A GENERATIVE ADVERSARIAL NETWORK APPROACH TO CALIBRATION OF LOCAL STOCHASTIC VOLATILITY MODELS

CHRISTA CUCHIERO, WAHID KHOSRAWI AND JOSEF TEICHMANN

Abstract. We propose a fully data driven approach to calibrate local stochastic volatility (LSV) models, circumventing in particular the ad hoc interpolation of the volatility surface. To achieve this, we parametrize the leverage function by a family of feed forward neural networks and learn their parameters directly from the available market option prices. This should be seen in the context of neural SDEs and (causal) generative adversarial networks: we *generate* volatility surfaces by specific neural SDEs, whose quality is assessed by quantifying, in an *adversarial* manner, distances to market prices. The minimization of the calibration functional relies strongly on a variance reduction technique based on hedging and deep hedging, which is interesting in its own right: it allows to calculate model prices and model implied volatilities in an accurate way using only small sets of sample paths. For numerical illustration we implement a SABR-type LSV model and conduct a thorough statistical performance analyis on many samples of implied volatility smiles, showing the accuracy and stability of the method.

1. Introduction

Each day a crucial task is performed in financial institutions all over the world: the calibration of stochastic models to current market or historical data. So far the model choice was not only driven by the capacity of capturing well empirically observed market features, but also by the computational tractability of the calibration process. This is now undergoing a big change since machine learning technologies offer new perspectives on model calibration.

Calibration is the choice of *one* model from a *pool of models*, given current market and historical data, possibly with some information on their significance. Depending on the nature of data

2010 *Mathematics Subject Classification.* 91G60, 93E35.
Key words and phrases. LSV calibration, neural SDEs, generative adversarial networks, deep hedging, variance reduction, stochastic optimization.

Christa Cuchiero gratefully acknowledges financial support by the Vienna Science and Technology Fund (WWTF) under grant MA16-021.

Christa Cuchiero
University of Vienna, Department of Statistics and Operations Research, Data Science @ Uni Vienna, Oskar-Morgenstern-Platz 1, 1090 Wien, Austria
E-mail: christa.cuchiero@univie.ac.at

Wahid Khosrawi
ETH Zürich, D-MATH, Rämistrasse 101, CH-8092 Zürich, Switzerland
E-mail: wahid.khosrawi@math.ethz.ch

Josef Teichmann
ETH Zürich, D-MATH, Rämistrasse 101, CH-8092 Zürich, Switzerland
E-mail: josef.teichmann@math.ethz.ch .

this is considered as an inverse problem or a problem of statistical inference. We consider here current market data, e.g. volatility surfaces, therefore we rather emphasize the inverse problem point of view. We however stress that it is the ultimate goal of calibration to include both data sources simultaneously. In this respect machine learning might help considerably.

We can distinguish three kinds of machine learning inspired approaches for calibration to current market prices: First, having solved the inverse problem already several times, one can learn from this experience (i.e. training data) the calibration map from market data to model parameters directly. Let us here mention one of the pioneering papers by A. Hernandez [26] that applied neural networks to learn this calibration map in the context of interest rate models. This was taken up in [11] for calibrating more complex mixture models. Second, one can learn the map from model parameters to market prices and then invert this map possibly with machine learning technology. In the context of rough volatility modeling, see [17], such approaches turned out to be very successful: we refer here to [3] and the references therein. Third, the calibration problem is considered as search for a model which *generates* given market prices and technology from generative adversarial networks, first introduced by [19], is used. This means parametrizing the model pool in a way which is accessible for machine learning techniques and interpreting the inverse problem as an training task of a generative network, whose quality is assessed by an *adversary.* We pursue this approach in the present article.

1.1. **Generative adversarial approaches in finance.** At first sight it might seem a bit unexpected to embed calibration problems in the realm of *generative adversarial networks* which are rather applied in areas like photorealistic image generation: a generative adversarial network, here mainly in its causal interpretation, is a neural network (mostly of recurrent type) G^θ depending on parameters θ, which transports a standard input law $\mathbb{P}_I$ (e.g., in our case of LSV modeling the law of a Brownian motion W together with an exogenously given stochastic volatility process α) to a target output law $\mathbb{P}_O$. In the context of finanical modeling $\mathbb{P}_O$ corresponds to the "true" law of the market, deduced from data (e.g., the empirical measure) and not necessarily fully specified. It can either correspond to the physical or to a risk neutral measure or even both, depending on the goal.

Denoting the push-forward of $\mathbb{P}_I$ under the transport map G^θ by $G^\theta_*\mathbb{P}_I$, the goal is to find parameters θ such that $G^\theta_*\mathbb{P}_I \approx \mathbb{P}_O$. For this purpose appropriate distance functions have to be used. Standard examples include entropies, integral distances Wasserstein or Radon distances, etc. The adversarial aspect appears when the chosen distance is represented as supremum over certain classes of functions, which can themselves be parametrized via neural networks of a certain type. This leads to a game, often of zero-sum type, between the generator and the adversary. In our case the measure $\mathbb{P}_O$ is not even fully specified, but only certain functionals, namely the given set of market prices, are provided. Therefore we shall measure the distance by the following neo-classical calibration functional

$$\inf_\theta \sup_\gamma \sum_{C\in\mathcal{C}} w_C^\gamma \Big(\underbrace{\mathbb{E}_{G^\theta_*\mathbb{P}_I}[C]}_{\text{model price}} - \underbrace{\mathbb{E}_{\mathbb{P}_O}[C]}_{\text{market price}}\Big)^2 \tag{1.1}$$

where $\mathcal{C}$ is the class of option payoffs and w_C^γ weights, which are be parametrized by γ to account for the adversarial part. Similary one could also consider distances of Wasserstein type (in its dual representation)

$$\inf_{\theta\in\Theta} \sup_\gamma \left(\mathbb{E}_{G^\theta_*\mathbb{P}_I}[C^\gamma] - \mathbb{E}_{\mathbb{P}_O}[C^\gamma]\right)$$

where γ parametrizes appropriate function classes, e.g. approximations of Lipschitz functions, neural networks or again payoffs of options.

In general we can consider distance functions d^γ such that the game between generator and adversary appears as

$$\inf_\theta \sup_\gamma d^\gamma(G_*^\theta \mathbb{P}_I, \mathbb{P}_O)\,. \tag{1.2}$$

A distance function here just maps two laws on path space to a non-negative real number allowing for minima if $\mathbb{P}_O$ is fixed: we do neither assume symmetry, nor a triangle inequality nor definiteness.

The advantage of this point of view is two-fold:

(i) we have access to the unreasonable effectiveness of modeling by neural networks, i.e. the counter-intuitive highly over-parametrized ansatz to consider a local stochastic volatility model as recurrent neural network is beneficial through its miraculous generalization and regularization properties;
(ii) the game theoretic view disentangles realistic price generation from discriminating certain options, e.g. by putting higher weights. Notice that (1.1) is not the usual form of GAN problems, since the adversary distance d^γ is non-linear in $\mathbb{P}_I$ and $\mathbb{P}_O$, but we believe that it is worth taking this abstract point of view.

There is no reason why these generative models, if sufficient computing power is available, should not take market price data as inputs, too. This would correspond, from the point of view of generative adversarial networks, to actually learn a map $G^{\theta,\text{market prices}}$, such that for any price configuration of market prices one has instantaneously a generative model given, which produces those prices. This requires just a rich data source of typical market prices (and computing power!). This would ultimately connect the first and third approach of calibration.

Even though it is usually not considered like that, one can also view the generative model as an engine producing a likelihood on path space: given historic data this would allow, with precisely the same technology, a maximum likelihood approach. In this case $\mathbb{P}_O$ is just the empirical measure of the one observed trajectory, that is inserted in the likelihood function on path space. This then falls in the realm of generative approaches that appear in the literature under the name "market generators". Here the goal is to mimick precisely the behavior and features of historical market trajectories. This line of research has been recently pursued in e.g. [31, 43, 25, 6, 2].

From a bird's eye perspective this machine learning approach to calibraiton might just look like a standard inverse problem with another parametrized family of functions. We, however, insist on one important difference. Implicit or explicit regularization (see e.g., [24]), which always appear in machine learning applications and which are cumbersome to mimick in classical inverse problems, are one of the secrets of this success.

In general, machine learning approaches are becoming more and more prolific in mathematical finance. Concrete applications include hedging [5], portfolio selection [16], stochastic portfolio theory [37, 12], optimal stopping [4], optimal transport and robust finance [15], stochastic games and control problems [28] as well as high dimensional non-linear partial differential equations (PDEs) [22, 29]. Machine learning also allows for new insights into structural properties

of financial markets as investigated in [39]. For an exhaustive overview on machine learning applications in mathematical finance, in particular for option pricing and hedging we refer to [36].

1.2. **Local stochastic volatiliy models as neural SDEs.** In the present article we focus on calibration of *local stochastic volatility (LSV) models*, which is still an intricate task, both from a theoretical as well as a practical point of view. LSV models, going back to [32, 34], combine classical stochastic volatility with local volatility to achieve both, a good fit to time series data and in principle a perfect calibration to the implied volatility smiles and skews. In these models, the discounted price process $(S_t)_{t\geq 0}$ of an asset satisfies

$$dS_t = S_t L(t, S_t)\alpha_t dW_t, \tag{1.3}$$

where $(\alpha_t)_{t\geq 0}$ is some stochastic process taking values in $\mathbb{R}$, and (a sufficiently regular function) $L(t, s)$ the so-called *leverage function* depending on time and the current value of the asset and W a one-dimensional Brownian motion. Note that the stochastic volatility process α can be very general and could for instance be chosen as rough volatility model. By slight abuse of terminology we call α stochastic volatility even though it is strictly speaking not the volatility of the log price of S (we can, however, imagine L to be close to 1).

For notational simplicity we consider here the one-dimensional case, but the setup easily translates to a multivariate situation with several assets and a matrix valued analog of α as well as a matrix valued leverage function.

The leverage function L is the crucial part in this model. It allows in principle to perfectly calibrate the implied volatility surface seen on the market. In order to achieve this goal L has to satisfy

$$L^2(t, s) = \frac{\sigma^2_{\text{Dup}}(t, s)}{\mathbb{E}[\alpha_t^2 | S_t = s]}, \tag{1.4}$$

where σ_{Dup} denotes Dupire's local volatility function (see [14]). For the derivation of (1.4), we refer to [21]. Note that (1.4) is an implicit equation for L as it is needed for the computation of $\mathbb{E}[\alpha_t^2 | S_t = s]$. This in turn means that the stochastic differential equation (SDE) for the price process $(S_t)_{t\geq 0}$ is actually a McKean-Vlasov SDE, since the law of S_t enters the characteristics of the equation. Existence and uniqueness results for this equation are not at all obvious, since the coefficients do not satisfy any kind of standard conditions like for instance Lipschitz continuity in the Wasserstein space. Existence of a short-time solution of the associated non-linear Fokker-Planck equation for the density of $(S_t)_{t\geq 0}$ was shown in [1] under certain regularity assumptions on the initial distribution. As stated in [20] (see also [30], where existence of a simplified version of an LSV model is proved) a very challenging and still open problem is to derive the set of stochastic volatility parameters for which LSV models exist uniquely for a given market implied volatility surface.

Despite these intriguing existence issues, LSV models have attracted – due to their appealing feature of a potentially perfect smile calibration and their econometric properties – a lot of attention from the calibration and implementation point of view. We refer to [20, 21, 10] for Monte Carlo methods, to [34, 40] for PDE methods based on non-linear Fokker-Planck equations and to [38] for inverse problem techniques. Within these approaches the particle approximation method for the McKean-Vlasov SDE proposed in [20, 21] works impressively well, as very few paths have to be used in order to achieve very accurate calibration results.

In the current paper we propose an alternative, fully data driven approach circumventing in particular the interpolation of the volatility surface, being necessary in several other approaches in order to compute Dupire's local volatility. To achieve this we *learn* or *train* the leverage function L in order to generate the available market option prices accurately. This approach allows in principle to take all kind of options into account. In other words we are not limited to plain vanilla European call options, in contrast to many existing methods. Setting $T_0 = 0$ and denoting by $T_1 < T_2 \cdots < T_n$ the maturities of the available options, we parametrize the leverage function $L(t, s)$ via a family of neural networks $F_i : \mathbb{R} \to \mathbb{R}$ with weights $\theta_i \in \Theta_i$, i.e.

$$L(t, s) = 1 + F_i(s \,|\, \theta_i), \quad t \in [T_{i-1}, T_i), \quad i \in \{1, \dots, n\}.$$

We here consider for simplicity only the univariate case. The multivariate situation just means that 1 is replaced by the identity matrix and the neural networks F_i are maps from $\mathbb{R}^d \to \mathbb{R}^{d\times d}$.

This is the way how we parametrize the transport map G^θ introduced in Section 1.1. Indeed, this leads to so-called neural SDEs (see [18] for related work), which in the case of time-inhomogenous Itô SDEs, just means to parametrize the drift $\mu(\cdot, \cdot \,|\, \theta)$ and volatility $\sigma(\cdot, \cdot \,|\, \theta)$ by neural networks with parameters θ, i.e.,

$$dX_t(\theta) = \mu(X_t(\theta), t \,|\, \theta) + \sigma(X_t(\theta), t \,|\, \theta) dW_t, \quad X_0(\theta) = x.$$

In our case, there is no drift and the volatility function (for the price) reads as

$$\sigma(S_t(\theta), t \,|\, \theta) = S_t(\theta) \left(1 + \sum_{i=1}^{n} F^i(S_t(\theta) \,|\, \theta_i) 1_{[T_{i-1}, T_i)}(t) \right) \alpha_t$$

The solution measure of the neural SDE corresponds to the transport $G^\theta_* \mathbb{P}_I$ where $\mathbb{P}_I$ is the law of (W, α).

Progressively for each maturity, the weights of the neural networks are learned by optimizing the calibration criterion (1.1), where we allow for more general loss functions than just the square, i.e.

$$\inf_\theta \sup_\gamma \sum_{j=1}^{J} w_j^\gamma \ell^\gamma(\pi_j^{\mathrm{mod}}(\theta) - \pi_j^{\mathrm{mkt}}) \,. \tag{1.5}$$

We here write $\pi_j^{\mathrm{mod}}(\theta)$ and π_j^{mkt} for the model and market option prices, which correspond exactly to the expected values under $G^\theta \mathbb{P}_I$ and $\mathbb{P}_O$, respectively. Moreover, for every fixed γ, ℓ^γ is a nonlinear nonnegative convex function with $\ell^\gamma(0) = 0$ and $\ell^\gamma(x) > 0$ for $x \neq 0$, measuring the distance between model and market prices. The parameters w_j^γ, for fixed γ, denote some weights, e.g., of vega type (compare [9]), which allow us to match implied volatility data rather then pure prices, our actual goal, very well.

Notice that, somehow quite typical for financial applications, we need to guarantee a very high accuracy of approximation, whence a variance reduction technique to approximate the model prices is crucial for this learning task. Notice also that due to the general structure of the adversary no diversification effects can be expected: we have to deal with nonlinear functions of expectations with respect to $\mathbb{P}_I$.

The precise algorithm is outlined in Section 3 and Section 4, where we also conduct a thorough statistical performance analysis. The implemention relies strongly on a variance reduction technique based on hedging and deep hedging: it allows to compute accurate model prices $\pi^{\mathrm{mod}}(\theta)$ for training purposes with only up to $5 \cdot 10^4$ trajectories. Let us remark that we do not aim

to compete with existing algorithms, as e.g. the particle method by [20, 21], in terms of speed but rather provide a generic data driven algorithm that is universally applicable for all kind of options, also in multivariate situations, without resorting to Dupire type volatilities.

The remainder of the article is organized as follows. Section 2 introduces the variance reduction technique based on hedge control variates, which is crucial in our optimization tasks. In Section 3 we explain our calibration method, in particular how to optimize (1.5). The details of the numerical implementation and the results of the statistical performance analysis are then given in Section 4 as well as Section 6. In Appendix A we state stability theorems for stochastic differential equations depending on parameters. This is applied to neural SDEs when calculating derivatives with respect to the parameters of the neural networks. In Appendix B we recall preliminaries on deep learning by giving a brief overview on universal approximation properties of artificial neural networks and briefly explaining stochastic gradient descent. Finally Appendix C contains alternative optimization approaches to (1.5).

2. Variance reduction for pricing and calibration via hedging and deep heding

This section is dedicated to introduce a generic variance reduction technique for Monte Carlo pricing and calibration by using hedging portfolios as control variates. This method will be crucial in our LSV calibration presented in Section 3. For similar considerations we refer to [41].

Consider on a finite time horizon $T > 0$, a financial market in discounted terms with r traded instruments $(Z_t)_{t\in[0,T]}$ following an $\mathbb{R}^r$-valued stochastic process on some filtered probability space $(\Omega, (\mathcal{F}_t)_{t\in[0,T]}, \mathcal{F}, \mathbb{Q})$. Here, $\mathbb{Q}$ is a risk neutral measure and $(\mathcal{F}_t)_{t\in[0,T]}$ is supposed to be right continuous. In particular, we suppose that $(Z_t)_{t\in[0,T]}$ is an r-dimensional square integrable martingale with càdlàg paths.

Let C be an $\mathcal{F}_T$-measurable random variable describing the payoff of some European option at maturity $T > 0$. Then the usual Monte Carlo estimator for the price of this option is given by

$$\pi = \frac{1}{N}\sum_{n=1}^{N} C_n, \tag{2.1}$$

where $(C_1, \dots, C_N)$ are i.i.d with the same distribution as C and $N \in \mathbb{N}$. This estimator can easily be modified by adding a stochastic integral with respect to Z. Indeed, consider a strategy $(h_t)_{t\in[0,T]} \in L^2(Z)$ and some constant c. Denote the stochastic integral with respect to Z by $I = (h \bullet Z)_T$ and consider the following estimator

$$\widehat{\pi} = \frac{1}{N}\sum_{n=1}^{N} (C_n - cI_n), \tag{2.2}$$

where $(I_1, \dots, I_N)$ are i.i.d with the same distribution as I. Then, for any $(h_t)_{t\in[0,T]} \in L^2(Z)$ and c, this estimator is still an unbiased estimator for the price of the option with payoff C since the expected value of the stochastic integral vanishes. If we denote by

$$H = \frac{1}{N}\sum_{n=1}^{N} I_n,$$

then the variance of $\widehat{\pi}$ is given by

$$\mathrm{Var}(\widehat{\pi}) = \mathrm{Var}(\pi) + c^2\mathrm{Var}(H) - 2c\mathrm{Cov}(\pi, H).$$

This becomes minimal by choosing

$$c = \frac{\mathrm{Cov}(\pi, H)}{\mathrm{Var}\,(H)}.$$

With this choice, we have

$$\mathrm{Var}(\widehat{\pi}) = (1 - \mathrm{Corr}^2(\pi, H))\mathrm{Var}(\pi).$$

In particular, in the case of a perfect (pathwise) hedge we have $\mathrm{Corr}(\pi, H) = 1$ and the estimator $\widehat{\pi}$ has 0 variance since in this case

$$\mathrm{Var}(\pi) = \mathrm{Var}(H) = \mathrm{Cov}(\pi, H).$$

Therefore it is crucial to find a good approximate hedging portfolio such that $\mathrm{Corr}^2(\pi, H)$ becomes large. This is subject of Section 2.1 and 2.2 below.

2.1. **Black-Scholes Delta hedge.** In many cases of local stochastic volatility models as of form (1.3) and options depending only on the terminal value of the price process, a Delta hedge of the Black-Scholes model works well. Indeed, let $C = g(S_T)$ and let $\pi^g_{\mathrm{BS}}(t, T, s, \sigma)$ be the price at time t of this claim in the Black-Scholes model. Here, s stands for the price variable and σ for the volatility parameter in the Black Scholes model. Moreover, we indicate the dependency on the maturity T as well. Then choosing as hedging instrument only the price S itself and as approximate hedging strategy

$$h_t = \partial_s \pi^g_{\mathrm{BS}}(t, T, S_t, L(t, S_t)\alpha_t) \tag{2.3}$$

usually already yields a considerable variance reduction. In fact it is even sufficient to consider α_t alone to achieve satisfying results, i.e., one has

$$h_t = \partial_s \pi^g_{\mathrm{BS}}(t, T, S_t, \alpha_t), \tag{2.4}$$

This reduces the computational costs for the evaluation of the hedging strategies even further.

2.2. **Hedging strategies as neural networks - deep hedging.** Alternatively, in particular when the number of hedging instruments becomes higher, one can learn the hedging strategy by parametrizing it via neural networks. For a brief overview on neural networks and relevant notation used below, we refer to Appendix B.

Let the payoff be again a function of the terminal values of the heging instruments, i.e., $C = g(Z_T)$. Then in Markovian models it makes sense to specify the hedging strategy via a function $h : \mathbb{R}_+ \times \mathbb{R}^r \to \mathbb{R}^r$

$$h_t = h(t, z),$$

which in turn will correspond to an artificial neural network $(t, z) \mapsto h(t, z \,|\, \delta) \in \mathcal{NN}_{r+1,r}$ with weights denoted by δ in some parameter space Δ (see Notation[1] B.4). Following the approach in [5, Remark 3], an optimal hedge for the claim C with given market price π^{mkt} can be computed via

$$\inf_{\delta \in \Delta} \mathbb{E}\left[u\left(-C + \pi^{\mathrm{mkt}} + (h(\cdot, Z_{\cdot -} \,|\, \delta) \bullet Z_\cdot)_T\right)\right]$$

for some convex loss function $u : \mathbb{R} \to \mathbb{R}_+$. If $u(x) = x^2$, which is often used in practice, this then corresponds to a quadratic hedging criterion.

[1]We here use δ to denote the parameters of the hedging neural networks, as θ shall be used for the networks of the leverage function.

In order to tackle this optimization problem, we can apply stochastic gradient descent, because we fall in the realm of problem (B.1). Indeed, the stochastic objective function $Q(\delta)(\omega)$ is given by

$$Q(\delta)(\omega) = u(-C(\omega) + \pi^{\mathrm{mkt}} + (h(\cdot, Z_{\cdot-} \,|\, \delta)(\omega) \bullet Z_{\cdot}(\omega))_T).$$

The optimal hedging strategy $h(\cdot, \cdot \,|\, \delta^*)$ for an optimizer δ^* can then be used to define

$$(h(\cdot, Z_{\cdot-} \,|\, \delta^*) \bullet Z_{\cdot})_T$$

which is in turn used in (2.2).

As always in this article we shall assume that activation functions σ of the neural network as well as the convex loss function u are smooth, hence we can calculate derivatives with respect to δ in a straight forward way. This is important to apply stochastic gradient descent, see Section B.2. We shall show that the gradient of $Q(\delta)$ is given by

$$\nabla_\delta Q(\delta)(\omega) = u'(-C(\omega) + \pi^{\mathrm{mkt}} + (h(\cdot, Z_{\cdot-} \,|\, \delta)(\omega) \bullet Z_{\cdot}(\omega))_T)(\nabla_\delta h(\cdot, Z_{\cdot-} \,|\, \delta)(\omega) \bullet Z_{\cdot}(\omega))_T,$$

i.e. we are allowed to move the gradient inside the stochastic integral, and that approximations with simple processes, as we shall do in practice, converge to the correct quantities. To ensure this property, we shall apply the following theorem, which follows directly from results in Section A.

Theorem 2.1. *For $\varepsilon \geq 0$, let Z^ε be a solution of a stochastic differential equation as described in Theorem A.3 with drivers $Y = (Y^1, \dots, Y^d)$, functionally Lipschitz operators $F_j^{\varepsilon,i}$, $i = 1, \dots, r$, $j = 1, \dots, d$ and a process $(J^{\varepsilon,1}, \dots J^{\varepsilon,r})$, which is here for all $\varepsilon \geq 0$ simply $J1_{\{t=0\}}(t)$ for some constant vector $J \in \mathbb{R}^r$. Let $(\varepsilon, t, z) \mapsto f^\varepsilon(t, z)$ be a map, such that the bounded càglàd process $f^\varepsilon := f^\varepsilon(.-, Z^0_{\cdot-})$ converges ucp to $f^0 := f^0(.-, Z^0_{\cdot-})$, then*

$$\lim_{\varepsilon \to 0} (f^\varepsilon \bullet Z^\varepsilon) = \left(f^0 \bullet Z^0\right)$$

holds true.

Proof. Consider the extended system

$$d(f^\varepsilon \bullet Z^\varepsilon) = \sum_{j=1}^{d} f^\varepsilon(t-, Z^\varepsilon_{t-})\, F_j^{\varepsilon,i}(Z^\varepsilon)_{t-} dY_t^j$$

and

$$dZ_t^{\varepsilon,i} = \sum_{j=1}^{d} F_j^{\varepsilon,i}(Z^\varepsilon)_{t-} dY_t^j,$$

where we obtain existence, uniqueness and stability for the second equation by Theorem A.3, and wherefrom we obtain ucp convergence of the integrand of the first equation: since stochastic integration is continuous with respect to the ucp topology we obtain the result. □

The following corollary implies the announced properties, namely that we can move the gradient inside the stochastic integral and that the derivatives of a discretized integral with a discretized version of Z and approximations of the hedging strategies are actually close to the derivatives of the limit object.

Corollary 2.2. *Let, for $\varepsilon > 0$, Z^ε denote a discretization of the process of heding instruments $Z \equiv Z^0$ such that the conditions of Theorem 2.1 are satisfied. Denote, for $\varepsilon \geq 0$, the corresponding hegding strategies by $(t, z, \delta) \mapsto h^\varepsilon(t, z \,|\, \delta)$ given by neural networks $\mathcal{NN}_{r+1,r}$, whose activation functions are bounded and C^1, with bounded derivatives.*

(i) *Then the derivative* $\nabla_\delta(h(\cdot, Z_{\cdot-} \,|\, \delta) \bullet Z)$ *in direction* δ *at* δ_0 *satisfies*

$$\nabla_\delta(h(\cdot, Z_{\cdot-} \,|\, \delta_0) \bullet Z) = (\nabla_\delta h(\cdot, Z_{\cdot-} \,|\, \delta_0) \bullet Z).$$

(ii) *If additionally the derivative in direction* δ *at* δ_0 *of* $\nabla_\delta h^\varepsilon(\cdot, Z_{\cdot-} \,|\, \delta_0)$ *converges ucp to* $\nabla_\delta h(\cdot, Z_{\cdot-} \,|\, \delta_0)$ *as* $\varepsilon \to 0$*, then the directional derivative of the discretized integral, i.e.* $\nabla_\delta(h^\varepsilon(\cdot, Z^\varepsilon_{\cdot-} \,|\, \delta_0) \bullet Z^\varepsilon)$ *or equivalently* $(\nabla_\delta h^\varepsilon(\cdot, Z^\varepsilon_{\cdot-} \,|\, \delta_0) \bullet Z^\varepsilon)$*, converges, as the discretization mesh* $\varepsilon \to 0$*, to*

$$\lim_{\varepsilon\to 0}(\nabla_\delta h^\varepsilon(\cdot, Z^\varepsilon_{\cdot-} \,|\, \delta_0) \bullet Z^\varepsilon) = (\nabla_\delta h(\cdot, Z_{\cdot-} \,|\, \delta_0) \bullet Z).$$

Proof. To prove (i), we apply Theorem 2.1 with

$$f^\varepsilon(\cdot, Z_{\cdot-}) = \frac{h(\cdot, Z_{\cdot-} \,|\, \delta_0 + \varepsilon\delta) - h(\cdot, Z_{\cdot-} \,|\, \delta_0)}{\varepsilon},$$

which converges ucp to $f^0 = \nabla_\delta h(\cdot, Z_{\cdot-} \,|\, \delta_0)$. Indeed, by the neural network assumptions, we have (with the sup over some compat set)

$$\lim_{\varepsilon\to 0} \sup_{(t,z)} \left\| \frac{h(t, z \,|\, \delta_0 + \varepsilon\delta) - h(t, z \,|\, \delta_0)}{\varepsilon} - \nabla_\delta h(t, z \,|\, \delta_0) \right\| = 0,$$

by equicontinuity of $\{(t, z) \mapsto \nabla_\delta h(t, z \,|\, \delta_0 + \varepsilon\delta) \,|\, \varepsilon \in [0, 1]\}$.

Concerning (ii) we apply again Theorem 2.1, this time with

$$f^\varepsilon(\cdot, Z_{\cdot-}) = \nabla_\delta h^\varepsilon(\cdot, Z_{\cdot-} \,|\, \delta_0),$$

which converges by Assumption (ii) ucp to $f^0 = \nabla_\delta h(\cdot, Z_{\cdot-} \,|\, \delta_0)$.

□

3. Calibration of LSV models

Consider an LSV model as of (1.3) defined on some filtered probability space $(\Omega, (\mathcal{F}_t)_{t\in[0,T]}, \mathcal{F}, \mathbb{Q})$, where $\mathbb{Q}$ is a risk neutral measure. We assume the stochastic process α to be fixed. This can for instance be achieved by first calibrating the pure stochastic volatility model with $L \equiv 1$ and by fixing the corresponding parameters.

Our main goal is to determine the leverage function L in perfect accordance with market data. We here consider only European call options, but our approach allows in principle to take all kind of other options into account.

Due to the universal approximation properties outlined in Appendix B (Theorem B.3) and in spirit of neural SDEs, we choose to parametrize L via neural networks. More precisely, set $T_0 = 0$ and let $0 < T_1 \cdots < T_n = T$ denote the maturities of the available European call options to which we aim to calibrate the LSV model. We then specify the leverage function $L(t, s)$ via a family of neural networks, i.e.,

$$L(t, s \,|\, \theta) = \left(1 + \sum_{i=1}^n F^i(s \,|\, \theta_i) 1_{[T_{i-1}, T_i)}(t)\right) \tag{3.1}$$

where $F_i \in \mathcal{NN}_{1,1}$ for $i = 1, \dots, n$ (see Notation B.4). For notational simplicity we shall often omit the dependence on $\theta_i \in \Theta_i$. However, when needed we write for instance $S_t(\theta)$, where θ then stands for all parameters θ_i used up to time t.

For purposes of training, similarly as in Section 2.2, we shall need to calculate derivatives of the LSV process with respect to θ.

Theorem 3.1. *Let $(t, s, \theta) \mapsto L(t, s \,|\, \theta)$ be of form (3.1) where the neural networks $(s, \theta_i) \mapsto F^i(s \,|\, \theta_i)$ are bounded and C^1, with bounded and Lipschitz continuous derivatives[2], for all $i = 1, \dots, n$. Then the directional derivative in direction θ at $\widehat{\theta}$ satisfies the following equation*

$$\begin{aligned} d\Big(\nabla_\theta S_t(\widehat{\theta})\Big) = \Big(&\nabla_\theta S_t(\widehat{\theta}) L(t, S_t(\widehat{\theta}) \,|\, \widehat{\theta}) + S_t(\widehat{\theta}) \partial_s L(t, S_t(\widehat{\theta}) \,|\, \widehat{\theta}) \nabla_\theta S_t(\widehat{\theta}) \\ &+ S_t(\widehat{\theta}) \nabla_\theta L(t, S_t(\widehat{\theta}) \,|\, \widehat{\theta})\Big) \alpha_t dW_t \,, \end{aligned} \tag{3.2}$$

with initial value 0. This can be solved by variation of constants and leads to well-known backward propagation schemes.

Proof. First note that Theorem A.2 implies the existence and uniqueness of

$$dS_t(\theta) = S_t(\theta) L(t, S_t(\theta) \,|\, \theta) \alpha_t dW_t \,,$$

for every θ. Here, the driving process is one dimensional and given by $Y = \int_0^\cdot \alpha_s dW_s$. Indeed, according to Remark A.4, if $(t, s) \mapsto L(t, s \,|\, \theta)$ is bounded, càdlàg in t and Lipschitz in s with a Lipschitz constant independent of t, $S_\cdot \mapsto S_\cdot(\theta) L(\cdot, S_\cdot(\theta) \,|\, \theta)$ is functionally Lipschitz and Theorem A.2 implies the assertion. These conditions are implied by the form of $L(t, s \,|\, \theta)$ and the conditions on the neural networks F^i.

To prove the form of the derivative process we apply Theorem A.3 to the following system: consider

$$dS_t(\widehat{\theta}) = S_t(\widehat{\theta}) L(t, S_t(\widehat{\theta}) \,|\, \widehat{\theta}) \alpha_t dW_t \,,$$

together with

$$dS_t(\widehat{\theta} + \varepsilon\theta) = S_t(\widehat{\theta} + \varepsilon\theta) L(t, S_t(\widehat{\theta} + + \varepsilon\theta) \,|\, \widehat{\theta} + \varepsilon\theta) \alpha_t dW_t \,,$$

as well as

$$\begin{aligned} d\,\frac{S_t(\widehat{\theta} + \varepsilon\theta) - S_t(\widehat{\theta})}{\varepsilon} &= \frac{S_t(\widehat{\theta} + \varepsilon\theta) L(t, S_t(\widehat{\theta} + \varepsilon\theta) \,|\, \widehat{\theta} + \varepsilon\theta) - S_t(\widehat{\theta}) L(t, S_t(\widehat{\theta}) \,|\, \widehat{\theta})}{\varepsilon} \alpha_t dW_t \\ &= \Big(\frac{S_t(\widehat{\theta} + \varepsilon\theta) - S_t(\widehat{\theta})}{\varepsilon} L(t, S_t(\widehat{\theta} + \varepsilon\theta) \,|\, \widehat{\theta} + \varepsilon\theta) \\ &\quad + S_t(\widehat{\theta}) \frac{L(t, S_t(\widehat{\theta} + \varepsilon\theta) \,|\, \widehat{\theta} + \varepsilon\theta) - L(t, S_t(\widehat{\theta}) \,|\, \widehat{\theta})}{\varepsilon} \Big) \alpha_t dW_t. \end{aligned}$$

In the terminology of Theorem A.3, $Z^{\varepsilon,1} = S(\widehat{\theta})$, $Z^{\varepsilon,2} = S(\widehat{\theta} + \varepsilon\theta)$ and $Z^{\varepsilon,3} = \frac{S_t(\widehat{\theta} + \varepsilon\theta) - S_t(\widehat{\theta})}{\varepsilon}$. Moreover, $F^{\varepsilon,3}$ is given by

$$\begin{aligned} F^{\varepsilon,3}(Z_t^0) = Z_t^{0,3} L(t, Z_t^{0,2} \,|\, \widehat{\theta} + \varepsilon\theta) + Z_t^{0,1} \partial_s L(t, Z_t^{0,1} \,|\, \widehat{\theta}) Z_t^{0,3} + \mathcal{O}(\varepsilon) \\ + Z_t^{0,1} \frac{L(t, Z_t^{0,1} \,|\, \widehat{\theta} + \varepsilon\theta) - L(t, Z_t^{0,1} \,|\, \widehat{\theta})}{\varepsilon}, \end{aligned} \tag{3.3}$$

[2]This just means that the activation function is bounded and C^1, with bounded and Lipschitz continuous derivatives.

which converges ucp to

$$F^{0,3}(Z_t^0) = Z_t^{0,3} L(t, Z_t^{0,2} \,|\, \widehat{\theta}) + Z_t^{0,1} \partial_s L(t, Z_t^{0,1} \,|\, \widehat{\theta}) Z_t^{0,3} + Z_t^{0,1} \nabla_\theta L(t, Z_t^{0,1} \,|\, \widehat{\theta}).$$

Indeed, for every fixed t, the family $\{s \mapsto L(t, s \,|\, \widehat{\theta} + \varepsilon\theta), \,|\, \varepsilon \in [0,1]\}$ is due to the form of the neural networks equicontinuous. Hence pointwise convergence implies uniform convergence in s. This together with $L(t, s \,|\, \theta)$ being piecewise constant in t yields

$$\lim_{\varepsilon \to 0} \sup_{(t,s)} |L(t, s \,|\, \widehat{\theta} + \varepsilon\theta) - L(t, s \,|\, \widehat{\theta})| = 0,$$

whence ucp convergence of the first term in (3.3). The convergence of term two and three is clear. The last one follows again from that fact the family $\{s \mapsto \nabla_\theta L(t, s \,|\, \widehat{\theta} + \varepsilon\theta) \,|\, \varepsilon \in [0,1]\}$ is equicontinuous, which is again a consequence of the form of the neural networks.

By the assumptions on the derivatives, $F^{0,3}$ is functionally Lipschitz. Hence Theorem A.2 yields the existence of a unique solution to (3.2) and Theorem A.3 implies convergence. □

Remark 3.2.

(i) For the pure existence and uniqueness of

$$dS_t(\theta) = S_t(\theta) L(t, S_t(\theta) \,|\, \theta) \alpha_t dW_t \,,$$

with $L(t, s \,|\, \theta)$ of form (3.1), it suffices that the neural networks $s \mapsto F^i(s \,|\, \theta_i)$ are bounded and Lipschitz, for all $i = 1, \dots, n$ (see also Remark A.4).

(ii) Similarly as in Theorem 2.1 we can also consider a discretization $S^\varepsilon(\theta)$ and conclude analogously as above the form of its derivative process. If the corresponding coefficients then converge in ucp, then also the derivative process does so.

Theorem 3.1 guarantees the existence and uniqueness of the derivative process. This thus allows to set up gradient based search algorithms for training.

In view of this let us now come to the precise optimization task as already outlined in Section 1.2. To ease the notation we shall here omit the dependence of the weigths w and the loss function ℓ on the parameter γ. For each maturity T_i, we assume to have J_i options with strikes K_{ij}, $j \in \{1, \dots, J_i\}$. The calibration functional for the i-th maturity is then of the form

$$\operatorname*{argmin}_{\theta_i \in \Theta_i} \sum_{j=1}^{J_i} w_{ij} \ell(\pi_{ij}^{\mathrm{mod}}(\theta_i) - \pi_{ij}^{\mathrm{mkt}}), \quad i \in \{1, \dots, n\}, \tag{3.4}$$

where $\pi_{ij}^{\mathrm{mod}}(\theta_i)$ (π_{ij}^{mkt} respectively) denotes the model (market resp.) price of an option with maturity T_i and Strike K_{ij}. Moreover, $\ell : \mathbb{R} \to \mathbb{R}_+$ is some nonnegative, nonlinear, convex loss function (e.g. square or absolute value) with $\ell(0) = 0$ and $\ell(x) > 0$ for $x \neq 0$, measuring the distance between market and model prices. Finally, w_{ij} denote some weights, e.g., of vega type (compare [9]), which allows to match implied volatility data rather then pure prices, our actual goal, very well. Notice that we allow the weights to be trained and adapted during a run of our algorithm.

We solve the minimization problems (3.4) iteratively: we start with maturity T_1 and fix θ_1. This then enters in the computation of $\pi_{2j}^{\mathrm{mod}}(\theta_2)$ and thus in (3.4) for maturity T_2, etc. To simplify the notation in the sequel, we shall therefore leave the index i away so that for a generic maturity $T > 0$, (3.4) becomes

$$\operatorname*{argmin}_{\theta \in \Theta} \sum_{j=1}^{J} w_j \ell(\pi_j^{\mathrm{mod}}(\theta) - \pi_j^{\mathrm{mkt}}).$$

Since the model prices are given by

$$\pi_j^{\mathrm{mod}}(\theta) = \mathbb{E}[(S_T(\theta) - K_j)^+], \tag{3.5}$$

we have $\pi_j^{\mathrm{mod}}(\theta) - \pi_j^{\mathrm{mkt}} = \mathbb{E}\left[Q_j(\theta)\right]$ where

$$Q_j(\theta)(\omega) := (S_T(\theta)(\omega) - K_j)^+ - \pi_j^{\mathrm{mkt}}. \tag{3.6}$$

The calibration task then amounts to finding a minimum of

$$f(\theta) := \sum_{j=1}^{J} w_j \ell(\mathbb{E}\left[Q_j(\theta)\right]). \tag{3.7}$$

As ℓ is a nonlinear function, this is not of the expected value form of problem (B.1). Hence standard stochastic gradient descent, as outlined in Appendix B.2, can not be applied in a straightforward manner.

We shall tackle this problem via hedge control variates as introduced in Section 2. In the following we explain this in more detail.

3.1. **Minimizing the calibration functional.** Consider the standard Monte Carlo estimator for $\mathbb{E}[Q_j(\theta)]$ so that (3.7) is estimated by

$$f^{\mathrm{MC}}(\theta) := \sum_{j=1}^{J} w_j \ell\left(\frac{1}{N}\sum_{n=1}^{N} Q_j(\theta)(\omega_n)\right), \tag{3.8}$$

for i.i.d samples $\{\omega_1, \dots, \omega_N\} \in \Omega$. Since the Monte Carlo error decreases as $\frac{1}{\sqrt{N}}$, the number of simulation N has to be chosen large ($\approx 10^8$) in order to approximate well the true model prices in (3.5). Note that implied volatility to which we actually aim to calibrate is even more sensitive. As stochastic gradient descent is not directly applicable due to the nonlinearity of ℓ, it seems necessary at first sight to compute the gradient of the whole function $\widehat{f}(\theta)$ to minimize (3.8). As $m \approx 10^8$, this is however computationally very expensive, leads to numerical instabilities and does not allow to find a minimum in the high dimensional parameter space Θ in a reasonable amount of time.

One very expedient remedy is to apply hedge control variates as introduced in Section 2 as variance reduction technique. This allows to reduce the number of samples N in the Monte Carlo estimator considerably to only up to $5 \cdot 10^4$ sample paths.

Assume that we have r hedging instruments (including the price process S) denoted by $(Z_t)_{t\in[0,T]}$ which are square integrable martingales under $\mathbb{Q}$ and take values in $\mathbb{R}^r$. Consider, for $j = 1, \dots, J$, strategies $h_j : [0,T] \times \mathbb{R}^r \to \mathbb{R}^r$ such that $h(\cdot, Z_\cdot) \in L^2(Z)$ and some constant c. Define

$$X_j(\theta)(\omega) := (S_T(\theta)(\omega) - K_j)^+ - c(h_j(\cdot, Z_{\cdot-}(\theta)(\omega)) \bullet Z_\cdot(\theta)(\omega))_T - \pi_j^{\mathrm{mkt}} \tag{3.9}$$

The calibration functionals (3.7) and (3.8), can then simply be defined by replacing $Q_j(\theta)(\omega)$ by $X_j(\theta)(\omega)$ so that we end up minimizing

$$\widehat{f}(\theta)(\omega_1, \dots, \omega_N) = \sum_{j=1}^{J} w_j \ell\left(\frac{1}{N}\sum_{n=1}^{N} X_j(\theta)(\omega_n)\right). \tag{3.10}$$

To tackle this task, we apply the following variant of gradient descent: starting with an initial guess $\theta^{(0)}$, we iteratively compute

$$\theta^{(k+1)} = \theta^{(k)} - \eta_k \nabla \widehat{f}(\theta)(\omega_1^{(k)}, \dots, \omega_N^{(k)}) \tag{3.11}$$

for some learning rate η_k and i.i.d samples $(\omega_1^{(k)}, \dots, \omega_N^{(k)})$. These samples can either be chosen to be the same in each iteration or to be newly sampled in each update step. The difference between these two approaches is negligable, since N is chosen so as to yield a small Monte Carlo error, whence the gradient is nearly deterministic. In our numerical experiments we newly sample in each update step.

Concerning the choice of the hedging strategies, we can parametrize them as in Section 2.2 via neural networks and find the optimal weights δ by computing

$$\operatorname*{argmin}_{\delta \in \Delta} \frac{1}{N} \sum_{n=1}^{N} u(-X_j(\theta, \delta)(\omega_n)). \tag{3.12}$$

for i.i.d samples $\{\omega_1, \dots, \omega_N\} \in \Omega$ and some loss function u when θ is fixed. Here,

$$X_j(\theta, \delta)(\omega) = (S_T(\theta)(\omega) - K_j)^+ - (h_j(\cdot, Z_{\cdot -}(\theta)(\omega)|\delta) \bullet Z_{\cdot}(\theta)(\omega))_T - \pi_j^{\mathrm{mkt}}.$$

This means to iterate the two optimization procedures, i.e. minimizing (3.10) for θ (with fixed δ) and (3.12) for δ (with fixed θ). Clearly the Black-Scholes-Hedge ansatz as of Section 2.1 works as well, in this case without additional optimization with respect to the hedging strategies.

For alternative approaches how to minimize (3.7), we refer to Appendix C.

4. Numerical Implementation

In this section we discuss the numerical implementation of the proposed calibration method.

We implement our approach via tensorflow, taking advantage of gpu-accelerated computing. All computations are performed on a single-gpu nvidea GEFORCE RTX 2080 machine. For the implied volatility computations, we rely on the python `py_vollib` library.[3]

Recall that a LSV model is given on some filtered probability space $(\Omega, (\mathcal{F}_t)_{t\in[0,T]}, \mathcal{F}, \mathbb{Q})$ by

$$dS_t = S_t \alpha_t L(t, S_t) dW_t, \quad S_0 > 0,$$

for some stochastic process α. When calibrating to data, it is therefore necessary to make further specifications. We calibrate the follwing SABR-type LSV model.

Definition 4.1. *The SABR-LSV model is specified via the SDE,*

$$\begin{aligned} dS_t &= S_t L(t, S_t) \alpha_t dW_t, \\ d\alpha_t &= \nu \alpha_t dB_t, \\ d\langle W, B\rangle_t &= \varrho dt, \end{aligned}$$

with parameters $\nu \in \mathbb{R}$, $\varrho \in [-1, 1]$ and initial values $\alpha_0 > 0$, $S_0 > 0$. Here, B and W are two correlated Brownian motions.

[3] See http://vollib.org/.

Remark 4.2. We shall often work in log-price coordinates for S. In particular, we can then consider L as a function of $X := \log S$ rather then S. By denoting this parametrization again with L, we therefore have $L(t, X)$ instead of $L(t, S)$ and the model dynamics read as

$$\begin{aligned} dX_t &= \alpha_t L(t, X_t) dW_t - \frac{1}{2}\alpha_t^2 L^2(t, X_t) dt, \\ d\alpha_t &= \nu \alpha_t dB_t, \\ d\langle W, B\rangle_t &= \varrho dt. \end{aligned}$$

Note that α is a geometric Brownian motion, in particular, the closed form solution for α is available and given by

$$\alpha_t = \alpha_0 \exp\left(-\frac{\nu^2}{2}t + \nu B_t\right).$$

For the rest of the paper we shall set $S_0 = 1$.

4.1. **Implementation of the calibration method.** We now present a proper numerical test and demonstrate the effectiveness of our approach on a family of typical market smiles (instead of just one calibration example). We consider as *ground truth* a situation where market smiles are produced by a parametric family. By randomly sampling smiles from this family we then show that they can be calibrated up to small errors, which we analyze statistically.

4.1.1. *Ground truth assumption.* We start by specifying the ground truth assumption. It is known that a discrete set of prices can be exactly calibrated by a local volatility model using Dupire's volatility function, if an appropriate interpolation method is chosen. Hence, any market observed smile data can be reproduced by the following model (we assume zero riskless rate and define $X = \log(S)$),

$$dS_t = \sigma_{\text{Dup}}(t, X_t) S_t dW_t,$$

or equivalently

$$dX_t = -\frac{1}{2}\sigma_{\text{Dup}}^2(t, X_t) dt + \sigma_{\text{Dup}}(t, X_t) dW_t, \tag{4.1}$$

where σ_{Dup} denotes Dupire's local volatility function [14]. Our ground truth assumption consist in supposing that the function σ_{Dup} (or to be more precise σ_{Dup}^2) can be chosen from a parametric family. Such parametric families for local volatility models have been discussed in the literature, consider e.g. [8] or [7]. In the latter, the authors introduce a family of local volatility functions $\widetilde{a}_\xi$ indexed by parameters

$$\xi = (p_1, p_2, \sigma_0, \sigma_1, \sigma_2)$$

and $p_0 = 1 - (p_1 + p_2)$ satisfying the constraints

$$\sigma_0, \sigma_1, \sigma_2, p_1, p_2 > 0 \text{ and } p_1 + p_2 \leq 1.$$

Setting $k(t, x, \sigma) = \exp\left(-x^2/(2t\sigma^2) - t\sigma^2/8\right)$, $\widetilde{a}_\xi$ is then defined as

$$\widetilde{a}_\xi^2(t, x) = \frac{\sum_{i=0}^2 p_i \sigma_i k(t, x, \sigma_i)}{\sum_{i=0}^2 (p_i/\sigma_i) k(t, x, \sigma_i)}.$$

In Figure 1(a) we show plots of implied volatilities for different slices for a realistic choice of parameters. As one can see, the produced smiles seem to be unrealistically flat. Hence we modify the local volatility function $\widetilde{a}_\xi$ to produce more pronounced and more realistic smiles. To

γ_1	γ_2	λ_1	λ_2	β_1	β_2	κ
1.1	20	10	10	0.005	0.001	0.5

TABLE 1. Fixed Parameters for the ground truth assumption a_ξ^2.

be precise, we define a new family of local volatility functions a_ξ indexed by the set of parameters ξ as

$$a_\xi^2(t,x) = \left| \frac{\left(\sum_{i=0}^2 p_i \sigma_i k(t,x,\sigma_i) + \Lambda(t,x)\right)\left(1 - 0.6 \times \mathbb{1}_{(t>0.1)}\right)}{\sum_{i=0}^2 (p_i/\sigma_i) k(t,x,\sigma_i) + 0.01} \right|, \tag{4.2}$$

with

$$\Lambda(t,x) := \left(\frac{\mathbb{1}_{(t\leq 0.1)}}{1+0.1t}\right)^{\lambda_2} \min\left\{\left(\gamma_1 (x-\beta_1)_+ + \gamma_2 (-x-\beta_2)_+\right)^\kappa, \lambda_1\right\}.$$

We fix the choice of the paramers $\gamma_i, \beta_i, \lambda_i, \kappa$ as given in Table 1. Note that a_ξ^2 is not defined at $t=0$. When doing a Monte Carlo simulation, we simply replace $a_\xi^2(0,x)$ with $a_\xi^2(\Delta_t, x)$, where Δ_t is the time increment of the Monte Carlo simulation.

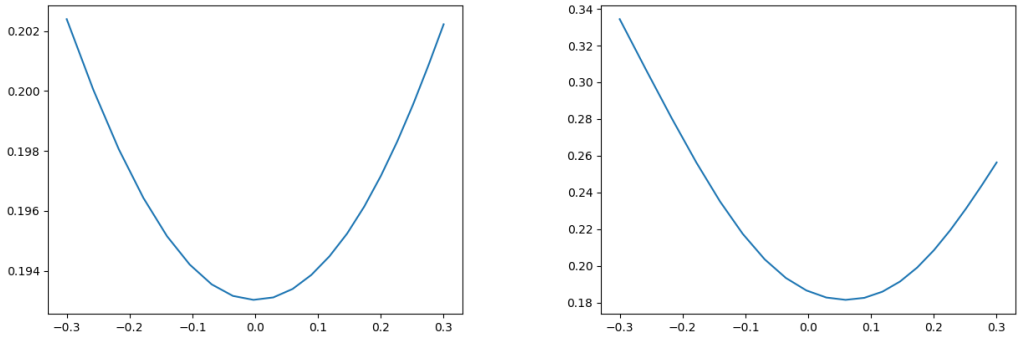
What is left to be specified are the parameters

$$\xi = (p_1, p_2, \sigma_0, \sigma_1, \sigma_2)$$

with $p_0 = 1 - p_1 - p_2$. This motivates our statistical test for the performance evaluation of our method. To be precise, our ground truth assumption is, that all observable market prices are explained by a variation of the parameters ξ. For illustration, we plot implied volatilities for this modified local volatility function in Figure 1(b) for a specific parameter set ξ.

Our ground truth model is now specified as in (4.1) with σ_{Dup} replaced by a_ξ, i.e.

$$dX_t = -\frac{1}{2} a_\xi^2(t, X_t) dt + a_\xi(t, X_t) dW_t. \tag{4.3}$$



(a) (b)

FIG. 1. Implied volatility of the original parametric family $\widetilde{a}_\xi$ (a) versus our modification a_ξ (b).

T_1	T_2	T_3	T_4
0.15	0.25	0.5	1.0

(a)

k_1	k_2	k_3	k_4
0.1	0.2	0.3	0.5

(b)

FIG. 2. (a) Maturities used to generate data for the calibration test. (b) Parameters that define the strikes of the call options to which we calibrate.

4.1.2. *Performance test.* We now come to the evaluation of our proposed method. We want to calibrate the SABR-LSV model to market prices generated by the previously formulated ground truth assumption. This corresponds to randomly sampling the parameter ξ of the local volatily function a_ξ and to compute prices according to (4.3). Calibrating the SABR-LSV model, i.e. finding the parameters ν, ϱ, the initial volatility α_0 and the unknown leverage function L, to these prices and repeating this multiple times then allows for a statistical analysis of the errors.

As explained in Section 3, we consider European call options with maturities $T_1 < \cdots < T_n$ and denote the strikes for a given maturity T_i by K_{ij}, $j \in \{1, \dots, J_i\}$. To compute the ground truth prices for these European calls we use a Euler-discretization of (4.3) with time step $\Delta_t = 1/100$. Prices are then obtained by a variance reduced Monte Carlo estimator using 10^7 Brownian paths and a Black-Scholes Delta hedge variance reduction as described previously. For a given parameter set ξ, we use the same Brownian paths for all strikes and maturities.

Overall, in this test, we consider $n = 4$ maturities with $J_i = 20$ strike prices for all $i = 1, \dots, 4$. The values for T_i are given in Figure 2(a). For the choice of the strikes K_i, we choose evenly spaced points, i.e.

$$K_{i,j+1} - K_{i,j} = \frac{K_{i,20} - K_{i,1}}{19}.$$

For the smallest and largest strikes per maturity we choose

$$K_{i,1} = \exp(-k_i), \; K_{i,20} = \exp(k_i),$$

with the values of k_i given in Figure 2(b).

We now specify a distribution under which we draw the parameters

$$\xi = (p_1, p_2, \sigma_0, \sigma_1, \sigma_2,)$$

for our test. The components are all drawn independently from each other under the uniform distribution on the respective intervals given below.

- $I_{p_1} = [0.4, 0.5]$
- $I_{p_2} = [0.4, 0.7]$
- $I_{\sigma_0} = [0.5, 1.7]$
- $I_{\sigma_1} = [0.2, 0.4]$
- $I_{\sigma_2} = [0.5, 1.7]$

When necessary we adjust p_2 so that $1 - p_1 - p_2 \geq 0$ is satisfied.

We can now generate data by the following scheme.

- For $m = 1, \dots, 150$ simulate parameters ξ_m under the law described above.
- For each m, compute prices of European calls for maturities T_i and strikes K_{ij} for $i = 1, \dots, n = 4$ and $j = 1, \dots, 20$ according to (4.3) using 10^7 Brownian trajectories (for each m we use new trajectories).

- Store these prices.

The second part consists in calibrating each of these surfaces and storing pertinent values for which we conduct a statistical analysis. In the following we describe the procedure in detail:

Recall that we specify the leverage function $L(t,x)$ via a family of neural networks, i.e.,

$$L(t,x) = 1 + F_i(x) \quad t \in [T_{i-1}, T_i), \quad i \in \{1, \dots, n = 4\},$$

where $F_i \in \mathcal{NN}_{1,1}$ (see Notation B.4). Each F_i is specified as a 3-hidden layer feed forward network where the dimension of each of the hidden layers is 50. As activation function we choose $\sigma = \tanh$. As before we denote the parameters of F_i by θ_i and the corresponding parameter space by Θ_i.

As closed form pricing formulas are not available for such an LSV model, let us here briefly specify our pricing method. For the variance reduced Monte Carlo estimator as of (3.10) we always use a standard Euler-SDE discretization with step size $\Delta_t = 1/100$. As variance reduction method, we implement the running Black-Scholes Delta hedge with instantanous running volatility of the price process, i.e. $L(t, S_t)\alpha_t$ is plugged in the formula for the Black-Scholes Delta as in (2.3). The only parameter that remains to be specified, is the number of trajectories used for the Monte Carlo estimator which is done in Algorithm 4.3 and 4.4 below.

As a first calibration step, we calibrate the SABR model (i.e. (4.1) with $L \equiv 1$) to the market prices and fix the calibrated SABR parameters ν, ϱ and α_0. For the remaining parameters θ_i, $i = 1, \dots, 4$, we apply the following algorithm until all parameters are calibrated.

Algorithm 4.3. *In the subsequent pseudocode, the index i stands for the maturities, N for the number of samples used in the variance reduced Monte Carlo estimator as of (3.10) and k for the updating step in the gradient descent:*

```
# Initialize the network parameters
initialize  θ_1,...,θ_4
# Define initial number of trajectories and initial step
N, k = 400, 1
# The time discretization for the MC simulations and the
# abort criterion
Δ_t, tol = 0.01, 0.0045

for i = 1,...,4:
  nextslice = False
  while nextslice == False:
    do:
      Simulate N trajectories of the SABR-LSV process up
      to time T_i, compute the payoffs.
    do:
      Compute the stochastic integral of the Black-Scholes
      Delta hedge against these trajectories as of (2.3)
      for maturity T_i
    do:
      Compute the calibration functional as of (3.10)
```

```
        with ℓ(x) = x^2 and normalized vega weights w_j for strike K_ij
        given by w_j = w̃_j / Σ_{l=1}^{20} w̃_l with w̃_j = 1/v_ij, where v_ij is the
        Black-Scholes vega for strike K_ij, the corresponding
        market implied volatility and the maturity T_i
    do:
        Make an optimization step from θ_i^(k-1) to θ_i^(k), similary
        as in (3.11) but with the more sophisticated ADAM-
        optimizer with learning rate 10^-3.
    do:
        Update the parameter N, the condition nextslice and
        compute model prices according to Specification 4.4.
    do:
        k = k + 1
```

Specfication 4.4. *We update the parameters in the above algorithm according to the following rules:*

```
if k == 500:
  N = 2000
else if k == 1500:
  N = 10000
else if k == 4000:
  N = 50000

if k >= 5000 and k mod 1000 == 0:
  do:
    Compute model prices π_model for slice i via MC simulation
    using 10^7 trajectories. Apply the Black-Scholes Delta
    hedge for variance reduction.
  do:
    Compute implied volatilities iv_model from the model prices π_model.
  do:
    Compute the maximum error of model implied volatilities
    against market implied volatilities:
    err_cali = || iv_model - iv_market ||_max
    if err_cali ≤ tol or k == 12000:
      nextslice = True
    else:
      Apply the adversarial part: Adjust the weights w_j
      according to:
      for j = 1,...,20:
        w_j = w_j + 0.1 * | iv_model_j - iv_market_j |
      This puts heigher weights on the options where the fit
      can still be improved
      Normalize the weights:
      for j = 1,...,20:
```

30 $w_j = w_j / \sum_{\ell=1}^{20} w_\ell$

4.2. **Numerical results for the calibration test.** We now discuss the results of our test.

We start by pointing out that from the 150 sampled market smiles, two smiles caused difficulty, in the sense that our implied volatility computation failed due to the remaining Monte-Carlo error. By increasing the training parameters slightly, this issue can be mitigated but the resulting calibrated implied volatility errors stay large out of the money where the smiles are extreme. Hence, we opt to remove those two samples from the following statistical analysis.

Further, we identify six smiles, for which the calibration of at least one slice has failed. Here, we say the calibration of a slice has failed if the maximum error of implied volatility is larger than 0.01. Let us make the following remark. In all these examples the training got stuck in a local minimum and a second drawing of the initial parameters actually led to satisfying results. A straight forward parallelization to reduce the additional time due to redrawing parameters is of course possible. In the following however, we keep the data of the failed 6 calibrations as they are when presenting results.

In Figure 3 we show calibration results for a typical example of randomly generated market data. From this it is already visible that the worst case calibration error (which occurs out of the money) ranges typically between 20 and 40 basis points. The corresponding calibration result for the leverage function L^2 is given in Figure 4.

Let us note that our method achieves a high calibration accuracy for the considered range of strikes and across all considered maturities. However, the further away from ATM, the more challenging this calibration becomes as can be seen in the results of a worst case analysis of calibration errors in Figures 6 and 7. There we show the mean as well as different quantiles of the data.

We present a histogram of calibration times in Figure 5. There, we see that the typical calibration of all slices finishes under 30 minutes and only rarely we face the situation where a higher number of optimization steps is needed before the abort criterion is activated. Since our approach allows for straight forward parallelization strategies, these times can be reduced significantly by changing to a multi gpu setup.

5. Conclusion

We have demonstrated how the parametrization by means of neural networks can be used to calibrate local stochastic volatility models to implied volatility data. We make the following remarks:

(i) The method we presented does not require any form of interpolation for the implied volatility surface since we do not calibrate via Dupire's formula. As the interpolation is usually done ad hoc, this is a desirable feature of our method.
(ii) It is possible to "plug in" any stochastic variance process such as rough volatility processes as long as an efficient simulation of trajectories is possible.
(iii) The multivariate extension is straight forward.

(iv) The level of accuracy of the calibration result is of a very high degree, making the presented method already of interest by this feature alone.
(v) The method can be significantly accelerated by applying distributed computation methods in the context of multi-gpu computational concepts.
(vi) The presented algorithm is further able to deal with path-dependent options since all computations are done by means of Monte Carlo simulations. In particular, it qualifies for joint calibration to S&P and VIX options. This is investigated in a companion paper.
(vii) We stress again the advantages of the generative adversarial network point of view. Indeed, the adversarial choice of the weights leads to very accurate generative neural SDE models. We believe that this is a crucial feature in the joint calibration of S&P and VIX options.

6. Plots

This section contains the relevant plots for the numerical test outlined in Section 4.

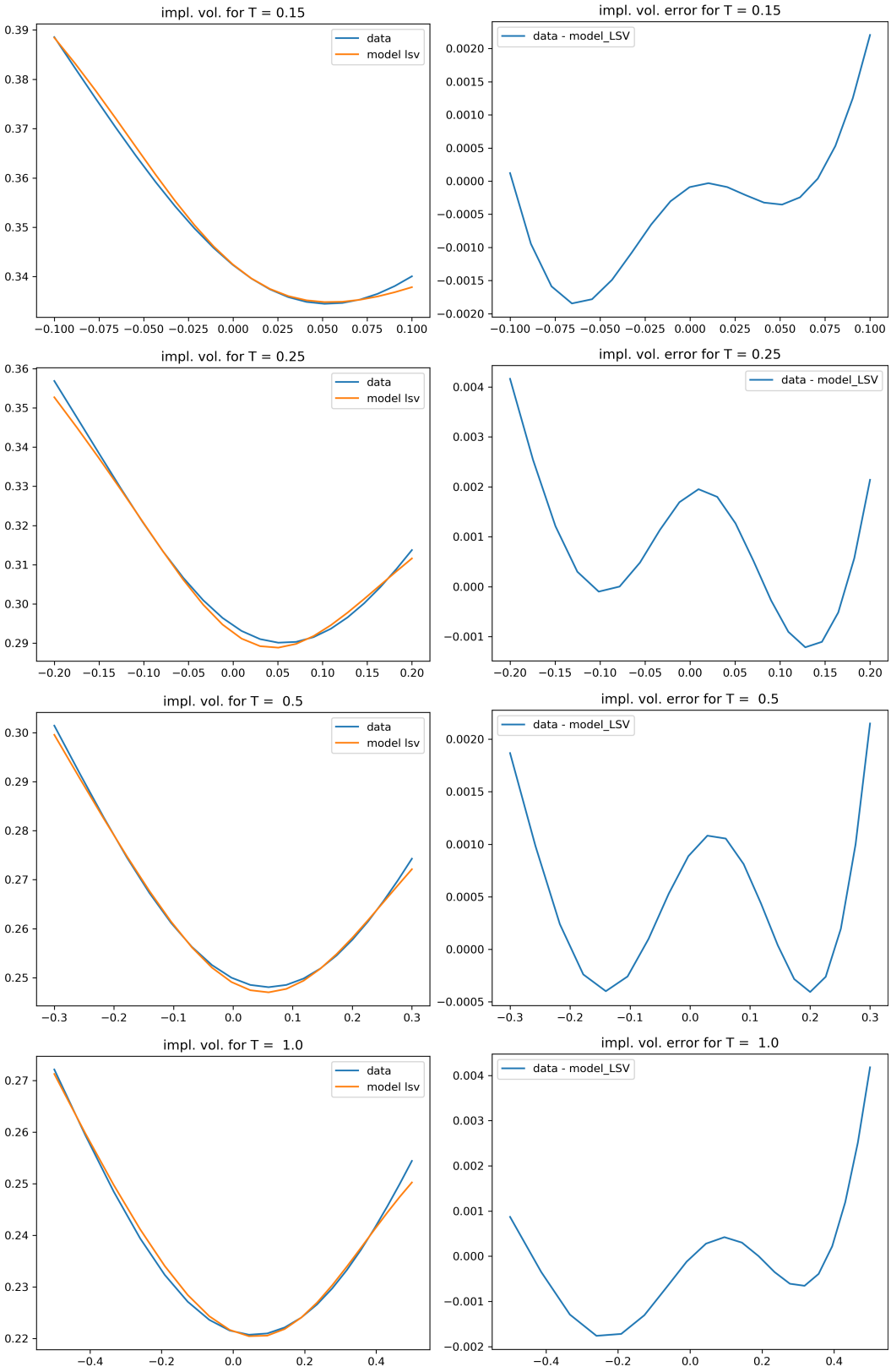


FIG. 3. Left column: Implied volatilities for the calibrated model together with the data (market) implied volatilities for a typical example of sampled market prices. Right column: Calibration errors by subtracting model implied volatilities from the data implied volatilities. Each row corresponds to one of the four available maturities.

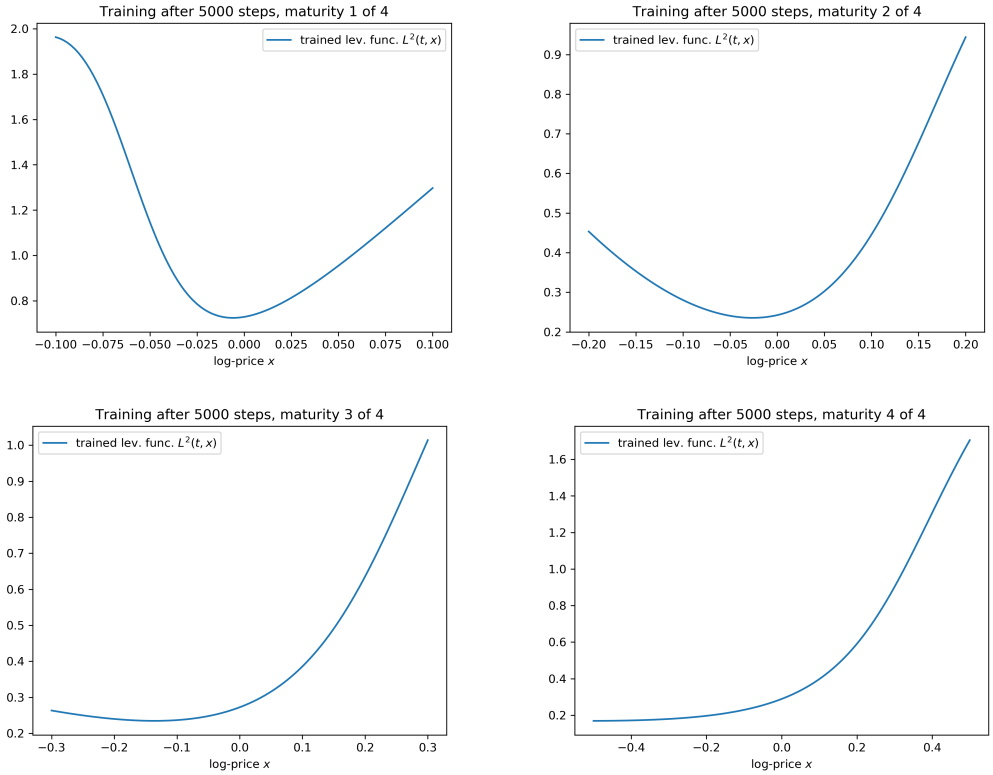


FIG. 4. Plot of the calibrated leverage function $L^2(t,\cdot)$ at $t \in \{0, T_1, T_2, T_3\}$ in the example shown in Figure 3.

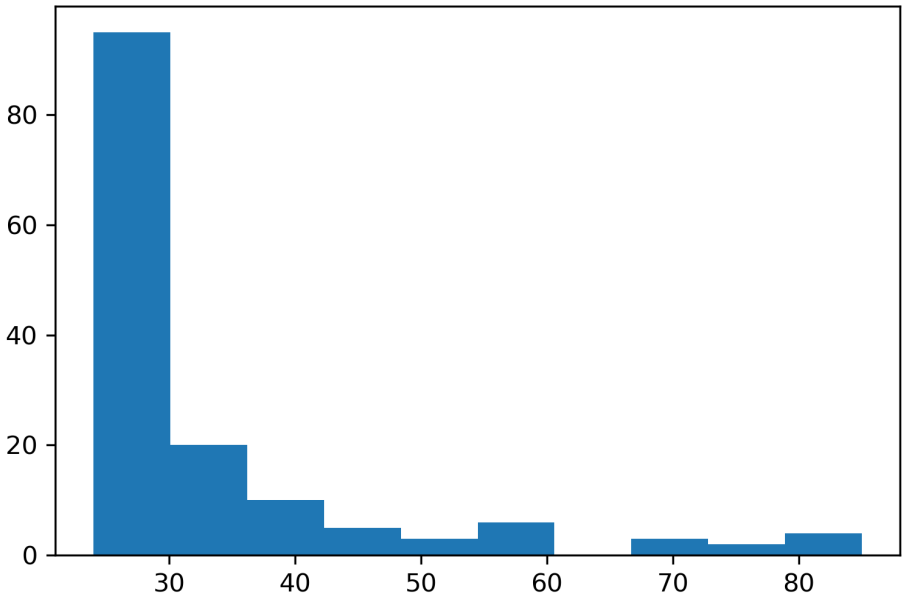


FIG. 5. Histogram of calibration times of the statistical test introduced in Section 4.1.2. One calibration in this histogram corresponds to the number of minutes, before the calibration of all slices is finished.

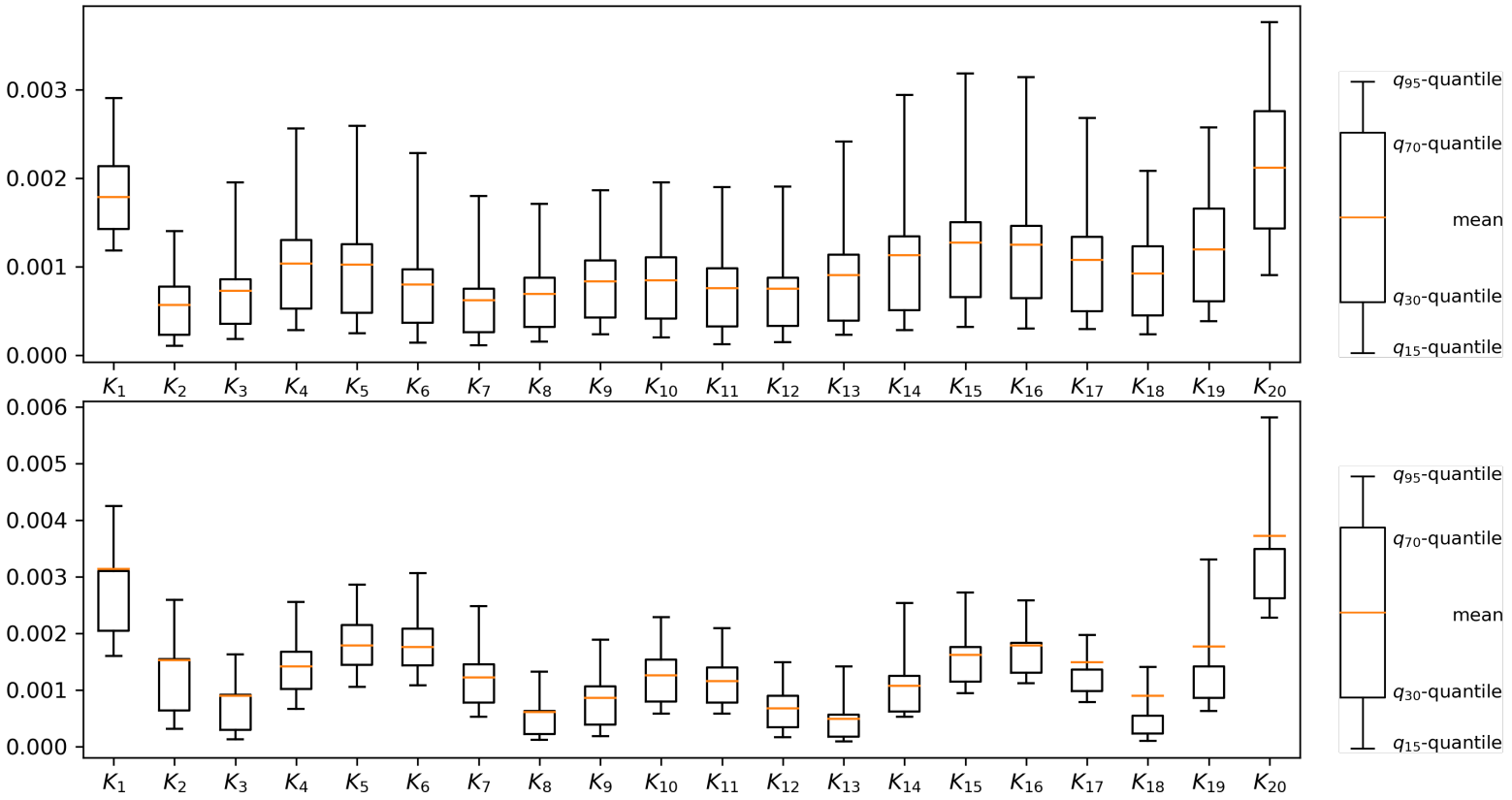


FIG. 6. Boxplots for the first (above) and second (below) slice. Depicted are the mean (horizontal line), as well as the $0.95, 0.70, 0.3, 0.15$ quantiles for the maximum of absolute calibration errors along all strikes.

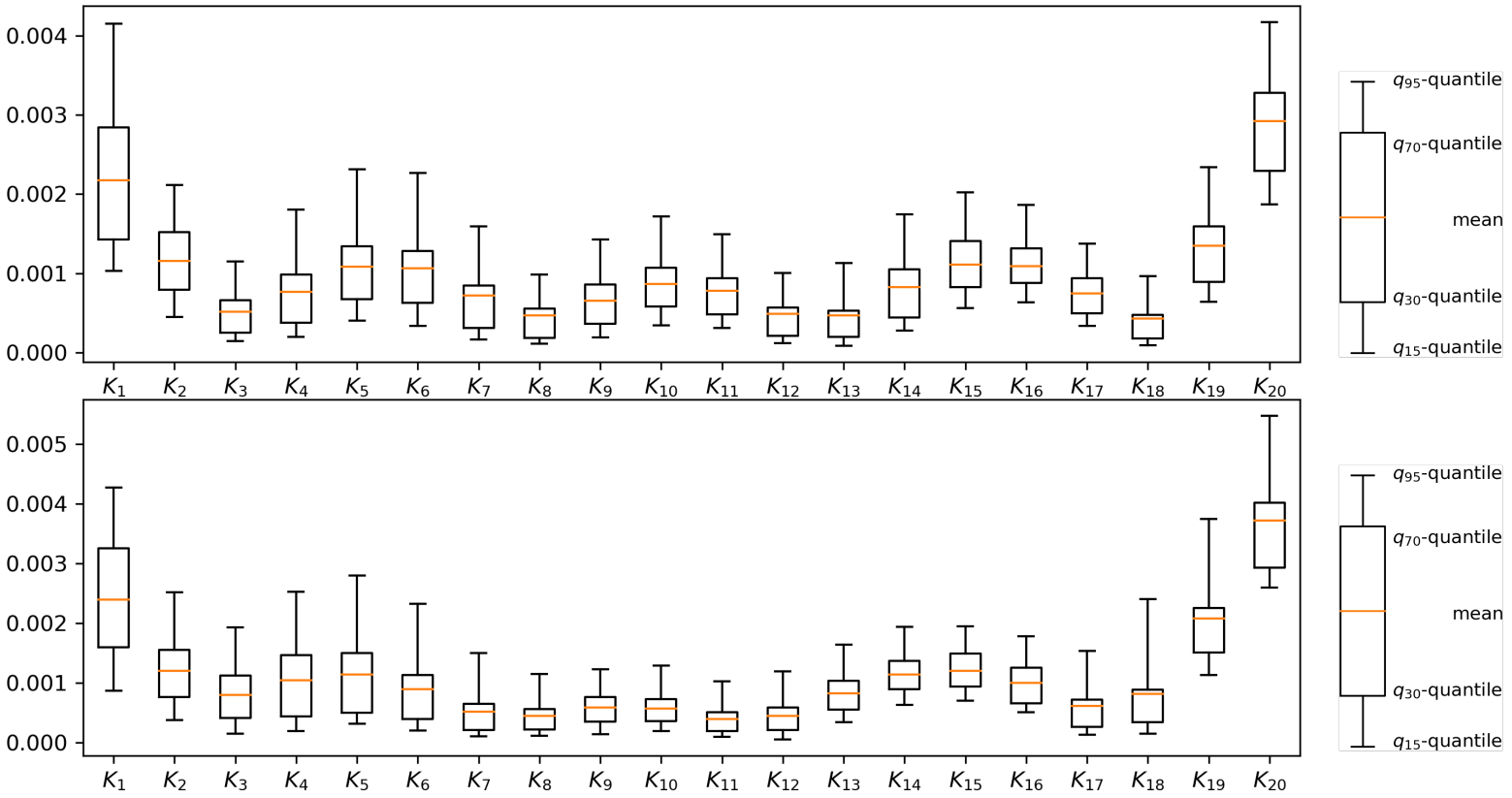


FIG. 7. Boxplot for the third (above) and fourth (below) slice. Depicted are the mean (horizontal line), as well as the $0.95, 0.70, 0.3, 0.15$ quantiles for the maximum of absolute calibration errors along all strikes.

Appendix A. Variations of stochastic differential equations

We follow here the excellent exposition of Philipp Protter in [33] in order to understand the dependence of solutions of stochastic differential equations on parameters, in particular when we aim to calculate derivatives with respect to parameters of neural networks.

Let us denote by $\mathbb{D}$ the set of real-valued, càdlàg, adapted processes on a given stochastic basis $(\Omega, \mathcal{F}, \mathbb{Q})$ with a filtration (satisfying usual conditions). By $\mathbb{D}^n$ we denote the set of $\mathbb{R}^n$-valued, càdlàg, adapted processes on the same basis.

Definition A.1. *An operator F from $\mathbb{D}^n$ to $\mathbb{D}$ is called* functional Lipschitz *if for any $X, Y \in \mathbb{D}^n$*

(i) *the property $X^{\tau-} = Y^{\tau-}$ implies $F(X)^{\tau-} = F(Y)^{\tau-}$ for any stopping time τ,*
(ii) *there exists an increasing process $(K_t)_{t\geq 0}$ such that for $t \geq 0$*

$$\|F(X)_t - F(Y)_t\| \leq K_t \sup_{r\leq t} \|X_r - Y_r\|.$$

Functional Lipschitz assumptions are sufficient to obtain existence and uniqueness for general stochastic differential equations, see [33, Theorem V 7].

Theorem A.2. *Let $Y = (Y^1, \dots, Y^d)$ be a vector of semimartingales starting at $Y_0 = 0$, $(J^1, \dots, J^n) \in \mathbb{D}^n$ a vector of processes and let F^i_j, $i = 1, \dots, n$, $j = 1, \dots, d$ be functionally Lipschitz operators. Then there is a unique process $Z \in \mathbb{D}^n$ satisfying*

$$Z^i_t = J^i_t + \sum_{j=1}^{d} \int_0^t F^i_j(Z)_{s-} dY^j_s$$

for $t \geq 0$ and $i = 1, \dots, n$. If J is a semimartingale, then Z is a semimartingale as well.

With an additional uniformity assumption on a sequence of stochastic differential equations with converging coefficients and initial data we obtain stability, see [33, Theorem V 15].

Theorem A.3. *Let $Y = (Y^1, \dots, Y^d)$ be vector of semimartingales starting at $Y_0 = 0$. Consider for $\varepsilon \geq 0$, a vector of processes $(J^{\varepsilon,1}, \dots, J^{\varepsilon,n}) \in \mathbb{D}^n$ and functionally Lipschitz operators $F^{\varepsilon,i}_j$ for $i = 1, \dots, n$, $j = 1, \dots, d$. Then, for $\varepsilon \geq 0$, there is a unique process $Z^\varepsilon \in \mathbb{D}^n$ satisfying*

$$Z^{\varepsilon,i}_t = J^{\varepsilon,i}_t + \sum_{j=1}^{d} \int_0^t F^{\varepsilon,i}_j(Z^\varepsilon)_{s-} dY^j_s$$

for $t \geq 0$ and and $i = 1, \dots, n$. If $J^\varepsilon \to J^0$ in ucp, $F^\varepsilon(Z^0) \to F^0(Z^0)$ in ucp, then $Z^\varepsilon \to Z^0$ in ucp.

Remark A.4. We shall apply these theorems to a local stochastic volatility model of the form

$$dS_t(\theta) = S_t(\theta) L(t, S_t(\theta) \,|\, \theta) \alpha_t dW_t \,,$$

where $\theta \in \Theta$, (W, α) some Brownian motion together with an adapted, càdlàg stochastic process α (all on a given stochastic basis) and $S_0 > 0$ some real number.

We assume that for each $\theta \in \Theta$

$$(t, s) \mapsto L(t, s \,|\, \theta) \tag{A.1}$$

is bounded, càdlàg in t (for fixed $s > 0$), and globally Lipschitz in s with a Lipschitz constant independent of t on compact intervals . In this case the map

$$S \mapsto S.L(\cdot, S. \,|\, \theta)$$

is functionally Lipschitz and therefore the above equation has a unique solution for all times t and any θ by Theorem A.2. If, additionally,

$$\lim_{\theta \to \widehat{\theta}} \sup_{(t,s)} |L(t, s \,|\, \theta) - L(t, s \,|\, \widehat{\theta})| = 0, \tag{A.2}$$

where the sup is taken over some compact set, then we also have that the solutions $S(\theta)$ converge ucp to $S(\widehat{\theta})$, as $\theta \to \widehat{\theta}$ by Theorem A.3.

Appendix B. Preliminaries on deep learning

We shall here briefly introduce two core concepts in deep learning, namely *artificial neural networks* and *stochastic gradient descent*. The latter is a widely used optimization method for solving maximization or minimization problems involving the first. In standard machine learning terminology, the optimization procedure is usually referred to as "training". We shall use both terminologies interchangeably.

B.1. **Artificial neural networks.**

We start with the definition of *feed forward neural networks*. These are functions obtained by composing layers consisting of an affine map and a componentwise nonlinearity. They serve as universal approximation class which is stated in Theorem B.3. Moreover, derivatives of these functions can be efficiently expressed iteratively (see e.g. [23]), which is a desirable feature from an optimization point of view.

Definition B.1. *Let $M, N_0, N_1, \dots, N_M \in \mathbb{N}$, $\sigma : \mathbb{R} \to \mathbb{R}$ and for any $m \in \{1, \dots, M\}$, let $w_m : \mathbb{R}^{N_{m-1}} \to \mathbb{R}^{N_m}$, $x \mapsto A_m x + b_m$ be an affine function with $A_m \in \mathbb{R}^{N_m \times N_{m-1}}$ and $b_m \in \mathbb{R}^{N_m}$. A function $\mathbb{R}^{N_0} \to \mathbb{R}^{N_M}$ defined as*

$$F(x) = w_M \circ F_{M-1} \circ \cdots \circ F_1, \quad \text{with } F_m = \sigma \circ w_m \quad \text{for } m \in \{1, \dots, M-1\}$$

is called a feed forward neural network. *Here the* activation function σ *is applied componentwise. $M-1$ denotes the number of hidden layers and $N_1, \dots, N_{M-1}$ denote the dimensions of the hidden layers and N_0 and N_M the dimension of the input and output layers.*

Remark B.2. Unless otherwise stated, the activation functions σ used in this article are always assumed to be smooth, globally bounded with bounded first derivative.

The following version of the so-called *universal approximation theorem* is due to K. Hornik [27]. An earlier version was proved by G. Cybenko [13]. To formulate the result we denote the set of all feed forward neural networks with activation function σ, input dimension N_0 and output dimension N_M by $\mathcal{NN}^{\sigma}_{\infty, N_0, N_M}$.

Theorem B.3 (Hornik (1991))**.** *Suppose σ is bounded and nonconstant. Then the following statements hold:*

(i) *For any finite measure μ on $(\mathbb{R}^{N_0}, \mathcal{B}(\mathbb{R}^{N_0}))$ and $1 \le p < \infty$, the set $\mathcal{NN}^{\sigma}_{\infty, N_0, 1}$ is dense in $L^p(\mathbb{R}^{N_0}, \mathcal{B}(\mathbb{R}^{N_0}), \mu)$.*
(ii) *If in addition $\sigma \in C(\mathbb{R}, \mathbb{R})$, then $\mathcal{NN}^{\sigma}_{\infty, N_0, 1}$ is dense in $C(\mathbb{R}^{N_0}, \mathbb{R})$ for the topology of uniform convergence on compact sets.*

Since each component of an $\mathbb{R}^{N_M}$-valued neural network is an $\mathbb{R}$-valued neural network, this result easily generalizes to $\mathcal{NN}^{\sigma}_{\infty,N_0,N_M}$ with $N_M > 1$.

Notation B.4. *We denote by $\mathcal{NN}_{N_0,N_M}$ the set of all neural networks in $\mathcal{NN}^{\sigma}_{\infty,N_0,N_M}$ with a* fixed architecture, *i.e. a fixed number of hidden layers $M-1$, fixed input and output dimensions N_m for each hidden layer $m \in \{1,\dots,M-1\}$ and a fixed activation function σ. This set can be described by*

$$\mathcal{NN}_{N_0,N_M} = \{F(\cdot|\theta) \,|\, F \text{ feed forward neural network and } \theta \in \Theta\},$$

with parameter space $\Theta \in \mathbb{R}^q$ for some $q \in \mathbb{N}$ and $\theta \in \Theta$ corresponding to the entries of the matrices A_m and the vectors b_m for $m \in \{1,\dots,M\}$.

B.2. **Stochastic gradient descent.** In light of Theorem B.3, it is clear that neural networks can serve as function approximators. To implement this, the entries of the matrices A_m and the vectors b_m for $m \in \{1,\dots,M\}$ are subject to optimization. If the unknown function can be expressed as the expected value of a stochastic objective function, one widely applied optimization method is *stochastic gradient descent,* which we shall review below.

Indeed, consider the following minimization problem

$$\min_{\theta\in\Theta} f(\theta) \quad \text{with} \quad f(\theta) = \mathbb{E}[Q(\theta)] \tag{B.1}$$

where Q denotes some stochastic objective function[4] $Q : \Omega \times \Theta \to \mathbb{R}$, $(\omega,\theta) \mapsto Q(\theta)(\omega)$ that depends on parameters θ taking values in some space Θ.

The classical method how to solve generic optimization problems for some differentiable objective function f (not necessarily of the expected value form as in (B.1)) is to apply a *gradient descent* algorithm: starting with an initial guess $\theta^{(0)}$, one iteratively defines

$$\theta^{(k+1)} = \theta^{(k)} - \eta_k \nabla f(\theta^{(k)}) \tag{B.2}$$

for some learning rate η_k. Under suitable assumptions, $\theta^{(k)}$ converges for $k \to \infty$ to a local minimum of the function f.

One insight of deep learning is that *stochastic gradient descent* methods, going back to stochastic approximation algorithms proposed by Robbins and Monroe [35], are much more efficient. To apply this, it is crucial that the objective function f is linear in the sampling probabilities. In other words, f needs to be of the expected value form as in (B.1). In the simplest form of stochastic gradient descent, under the assumption that

$$\nabla f(\theta) = \mathbb{E}[\nabla Q(\theta)],$$

the true gradient of f is approximated by a gradient at a single sample $Q(\theta)(\omega)$ which reduces the computational cost considerably. In the updating step for the parameters θ as in (B.2), f is then replaced by $Q(\theta)(\omega_k)$, hence

$$\theta^{(k+1)} = \theta^{(k)} - \eta_k \nabla Q(\theta^{(k)})(\omega_k). \tag{B.3}$$

The algorithm passes through all samples ω_k of the so-called training data set, possibly several times (specified by the number of epochs), and performs the update until an approximate minimum is reached.

[4]We shall often omit the dependence on ω.

A compromise between computing the true gradient of f and the gradient at a single sample $Q(\theta)(\omega)$ is to compute the gradient of a subsample of size N_{batch}, called (mini)-batch, so that $Q(\theta^{(k)})(\omega_k)$ used in the update (B.3) is replaced by

$$Q^{(k)}(\theta) = \frac{1}{N_{\text{batch}}} \sum_{n=1}^{N_{\text{batch}}} Q(\theta)(\omega_{n+kN_{\text{batch}}}), \quad k \in \{0, 1, ..., \lfloor N/N_{\text{batch}} \rfloor - 1\}, \tag{B.4}$$

where N is the size of the whole training data set. Any other unbiased estimators of $\nabla f(\theta)$ can of course also be applied in (B.3).

Appendix C. Alternative approaches for minimizing the calibration functional

We consider here alternative algorithms for minimizing (3.7).

C.1. **Stochastic compositional gradient descent.** One alternative is stochastic compositional gradient descent as developed e.g. in [42]. Applied to our problem this algorithm (in its simplest form) works as follows: starting with an initial guess $\theta^{(0)}$, and $y_j^{(0)}$, $j = 1, \dots, J$ one iteratively defines

$$\begin{aligned} y_j^{(k+1)} &= (1 - \beta_k) y_j^{(k)} + \beta_k Q_j(\theta^{(k)})(\omega_k) \quad j = 1, \dots, J, \\ \theta^{(k+1)} &= \theta^{(k)} - \eta_k \sum_{j=1}^{J} w_j \ell'(y_j^{(k+1)}) \nabla Q_j(\theta^{(k)})(\omega_k) \end{aligned}$$

for some learning rates $\beta_k, \eta_k \in (0, 1]$. Note that $y^{(k)}$ is an auxiliary variable to track the quantity $\mathbb{E}[Q(\theta^{(k)})]$ which has to be plugged in ℓ' (other faster converging estimates have also been developed). Of course $\nabla Q_j(\theta^{(k)})(\omega_k)$ can also be replaced by other unbiased estimates of the gradient, e.g. the gradient of the (mini)-batches as in (B.4). For convergence results in the case when $\theta \mapsto \ell(\mathbb{E}[Q_j(\theta)])$ is convex we refer to [42, Theorem 5]. Of course the same algorithm can be applied when we replace $Q_j(\theta)$ in (3.7) with $X_j(\theta)$ as defined in (3.9) for the variance reduced case.

C.2. **Estimators compatible with stochastic gradient descent.** Our goal here is to apply at least in special cases of the nonlinear function ℓ (variant (B.4) of) stochastic gradient descent to the calibration functional (3.7). This means that we have to cast (3.7) into expected value form. We focus on the case when $\ell(x)$ is given by $\ell(x) = x^2$ and write $f(\theta)$ as

$$f(\theta) = \sum_{j=1}^{J} w_j \mathbb{E}\left[Q_j(\theta) \widetilde{Q}_j(\theta) \right]$$

for some independent copy $\widetilde{Q}_j(\theta)$ of $Q_j(\theta)$, which is clearly of the expected value form required in (B.1). A Monte Carlo estimator of $f(\theta)$ is then constructed by

$$\widehat{f}(\theta) = \frac{1}{N} \sum_{n=1}^{N} \sum_{j=1}^{J} w_j Q_j(\theta)(\omega_n) \widetilde{Q}_j(\theta)(\omega_n).$$

for independent draws $\omega_1, \dots, \omega_N$ (the same N samples can be used for each strike K_j). Equivalently we have

$$\widehat{f}(\theta) = \frac{1}{N} \sum_{n=1}^{N} \sum_{j=1}^{J} w_j Q_j(\theta)(\omega_n) Q_j(\theta)(\omega_{n+m}). \tag{C.1}$$

for independent draws $\omega_1, \dots, \omega_{2N}$. The analog of (B.4) is then given by

$$Q^{(k)}(\theta) = \frac{1}{N_{\text{batch}}} \sum_{l=1}^{N_{\text{batch}}} \sum_{j=1}^{J} w_j Q_j(\theta)(\omega_{l+2kN_{\text{batch}}}) Q_j(\theta)(\omega_{l+(2k+1)N_{\text{batch}}})$$

for $k \in \{0, 1, ..., \lfloor N/N_{\text{batch}} \rfloor - 1\}$.

Clearly we can now modify and improve the estimator by using again hedge control variates and replace $Q_j(\theta)$ by $X_j(\theta)$ as defined in (3.9).

References

[1] F. Abergel and R. Tachet. A nonlinear partial integro-differential equation from mathematical finance. *Discrete and Continuous Dynamical Systems-Series A*, 27(3):907–917, 2010.

[2] B. Acciaio and T. Xu. Learning dynamic gans via causal optimal transport. *Working paper*, 2020.

[3] C. Bayer, B. Horvath, A. Muguruza, B. Stemper, and M. Tomas. On deep calibration of (rough) stochastic volatility models. *Preprint, available at arXiv:1908.08806*, 2019.

[4] S. Becker, P. Cheridito, and A. Jentzen. Deep optimal stopping. *Journal of Machine Learning Research*, 20, 2019.

[5] H. Buehler, L. Gonon, J. Teichmann, and B. Wood. Deep hedging. *Quantitative Finance*, 19(8):1271–1291, 2019.

[6] H. Buehler, B. Horvath, I. Perez Arribaz, T. Lyons, and B. Wood. A data-driven market simulator for small data environments. *Working paper*, 2020.

[7] R. Carmona, I. Ekeland, A. Kohatsu-Higa, J.-M. Lasry, P.-L. Lions, H. Pham, and E. Taflin. *HJM: A Unified Approach to Dynamic Models for Fixed Income, Credit and Equity Markets*, volume 1919, pages 1–50. 10 2007.

[8] R. Carmona and S. Nadtochiy. Local volatility dynamic models. *Finance and Stochastics*, 13(1):1–48, 2009.

[9] R. Cont and S. Ben Hamida. Recovering volatility from option prices by evolutionary optimization. *Journal of Computational Finance*, 8(4):43–76, 2004.

[10] A. Cozma, M. Mariapragassam, and C. Reisinger. Calibration of a hybrid local-stochastic volatility stochastic rates model with a control variate particle method. *Preprint, available at arXiv:1701.06001*, 2017.

[11] C. Cuchiero, A. Marr, M. M., A. Mitoulis, S. Singh, and J. Teichmann. Calibration of mixture interest rate models with neural networks. *Technical report*, 2018.

[12] C. Cuchiero, P. Schmocker, and T. Josef. Deep stochastic portfolio theory. *Working paper*, 2020.

[13] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Math. Control Signal Systems*, 5(455), 1992.

[14] B. Dupire. A unified theory of volatility. *Derivatives pricing: The classic collection*, pages 185–196, 1996.

[15] S. Eckstein and M. Kupper. Computation of optimal transport and related hedging problems via penalization and neural networks. *Applied Mathematics & Optimization*, pages 1–29, 2019.

[16] X. Gao, S. Tu, and L. Xu. A* tree search for portfolio management. *Preprint, available at arXiv:1901.01855*, 2019.

[17] J. Gatheral, T. Jaisson, and M. Rosenbaum. Volatility is rough. *Quantitative Finance*, 18(6):933–949, 2018.

[18] P. Gierjatowicz, M. Sabate, D. Siska, and L. Szpruch. Robust finance via neural sdes. *Working Paper*, 2020.

[19] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[20] J. Guyon and P. Henry-Labordere. The smile calibration problem solved. *Preprint, available at https://ssrn.com/abstract=1885032*, 2011.

[21] J. Guyon and P. Henry-Labordère. *Nonlinear option pricing*. CRC Press, 2013.

[22] J. Han, A. Jentzen, and E. Weinan. Overcoming the curse of dimensionality: Solving high-dimensional partial differential equations using deep learning. *Preprint, available at arXiv:1707.02568*, pages 1–13, 2017.

[23] R. Hecht-Nielsen. Theory of the backpropagation neural network. In *Neural networks for perception*, pages 65–93. Elsevier, 1992.
[24] J. Heiss, J. Teichmann, and H. Wutte. How implicit regularization of neural networks affects the learned function–part i. *Preprint arXiv:1911.02903*, 2019.
[25] P. Henry-Labordere. Generative models for financial data. *Preprint, Available at SSRN 3408007*, 2019.
[26] A. Hernandez. Model calibration with neural networks. *Risk*, 2017.
[27] K. Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.
[28] C. Huré, H. Pham, A. Bachouch, and N. Langrené. Deep neural networks algorithms for stochastic control problems on finite horizon, part i: convergence analysis. *Preprint, available at arXiv:1812.04300*, 2018.
[29] C. Huré, H. Pham, and X. Warin. Some machine learning schemes for high-dimensional nonlinear PDEs. *Preprint, available at arXiv:1902.01599*, 2019.
[30] B. Jourdain and A. Zhou. Existence of a calibrated regime switching local volatility model and new fake brownian motions. *Preprint available at arXiv:1607.00077*, 2016.
[31] A. Kondratyev and C. Schwarz. The market generator. *Available at SSRN*, 2019.
[32] A. Lipton. The vol smile problem. *Risk Magazine*, 15:61–65, 2002.
[33] P. Protter. *Stochastic integration and differential equations*, volume 21 of *Applications of Mathematics (New York)*. Springer-Verlag, Berlin, 1990. A new approach.
[34] Y. Ren, D. Madan, and M. Q. Qian. Calibrating and pricing with embedded local volatility models. *London Risk Magazine Limited-*, 20(9):138, 2007.
[35] H. Robbins and S. Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.
[36] J. Ruf and W. Wang. Neural networks for option pricing and hedging: a literature review. *Available at SSRN 3486363*, 2019.
[37] Y.-L. K. Samo and A. Vervuurt. Stochastic portfolio theory: a machine learning perspective. *Preprint, available at arXiv:1605.02654*, 2016.
[38] Y. F. Saporito, X. Yang, and J. P. Zubelli. The calibration of stochastic-local volatility models-an inverse problem perspective. *Preprint, available at arXiv:1711.03023*, 2017.
[39] J. Sirignano and R. Cont. Universal features of price formation in financial markets: perspectives from deep learning. *Quantitative Finance*, 19(9):1449–1459, 2019.
[40] Y. Tian, Z. Zhu, G. Lee, F. Klebaner, and K. Hamza. Calibrating and pricing with a stochastic-local volatility model. *Journal of Derivatives*, 22(3):21, 2015.
[41] M. S. Vidales, D. Siska, and L. Szpruch. Unbiased deep solvers for parametric pdes. *Preprint arXiv:1810.05094*, 2018.
[42] M. Wang, E. X. Fang, and H. Liu. Stochastic compositional gradient descent: algorithms for minimizing compositions of expected-value functions. *Mathematical Programming*, 161(1-2):419–449, 2017.
[43] M. Wiese, L. Bai, B. Wood, and H. Buehler. Deep hedging: learning to simulate equity option markets. *Preprint, Available at SSRN 3470756*, 2019.