, each calibration is performed as described in Section 4.2.1, starting from $N_{cal} = 63$ different prices corresponding to the same $\theta \in \Theta$, to be estimated.

In Figure 8 we report the average relative error and the maximum relative error in the approximation step after clustering, as described above. For both, we notice a better accuracy compared to the grid-based approaches. Still, the worst accuracy is for contracts with the highest strike price. However, when it comes to calibration, the pointwise approach turns out to be the worst. Indeed, the relative errors for $\hat{\theta}$ in Figure 9 are on average worse than in the previous experiments. It can be noticed in particular, that for a, b and k, the error is concentrated around, respectively, 45%, 30% and 5%, while before it was much more spread starting from 0%. The accuracy for a is not very good, almost reaching 50%, and the accuracy for b is the worst among the three methods. On the other hand, the accuracy for α_2 has substantially improved. In Figure 10 we can see that the relative error for $\hat{\Pi}$ after calibration is also the worst among the three methods.

6.4 Bid-ask constraints

In the setting of the grid-based learning approach with dense network of Section 6.1, we test the bid-ask loss function defined in Section 4.3. In particular, we consider the dense neural network previously trained, and we use the bid-ask loss function for the calibration step. Starting from the test set of the grid-based approach, we obtain bid and ask prices by considering 90% and 110%, respectively, of the original prices. In Figures 11 we report the relative error for the model parameters calibrated, which are a bit worse than the values found in the experiment with the same neural network but exact prices. However, considering that the information given to the network is much weaker due to the relatively wide bid-ask range, the results are surprisingly good and the overall error is of similar magnitude as in the rest of the experiments.

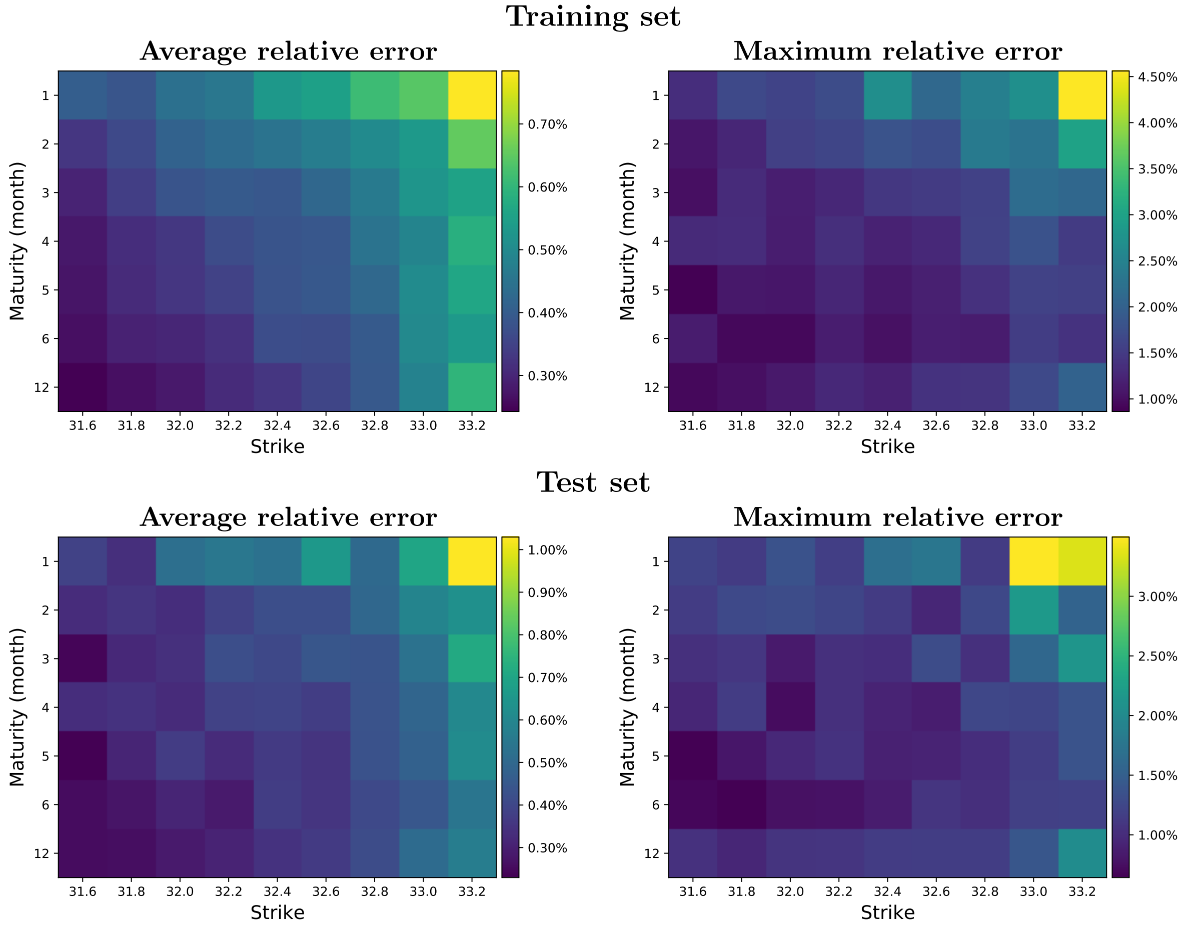
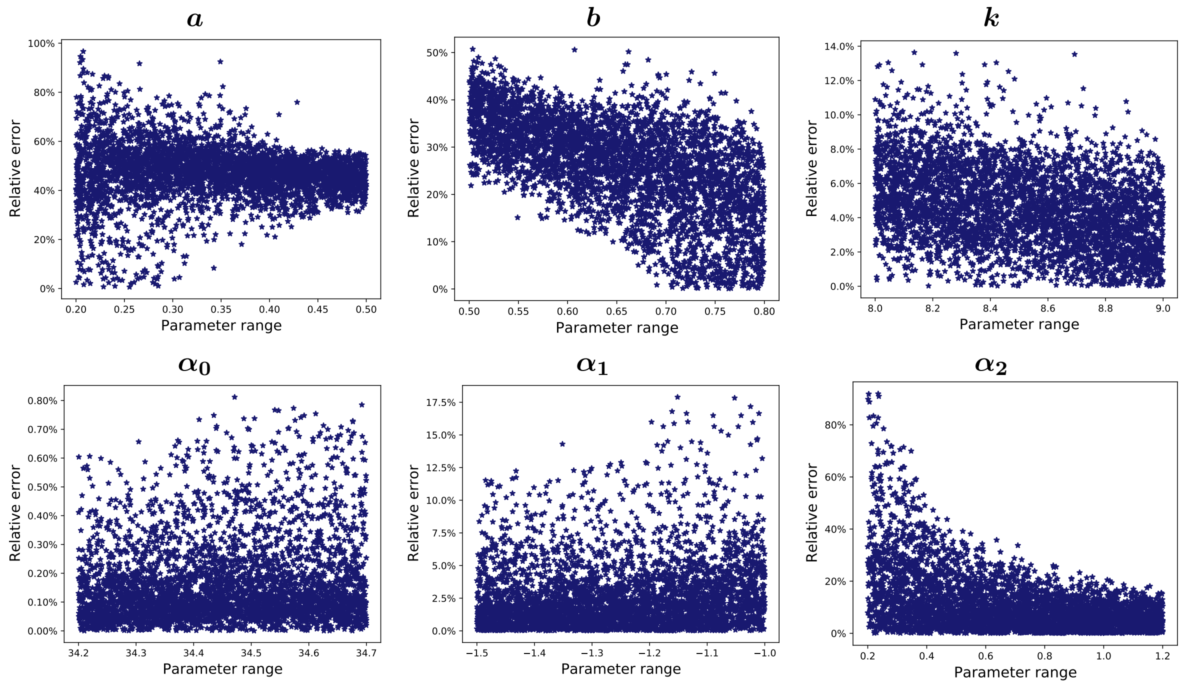
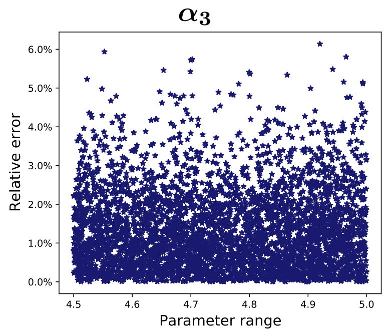


Figure 8: Average relative error and Maximum relative error in the approximation step with the pointwise learning approach.





	Mean	Median
a:	46.9%	47.3%
b:	26.3%	27.6%
k:	4.59%	4.50%
α_0:	0.17%	0.12%
α_1:	2.72%	1.89%
α_2:	11.2%	7.67%
α_3:	1.33%	1.12%

Figure 9: Relative error for the model parameters calibrated with the pointwise learning approach.

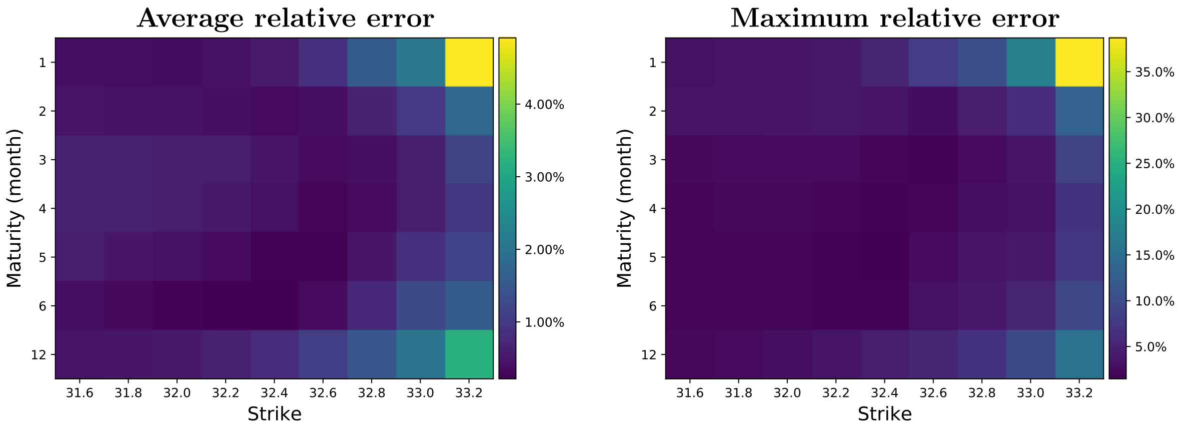
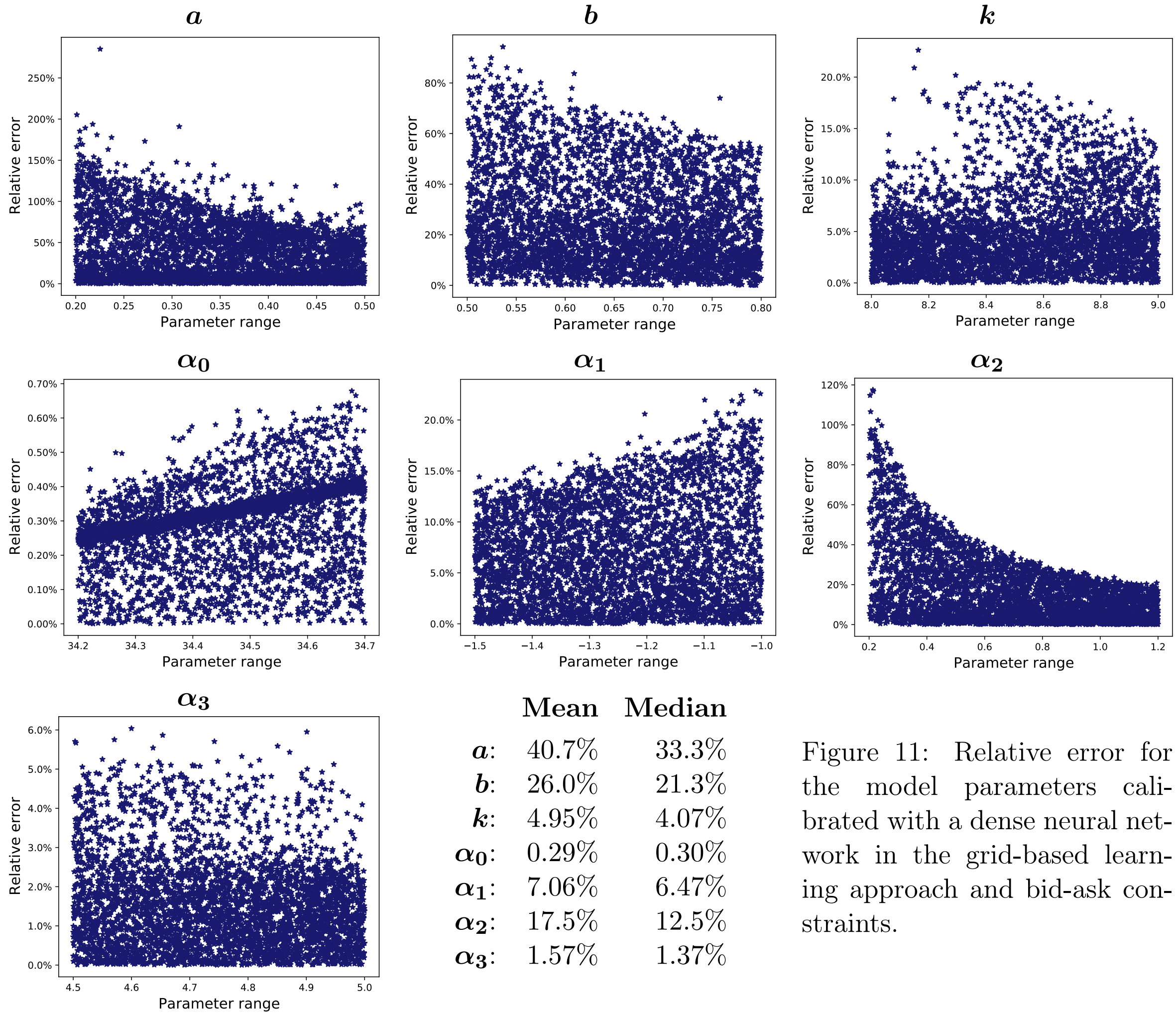


Figure 10: Average relative error and Maximum relative error after calibration with the pointwise learning approach.

We calculate the rate of mismatched prices, namely the percentage of prices which does not lie within the bid-ask constraints. In Figure 12, on the left we have the plot of the mismatched prices rate that we observe before calibration, namely the rate we obtain by plugging in the neural network the starting parameters that we need in order to initialize the optimization routine. On the right, the plot shows the percentage of mismatched prices after calibration. As we can see, the final rate is 0% for almost all the contracts, except for $(\tau, K) = (1, 33.2)$ and for $(\tau, K) = (1/12, 33.2)$. A random sub-sample of 100 prices have been reported in Figure 13 where, on the top panel (corresponding to $(\tau, K) = (1, 33.2)$), we can notice several prices being outside the constraints (marked with the symbol $\bigstar$), in the mid panel (corresponding to $(\tau, K) = (1/12, 33.2)$) only some of the prices are outside the constraints, while in the bottom panel (corresponding to $(\tau, K) = (4/12, 32.2)$) all the prices are within the constraints (marked with the symbol $\times$), referring to a rate of mismatching equal to 0%. However, the prices lying outside the bid-ask interval are indeed very close to the boundaries, which confirms the suitability of the bid-ask loss function whenever exact prices are not available. We have been testing the bid-ask loss function also with more narrow constraints. In this case, it is sufficient to increase the number of iterations for the optimizing routine to obtain very similar results.



	Mean	Median
$\boldsymbol{a}$:	40.7%	33.3%
$\boldsymbol{b}$:	26.0%	21.3%
$\boldsymbol{k}$:	4.95%	4.07%
$\boldsymbol{\alpha_0}$:	0.29%	0.30%
$\boldsymbol{\alpha_1}$:	7.06%	6.47%
$\boldsymbol{\alpha_2}$:	17.5%	12.5%
$\boldsymbol{\alpha_3}$:	1.57%	1.37%

Figure 11: Relative error for the model parameters calibrated with a dense neural network in the grid-based learning approach and bid-ask constraints.

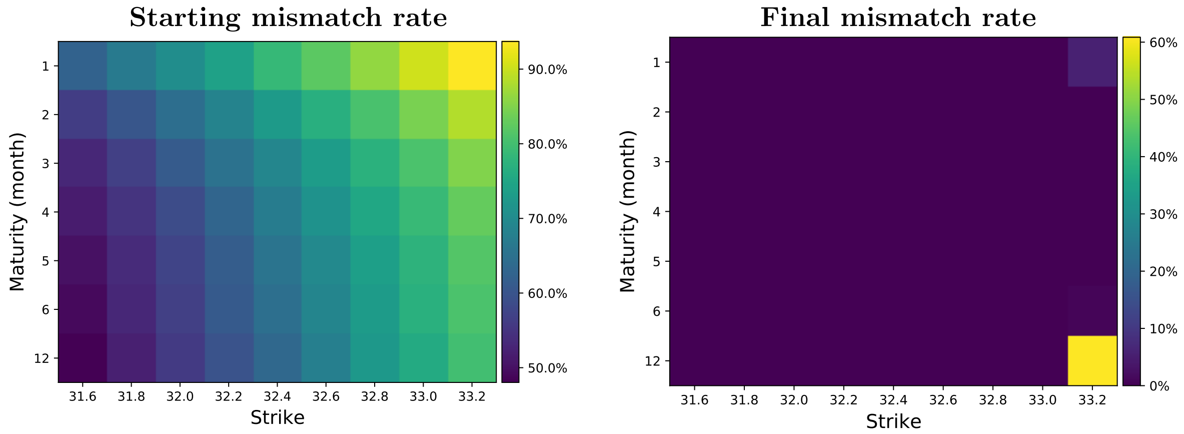


Figure 12: Starting mismatch rate and Final mismatch rate after calibration with a dense network in the grid-based learning approach and bid-ask constraints.

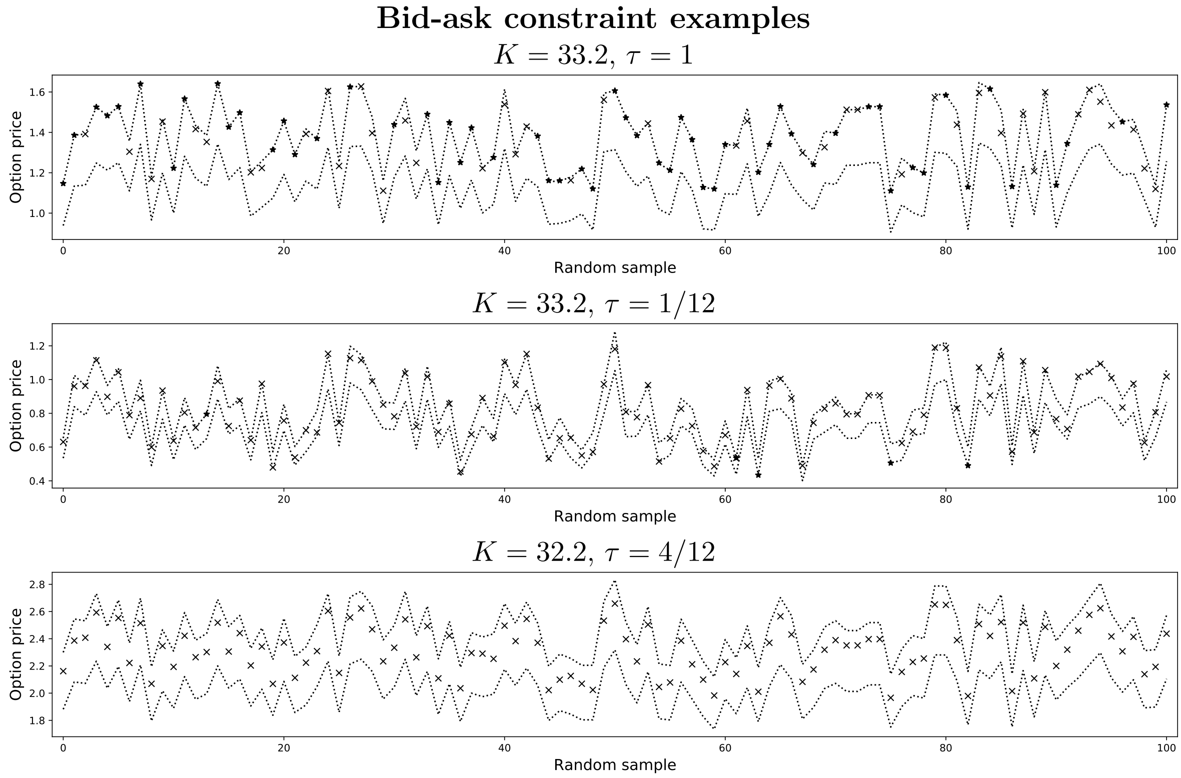


Figure 13: Examples of a 20% bid-ask constraint with a dense neural network and the grid-based learning approach. The symbol × indicates the prices which after calibration are inside the constraints, the symbol ★ the ones outside.

6.5 The non-injectivity issue

In all the experiments described above, a common issue appears. The accuracy achieved in calibration is not particularly convincing, especially for the parameters regarding the volatility and the covariance operator, a, b and k. Slightly better results was obtained for the Nelson-Siegel curve parameters, α_0, α_1 and α_3, with the exception of α_2. See Figure 3, 6, 9 and 11. On the other hand, the relative error for the price approximation after calibration shows high degree of accuracy. See Figure 4, 7, 10 and 12. This phenomenon appears as a discrepancy.

From this, one may conclude that neural networks can be used for approximation and calibration of sophisticated stochastic models, such as the HJM model considered here. However, the original meaning of the model parameters may get lost in the approximation step. As pointed out in [6], it is somehow to be expected that the neural network $\mathcal{N}$ is non-injective in the input parameters on large part of the inputs domain. We shall briefly analyse this.

The price formula (5.1), once fixed the strike K and the time to maturity τ, crucially depends on ξ as derived in Proposition 5.3 and on $\mu(g_t)$ (see equation (5.6)):

$$\Pi(t) = e^{-r(\tau-t)} \left\{ \xi\phi\left(\frac{\mu(g_t)-K}{\xi}\right) + (\mu(g_t)-K)\,\Phi\left(\frac{\mu(g_t)-K}{\xi}\right) \right\}.$$

However, a first observation is that ξ is only a scale, while $\mu(g_t)$ defines the distance from the strike price K, hence it is more influencial on the final price level. Let us focus on ξ:

$$\xi^2 = \frac{a^2}{kb^5\ell^2}\left(e^{-2b(T_1-\tau)} - e^{-2b(T_1-t)}\right) B(b^2, e^{-b\ell}),$$

where $B(b^2, e^{-b\ell})$ simply indicates a term proportional to b^2 and $e^{-b\ell}$. In the front coefficient, the three parameters a, b, and k are multiplied together directly or inversely, and a decrease in a might be, for example, compensated by a decrease in b or k, and vice versa, meaning that several combinations of values for a, b, and k lead to the same overall ξ. Thus we may suspect that it can be hard for the neural network to identify the right vector of parameters in the calibration step, despite reaching good level of accuracy for the price.

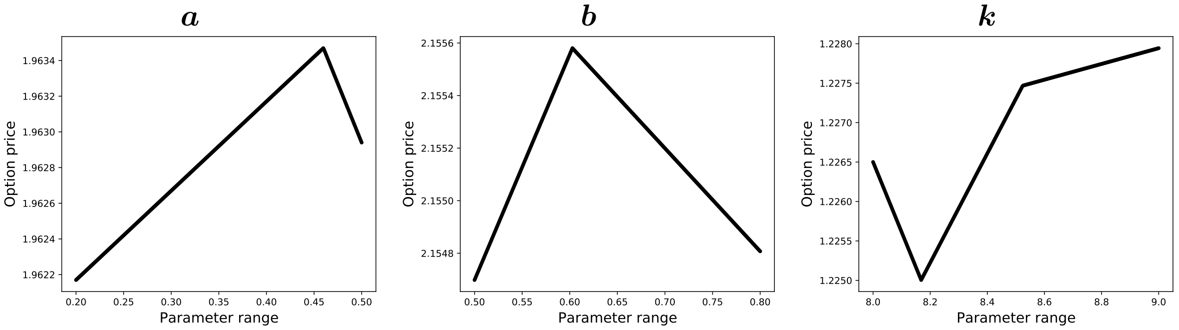
In Figure 14 we report an example of non-injectivity with respect to the parameters a, b and k that we have observed for the dense network trained in the grid-based learning approach. Here we notice indeed that the neural network is not injective as a map when all the parameters, except one, are fixed. We also notice that the map is only little sensitive to the change in the parameters a, b and k, which also explains the struggle in recovering these three parameters.

Similar observations can be done for the drift:

$$\mu(g_t) = \alpha_0 + \frac{e^{-\alpha_3(T_1-t)}}{\alpha_3\ell}\left(\alpha_1+\alpha_2+\alpha_2\alpha_3(T_1-t)\right) - \frac{e^{-\alpha_3(T_1+\ell-t)}}{\alpha_3\ell}\left(\alpha_1+\alpha_2+\alpha_2\alpha_3(T_1+\ell-t)\right).$$

Here the role of α_0 is specific since it defines the starting level of the curve, and indeed α_0 is the parameter that gets the best accuracy in estimation. However, α_2 appears in two positions: first added to α_1 and then multiplied by α_3. It is thus difficult for the neural network to outline the role of α_2 in the drift. In the Nelson-Siegel curve definition in equation (5.5), α_2 defines the position of the "bump" in the curve. Then, the drift $\mu(g_t)$ is obtained my integrating the curve within the delivery period of the contract. This integration might then smoothen the curve, and make it difficult to locate the "bump". This might explain why the accuracy in estimating α_2 is worse than for the other Nelson-Siegel parameters.

The problem described above may arise also with more traditional calibration techniques. It might simply be due to the fact that one is dealing with a non-convex optimisation problem with potentially several local minima. What is unique to the neural network is that the approximated price function $\mathcal{N}(x)$ might violate some basic arbitrage conditions. One particular example of these conditions is that the call price should be an increasing function of the volatility. In our case, in particular, since a arises as a constant factor in the volatility, the price should be increasing in a, when all the other parameters are fixed. In Figure 14 we observe that this is violated here. The violation is however relatively small since overall the prices are well fit.



$K = 32.4$, $\tau = 3/12$ $\theta = [a, 0.7, 8.0, 34.4, -1.0, 0.5, 5.0]$

$K = 33.2$, $\tau = 1$ $\theta = [0.45, b, 8.5, 34.3, -1.0, 0.5, 5.0]$

$K = 31.8$, $\tau = 1/12$ $\theta = [0.5, 0.8, k, 34.4, -1.0, 0.5, 5.0]$

Figure 14: Examples of non injectivity for the dense neural network trained in the grid-based learning approach. In each image, only one of the parameters is varying, while the rest is fixed.

We conclude the article with the following Theorem showing that it is possible to construct ReLU neural networks which act as linear maps.

Theorem 6.1. *Let $A \in \mathbb{R}^{p \times d}$. Then for any $L \geq 2$ and any $\mathbf{n} = (d, n_1, \dots, n_{L-1}, p)$ with $n_i \geq 2d$, $i = 1, \dots, (L-1)$, there exists an L-layer ReLU neural network $\mathcal{N} \colon \mathbb{R}^d \to \mathbb{R}^p$ with dimension $\mathbf{n}$, which satisfies*

$$\mathcal{N}(x) = Ax, \qquad \textit{for all } x \in \mathbb{R}^d.$$

Proof. See Appendix A.8.

Theorem 6.1 proves that we can construct a ReLU L-layer neural network which acts as a linear map. As there are infinitely many non-injective linear maps (the zero-map being a trivial example), it is then possible to construct infinitely many non-injective ReLU neural networks. Obviously, this does not show that a non-injective network, such as the one constructed in the proof of Theorem 6.1, will also minimize the objective function used for training. It however may represent an important starting point to (try to) understand the injectivity issue.

7 Conclusions, remarks and further ideas

We consider a state-dependent HJM model defining the forward curve dynamics under the risk-neutral measure. To specify the model, we construct a volatility operator which turns elements from the noise space $L^2(\mathcal{O})$ into elements in the forward curves space, the Filipović space $\mathcal{H}_\alpha(\mathbb{R}_+)$ defined in Section 2. This is achieved by introducing a class of integral operators. We prove some conditions which ensure that an operator of this class is well defined, and that the resulting volatility satisfies the Lipschitz and linear growth conditions required for existence of a mild solution to the HJM equation. We then restrict our attention to a parametrized deterministic volatility operator. It captures some important properties studied in the literature, such as a pronounced seasonality and the Samuelson effect.

We focus then on forward contracts with delivery periods, called swaps, and options written on them. By considering a parametric covariance operator and a parametric model for the initial forward curve, namely the Nelson-Siegel curve, we obtain a vector θ of parameters that determines the option prices in the model. We derive the respective pricing functional and train a neural network to approximate it. Then this neural network is used in a calibration step to find the best model parameters (in a mean squared error sense) to fit the (artificial) market price data. We do this both with the pointwise and the grid-based learning approach described in Section 4.2, and we test different neural network architectures.

From all the experiments, we can conclude that neural networks perform very well in approximating the price functional, with average relative error in the test set reaching levels below 1%. However, the original meaning of the parameters in the underlying model might get lost during the approximation step. This becomes apparent in the calibration step, where the average relative error for the parameters can be of magnitude 50%, with peaks above 100% in the worst cases. On the other hand, the prices obtained with the neural network and the estimated parameters are accurate (1-4% for the average relative error), showing that the prices can be recovered even with a vector of parameters θ different from the true one.

As discussed in Section 6.5, this may be due to the non-injectivity of the neural network. Moreover, for an option with fixed maturity and strike, different combinations of the parameters can result in the same price. As stated in [6], computing the distance between the true model parameters and the estimated ones as we do here, might not be the optimal way to quantify the accuracy of the two steps approach. In [6] the authors suggest to switch to a Bayesian viewpoint and observe the posterior distribution of the model parameters θ considered as a random variable. However, this kind of analysis was beyond the scope of the current research. While the problem in recovering the true parameters stresses the importance of proper benchmarking of a neural network based approach, the accuracy in pricing suggests that neural networks might in fact turn out promising in making infinite dimensional models more tractable.

To improve accuracy in calibration one might consider a different structure for the input vectors in the training set. In our experiments, we consider a uniform grid for each of the dimension and randomly match them to generate the input set. Then, a value, for example, in the first dimension appears only once in the whole training set. It might help to construct the set in such a way that every value in the first dimension is combined with every value in the second dimension, and so on to generate a uniform grid on Θ. As pointed out before, since Θ has dimension $n = 7$, if one wants for example to train the network with 10 different values for each of the parameters (which is still a low number), the training set must be of size 10^7.

The new loss function introduced for calibration based on bid-ask prices instead of exact prices, works well for the calibration step. It penalizes the values lying outside the bid-ask range interval, and considering that the information given to the network is weaker than with exact prices, the results are promising. Indeed, after calibration, for almost all the contracts considered the price lies within the bid-ask range, or very close to the boundaries otherwise. If the prices for some of the contracts are available, while for others one has only bid and ask prices, one could for example consider a hybrid loss function, corresponding to the mean squared error for the exact prices, and to the newly defined bid-ask loss function for the bid and ask price ranges. This allows to exploit all the available information at once.

Since forward contracts are traded themselves, it may be interesting to test a different approach. As done here, the training of the neural network includes the parameters for the initial forward curve. Once trained the neural network in the off-line step, one could divide the second step in two parts. By standard techniques for interpolation, one could use the forward prices to calibrate the initial forward curve parameters. These can then be considered as fixed in the proper calibration step, so that the vector of parameters to be recovered with the neural networks is smaller; most likely leading to better results in the calibration.

One might also decide to fix the initial forward curve already at the level of the approximation step, and then train the neural network with a lower dimensional input vector θ. This would require to run the approximation step much more frequently to be updated with the current market information, but most likely improves accuracy. Another possibility without changing the setup, is to include forward contracts as options with strike $K = 0$ and time to maturity $\tau = t$ equal to the evaluation time. These contracts most efficiently allow to recover the parameters for the Nelson-Siegel curve. Given these considerations, our approach to calibrate the entire parameter vector θ only based on call options, is somehow likely to lead to a larger error than the one obtained in practice. For instance, when pricing standard options in equity markets based on a Black Scholes model, only the implied volatility is calibrated, while the current stock price, dividends and interest rate are obtained from other sources.

A Proofs of the main results

We report in this Section the proofs to the main results in the order they appear in the paper.

A.1 Proof of Theorem 2.2

For every $x \in \mathbb{R}_+$ and $f \in \mathcal{H}_\alpha$, by the Cauchy-Schwarz inequality we can write that

$$\int_{\mathcal{O}} |\kappa_t(x, y, f)h(y)|\, dy \leq \left(\int_{\mathcal{O}} \kappa_t(x, y, f)^2 dy\right)^{1/2} \left(\int_{\mathcal{O}} h(y)^2 dy\right)^{1/2} < \infty,$$

which is bounded, since $\kappa_t(x, \cdot, f) \in \mathcal{H}$ for every $x \in \mathbb{R}_+$ and every $f \in \mathcal{H}_\alpha$ by Assumption 1, and because $h \in \mathcal{H}$. Thus $\sigma_t(f)h$ is well defined for all $h \in \mathcal{H}$.

We need to show that $\sigma_t(f)h \in \mathcal{H}_\alpha$ for every $f \in \mathcal{H}_\alpha$. We start by noticing that for every $x \in \mathbb{R}_+$ the following equality holds:

$$\frac{\partial \sigma_t(f)h(x)}{\partial x} = \int_{\mathcal{O}} \frac{\partial \kappa_t(x, y, f)}{\partial x} h(y) dy,$$

where the differentiation under the integral sign is justified by Dominated Convergence because of Assumption 2 and $\int_{\mathcal{O}} \bar{\kappa}_x(y)h(y)dy < \infty$. Moreover, by Assumption 3 and the Cauchy-Schwarz inequality, we find that

$$\int_{\mathbb{R}_+} \left(\int_{\mathcal{O}} \frac{\partial \kappa_t(x, y, f)}{\partial x} h(y)dy\right)^2 \alpha(x)dx \leq \|h\|^2 \int_{\mathbb{R}_+} \left\|\frac{\partial \kappa_t(x, \cdot, f)}{\partial x}\right\|^2 \alpha(x)dx < \infty,$$

which shows that $\sigma_t(f)h \in \mathcal{H}_\alpha$ and boundedness of the operator $\sigma_t(f)$ for every $f \in \mathcal{H}_\alpha$. □

A.2 Proof of Theorem 2.3

We first observe that for every $h \in \mathcal{H}$ and every $f, f_1 \in \mathcal{H}_\alpha$, it holds that

$$\int_{\mathbb{R}_+} \int_{\mathbb{R}_+} \left|\frac{\partial \kappa_t(x, y, f)}{\partial x} f_1'(x)\alpha(x)h(y)\right| dydx = \int_{\mathbb{R}_+} |f_1'(x)\alpha(x)| \int_{\mathbb{R}_+} \left|\frac{\partial \kappa_t(x, y, f)}{\partial x}\right| |h(y)|\, dydx$$

$$\leq \int_{\mathbb{R}_+} |f_1'(x)|\alpha^{1/2}(x) \left\|\frac{\partial \kappa_t(x, \cdot, f)}{\partial x}\right\| \alpha^{1/2}(x)\, \|h\|\, dx \leq \|h\|\, \|f_1\|_\alpha \left(\int_{\mathbb{R}_+} \left\|\frac{\partial \kappa_t(x, \cdot, f)}{\partial x}\right\|^2 \alpha(x)dx\right)^{1/2}$$

where we used the Cauchy-Schwarz inequality twice. By Assumption 3. this is bounded and allows us to apply the Fubini Theorem and calculate as follows:

$$\begin{aligned}\langle \sigma_t(f)h, f_1\rangle_\alpha &= f_1(0)\int_{\mathcal{O}} \kappa_t(0,y,f)h(y)dy + \int_{\mathbb{R}_+} \frac{\partial \sigma_t(f)h(x)}{\partial x} f_1'(x)\alpha(x)dx \\ &= f_1(0)\int_{\mathcal{O}} \kappa_t(0,y,f)h(y)dy + \int_{\mathbb{R}_+}\int_{\mathcal{O}} \frac{\partial \kappa_t(x,y,f)}{\partial x} h(y)dy f_1'(x)\alpha(x)dx \\ &= \int_{\mathcal{O}} \left(f_1(0)\kappa_t(0,y,f) + \int_{\mathbb{R}_+} \frac{\partial \kappa_t(x,y,f)}{\partial x} f_1'(x)\alpha(x)dx \right) h(y)dy \\ &= \int_{\mathcal{O}} \sigma_t(f)^* f_1(y)h(y)dy = \langle h, \sigma_t(f)^* f_1\rangle,\end{aligned}$$

for $\sigma_t(f)^* f_1$ defined by

$$\sigma_t(f)^* f_1(y) := f_1(0)\kappa_t(0,y,f) + \int_{\mathbb{R}_+} \frac{\partial \kappa_t(x,y,f)}{\partial x} f_1'(x)\alpha(x)dx = \langle \kappa_t(\cdot,y,f), f_1\rangle_\alpha.$$

From [36, Theorem 6.1], $\sigma_t(f)^*$ is the unique adjoint operator of $\sigma_t(f)$, for $f \in \mathcal{H}_\alpha$. □

A.3 Proof of Theorem 2.4

We start with the growth condition. For $h \in \mathcal{H}$ and $f_1 \in \mathcal{H}_\alpha$ we can write that

$$\begin{aligned}\|\sigma_t(f_1)h\|_\alpha^2 &= (\sigma_t(f_1)h(0))^2 + \int_{\mathbb{R}_+} \left(\frac{\partial \sigma_t(f_1)h(x)}{\partial x} \right)^2 \alpha(x)dx \\ &= \left(\int_{\mathcal{O}} \kappa_t(0,y,f_1)h(y)dy \right)^2 + \int_{\mathbb{R}_+} \left(\int_{\mathcal{O}} \frac{\partial \kappa_t(x,y,f_1)}{\partial x} h(y)dy \right)^2 \alpha(x)dx \\ &\leq \|\kappa_t(0,\cdot,f_1)\|^2 \|h\|^2 + \int_{\mathbb{R}_+} \left\| \frac{\partial \kappa_t(x,\cdot,f_1)}{\partial x} \right\|^2 \|h\|^2 \alpha(x)dx \\ &\leq C(t)^2(1+|f_1(0)|)^2 \|h\|^2 + \int_{\mathbb{R}_+} C(t)^2 f_1'(x)^2 \|h\|^2 \alpha(x)dx \\ &\leq 2C(t)^2(1+\|f_1\|_\alpha)^2 \|h\|^2,\end{aligned}$$

where we have used the Cauchy-Schwarz inequality, together with the inequality $|f_1(0)| \leq \|f_1\|_\alpha$ and Assumption 2. With some abuse of notation, it follows that $\|\sigma_t(f_1)\|_{\mathcal{L}(\mathcal{H},\mathcal{H}_\alpha)} \leq C(t)(1 + \|f_1\|_\alpha)$ for a suitably chosen constant $C(t)$. Similarly, from Assumption 1 it follows that

$$\begin{aligned}\|(\sigma_t(f_1) - \sigma_t(f_2))h\|_\alpha^2 &= \left(\int_{\mathcal{O}} (\kappa_t(0,y,f_1) - \kappa_t(0,y,f_2))\, h(y)dy \right)^2 + \\ &\quad + \int_{\mathbb{R}_+} \left(\int_{\mathcal{O}} \left(\frac{\partial \kappa_t(x,y,f_1)}{\partial x} - \frac{\partial \kappa_t(x,y,f_2)}{\partial x} \right) h(y)dy \right)^2 \alpha(x)dx \\ &\leq \|\kappa_t(0,\cdot,f_1) - \kappa_t(0,\cdot,f_2)\|^2 \|h\|^2 + \int_{\mathbb{R}_+} \left\| \frac{\partial \kappa_t(x,\cdot,f_1)}{\partial x} - \frac{\partial \kappa_t(x,\cdot,f_2)}{\partial x} \right\|^2 \|h\|^2 \alpha(x)dx \\ &\leq C(t)^2 |f_1(0) - f_2(0)|^2 \|h\|^2 + \int_{\mathbb{R}_+} C(t)^2 (f_1'(x) - f_2'(x))^2 \|h\|^2 \alpha(x)dx \\ &\leq 2C(t)^2 \|f_1 - f_2\|_\alpha^2 \|h\|^2,\end{aligned}$$

from which $\|\sigma_t(f_1) - \sigma_t(f_2)\|_{\mathcal{L}(\mathcal{H},\mathcal{H}_\alpha)} \leq C(t)\,\|f_1 - f_2\|_\alpha$ for a suitably chosen $C(t)$, which proves the Lipschitz continuity of the volatility operator, and concludes the proof. □

A.4 Proof of Proposition 2.5

In order for the volatility operator σ_t to be well defined, we need to check that the function κ_t introduced in equation (2.4) satisfies the assumptions of Theorem 2.2. We start by observing that $\kappa_t(x,\cdot) \in \mathcal{H}$ if and only if $\omega \in \mathcal{H}$. Then, we can calculate the derivative

$$\frac{\partial \kappa_t(x,y)}{\partial x} = a(t)e^{-bx}\left(\omega'(x-y) - b\omega(x-y)\right),$$

which, in particular, by Assumption 2 is bounded by

$$\left|\frac{\partial \kappa_t(x,y)}{\partial x}\right| \leq a(t)e^{-bx}\bar{\omega}_x(y).$$

For the $\mathcal{H}$-norm we then have that

$$\left\|\frac{\partial \kappa_t(x,\cdot)}{\partial x}\right\|^2 = \int_{\mathcal{O}}\left(\frac{\partial \kappa_t(x,y)}{\partial x}\right)^2 dy \leq a(t)^2 e^{-2bx} C_1^2 < \infty,$$

where we have used that $\|\bar{\omega}_x\| \leq C_1$, which implies that Assumption 3 in Theorem 2.2 is satisfied for α such that $\int_{\mathbb{R}_+} e^{-2bx}\alpha(x) < \infty$. Finally, the Lipschitz condition is trivially satisfied and the growth condition is fulfilled because $a(t)$ is bounded. □

A.5 Proof of Lemma 3.1

For w in equation (3.2), we get that $w_\ell(v) = \frac{1}{\ell}$ and $\mathcal{W}_\ell(u) = \frac{u}{\ell}$. Then

$$q_\ell^w(x,y) = \frac{1}{\ell}\left(\ell - y + x\right)\mathbb{I}_{[0,\ell]}(y-x),$$

and from equations (3.3) and (3.5) we can write that

$$\mathcal{D}_\ell^w(g_t)(x) = g_t + \frac{1}{\ell}\int_0^\infty \left(\ell - y + x\right)\mathbb{I}_{[0,\ell]}(y-x)g_t'(y)dy = g_t + \frac{1}{\ell}\int_x^{x+\ell}\left(\ell - y + x\right)g_t'(y)dy.$$

Integration by parts gives the result. □

A.6 Proof of Proposition 3.2

Let $f := \mathcal{D}_\ell^{w*}\delta_{T_1-s}^*(1)$. We start by applying the covariance operator to $h := \sigma_s(g_s)^* f$:

$$\begin{aligned}\left(\mathcal{Q}\sigma_s(g_s)^* f\right)(x) &= \int_{\mathcal{O}} q(x,y)\sigma_s(g_s)^* f(y)dy \\ &= \int_{\mathcal{O}} q(x,y)\left\langle \kappa_s(\cdot,y,g_s), f\right\rangle_\alpha dy = \left\langle \int_{\mathcal{O}} q(x,y)\kappa_s(\cdot,y,g_s)dy, f\right\rangle_\alpha,\end{aligned}$$

where we used Theorem 2.3 and the linearity of the scalar product. Further, we apply $\sigma_s(g_s)$:

$$
\begin{aligned}
\left(\sigma_s(g_s)\mathcal{Q}\sigma_s(g_s)^* f\right)(x) &= \int_{\mathcal{O}} \kappa_s(x,z,g_s)\left(\mathcal{Q}\sigma_s(g_s)^* f\right)(z)dz \\
&= \left\langle \int_{\mathcal{O}}\int_{\mathcal{O}} \kappa_s(x,z,g_s)q(z,y)\kappa_s(\cdot,y,g_s)dydz, f\right\rangle_\alpha = \left\langle \Psi_s(x,\cdot), f\right\rangle_\alpha,
\end{aligned}
$$

for $\Psi_s(x,\cdot) := \int_{\mathcal{O}}\int_{\mathcal{O}} \kappa_s(x,z,g_s)q(z,y)\kappa_s(\cdot,y,g_s)dydz$. We go now back to the definition of f:

$$
\begin{aligned}
&\left(\sigma_s(g_s)\mathcal{Q}\sigma_s(g_s)^*\right)\left(\mathcal{D}_\ell^{w*}\delta_{T_1-s}^*(1)\right)(x) = \left\langle \Psi_s(x,\cdot), \mathcal{D}_\ell^{w*}\delta_{T_1-s}^*(1)\right\rangle_\alpha \\
&= \left\langle \mathcal{D}_\ell^{w}\Psi_s(x,\cdot), \delta_{T_1-s}^*(1)\right\rangle_\alpha = \delta_{T_1-s}\left(\mathcal{D}_\ell^{w}\Psi_s(x,\cdot)\right) = \left(\mathcal{D}_\ell^{w}\Psi_s\right)(x,T_1-s).
\end{aligned}
$$

By Lemma 3.1 we can write that

$$
\begin{aligned}
\left(\mathcal{D}_\ell^{w}\Psi_s\right)(x,T_1-s) &= \int_{\mathbb{R}_+} d_\ell(T_1-s,u)\Psi_s(x,u)du \\
&= \int_{\mathbb{R}_+}\int_{\mathcal{O}}\int_{\mathcal{O}} d_\ell(T_1-s,u)\kappa_s(x,z,g_s)q(z,y)\kappa_s(u,y,g_s)dydzdu,
\end{aligned}
$$

to which, finally, we apply the operator $\delta_{T_1-s}\mathcal{D}_\ell^w$:

$$
\begin{aligned}
&\delta_{T_1-s}\mathcal{D}_\ell^w\left(\sigma_s(g_s)\mathcal{Q}\sigma_s(g_s)^*\right)\left(\mathcal{D}_\ell^{w*}\delta_{T_1-s}^*(1)\right) \\
&= \int_{\mathbb{R}_+} d_\ell(T_1-s,v)\left(\sigma_s(g_s)\mathcal{Q}\sigma_s(g_s)^*\right)\left(\mathcal{D}_\ell^{w*}\delta_{T_1-s}^*(1)\right)(v)dv \\
&= \int_{\mathbb{R}_+}\int_{\mathbb{R}_+}\int_{\mathcal{O}}\int_{\mathcal{O}} d_\ell(T_1-s,v)d_\ell(T_1-s,u)\kappa_s(v,z,g_s)q(z,y)\kappa_s(u,y,g_s)dydzdudv,
\end{aligned}
$$

finalizing the proof. □

A.7 Proof of Proposition 5.2

We consider the representation

$$
\Sigma^2(s) = a^2\int_{\mathbb{R}_+}\int_{\mathbb{R}_+} e^{-bu}e^{-bv}d_\ell(T_1-s,u)d_\ell(T_1-s,v)\mathcal{A}(u,v)dudv, \tag{A.1}
$$

where we have introduced

$$
\mathcal{A}(u,v) := \int_{\mathbb{R}}\int_{\mathbb{R}} \omega(v-z)q(z,y)\omega(u-y)dydz, \qquad u,v\in\mathbb{R}_+.
$$

By applying (repeatedly) the integration by parts, and since ω'' is null, we obtain

$$
\begin{aligned}
\mathcal{A}(u,v) &= \int_{\mathbb{R}} \omega(v-z)\left(\int_{\mathbb{R}} e^{-k|z-y|}\omega(u-y)dy\right)dz \\
&= \int_{\mathbb{R}} \omega(v-z)\left(\int_{-\infty}^{z} e^{-k(z-y)}\omega(u-y)dy + \int_{z}^{\infty} e^{-k(y-z)}\omega(u-y)dy\right)dz \\
&= \frac{2}{k}\int_{\mathbb{R}} \omega(v-z)\omega(u-z)dz. \qquad\qquad (A.2)
\end{aligned}
$$

By substituting equation (A.2) into (A.1), we get that

$$\begin{aligned}\Sigma^2(s) &= \frac{2a^2}{k}\int_{\mathbb{R}}\int_{\mathbb{R}_+}\int_{\mathbb{R}_+} e^{-bu}e^{-bv}d_\ell(T_1-s,u)d_\ell(T_1-s,v)\omega(v-z)\omega(u-z)dzdudv\\ &= \frac{2a^2}{k}\int_{\mathbb{R}}\left(\int_{\mathbb{R}_+} e^{-bu}d_\ell(T_1-s,u)\omega(u-z)du\right)^2 dz\\ &= \frac{2a^2}{k\ell^2}\int_{\mathbb{R}}\left(\int_{T_1-s}^{T_1-s+\ell} e^{-bu}\omega(u-z)du\right)^2 dz,\end{aligned}$$

where we used the definition of d_ℓ in Lemma 3.1. By integration by parts, we get

$$\begin{aligned}\Sigma^2(s) = \frac{2a^2}{k\ell^2b^4}\int_{\mathbb{R}}\Big(&e^{-b(T_1-s)}\left(b\omega(T_1-s-z)+\omega'(T_1-s-z)\right)+\\ &-e^{-b(T_1-s+\ell)}\left(b\omega(T_1-s+\ell-z)+\omega'(T_1-s+\ell-z)\right)\Big)^2 dz\\ = \frac{2a^2}{k\ell^2b^4}\Big(&e^{-2b(T_1-s)}\mathcal{B}_1(s)-2e^{-2b(T_1-s)}e^{-b(T_1-s+\ell)}\mathcal{B}_2(s)+e^{-2b(T_1-s+\ell)}\mathcal{B}_3(s)\Big),\end{aligned}$$

where we introduced

$$\begin{aligned}\mathcal{B}_1(s) &:= \int_{\mathbb{R}}\left(b\omega(T_1-s-z)+\omega'(T_1-s-z)\right)^2 dz,\\ \mathcal{B}_2(s) &:= \int_{\mathbb{R}}\left(b\omega(T_1-s-z)+\omega'(T_1-s-z)\right)\left(b\omega(T_1-s+\ell-z)+\omega'(T_1-s+\ell-z)\right)dz,\\ \mathcal{B}_3(s) &:= \int_{\mathbb{R}}\left(b\omega(T_1-s+\ell-z)+\omega'(T_1-s+\ell-z)\right)^2 dz.\end{aligned}$$

By using the definition of ω in equation (5.3), we get that

$$\begin{aligned}\mathcal{B}_1(s) &= \int_{T_1-s-1}^{T_1-s+1}\left(b(1-|T_1-s-z|)-\operatorname{sgn}(T_1-s-z)\right)^2 dz\\ &= \int_{T_1-s-1}^{T_1-s}\left(b(1-T_1+s+z)-1\right)^2 dz+\int_{T_1-s}^{T_1-s+1}\left(b(1+T_1-s-z)+1\right)^2 dz\\ &= \frac{2}{3}\left(b^2+3\right),\end{aligned}$$

where sgn denotes the sign function. Similarly,

$$\mathcal{B}_2(s) = \frac{b^2}{6}\left(3(\ell-2)\ell^2+4\right)-3\ell+2, \qquad \mathcal{B}_3(s) = \frac{2}{3}\left(b^2+3\right).$$

By substituting these findings and rearranging the terms, we get that

$$\Sigma^2(s) = \frac{2a^2}{kb^4\ell^2}\left\{\frac{2}{3}\left(b^2+3\right)\left(1+e^{-2b\ell}\right)-2e^{-b\ell}\left(\frac{b^2}{6}\left(3(\ell-2)\ell^2+4\right)-3\ell+2\right)\right\}e^{-2b(T_1-s)},$$

which concludes the proof. □

A.8 Proof of Theorem 6.1

We follow a similar approach to [24, Section 8.5]. Let $\nu_i \geq 0$ be such that $n_i = 2d + \nu_i$ for $i = 1, \ldots, (L-1)$. For I_d the identity matrix of dimension d, we define the following weights:

$$\begin{aligned}
V_1 &:= \begin{bmatrix} I_d & -I_d & O_1 \end{bmatrix}^\top, \\
V_i &:= \begin{bmatrix} I_d & -I_d & O_i \end{bmatrix} \begin{bmatrix} I_d & -I_d & O_{i-1} \end{bmatrix}^\top, \quad i = 2, \ldots, (L-1), \\
V_L &:= A \begin{bmatrix} I_d & -I_d & O_{L-1} \end{bmatrix},
\end{aligned}$$

where $\top$ denotes the transpose operator. Here $O_i \in \mathbb{R}^{d \times \nu_i}$ are matrices with all entries equal to 0 to compensate the matrix dimension in such a way that $V_i \in \mathbb{R}^{n_i \times n_{i-1}}$ for $i = 1, \ldots, (L-1)$. By considering zero-biases vectors v_i, the linear maps H_i introduced in the neural network definition in equation (4.1) coincide then with the matrices V_i.

We observe that for every $x \in \mathbb{R}^d$, the ReLU activation function satisfies

$$x = \rho(x) - \rho(-x) = \begin{bmatrix} I_d & -I_d \end{bmatrix} \rho\left(\begin{bmatrix} I_d & -I_d \end{bmatrix}^\top x \right),$$

where the activation function is meant to act component wise. By straightforward calculation, one can then see that the neural network defined here satisfies the equality $\mathcal{N}(x) = Ax$ for every $x \in \mathbb{R}^d$, which means that it acts on x as a linear map. □

References

[1] Andresen, Arne, Steen Koekebakker and Sjur Westgaard (2010). *Modeling electricity forward prices using the multivariate normal inverse Gaussian distribution.* The Journal of Energy Markets, 3(3), 3.

[2] Barth, Andrea and Annika Lang (2012). *Simulation of stochastic partial differential equations using finite element methods.* Stochastics An International Journal of Probability and Stochastic Processes 84(2-3): 217-231.

[3] Barth, Andrea and Annika Lang (2012). *Multilevel Monte Carlo method with applications to stochastic partial differential equations.* International Journal of Computer Mathematics, 89(18): 2479-2498.

[4] Barth, Andrea, Annika Lang and Christoph Schwab (2013). *Multilevel Monte Carlo method for parabolic stochastic partial differential equations.* BIT Numerical Mathematics, 53(1): 3-27.

[5] Barth, Andrea and Fred E. Benth (2014). *The forward dynamics in energy markets – infinite-dimensional modelling and simulation.* Stochastics An International Journal of Probability and Stochastic Processes, 86(6), 932-966.

[6] Bayer, Christian and Benjamin Stemper (2018). *Deep calibration of rough stochastic volatility models.* arXiv preprint arXiv:1810.03399.

[7] Bayer, Christian, Blanka Horvath, Aitor Muguruza, Benjamin Stemper and Mehdi Tomas (2019). *On deep calibration of (rough) stochastic volatility models.* arXiv preprint arXiv:1908.08806.

[8] Benth, Fred E. (2015). *Kriging smooth energy futures curves.* Energy Risk.

[9] Benth, Fred E. and Steen Koekebakker (2008). *Stochastic modeling of financial electricity contracts.* Energy Economics, 30(3), 1116-1157.

[10] Benth, Fred E., Jūratė Š. Benth and Steen Koekebakker (2008). *Stochastic Modelling of Electricity and Related Markets.* Vol. 11. World Scientific, 2008.

[11] Benth, Fred E. and Paul Krühner (2014). *Representation of infinite-dimensional forward price models in commodity markets.* Communications in Mathematics and Statistics, 2(1), 47-106.

[12] Benth, Fred E. and Paul Krühner (2015). *Derivatives pricing in energy markets: an infinite-dimensional approach.* SIAM Journal on Financial Mathematics, 6(1), 825-869.

[13] Benth, Fred E. and Florentina Paraschiv (2018). *A space-time random field model for electricity forward prices.* Journal of Banking & Finance, 95, 203-216.

[14] Bühler, Hans, Lukas Gonon, Josef Teichmann and Ben Wood (2019). *Deep hedging.* Quantitative Finance, 19(8), 1271-1291.

[15] Carmona, René and Sergey Nadtochiy (2012). *Tangent Lévy market models.* Finance and Stochastics, 16(1): 63-104.

[16] Clewlow, Les and Chris Strickland (2000). *Energy Derivatives: Pricing and Risk Management.* Lacima Publications, London.

[17] Cuchiero, Christa, Wahid Khosrawi and Josef Teichmann (2020). *A generative adversarial network approach to calibration of local stochastic volatility models.* arXiv preprint arXiv:2005.02505.

[18] Da Prato, Giuseppe and Jerzy Zabczyk (2014). *Stochastic Equations in Infinite Dimensions.* Cambridge University Press.

[19] De Spiegeleer, Jan, Dilip B. Madan, Sofie Reyners and Wim Schoutens (2018). *Machine learning for quantitative finance: fast derivative pricing, hedging and fitting.* Quantitative Finance, 18(10), 1635-1643.

[20] Ferguson, Ryan and Andrew Green (2018). *Deeply learning derivatives.* arXiv preprint arXiv:1809.02233.

[21] Filipović, Damir (2001). *Consistency Problems for Heath-Jarrow-Morton Interest Rate Models.* Lecture notes in Mathematics, vol. 1760. Springer, Berlin.

[22] Frestad, Dennis (2008). *Common and unique factors influencing daily swap returns in the Nordic electricity market, 1997–2005.* Energy Economics, 30(3): 1081-1097.

[23] Goodfellow, Ian, Yoshua Bengio and Aaron Courville (2016). *Deep learning.* MIT press.

[24] Gottschling, Nina M., Vegard Antun, Ben Adcock and Anders C. Hansen (2020). *The troublesome kernel: why deep learning for inverse problems is typically unstable.* arXiv preprint arXiv:2001.01258.

[25] Heath, David, Robert Jarrow and Andrew Morton (1992). *Bond pricing and the term structure of interest rates: A new methodology for contingent claims valuation.* Econometrica: Journal of the Econometric Society: 77-105.

[26] Henry-Labordere, Pierre (2017). *Deep primal-dual algorithm for BSDEs: Applications of machine learning to CVA and IM.* Available at SSRN 3071506.

[27] Hernandez, Andres (2016). *Model calibration with neural networks.* Available at SSRN 2812140.

[28] Higham, Catherine F. and Desmond J. Higham (2019). *Deep learning: An introduction for applied mathematicians.* SIAM Review 61(4): 860-891.

[29] Horvath, Blanka, Aitor Muguruza and Mehdi Tomas (2019). *Deep learning volatility.* Available at SSRN 3322085.

[30] Kallsen, Jan and Paul Krühner (2015). *On a Heath – Jarrow – Morton approach for stock options.* Finance and Stochastics, 19(3): 583-615.

[31] Koekebakker, Steen and Fridthjof Ollmar (2005). *Forward curve dynamics in the Nordic electricity market.* Managerial Finance, 31(6): 73-94.

[32] Kondratyev, Alexei (2018). *Learning curve dynamics with artificial neural networks.* Available at SSRN 3041232.

[33] Kovács, Mihály, Stig Larsson and Fredrik Lindgren (2010). *Strong convergence of the finite element method with truncated noise for semilinear parabolic stochastic equations with additive noise.* Numerical Algorithms 53(2-3): 309-320.

[34] Nelson, Charles R. and Andrew F. Siegel (1987). *Parsimonious modeling of yield curves.* Journal of business: 473-489.

[35] Peszat, Szymon and Jerzy Zabczyk (2007). *Stochastic Partial Differential Equations with Lévy Noise: An evolution equation approach.* (Vol. 113). Cambridge University Press.

[36] Rynne, Bryan and Martin A. Youngson (2013). *Linear Functional Analysis.* Springer Science & Business Media.

[37] Tappe, Stefan (2012). *Some refinements of existence results for SPDEs driven by Wiener processes and Poisson random measures.* International Journal of Stochastic Analysis, 2012.

[38] Weinan, E, Jiequn Han and Arnulf Jentzen (2017). *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.