



Neural network empowered liquidity pricing in a two-price economy under conic finance settings

MATTEO MICHIELON*†‡, DIOGO FRANQUINHO  §, ALESSANDRO GENTILE¶,
ASMA KHEDHER‡ and PETER SPREIJ  ‡||

†Quantitative Analysis and Quantitative Development, ABN AMRO Bank N.V., Gustav Mahlerlaan 10, Amsterdam, 1082 PP, The Netherlands

‡Korteweg-de Vries Institute for Mathematics, University of Amsterdam, Science Park 105-107, Amsterdam, 1098 XG, The Netherlands

§Department of Mathematics, Instituto Superior Técnico, Universidade de Lisboa, Lisboa, 1049-001, Portugal

¶Energy Services B.V., Joan Muyskenweg 22, Amsterdam, 1096 CJ, The Netherlands

||Institute for Mathematics, Astrophysics and Particle Physics, Radboud University Nijmegen, Huygens building, Heyendaalseweg 135, Nijmegen, 6525 AJ, The Netherlands

(Received 9 February 2024; accepted 27 July 2024)

In the article at hand neural networks are used to model liquidity in financial markets, under conic finance settings, in two different contexts. That is, on the one hand this paper illustrates how the use of neural networks within a two-price economy allows to obtain accurate pricing and Greeks of financial derivatives, enhancing computational performances compared to classical approaches such as (conic) Monte Carlo. The methodology proposed for this purpose is agnostic of the underlying valuation model, and it easily adapts to all models suitable for pricing in conic financial markets. On the other hand, this article also investigates the possibility of valuing contingent claims under conic assumptions, using local stochastic volatility models, where the local volatility is approximated by means of a (combination of) neural network(s). Moreover, we also show how it is possible to generate hybrid families of distortion functions to better fit the implied liquidity of the market, as well as we introduce a conic version of the SABR model, based on the Wang transform, that still allows for analytical bid and ask pricing formulae.

Keywords: Bid-ask spread; Conic finance; Concave distortion; Liquidity; Neural network

JEL Classifications: C45, G12

1. Introduction

The article at hand investigates two methodologies, based on neural networks, that allow valuing and risk managing financial derivatives in markets with frictions.

With the constantly increasing complexity of valuation models used within the financial industry, and the need for high-performance calculations and versatile modeling frameworks, machine learning is carving out more and more importance in this area. In this paper we propose two approaches, based on neural networks, allowing to compute prices and sensitivities of financial derivatives in a *two-price* economy, i.e. an economy where prices are direction-dependent. We assume that financial markets can be described according to the *conic finance* paradigm of Cherny and Madan (2010). We

first illustrate how to use (vector-valued) neural networks to jointly calculate bid and ask prices of contingent claims, as well as their sensitivities. The methodology is shown to be able to produce accurate results. Furthermore, within the framework of local stochastic volatility (LSV) modeling we extend the work of Cuchiero *et al.* (2020), which entails approximating the local volatility component of a LSV model by means of a (combination of) neural network(s), to markets with bid-ask spreads. The results made available in this paper illustrate the advantages introduced by the two methodologies proposed, and thus support further developments in the aforementioned areas.

The use of machine learning techniques has found applications in almost all disciplines (see, e.g. Johri *et al.* 2020 for an overview). And the financial industry has also, in the last decade, proven to be a fertile ground for this subject, with applications, e.g. in financial fraud detection (Awoyemi

*Corresponding author. Email: matteo.michielon@nl.abnamro.com

et al. 2017, Sadgali *et al.* 2019) and anti-money laundering (Chen *et al.* 2021, Domashova and Mikhailina 2021) to name but two. And it is expected that financial modeling and quantitative analysis are not an exception in this respect. In fact, different methodologies driven by machine learning algorithms have flourished in the last years to ameliorate computational speed (almost) without compromising accuracy; see, e.g. De Spiegeleer *et al.* (2018) and Davis *et al.* (2021). We also recall (Liu *et al.* 2019a), where a data-driven approach utilizing neural networks is used to calibrate financial asset price models in high dimensional stochastic volatility settings. Additionally, in Liu *et al.* (2019b) a framework to value options and calculate implied volatilities, aiming to accelerate numerical methods is presented, achieving a significant reduction in computing time across various solvers. We also mention (Buehler *et al.* 2019), which entails hedging derivative portfolios in markets with frictions, employing deep reinforcement learning methods, and capable of accurately approximating optimal solutions. However, to the best of our knowledge, there is currently no literature addressing how to perform accurate and efficient bid-ask pricing in financial markets under conic settings by means of machine learning techniques. This article proposes two methodologies, both employing neural networks, aiming to provide novel contributions in this area.

We point out that one could look at applications of neural networks in the area of derivatives pricing from two main perspectives. On the one hand, one could use a neural network as a parsimonious parametric model calculator aiming to replicate model prices generated by a chosen model. On the other hand, one could use a neural network as a non-parametric model builder aiming to infer the hidden relationships between contract prices, through time. And each of these two points of view can be used with a different goal. In the first case one could use neural networks (or any other machine learning technique deemed appropriate) to value complex financial products as done in Ferguson and Green (2018) for basket options. On the other hand, in the second case one could attempt to use different machine learning techniques to exploit statistical arbitrage opportunities in financial markets, as done in Krauss *et al.* (2017) in the case of the S&P 500 index.

The idea of this paper to adopt neural networks as model-building tools is driven by several factors. To start with, neural networks offer a high degree of architectural flexibility. Further, due to their common usage in fields such as image recognition (and related areas), which require processing billions of pixels at an exceptionally high speed, neural network implementations naturally allow for GPU acceleration features, which enable to significantly speed up calculations compared to CPU architectures (Oh and Jung 2004, Nasse *et al.* 2009). Moreover, neural networks allow for easy multi-valued regression implementations (Borchani *et al.* 2015), which is one possible way of interpreting bid-ask pricing. Last but not least, the choice of using neural networks is further supported by the literature; see, e.g. Morelli *et al.* (2004), amongst others. However, as far as conic finance is concerned, at the time of writing (Chopra 2020) seem to represent one of the few attempts to use machine learning in a conic economy. Nevertheless, the research conducted therein mainly applies

to pairs trading and index performance tracking, which are outside the domain we are dealing with in this paper. Lastly, we also mention (Madan and Sharaiha 2020), where machine learning strategies based on Gaussian processes and least squares regressions are compared and generalized by means of distorted expectations, leading to their distorted counterparts.

This article aims to connect the use of neural networks with pricing and risk-managing contingent claims in markets governed by the conic finance paradigm of Cherny and Madan (2010) and results in five contributions. In particular, we (i) provide an accurate, fast, *model-agnostic*, and fully neural network-based architecture for direction-dependent derivative pricing consistent with Cherny and Madan (2010), and we (ii) generalize it to compute sensitivities as well. Additionally, in the area of LSV modeling we (iii) extend the work of Cuchiero *et al.* (2020) to incorporate bid-ask pricing through conic Monte Carlo simulations. As a supplementary endeavor, we also (iv) introduce a conic version of the SABR model (Hagan *et al.* 2014), based on the Wang distortion (Wang 2000), which still allows for analytical bid and ask pricing formulae. Lastly, we (v) provide simple techniques to generate arbitrary families of hybrid distortion functions to be used in bid-ask calibration routines.

The manuscript is organized as follows. Section 2 provides the necessary tools and essential notions related to conic pricing under the paradigm of Cherny and Madan (2009). In particular, therein we recall how to perform bid-ask valuation using the Wang transform (Wang 2000) and introduce the conic SABR model, as well as we delineate a simple methodology to generate arbitrary families of distortion functions. The two modeling approaches investigated in this paper are presented in sections 3 and 4. In particular, in section 3 we illustrate how to perform bid and ask pricing by means of a (vectorized) neural network regression. On the other hand, in section 4 our work related to LSV modeling can be found. Therein we outline how it is possible to approximate the local volatility component of a LSV model by means of (a combination of) neural network(s) following the work of Cuchiero *et al.* (2020), and extend the aforementioned technique to a two-price economy by means of conic Monte Carlo. In both sections 3 and 4 we provide a selection of real-world illustrative examples. Section 5 concludes.

2. Bid-ask pricing with distorted expectations: an intuitive introduction

We start by considering a probability space $(\Omega, \mathcal{F}, \mathbb{Q})$, where \mathbb{Q} denotes a chosen pricing (i.e. risk-neutral) probability measure. Furthermore, we consider a contingent claim, paying out at a given (known) future time, represented with a random variable X . For simplicity, and for the ease of notation as well, from here onwards (and without loss of generality) we will always assume the time value of money to be zero (otherwise, discounted prices could be considered, instead).

If the contingent claim X trades under the law of one price (still dependent on the probability measure \mathbb{Q}), then it can be bought or sold for the same amount $\mathbb{E}^{\mathbb{Q}}(X) = \int_{\Omega} X d\mathbb{Q}$, which

can be rewritten as

$$\mathbb{E}^{\mathbb{Q}}(X) = \int_{-\infty}^0 (\mathbb{Q}(X \geq x) - 1) dx + \int_0^{+\infty} \mathbb{Q}(X \geq x) dx, \tag{1}$$

whenever the two integrals in (1) are not both infinite of opposite signs.

Assume now that a concave (distortion) function $\psi : [0, 1] \rightarrow [0, 1]$ such that $\psi(0) = 0$ and $\psi(1) = 1$ is available. One can then note, after re-weighting the \mathbb{Q} -probabilities in (1) by means of $\psi(\cdot)$, that

$$\begin{aligned} & \int_{-\infty}^0 (\psi(\mathbb{Q}(X \geq x)) - 1) dx + \int_0^{+\infty} \psi(\mathbb{Q}(X \geq x)) dx \\ & \geq \mathbb{E}^{\mathbb{Q}}(X). \end{aligned} \tag{2}$$

Therefore, the left-hand-side of (2), named (asymmetric) Choquet integral of X with respect to the distorted probability measure $\psi \circ \mathbb{Q}$ (see Denneberg 1994, Sec. 5), can be naturally interpreted as an ask price for X . This is because, via (2), higher (lower) weights are given to the high (low) realizations of X . From here onwards we will denote the left-hand-side of (2) as (C) $\int_{\Omega} X d\psi \circ \mathbb{Q}$. By recognizing that buying X is equivalent to sell $-X$, one then obtains that $-(C) \int_{\Omega} -X d\psi \circ \mathbb{Q}$ can be interpreted as a natural candidate for the bid price of X . This is because, in this case, lower (higher) weights are assigned to high (low) payoffs of X .

However, the definitions just given for bid and ask prices are not ‘operational’ yet. That is, one would need the distortion function to depend on (at least) one parameter, in such a way that changing the value(s) of the latter would allow to replicate the bid and ask spread observed in the market (at least in a least-squares sense should this not be possible in an exact manner). To this purpose (Cherny and Madan 2009) propose to consider an increasing family of distortion functions $(\psi_{\gamma})_{\gamma \geq 0}$ from $[0, 1]$ to $[0, 1]$ and such that $\psi_0(\cdot)$ coincides with the identity function (in this context the term ‘increasing’ should be interpreted in a pointwise sense). As a consequence, one can then obtain a wide range of γ -dependent bid and ask prices. From here on, whenever the symbol γ is used, it would always refer to a distortion parameter ranging in $[0, +\infty)$. By assuming the dynamics of the underlying under \mathbb{Q} are fully specified and known, one can then, by changing the values of γ , attempt to calibrate the theoretical conic prices to their market-observable counterparts by means of the relationships

$$\text{ask}_{\gamma}(X) = (C) \int_{\Omega} X d\psi_{\gamma} \circ \mathbb{Q} \tag{3}$$

and

$$\text{bid}_{\gamma}(X) = -(C) \int_{\Omega} -X d\psi_{\gamma} \circ \mathbb{Q} \tag{4}$$

where, for clarity, we emphasize that in (3) and (4) the same γ is used. We recall that the parameter γ and the Choquet representations of bid and ask prices are strongly linked to the concepts of acceptability indices and coherent risk measures. We refer the reader to Cherny and Madan (2009, 2010) and Madan and Schoutens (2016, Sec. 4) for a more detailed elucidation of the matter.

2.1. Conic pricing of European options with the Wang transform

In the applications we provide in the article at hand great use will be made of the Wang transform (see, e.g. section 3.2.1). We highlight here the reason why this concave distortion function plays an important role in the valuation of contingent claims in a direction-dependent trading environment.

The Wang transform was initially introduced in Wang (2000) to value both financial and insurance risks using Choquet expectations. In particular, in the framework outlined in Wang (2000) it is shown how this distortion function allows for an alternative construction of the Black–Scholes option pricing model. In symbols, the Wang transform reads

$$\psi_{\gamma}(x) := \Phi(\Phi^{-1}(x) + \gamma), \tag{5}$$

where $\Phi(\cdot)$ denotes the cumulative distribution function of a standard normal random variable, and $\Phi^{-1}(\cdot)$ indicates its inverse. As it is clear from (5), the Wang transform shifts the quantile of standard normal random variables by the positive amount γ . Note that in the context of conic pricing γ is assumed to be non-negative (with the case $\gamma = 0$ reducing to the risk-neutral case). However, in other frameworks the Wang transform can be also considered for negative values of the distortion parameter (note that for $\gamma > 0$ the Wang transform is concave, while when $\gamma < 0$ it is convex).

Definition (5) (see also Madan and Schoutens 2016, Sec. 4.7.5) is very useful in practice. In particular, the Wang transform has the convenient property that, if a random variable is lognormally distributed, then by compounding the distribution function of the latter with the Wang transform, still a lognormal distribution is obtained (with adjusted mean). Similar considerations also hold for the normal case. In addition, it has been recently shown, see Michielon *et al.* (2021), that this property still holds whenever a normal random variable is transformed by means of a non-decreasing function. Therefore, the use of the Wang transform has become very popular in conic pricing, as it allows to still obtain analytical formulae for bid and ask prices of European options under both Bachelier (see Michielon *et al.* 2021) and Black–Scholes (see [Sec. 5.4] Madan and Schoutens 2016) settings (and, of course, in all those situations where (log)normal distributions relate to the relevant options pricing formulae; see, e.g. Haug (2007, Sec. 1.3) for some examples). From this it follows that the Wang transform is arguably the natural distortion to be used in option pricing given that vanilla options are often quoted in terms of their (lognormal or normal) implied volatilities.

2.1.1. Interlude: conic SABR. The observations made in section 2.1 concerning the Wang distortion allow us to make some interesting considerations concerning the SABR model (Hagan *et al.* 2002, 2014). This model is very popular in the financial industry and it has applications in different asset classes: mainly rates, but also applications of the SABR model (potentially with adaptations) can be found for foreign-exchange (van der Stoep *et al.* 2015) and equity (Overhaus *et al.* 2007, Sec. 2.1.2) markets, amongst others.

For an underlying process $(Y_t)_{t \geq 0}$, the standard SABR and specifications read

$$\begin{cases} dY_t = \sigma_t Y_t^\beta dW_t \\ d\sigma_t = \alpha \sigma_t dZ_t \\ d\langle W_t, Z_t \rangle = \rho dt \end{cases}, \tag{6}$$

where $\alpha > 0$, $\rho \in [-1, 1]$ (i.e. the correlation coefficient between the two standard Brownian motions $(W_t)_{t \geq 0}$ and $(Z_t)_{t \geq 0}$), and where $\beta \in [0, 1]$. In Hagan *et al.* (2014) it is shown that, up to higher order effects, the SABR specifications (6) still allow to use the Black formula for pricing European options, but this time with a ‘twicked’ implied volatility dependent on a set of (calibrated) parameters and strikes. Note that, in practice, the parameter β is often not calibrated but, instead, exogenously set equal to one (zero) in the case of lognormal (normal) SABR specifications, or to $\frac{1}{2}$ for ‘in-between’ dynamics. We denote, for a fixed maturity T and for a given strike price K , the SABR volatility as $\sigma^{\text{SABR}}(K, T)$, with

$$\sigma^{\text{SABR}}(K, T) = \frac{\sigma_0 \left\{ 1 + \left[\frac{(1-\beta)^2}{24} \frac{\sigma_0^2}{(Y_0 K)^{1-\beta}} + \frac{1}{4} \frac{\rho \beta \alpha \sigma_0}{(Y_0 K)^{(1-\beta)/2}} + \frac{2-3\rho^2}{24} \alpha^2 \right] T \right\}}{(Y_0 K)^{(1-\beta)/2}} \cdot \frac{z}{\chi(z)}, \tag{7}$$

$$\left[1 + \frac{(1-\beta)^2}{24} \ln^2 \left(\frac{Y_0}{K} \right) + \frac{(1-\beta)^4}{1920} \ln^4 \left(\frac{Y_0}{K} \right) \right]$$

where

$$z := \frac{\alpha}{\sigma_0} (Y_0 K)^{(1-\beta)/2} \ln \left(\frac{Y_0}{K} \right),$$

$$\chi(z) := \ln \left(\frac{\sqrt{1 - 2\rho z + z^2} + z - \rho}{1 - \rho} \right),$$

and with σ_0 the initial value of the volatility.

By virtue of the properties of the Wang transform outlined in section 2.1, we obtain that, under conic settings and SABR risk-neutral model specifications, call and put option prices can be still computed analytically. That is, for a given liquidity level γ and having defined $\delta (= \delta(K, T; \gamma)) := \frac{\gamma \sigma^{\text{SABR}}(K, T)}{\sqrt{T}}$, it holds that

$$\text{bid}_\gamma(\mathcal{C}(K, T)) = Y_0 e^{-\delta T} \Phi(d_1) - K \Phi(d_2), \tag{8}$$

$$\text{ask}_\gamma(\mathcal{C}(K, T)) = Y_0 e^{\delta T} \Phi(d_1) - K \Phi(d_2), \tag{9}$$

$$\text{bid}_\gamma(\mathcal{P}(K, T)) = K \Phi(-d_2) - Y_0 e^{\delta T} \Phi(-d_1), \tag{10}$$

and

$$\text{ask}_\gamma(\mathcal{P}(K, T)) = K \Phi(-d_2) - Y_0 e^{-\delta T} \Phi(-d_1), \tag{11}$$

where

$$d_1 := \frac{\ln \left(\frac{Y_0}{K} \right) + \left(-\delta + \frac{\sigma^{\text{SABR}}(K, T)}{2} \right) T}{\sigma^{\text{SABR}}(K, T) \sqrt{T}},$$

$$d_2 := d_1 - \sigma^{\text{SABR}}(K, T) \sqrt{T}.$$

In formulae (8), (9), (10) and (11) $\mathcal{C}(K, T)$ ($\mathcal{P}(K, T)$) denotes the (γ -dependent) price of a European call (put) option with strike K and expiry T .

2.2. Conic Monte Carlo

We now briefly describe how to perform bid and ask pricing of financial derivatives, within the conic finance framework, by means of simulations. This methodology will be used in the numerical examples provided later on in this paper (see sections 3 and 4).

Under risk-neutral settings, given a (potentially path-dependent) contingent claim paying out at a fixed future time, we assume that N scenarios $\omega_1, \dots, \omega_N$ have been sampled to obtain the corresponding payoffs $x_1 := X(\omega_1), \dots, x_N := X(\omega_N)$. One would just need to average the latter to obtain the current value of the contract (again, up to discounting). This is because each outcome is considered equally likely.

In a two-price economy, on the other hand, the realizations of the contingent claim across the different states of nature need to be weighted differently. Intuitively, giving more (less) weight to high (low) payoffs allows to obtain an ask (a bid) price for X (see beginning of section 2). That is, assume that the outcomes x_1, \dots, x_N are, without loss of generality, ordered in a non-decreasing way. To calculate the bid and ask prices of X one needs to weight the sampled realizations in an alternative fashion. Namely, for each $i \in \{1, \dots, N\}$ and given distortion parameter $\gamma \in [0, +\infty)$, the probability to be assigned to x_i to calculate the bid price of X equals

$$p_i^{\text{bid}_\gamma} := \psi_\gamma \left(\frac{i}{N} \right) - \psi_\gamma \left(\frac{i-1}{N} \right). \tag{12}$$

Contrariwise, that to be used to compute its ask counterpart should equal

$$p_i^{\text{ask}_\gamma} := \psi_\gamma \left(\frac{N-i+1}{N} \right) - \psi_\gamma \left(\frac{N-i}{N} \right). \tag{13}$$

Thus, denoting with $x := (x_1, \dots, x_N)$, $p^{\text{bid}_\gamma} := (p_1^{\text{bid}_\gamma}, \dots, p_N^{\text{bid}_\gamma})$ and $p^{\text{ask}_\gamma} := (p_1^{\text{ask}_\gamma}, \dots, p_N^{\text{ask}_\gamma})$, to value in a two-price economy one just needs to compute bid and ask prices by means of the scalar products

$$\text{bid}_\gamma(X) = \langle x, p^{\text{bid}_\gamma} \rangle \tag{14}$$

and

$$\text{ask}_\gamma(X) = \langle x, p^{\text{ask}_\gamma} \rangle, \tag{15}$$

respectively. From (12) and (13) it is obvious that the case where $\psi_\gamma(\cdot)$ equals the identity function the standard uniform weighting case is retrieved. For the derivations of formulae (14) and (15), which basically depend on a discretized version of the Choquet integral (see Wang and Klir 2009, Sec. 11.5), refer to Madan and Schoutens (2016, Sec. 5.2).

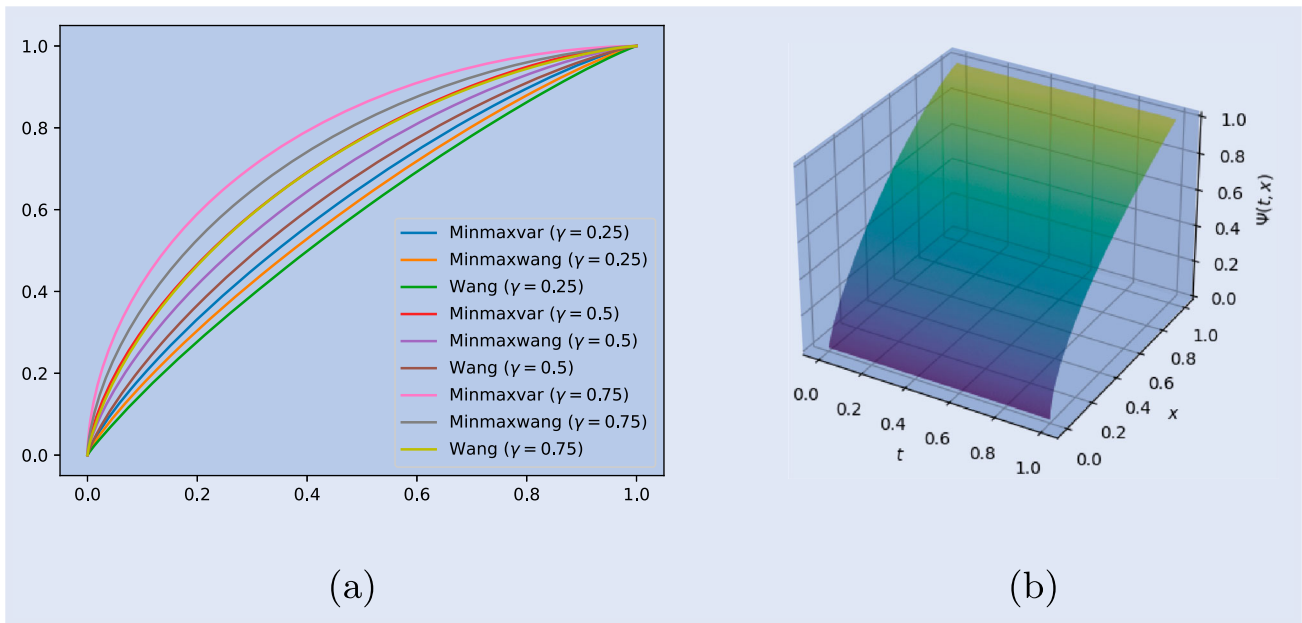


Figure 1. Graphical depiction of the Minmaxwang distortion given different levels of the distortion parameter γ , panel (a), and of the continuous mapping $H(\cdot, \cdot)$ transforming the Minmaxvar distortion into the Wang distortion, panel (b).

2.3. Hybrid distortions

In practical applications, the choice of the distortion function is somehow arbitrary. For instance, as seen in section 2.1 (and section 2.1.1 therein), the Wang transform is often used for European option pricing as it allows to obtain, under normal or lognormal dynamics, analytical pricing formulae. However, to the best of our knowledge, there is currently no methodology available providing some rules of thumb for distortion function selection. And, further, the ultimate aspect that a modeler is interested in is given by how well a given distortion function allows to replicate the bid ask spreads observed in the market. Thus, in this section, we propose some simple techniques that allow to generate arbitrary families of distortions, generalizing the currently available ones, and that can be used as test functions to choose whether one distortion is better than another, or whether something ‘in-between’ would fit even better. Therefore, one might seek to combine the characteristics and specifications of different distortion families to introduce more flexibility to the modeling framework. Here we provide some intuitive ways to do so. Observe that all the hybrid distortion functions we are going to define in this section will be dependent on a single distortion parameter. This is because, from a practical angle, one would be interested in quantifying liquidity risk by means of a single parameter (as, for instance, one would do in the case of implied volatility for options) and, more precisely, in terms of liquidity delta.

First of all, given N (γ -dependent) distortion functions $\psi_\gamma^1(\cdot), \dots, \psi_\gamma^N(\cdot)$, observe that any convex linear combination of them is still a concave distortion function. This means that, if we consider the combined distortion

$$\tilde{\psi}_\gamma(x) := \sum_{i=1}^N \alpha_i \psi_\gamma^i(x) \quad (16)$$

such that, for every $i \in \{1, \dots, N\}$, $\alpha_i \geq 0$ and with $\sum_{i=1}^N \alpha_i = 1$, we still obtain a concave distortion. Furthermore, it trivially follows that the family $(\tilde{\psi}_\gamma)_{\gamma \geq 0}$ is increasing with respect to γ and that $\tilde{\psi}_0(\cdot)$ coincides with the identity function on $[0, 1]$.

As an illustration of this, we take into account the Minmaxvar distortion, defined as $\psi_\gamma^{\text{Minmaxvar}}(x) := 1 - (1 - x^{\frac{1}{\gamma+1}})^{\gamma+1}$ (see Cherny and Madan 2009), and the Wang distortion. I.e. we set $N = 2$ and assume that $\psi_\gamma^1(\cdot) := \psi_\gamma^{\text{Minmaxvar}}(\cdot)$, and that $\psi_\gamma^2(\cdot) := \psi_\gamma^{\text{Wang}}(\cdot)$, with $\alpha_1 := 1 - t$ and $\alpha_2 := t$ (for $t \in [0, 1]$). Intuitively, this way we have created a family of distortions that, continuously, transforms the Minmaxvar distortion into the Wang distortion (and the other way round). That is, we are simply considering the straight-line homotopy between $\psi_\gamma^{\text{Minmaxvar}}(\cdot)$ and $\psi_\gamma^{\text{Wang}}(\cdot)$, i.e. the map $H : [0, 1] \times [0, 1] \rightarrow [0, 1]$ given by $H(t, x) := (1 - t) \cdot \psi_\gamma^{\text{Minmaxvar}}(x) + t \cdot \psi_\gamma^{\text{Wang}}(x)$ such that $H(0, x) = \psi_\gamma^{\text{Minmaxvar}}(x)$, while $H(1, x) = \psi_\gamma^{\text{Wang}}(x)$. In particular, we name the ‘average’ distortion, obtainable when $t = 0.5$, from here onwards, Minmaxwang, for the ease of terminology (and t -Minmaxwang when $t \neq 0.5$). The Minmaxwang distortion is illustrated in figure 1(a), while the homotopy $H(\cdot, \cdot)$ in figure 1(b).

The advantage of considering combined distortions as proposed here is that it makes it possible to have one (or more, if additional distortions are also considered) ‘helper parameter’ that allows for more flexibility in the calibration routine. Given that every distortion function on the right-hand-side of (16) can be obtained by setting the appropriate parameters to zero, then it is guaranteed that (16) will never worsen the calibration results for any set of distortion functions initially chosen.

REMARK 2.1 With reference to Madan and Schoutens (2016, Sec. 4.7), we recall that two desirable properties of a distortion

function are, for $\gamma > 0$, that

$$\lim_{x \rightarrow 0^+} \psi'_\gamma(x) = +\infty, \quad (17)$$

and that

$$\lim_{x \rightarrow 1^-} \psi'_\gamma(x) = 0. \quad (18)$$

One can then trivially observe that, for (17) to be verified by $\tilde{\psi}_\gamma(\cdot)$, then it is enough that one of the distortion functions $\psi_\gamma^1(\cdot), \dots, \psi_\gamma^N(\cdot)$ is such that (17) holds. On the other hand, in order for condition (18) to be verified for $\tilde{\psi}_\gamma(\cdot)$, then (18) needs to be verified for all the distortions $\psi_\gamma^1(\cdot), \dots, \psi_\gamma^N(\cdot)$.

We now propose another way of generating hybrid distortions. That is, we observe that the composition of the N distortion functions $\psi_\gamma^1(\cdot), \dots, \psi_\gamma^N(\cdot)$ is still a valid distortion function. Therefore, if we start by taking into account, as done above, the Minmaxvar and Wang distortions, then we can define the Minmaxvar-Wang distortion as $\psi_\gamma^{\text{Wang}} \circ \psi_\gamma^{\text{Minmaxvar}}(\cdot)$. However, note that defining distortions using the methodology just highlighted is not a commutative operation. Therefore, we could also opt for the Wang-Minmaxvar distortion by inverting the order of the composition, i.e. by taking into account $\psi_\gamma^{\text{Minmaxvar}} \circ \psi_\gamma^{\text{Wang}}(\cdot)$ and thus define the Wang-Minmaxvar function.

REMARK 2.2 Let us consider here the simple case given by the combination of two distortion functions, i.e. let

$$\hat{\psi}_\gamma(\cdot) := \psi_\gamma^1 \circ \psi_\gamma^2(\cdot).$$

It then results, for $x \in [0, 1]$, that

$$\hat{\psi}'_\gamma(x) = (\psi_\gamma^1)'(\psi_\gamma^2(x)) \cdot (\psi_\gamma^2)'(x).$$

By definition, both $\psi_\gamma^1(\cdot)$ and $\psi_\gamma^2(\cdot)$ map zero to zero and one to one. Therefore, we obtain that condition (17) is satisfied whenever one of the derivatives of $\psi_\gamma^1(\cdot)$ or $\psi_\gamma^2(\cdot)$ diverges at zero (provided that the other, if any, is non-zero). Similarly, condition (18) is verified whenever one of the derivatives of $\psi_\gamma^1(\cdot)$ or $\psi_\gamma^2(\cdot)$ tends to zero at one (provided that the other, if any, is finite).

3. Conic neural networks: liquidity pricing and hedging with vector-valued neural networks

In this section, we illustrate a methodology that allows to computation of bid and ask prices in a two-price economy under the conic finance paradigm of Cherny and Madan (2010). In particular, we outline how neural networks can be used for bid-ask calibration purposes. From here onwards we will call every neural network to be used to produce prices of contingent claims in a two-price economy a *conic neural network*, and we will denote it with the acronym CNN (not to be confused with the acronym for a convolutional neural network).

3.1. A motivating example

We first start by taking into account a simple example to provide a step-by-step illustration of how to calibrate (and, therefore, price) in a two-price economy under conic finance settings by using CNNs. We consider an economy such that, in a risk-neutral context, the dynamics of a selected (non-dividend paying) underlying asset of interest follow the specifications of the Black–Scholes model. We assume that, for the underlying chosen, a set of European call options, all with the same maturity, are quoted. Furthermore, we also hypothesize that the bid and ask prices of these quoted options under these simplistic settings are consistent with the conic paradigm. That is, we assume that they can all be obtained, starting from their risk-neutral prices, by means of applying the Wang distortion (Wang 2000) (see also section 2.1) with the same constant distortion parameter. Thus, we are in a simple economy where both risk-neutral and bid-ask prices can be obtained by means of analytical formulae (Madan and Schoutens 2016, Sec. 5.4) (see also section 2.1.1). Besides, assume that the underlying price $Y_0 = 10$, that the time to maturity $T = 1$, that the constant volatility parameter $\sigma = 25\%$, and that the distortion parameter $\gamma = 12.5\%$. Also, it is supposed that the discount rate is zero and that the strike prices K_1, \dots, K_{11} of the quoted options belong to $\{7.5, 8, 8.5, 9, 9.5, 10, 10.5, 11, 11.5, 12, 12.5\}$, i.e. that they correspond to the 75% to 125% option moneyness levels (equally spaced by 0.5 currency units). We denote the risk-neutral option prices as C_1, \dots, C_{11} , while their corresponding bid and ask prices as b_1, \dots, b_{11} and a_1, \dots, a_{11} , respectively. In this case, b_1, \dots, b_{11} and a_1, \dots, a_{11} represent our (synthetic) market dataset.

In order to train the neural network that will be used for model calibration, one needs to proceed per steps (we assume here that the market data, i.e. the underlying price Y_0 and the zero risk-free rate are fixed and static, for illustration purposes). Note, however, that this assumption is not restrictive, as it would be sufficient to enlarge the training dataset of the CNN by considering different values for the underlying and for the discount rate. First of all one needs to select large-enough intervals where the (to-be-estimated) volatility and liquidity parameters are assumed to belong. In this case, we consider $[0, 50\%]$ for the volatility, and $[0, 25\%]$ for the distortion. We also consider the range $[7, 13]$ for the strike price. One can then specify a grid $0 =: \sigma_1 < \dots < \sigma_N := 50\%$ for the volatility parameter, $0 =: \gamma_1 < \dots < \gamma_N := 25\%$ for the distortion parameter, and $\tilde{K}_1 := 7, \dots, \tilde{K}_N := 13$ for the strike price (with respect to this example we select 125 equally-distant points in all the cases considered, that is, $N = 100$). Here we use the tilde superscript for the training grid in the strike direction to distinguish this dataset from the ‘market-observables’ K_1, \dots, K_{11} . We remark that, in principle, also considering (random and) uniformly distributed parameters is a possible choice, as well as considering a different amount of training points for the volatility compared to that of the distortion parameter or to that of the strike price. Decisions should always be made on the base of experimentation depending on the model specifications considered. Note, however, that as far as the distortion parameter is concerned, zero should always belong to the grid, as needed for the preliminary

risk-neutral calibration as it will be shown later. For each pair $(\sigma_i, \gamma_j, K_k)$ (here, i, j and k all range in $\{1, \dots, 100\}$) one can then generate the corresponding theoretical bid and ask prices by using the analytical bid and ask option pricing formulae of Madan and Schoutens (2016, Sec. 5.3). Afterwards, the training of the CNN takes place. In this case, therefore, we are training a neural network which attempts to approximate the function, from $[0, +\infty)^3$ to $[0, +\infty)^2$, such that $(\sigma, \gamma, K) \mapsto (\text{bid}_\gamma(\mathcal{C}(\sigma, K)), \text{ask}_\gamma(\mathcal{C}(\sigma, K)))$, where $\text{bid}_\gamma(\mathcal{C}(\sigma, K))$ ($\text{ask}_\gamma(\mathcal{C}(\sigma, K))$) represents the bid (ask) price of a call option \mathcal{C} with, under conic Black–Scholes settings, an implied volatility of σ , a distortion parameter of γ , and a strike price equal to K . Therefore, the first component of the neural network, i.e. $\text{CNN}(\cdot, \cdot, \cdot)_1$, denotes the bid price, while the second one, i.e. $\text{CNN}(\cdot, \cdot, \cdot)_2$, the ask price, instead.

Once the training of the neural network has terminated, our model approximator is ready for the calibration phase. Thus, we first consider $\text{CNN}(\cdot, 0, \cdot)$ and perform the first minimization

$$\min_{\sigma \geq 0} \sum_{i=1}^{11} \left(\mathcal{C}_i - \frac{1}{2} (\text{CNN}(\sigma, 0, K_i)_1 + \text{CNN}(\sigma, 0, K_i)_2) \right)^2. \tag{19}$$

Note that in (19) we have considered the average of the bid and the ask prices, calculated using the CNN, after having set the distortion parameter equal to zero. Observe that, theoretically speaking, both of them should coincide with the corresponding risk-neutral prices of the options taken into account. Thus, in practice, for σ and K given, one could have just considered $\text{CNN}(\sigma, 0, K)_1$ or $\text{CNN}(\sigma, 0, K)_2$, instead of their average. However, due to numerical approximations, $\text{CNN}(\sigma, 0, K)_1$ and $\text{CNN}(\sigma, 0, K)_2$ are never exactly the same, so for instance one could choose to approximate the risk-neutral price with the average $\frac{1}{2}(\text{CNN}(\sigma, 0, K)_1 + \text{CNN}(\sigma, 0, K)_2)$. Figure 2 illustrates how $\text{CNN}(\cdot, 0, \cdot)$ approximates the synthetic risk-neutral prices $\mathcal{C}_1, \dots, \mathcal{C}_{11}$. Observe that, as far as the liquidity parameter is concerned, for the CNN zero is the left-most point that belongs to the training dataset. Therefore, one could also argue that considering the average $\frac{1}{2}(\text{CNN}(\sigma, \epsilon, K)_1 + \text{CNN}(\sigma, \epsilon, K)_2)$ for ϵ small would likewise be an appropriate choice to be sure that the approximated risk-neutral price belongs to the interpolation area of the CNN. However, from the practical experiments we have performed, it seems that both choices perform well enough with marginal and negligible differences.

Once an optimal σ has been estimated as described in the former step (which we denote with $\bar{\sigma}$), the bid-ask calibration routine can finally take place. This means that the second minimization step that aims to calculate the implied liquidity parameter can start. That is, the minimization

$$\min_{\gamma \geq 0} \sum_{i=1}^{11} \left((\text{bid}(\mathcal{C}_i) - \text{CNN}(\bar{\sigma}, \gamma, K_i)_1)^2 + (\text{ask}(\mathcal{C}_i) - \text{CNN}(\bar{\sigma}, \gamma, K_i)_2)^2 \right) \tag{20}$$

should be performed. Once an estimate for the optimal distortion parameter, denoted with $\bar{\gamma}$, has been found, then the

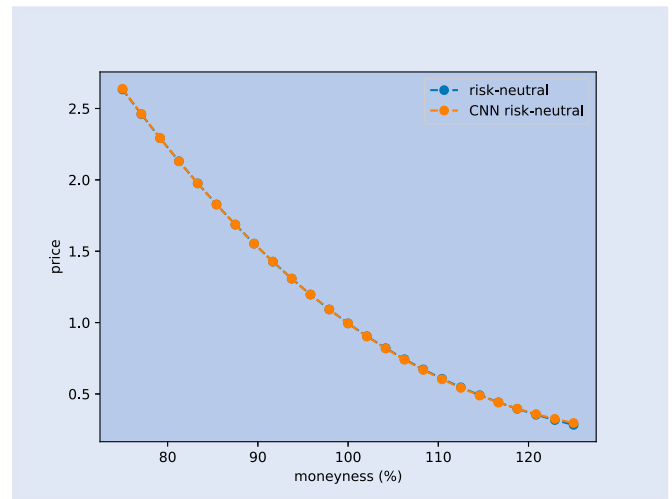


Figure 2. Synthetic risk-neutral prices approximated by $\text{CNN}(\cdot, 0, \cdot)$ by means of $\frac{1}{2}(\text{CNN}(\cdot, 0, \cdot)_1 + \text{CNN}(\cdot, 0, \cdot)_2)$.

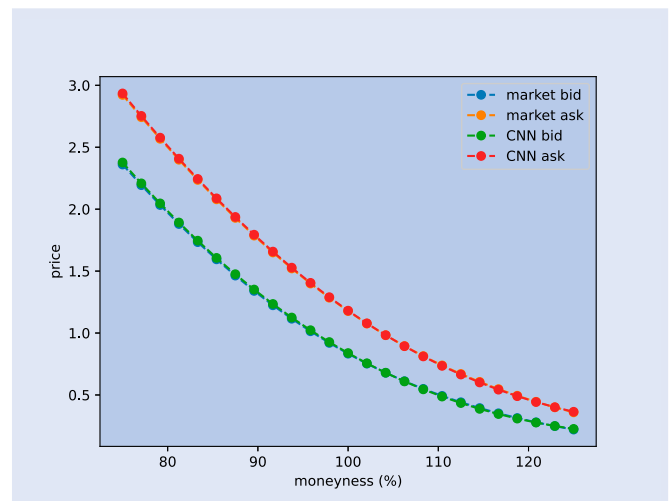


Figure 3. Synthetic bid and ask prices approximated by $\text{CNN}(\bar{\sigma}, \bar{\gamma}, \cdot)$.

two-price model has been calibrated, and it can thus be used for valuation and sensitivity calculation purposes. A graphical depiction of the calibrated bid and ask prices computed by means of the CNN has been provided in figure 3, which shows how the prices obtained by means of the CNN basically overlap with those of the synthetic market data set.

Please note that, as far as the distortion parameter is concerned, in practice one would need to estimate a different distortion parameter per strike (similar to what is done in practice with implied volatilities, which are computed in a strike-dependent manner). I.e. one would need to perform, for every i , the minimization

$$\min_{\gamma \geq 0} \left((\text{bid}(\mathcal{C}_i) - \text{CNN}(\bar{\sigma}, \gamma, K_i)_1)^2 + ((\text{ask}(\mathcal{C}_i) - \text{CNN}(\bar{\sigma}, \gamma, K_i)_2)^2 \right),$$

given that implied liquidity amounts are, in general, not constant across the strike direction. This will be shown in

section 3.2. In principle, one could also separate the calibration in two parts and determine a γ for the bid by minimizing $(\text{bid}(\mathcal{C}_i) - \text{CNN}(\bar{\sigma}, \gamma, K_i)_1)^2$, and one for the ask by minimizing $(\text{ask}(\mathcal{C}_i) - \text{CNN}(\bar{\sigma}, \gamma, K_i)_2)^2$. However, this would be slightly inconsistent, from a theoretical perspective, with (3) and (4), which state that the same value of the distortion parameter γ should be considered for the bid and the ask prices of the same contingent claim.

3.2. CNNs in use

In this section, we provide relevant applications of the idea concerning CNNs proposed in section 3.1. That is, we will highlight how the use of neural networks in quantitative analysis expresses its best in situations where more standard procedures still require a considerable amount of time to provide the desired results. This is particularly true in the case of Monte Carlo methods which, despite different techniques that can be adopted to improve their performances remain, generally speaking, computationally expensive.

From here onwards we will always assume, unless differently specified, that a given CNN has been previously trained. As already done in section 3.1, without loss of generality we will consider the training with respect to a set of parameters (generically denoted θ), the distortion γ , as well as the strike price given that, in this article, we are dealing with options. However, extending the methodology to the case where the training step is performed with respect to different market data sets (and/or parameters) is similar. Thus, the steps listed here should not be viewed as limiting in any way.

Given $\theta := (\theta_1, \dots, \theta_N) \in \mathbb{R}^N$ we consider M possible values for θ_1 , as well as for the remaining $N - 1$ parameters in θ , M possible values for the distortion parameter γ , and M possible values for the strike price K . That is, we construct a grid of points $\{\theta_1^1, \dots, \theta_1^M\}$ for θ_1 , until $\{\theta_N^1, \dots, \theta_N^M\}$ for θ_N , together with M points for the distortion parameter $\{\gamma^1, \dots, \gamma^M\}$ and for the strike price $\{K^1, \dots, K^M\}$ too. In this case, we have chosen an equal number of points per variable. However, in principle this is not a strict requirement. Furthermore, one can also choose whether to consider equally-spaced grids or not. One important remark is that γ^0 should always be equal to zero, as this is necessary due to the fact that training in a conic framework first requires calibrating under risk-neutral settings, a circumstance that can be retrieved when $\gamma^0 = 0$. For each of the $(N + 2)^M$ possible parameter combinations $(\theta_1^{i_1}, \dots, \theta_N^{i_N}, \gamma^{i_{N+1}}, K^{i_{N+2}})$, where i_1, \dots, i_{N+2} range in $\{1, \dots, M\}$, one should generate option prices with a given chosen model. The CNN can be then trained following the step-by-step procedure described in section 3.1. As already mentioned, note that the CNN is agnostic of the technique used to produce contract payoffs, so the latter can therefore be chosen at will.

The modeling framework introduced by the motivating example of section 3.1 (and that will be specified further in the remainder of the current section) aims to construct a neural network able to replicate bid and ask prices generated by a given model under conic settings in a parsimonious way from a computational perspective. That is, we aim to construct a (model dependent) neural network that is able to replace a

given computationally-expensive model with a ‘clone’ that is numerically more efficient. This means that the CNN will be a model surrogate used exactly as the initial chosen model was supposed to be used. For instance, the CNN would be recalibrated (but not re-trained) every time the market moves if the aim was that of using the CNN for live pricing, exactly as it would be the case for any pricing model. As the CNNs we are going to build aim to understand the relationship between model inputs and model outputs without knowing the functional form of the model considered, we say that CNNs are model-agnostic. We point out that the CNNs we introduce in this paper are trained given a set of parameters, and that this step needs to be performed only once (assuming the parameter range is wide enough). We also empathize that one might use neural networks to price (and risk manage) financial contracts in a different way. That is, one could train the neural network given (quoted bid and ask) prices directly. This way one could use a purely data-driven approach which would then be, as not dependent on any stochastic process for the underlying asset(s), *model-free* (up to the model choices related to the specifications of the neural network used). A similar example to this last possibility, within the area of LSV modeling, is that of avoiding assuming a specific functional and parametric form for the local volatility function but, instead, using a neural network to indirectly infer it, as done in Cuchiero *et al.* (2020) (see also section 4).

3.2.1. A fully-fledged arbitrage-free calibration routine with CNNs: an illustration. We highlight here an approach to perform bid and ask calibration of European call options by means of vector-valued neural networks (the case of put options would be analogous).

To begin with we assume, given a fixed maturity T , that L call options $\mathcal{C}_1, \dots, \mathcal{C}_L$, with strike prices $K_1 < \dots < K_L$, respectively, are quoted in the market for a given underlying. We denote the value of the underlying asset, at the current time, by Y_0 . We denote with b_1, \dots, b_L and with a_1, \dots, a_L the corresponding bid and ask option prices quoted in the market. We now illustrate the procedure to be followed.†

- (i) Foremost, recall that calibrating to bid and ask quotes under conic finance settings first requires to perform a risk-neutral calibration as initial mandatory step. We highlight that, if for a given option bid and ask prices are quoted, then (any of) their risk-neutral prices would lie in-between them, but would not necessarily coincide with the mid price. As a common way to compare and quote European options is by means of their implied volatilities, we consider here the framework introduced in Michielon *et al.* (2021). That is, for each of the quoted options, we compute the correspondent *liquidity-free* implied volatility and related distortion parameter. This means that, in symbols, for

† From here onwards, for brevity and convenience, the index i will be always assumed to be ranging in $\{1, \dots, L\}$.

the i th option we solve the system

$$\begin{cases} \text{bid}_{\gamma_i}^{\text{BS}}(\sigma_i, K_i) = b_i \\ \text{ask}_{\gamma_i}^{\text{BS}}(\sigma_i, K_i) = a_i \end{cases} \quad (21)$$

numerically with respect to σ_i and γ_i . In (21), $\text{bid}_{\gamma_i}^{\text{BS}}(\cdot, \cdot)$ and $\text{ask}_{\gamma_i}^{\text{BS}}(\cdot, \cdot)$ denote the conic Black–Scholes bid and ask functionals, assuming the Wang transform is used. Further, σ_i and γ_i represent the Black–Scholes implied volatility and the Wang implied distortion values for the option considered, respectively. The non-linear system in two equations and two unknowns (21) is guaranteed (under almost-always satisfied assumptions) to have a unique solution. We denote the so-calculated implied volatility and liquidity amounts with $\bar{\sigma}_i$ and $\bar{\gamma}_i$, respectively;

- (ii) Once we have successfully managed to compute the liquidity-free implied volatilities $\bar{\sigma}_1, \dots, \bar{\sigma}_L$, we can then calculate their corresponding risk-neutral Black–Scholes prices, which we denote as $\text{BS}(\bar{\sigma}_1, K_1), \dots, \text{BS}(\bar{\sigma}_L, K_L)$, respectively;
- (iii) Observe that, at risk-neutral level, the call prices $\text{BS}(\bar{\sigma}_1, K_1), \dots, \text{BS}(\bar{\sigma}_L, K_L)$ are not automatically guaranteed to be arbitrage-free. Therefore, for the relevant arbitrage-free conditions to be satisfied, we apply the option data pre-processing *filtering methodology* proposed in Moussa (2018), which provides an approach for adjusting option data when no-arbitrage conditions are violated. This means that option prices, when no-arbitrage conditions are infringed, get updated in an arbitrage-free manner with respect to the other options taken into account in the dataset until no-arbitrage bounds hold again. Observe that applying this procedure, and therefore updating risk-neutral prices whenever necessary, is both theoretically and practically justified. This since risk-neutral prices are neither tradable nor observable. Therefore, as soon as they lie within their corresponding bid and ask counterparts in an arbitrage-free way, applying (Moussa 2018) is a legitimate step;
- (iv) After the filtering procedure has been applied, the actual calibration can take place. Firstly, we assume a distortion parameter of zero as, under these circumstances, the calibration needs to be performed at risk-neutral level first. That is, one needs to perform the minimization procedure

$$\min_{\theta \in \Theta} \sum_{i=1}^L \left(\text{BS}(\bar{\sigma}_i, K_i) - \frac{1}{2}(\text{CNN}(\theta, 0, K_i)_1 + \text{CNN}(\theta, 0, K_i)_2) \right)^2,$$

where Θ indicates a large-enough set containing the ‘true’ risk-neutral model parameters. In order to find a good estimate for the optimal parameters in Θ one can for instance use a direction set method (Press *et al.* 2007, Sec. 10.7) or a (quasi-)Newton method (Press *et al.* 2007, Sec. 10.9), depending on the circumstances, after having estimated the initial guess

by means, e.g. of a genetic algorithm as per (Kaelo and Ali 2007). We denote the risk-neutral parameters estimated as just described by $\bar{\theta}$;

- (v) Now that $\bar{\theta}$ has been estimated and thus the CNN is calibrated under risk-neutral settings, the bid-ask calibration routine begins. That is, for each i , the implied liquidity parameter is computed by performing the minimization

$$\min_{\gamma_i \geq 0} \left((b_i - \text{CNN}(\bar{\theta}, \gamma_i, K_i)_1)^2 + (a_i - \text{CNN}(\bar{\theta}, \gamma_i, K_i)_2)^2 \right). \quad (22)$$

By experimenting we observe that, at least as far as the Wang transform is concerned, using $\frac{1}{2}\bar{\sigma}_i$ as initial guess for a local minimization routine seems to work well enough.

Now all the parameters of the CNN have been estimated in such a way that the CNN is capable of pricing back (in a least-square sense) the quoted options. This makes it suitable for pricing and risk-management purposes. In particular, bid and ask prices of non-quoted options can be calculated by feeding the CNN with the appropriate strike and liquidity levels (the liquidity levels of the market can be interpolated from the implied ones, for instance linearly or by using a monotonic interpolation Fritsch and Carlson 1980).

3.2.2. A real-world example. For illustrative purposes, we will perform experiments concerning options pricing based on the rough Bergomi (rBergomi) model (Bayer *et al.* 2016). Rough volatility models are becoming more and more popular in equity volatility modeling. However, they are in general expensive to simulate. In our tests we take into account the rBergomi model as (i) rough volatility modeling is still a relatively new framework in financial engineering. Thus, we believe that considering an up-to-date financial modeling environment as a benchmark adds value to the analysis provided here. Further, this choice (ii) provides a valuable example to illustrate the computational advantage coming from the use of neural networks for valuation and sensitivities calculation purposes in a market with frictions. And besides, (iii) to the best of our knowledge no examples of conic Monte Carlo combined with rough volatility modeling are available in the literature at the time of writing. Therefore, we believe this is a fit-for-purpose choice to illustrate the methodology described in the article at hand. We want to highlight, as already mentioned in section 1, that the methodology proposed in this article is model-agnostic. I.e. if another model were to take the place of the rBergomi model, then the methodology could be applied in a similar manner. Further, the same considerations hold as far as the payoffs are concerned, as well as for the pricing technique chosen (as soon as compatible with the conic finance paradigm). Therefore, the choice made here is neither restrictive nor specific.

In this example, we provide an illustration of the calibration routine described in section 3.2.1. In particular, we follow the modeling framework of Buehler (2010), given that we consider, in our example, equity options on potentially

dividend-paying underlying assets. That is, we start by assuming that a given underlying asset $(Y_t)_{t \geq 0}$ pays (cash) dividends at the dividend dates $\tau_1 < \dots < \tau_D$; we denote these amounts as d_1, \dots, d_D .[†] We denote with D_t the sum $\sum_{j: \tau_j > t} d_j$. From Buehler (2010) it follows, under the same pricing measure used for valuation purposes, that

$$Y_t = (F_t - D_t) \times U_t + D_t, \quad (23)$$

where in (23) $(U_t)_{t \geq 0}$ is a non-negative local martingale (under the same pricing measure) starting at one, and where $F_t := Y_t \times (\sum_{j: 0 < \tau_j \leq t} d_j)$. To evaluate a call option with maturity T one would need to compute the amount

$$C_Y(K, T) = (F_T - D_T) \times C_U\left(\frac{K - D_T}{F_T - D_T}, T\right), \quad (24)$$

where $C_Y(K, T)$ indicates that the option with strike price K and maturity T should be considered as having $(Y_t)_{t \geq 0}$ as underlying. The amount $C_U(\frac{K - D_T}{F_T - D_T}, T)$ should be interpreted as an option on $(U_t)_{t \geq 0}$, with the same maturity as the former but, this time, with an appropriately adjusted strike price.

As far as the rBergomi model (Bayer *et al.* 2016) is concerned, we recall here its functional form. Given relationship (24), the rBergomi model can be then used to describe the dynamics of the process $(U_t)_{t \geq 0}$ ($U_0 = 1$). That is, the set of equations

$$\begin{cases} U_t = \mathcal{E}\left(\int_0^t \sqrt{v_u} dW_u\right) \\ v_u = \sigma \times \mathcal{E}\left(\eta \sqrt{2H} \int_0^u (u-s)^{H-\frac{1}{2}} dZ_s\right) \\ d\langle W_t, Z_t \rangle = \rho dt \end{cases} \quad (25)$$

should be used, with $\sigma > 0$, $\eta > 0$, $H \in (0, \frac{1}{2})$ the Hurst exponent, $\rho \in [-1, 1]$ the correlation between the two Brownian motions $(W_t)_{t \geq 0}$ and $(Z_t)_{t \geq 0}$, and with $\mathcal{E}(\cdot)$ denoting the stochastic exponential. Observe that the second equation in (25) includes a Volterra process, which involves a fractional Brownian motion (Mandelbrot and van Ness 1968). For performing path-wise simulations under rBergomi dynamics (in a risk-neutral environment), a popular choice is that of discretizing the integral related to the Volterra process within the second equation in (25) by means of the hybrid scheme (Bennedsen *et al.* 2017) (see also McCrickerd and Pakkanen 2018 for additional details).

We now present some results with the aim of illustrating how CNNs perform in practice. In particular, we consider (European) equity options maturing in one year on a dividend-paying asset and, therefore, we follow the modeling approach (24). As far as the strike range is concerned, we take into account all the available options with a moneyness level in between the 75% and 125% range (in line with the example provided in section 3.1). The reason for this is given by the fact that, for deeply in-the-money or out-of-the-money options, often only bids or asks are quoted, depending

Table 1. Hyperparameter specifications for neural networks used to produce the results available in this section.

Parameter	Value
Hidden layers	1
Neurons	30
Activation	Sigmoid
Dropout rate	0.0
Batch-normalization	No
Initialization	Xavier uniform
Optimizer	Adam
Learning rate	10^{-3}
Batch size	10^4
Loss	Mean squared error

on whether the market trend is that of shorting or longing specific option positions. Therefore, we focus our analysis on the moneyness area where, given the data available, the observable market information is more reliable. We note, however, that this choice is by no means restrictive, as considering more (or fewer) options would just require feeding the CNN with an extended (shrunk) dataset. Everything else would work exactly the same manner. In order to train the CNN we consider as input ranges those that can be obtained by taking into account the calibrated parameters of the original model to which a buffer of $\pm 100\%$ (relative) is added. For each of the chosen intervals, the CNN is trained by taking into account a different number of points. That is, each CNN has been trained by taking into account, for each parameter range, a fixed number of equidistant points as done in section 3.1 (from here onwards, for simplicity we will use expressions like ‘number of points’ to refer to the fineness of the uniform discretization grid for each model parameter’s range). This is in order to show how, when a sufficiently high number of inputs in the training dataset is considered, the CNN allows to accurately replicate the quantities of interest, either implied or market-observable. In this section, we have taken into account a feed-forward neural network with a single hidden layer with characteristics and specifications as per table 1.

We start by taking into account the replication of the (calibrated) model prices given the input dataset. This is illustrated in figure 4(a–d). In particular, these illustrations show how the CNN, by increasing the number of training inputs, smoothly and accurately replicates the option prices quoted in the market and the respective model prices computed with the (calibrated) conic rBergomi model.

As figure 4(a–d) illustrate, the conic rBergomi model can be accurately replicated by the CNN. In particular, in implied volatility terms, with a 15-point (or finer) grid per model parameter’s range it is possible to replicate bid and ask prices generated by the conic rBergomi model with errors smaller than a basis point. However, by considering even finer grids, this amount can be further decreased.

We now introduce another important concept in conic pricing, i.e. the notion of *implied liquidity* coined in Corcuera *et al.* (2012). To intuitively explain this concept recall that the implied volatility is some sort of forward-looking forecast measure of the expected volatility of a given underlying that can be ‘implied’ from option quotes. That is, it somehow

[†] Recall that, throughout this article, for simplicity and without loss of generality we assume the time value of money to be zero (otherwise, discounted prices could be considered, instead).

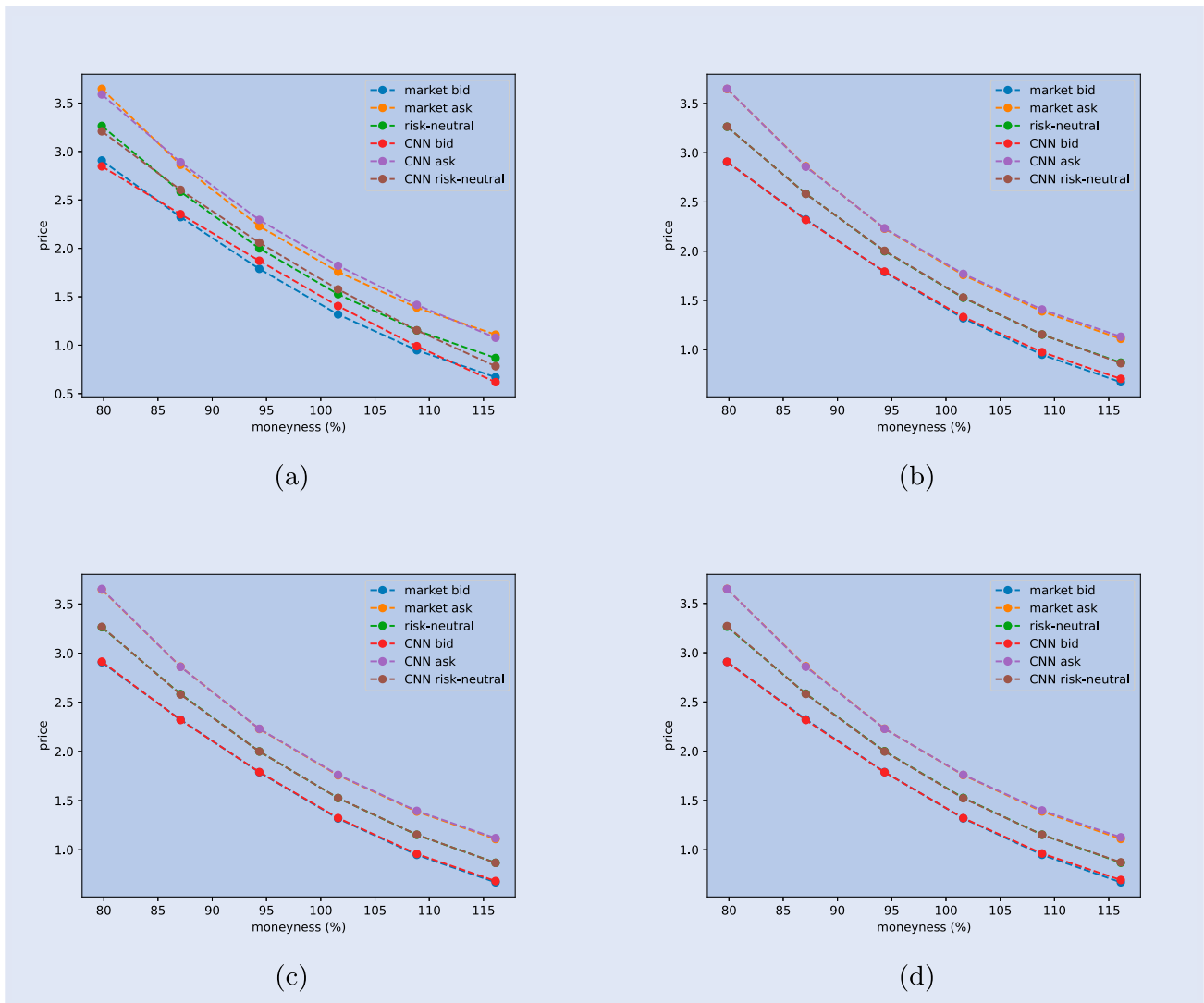


Figure 4. Replication of quoted bid and ask option prices, as well as their risk-neutral counterparts, using the conic rBergomi model and its CNN extension. Risk-neutral prices have been calculated by considering $\gamma = 0$. Panel (a) corresponds to a CNN trained given an input dataset where each model parameter's range has been discretized into a uniform grid with 5 evenly spaced points, while in panels (b), (c) and (d) this number has been increased to 10, 15 and 20, respectively.

represents and quantifies the market's overall expectation of the variability of the underlying within a given horizon. Similarly, the implied liquidity (also known as *implied distortion* given its definition in mathematical terms, see (20) and (22)) for a given option represents, assuming market agents trade options according to the conic finance paradigm (Cherny and Madan 2010), the implicit willingness to trade a given contract. This amount is, therefore, essential as far as hedging considerations are concerned and therefore must be taken into account in markets with frictions. Graphical depictions of how implied liquidity parameters for different strikes can be replicated using the CNN are available in figures from 5(a) to 6(b).

As figures 5(a) to 6(b) illustrate, the implied liquidity parameters produced by the conic rBergomi model can be accurately replicated by their CNN counterparts, in a similar manner it was observed in the case of (bid, ask and risk-neutral) option prices as per figure 4(a–d). That is, as soon as enough input points are considered in the training set, implied liquidity parameters can be replicated with very

good accuracy. In the case of a 20-point grid, implied liquidity parameters between the conic rBergomi model and its corresponding CNN differ for less than one basis point.

We recall here the definitions of mean squared error (MSE), of root mean squared error (RMSE), and of mean average percentage error (MAPE). These error metrics will be used, from here onwards, throughout this article. That is, given a set of J data points y_1, \dots, y_J and their corresponding predictions $\hat{y}_1, \dots, \hat{y}_J$, we recall the following definitions:

- $\text{MSE} := \frac{1}{J} \sum_{j=1}^J (y_j - \hat{y}_j)^2$;
- $\text{RMSE} := \sqrt{\text{MSE}}$;
- $\text{MAPE} := \frac{100}{J} \sum_{j=1}^J \left| \frac{y_j - \hat{y}_j}{y_j} \right|$.

The parameters specifying the CNN used for the tests performed in the current section are based on experimentation, and we now want to briefly comment on the choice of the activation function we made to support our choice (for a study concerning the comparison of activation functions in (deep) neural networks we refer the reader, e.g. to Szandała 2021).

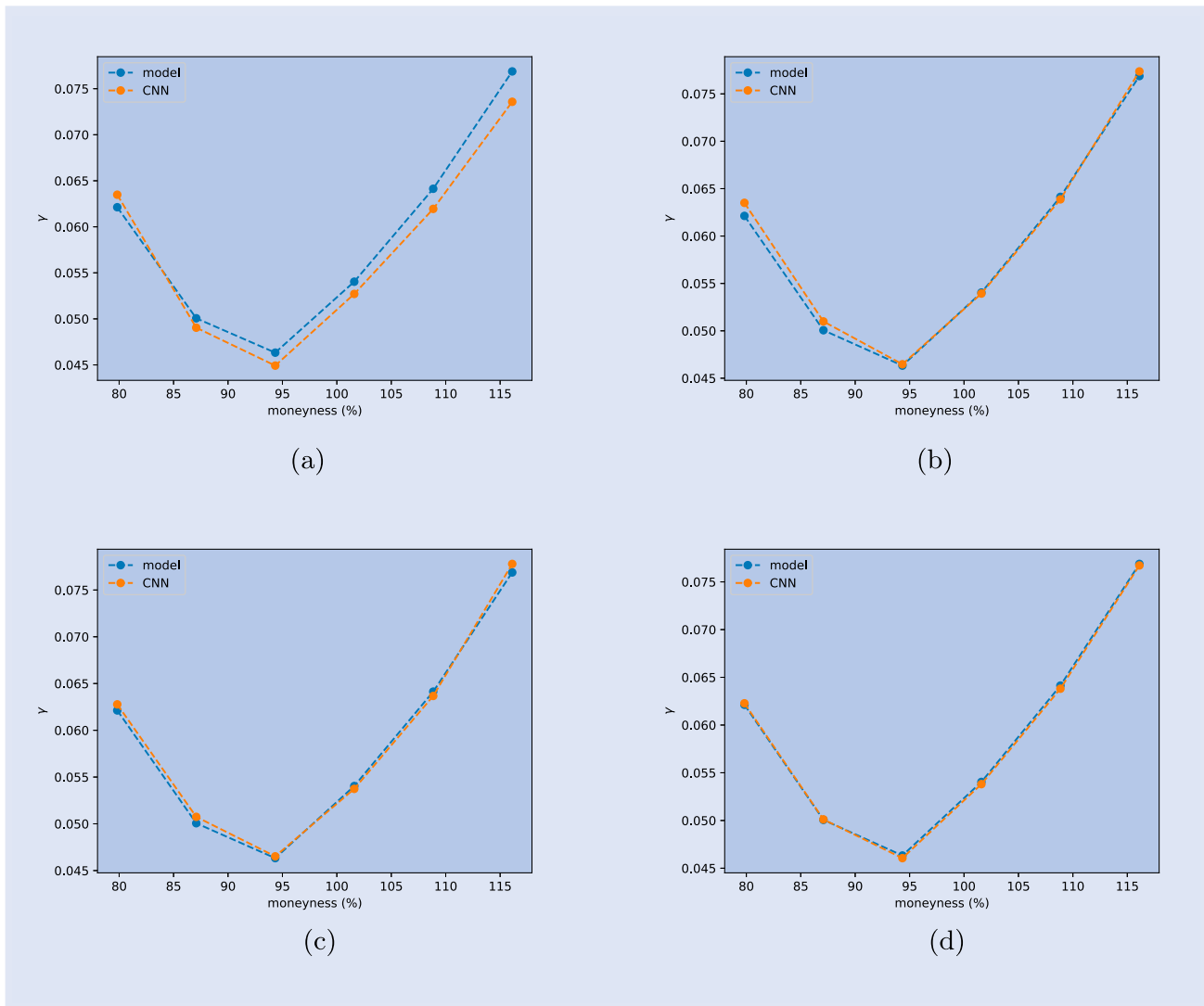


Figure 5. Replication of implied liquidity amounts computed using the conic rBergomi model by means of its CNN extension. Panel (a) corresponds to a CNN trained on an input dataset where each model parameter’s range has been discretized into a uniform grid with 5 evenly spaced points, while in panels (b), (c) and (d) this number has been increased to 10, 15 and 20, respectively.

For this purpose, given the generated training datasets computed using 5, 10, 15 and 20 points per parameter we compute, for different activation functions, the MSE resulting from the training of the CNN. We report the results in table 2 we provide a detailed quantification of the MSEs resulting from the training of the CNN.

What can be easily observed from the numbers provided in table 2 is that all the choices considered here result in an MSE of the order of 10^{-4} or 10^{-5} , except for the ReLU activation function, which has an error up to two orders of magnitude larger.

However, in the context of option pricing, models depend on a series of input parameters which are, often, positive. For instance, in the case of the rBergomi model (25) all the parameters except the correlation between the two model Brownian motions are positive. Furthermore, option premia are positive quantities and, besides, Choquet integrals are highly nonlinear functionals. Therefore, in order to obtain good model replication, the neural network considered should be ‘significantly nonlinear’ in the input parameters. Given however that model and market parameters are, in this situation, mainly

non-negative, one would expect the ReLU function not to perform as well as in other applications. For completeness, we have calibrated the CNN using the ReLU function as a neuron activation function and provided the sub-optimal outcomes in figure 7(a,b).

By using the Sigmoid activation function and 20 input points for bid (ask) prices we obtain a calibration error of $5.88 \cdot 10^{-5}$ ($4.26 \cdot 10^{-5}$), $7.67 \cdot 10^{-3}$ ($6.52 \cdot 10^{-3}$), and 3.02×10^{-2} (1.65×10^{-2}) in terms of MSE, RMSE, and MAPE, respectively. The corresponding calibrated parameters are $\sigma = 0.26$, $\nu = 0.11$, $H = -0.17$, and $\rho = 0.21$.

3.2.3. Performance analysis. In this section, we provide some details concerning the time performance of a CNN compared to conic Monte Carlo given the examples considered in section 3.2.2. All the calculations and measurements presented here have been performed using a laptop equipped with an Intel® Core™ i7-13620H processor (2.40GHz CPU), with 10 (16) physical (logical) cores, 32GB RAM, a 500GB hard drive, and with an NVIDIA RTX 4050 Laptop GPU. We report

Table 2. MSEs as a function of the number of points in the uniform discretization grid of each model parameter’s range.

Act. Fun.	# points			
	5	10	15	20
Exponential	$1.71 \cdot 10^{-1}$	$1.96 \cdot 10^{-2}$	$1.11 \cdot 10^{-2}$	$4.28 \cdot 10^{-4}$
GeLU	$1.60 \cdot 10^{-2}$	$1.80 \cdot 10^{-2}$	$3.61 \cdot 10^{-4}$	$2.23 \cdot 10^{-4}$
ReLU	$8.47 \cdot 10^{-2}$	$5.50 \cdot 10^{-2}$	$4.90 \cdot 10^{-2}$	$4.63 \cdot 10^{-2}$
Sigmoid	$1.79 \cdot 10^{-2}$	$1.20 \cdot 10^{-3}$	$1.16 \cdot 10^{-4}$	$2.75 \cdot 10^{-5}$
Softplus	$2.89 \cdot 10^{-2}$	$2.50 \cdot 10^{-3}$	$3.54 \cdot 10^{-4}$	$2.09 \cdot 10^{-4}$
Softsign	$2.32 \cdot 10^{-2}$	$1.10 \cdot 10^{-3}$	$1.42 \cdot 10^{-4}$	$8.52 \cdot 10^{-5}$
Swish	$3.84 \cdot 10^{-2}$	$2.50 \cdot 10^{-3}$	$7.33 \cdot 10^{-4}$	$3.00 \cdot 10^{-4}$
Tanh	$2.05 \cdot 10^{-2}$	$1.50 \cdot 10^{-2}$	$1.73 \cdot 10^{-4}$	$9.45 \cdot 10^{-5}$

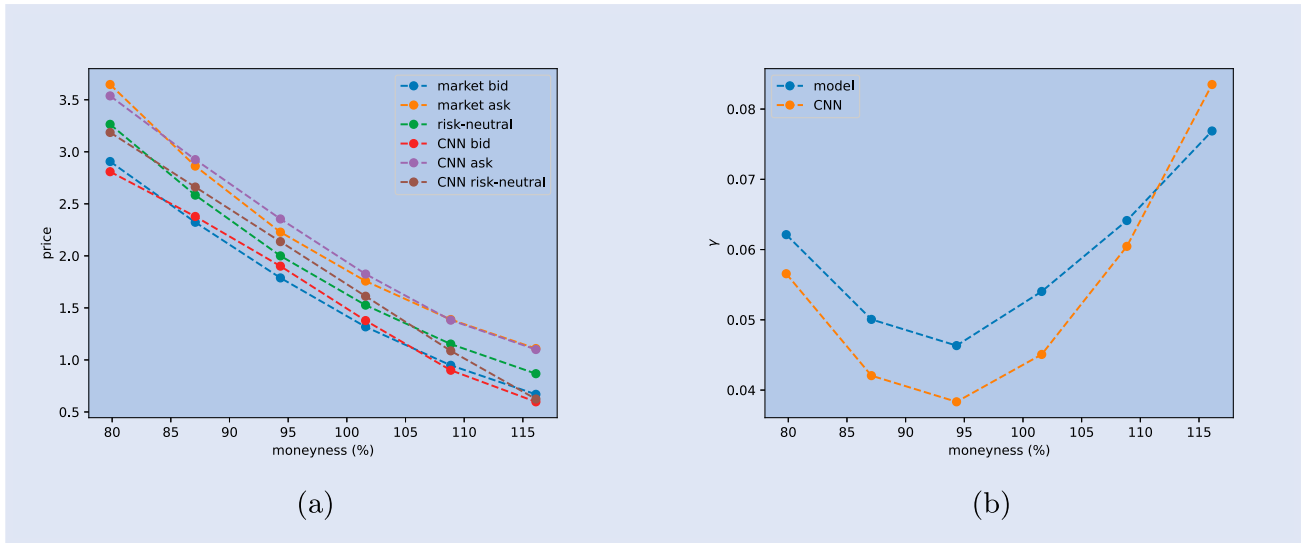


Figure 6. Replication of model prices, panel (a), and implied liquidity amounts, panel (b), by means of a CNN with ReLU activation function. In both cases the CNN has been trained given an input dataset where each model parameter’s range has been discretized into a uniform grid with 20 evenly spaced points.

Table 3. Average time to generate the training dataset and train the CNN of section 3.2.2 based on 10 runs and 10,000 simulations.

# points	Dataset generation time (hours)	Training time (seconds)
5	$1.28 \cdot 10^{-2}$	21.2
10	$2.79 \cdot 10^{-1}$	41.6
15	2.07	95.9
20	8.75	148.2

Note: The number of points refers to the fineness of the uniform discretization grid of each model parameter’s range.

in table 3 the (average) amounts of time necessary for generating the dataset and for calibrating the CNN. The generation of the training datasets has been implemented in a parallelized manner taking advantage of the multi-processing capabilities of the machine.

We now report some performance metrics, from a pricing perspective, in figure 8. We note that the advantage of performing valuation using a CNN is that, once calibrated, it provides constant computational time, independent of the maturity (approx. 16 milliseconds). Also, the time remains (almost) constant when sensitivities are computed, while this

would require double (triple) the effort in the case the latter are calculated by forward (central) finite difference approximations using (conic) Monte Carlo.

3.2.4. A no-arbitrage test. The model we consider as base risk-neutral one in section 3.2.2 (i.e. the rBergomi model described in (25)), as well as its conic extension, are by construction arbitrage-free. Given three strikes $K_L < K_M < K_H$, one could buy a call option struck at K_L , sell $\frac{K_H - K_L}{K_H - K_M}$ calls with strike K_M , and again buy $\frac{K_M - K_L}{K_H - K_M}$ calls with strike K_H (we assume here that the maturity T is fixed). One would then obtain (see Madan and Schoutens 2016, Sec.1.7.2) that inequality (26) would hold:

$$\frac{C(K_L, T) - C(K_M, T)}{K_M - K_L} \geq \frac{C(K_M, T) - C(K_H, T)}{K_H - K_M}. \quad (26)$$

However, as in practice risk-neutral prices are not tradable (an, thus, not observable in the market), one would need to consider a different version of (26) where the risk-neutral prices of the bought (sold) call options are then replaced by

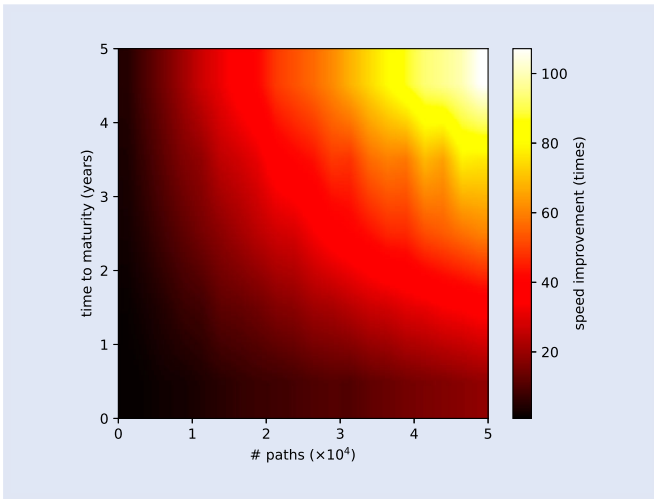


Figure 7. Average speed enhancement when pricing European options (one to five years to maturity) while approximating the rBergomi model by means of a CNN. Lighter (darker) colors correspond to higher (lower) speed improvements. Numbers are based on ten runs.

their ask (bid) counterparts, which reads

$$\frac{\text{ask}(\mathcal{C}(K_L, T)) - \text{bid}(\mathcal{C}(K_M, T))}{K_M - K_L} \geq \frac{\text{bid}(\mathcal{C}(K_M, T)) - \text{ask}(\mathcal{C}(K_H, T))}{K_H - K_M}. \quad (27)$$

For puts, similar versions of inequalities (26) and (27) hold. In particular, if put options are considered, (26) and (29) would read

$$\frac{\mathcal{P}(K_H, T) - \mathcal{P}(K_M, T)}{K_H - K_M} \geq \frac{\mathcal{P}(K_M, T) - \mathcal{P}(K_L, T)}{K_M - K_L} \quad (28)$$

and

$$\frac{\text{ask}(\mathcal{P}(K_H, T)) - \text{bid}(\mathcal{P}(K_M, T))}{K_H - K_M} \geq \frac{\text{bid}(\mathcal{P}(K_M, T)) - \text{ask}(\mathcal{P}(K_L, T))}{K_M - K_L}, \quad (29)$$

respectively. By taking into account the rBergomi model, one expects to obtain that, for any choice of the three strikes K_L , K_M , and K_H , inequalities (26) and (28) would hold at risk-neutral level. Furthermore, by computing bid and ask prices by means of distorted risk-neutral prices at a given level of γ , inequalities (27) and (29) should also hold. The question that it is natural to attempt to answer is therefore whether, if risk-neutral, bid and ask prices are approximated by means of a CNN, inequalities (26), (27), (28) and (29) would be still valid.

We therefore perform the following test after having taken into account the calibrated neural networks of section 3.2.2 for the rBergomi model based on 5, 10, 15 and 20 grid points per input parameter (see figure 4(a–d), respectively). Given that in the calibration step we have taken into account the 75%–125% moneyness level, we consider a grid (in the moneyness dimension) five times finer for each case (still given by

Table 4. In-sample bid calibration MSEs, RMSEs, and MAPEs given the example of in section 3.2.2.

Error	# points			
	5	10	15	20
MSE	$2.06 \cdot 10^{-2}$	$1.10 \cdot 10^{-3}$	$2.13 \cdot 10^{-4}$	$1.14 \cdot 10^{-4}$
RMSE	$1.44 \cdot 10^{-1}$	$3.31 \cdot 10^{-2}$	$1.46 \cdot 10^{-2}$	$1.07 \cdot 10^{-2}$
MAPE	$1.15 \cdot 10^{-1}$	$2.45 \cdot 10^{-2}$	$1.07 \cdot 10^{-2}$	$8.13 \cdot 10^{-3}$

Note: The number of points refers to the fineness of the uniform discretization grid of each model parameter's range.

equidistant points). That is, we consider a grid of 25 points in the first case, of 50 in the second case, of 75 in the third and of 100 in the last case. We fix the distortion parameter γ to 0.05 for testing purposes (note that this value is reasonable given that it belongs to the range of the implied distortions observed in figures 5(a,b) and 6(a,b)). We first count, for each calibrated CNN and for each of the correspondent grids constructed, the number of violations of inequality (26). The number of violations at the risk-neutral level equals to two in the case of the 25-point grid, while no violations are observed for the case the considered testing grid contains 50, 75 or 100 points. Afterwards, we perform a similar exercise and count the number of violations of inequality (27) at bid-ask level. We observe one violation in the case 25 points are considered, while zero violations for finer grids. Furthermore, in order to test the number of violations for put options we construct a synthetic dataset as follows. We interpolate linearly the (relative with respect to the mid-price) bid-ask spreads of the call options used in the calibration step and, for each put option with a moneyness level of $1 + m$ ($m \in [-0.25, 0.25]$) we apply the bid-ask spread corresponding to the call option with moneyness equal to $1 - m$. We then re-perform the calibration exercise (this time with respect to the synthetic put options) and construct four auxiliary grids the same way we did in the case of the call options. Finally, we again count the number of violations of inequality (28) at the risk-neutral level and of (29) in a two-price economy. We observe three violations in the first case (both at risk-neutral and bid-ask levels) when a grid of 25 points is considered, and no violations for finer (calibration) grids. This illustrates that, as soon as the number of input training points is large enough, the CNN approximates both risk-neutral and bid-ask prices well enough not to allow for butterfly spread arbitrages, and this holds both at risk-neutral as well as at the bid-ask level.

3.2.5. Training validation. We consider the error metrics introduced in section 3.2.2 for each of the cases considered in section 3.2.2 (i.e. figure 4(a–d)) i.e. the bid and ask in-sample MSEs, RMSEs and MAPEs the in-sample MSEs, RMSEs and MAPEs of bid-ask prices are used to quantify the error observed after model calibration using the procedure of section 3.2.2. The results are reported in tables 4 and 5.

Similarly to what is proposed in Spiegeleer *et al.* (2018), for each of the parameter ranges considered we take into account uniformly distributed points therein. For example, in the case the CNN was calibrated using 5 points per input parameter,

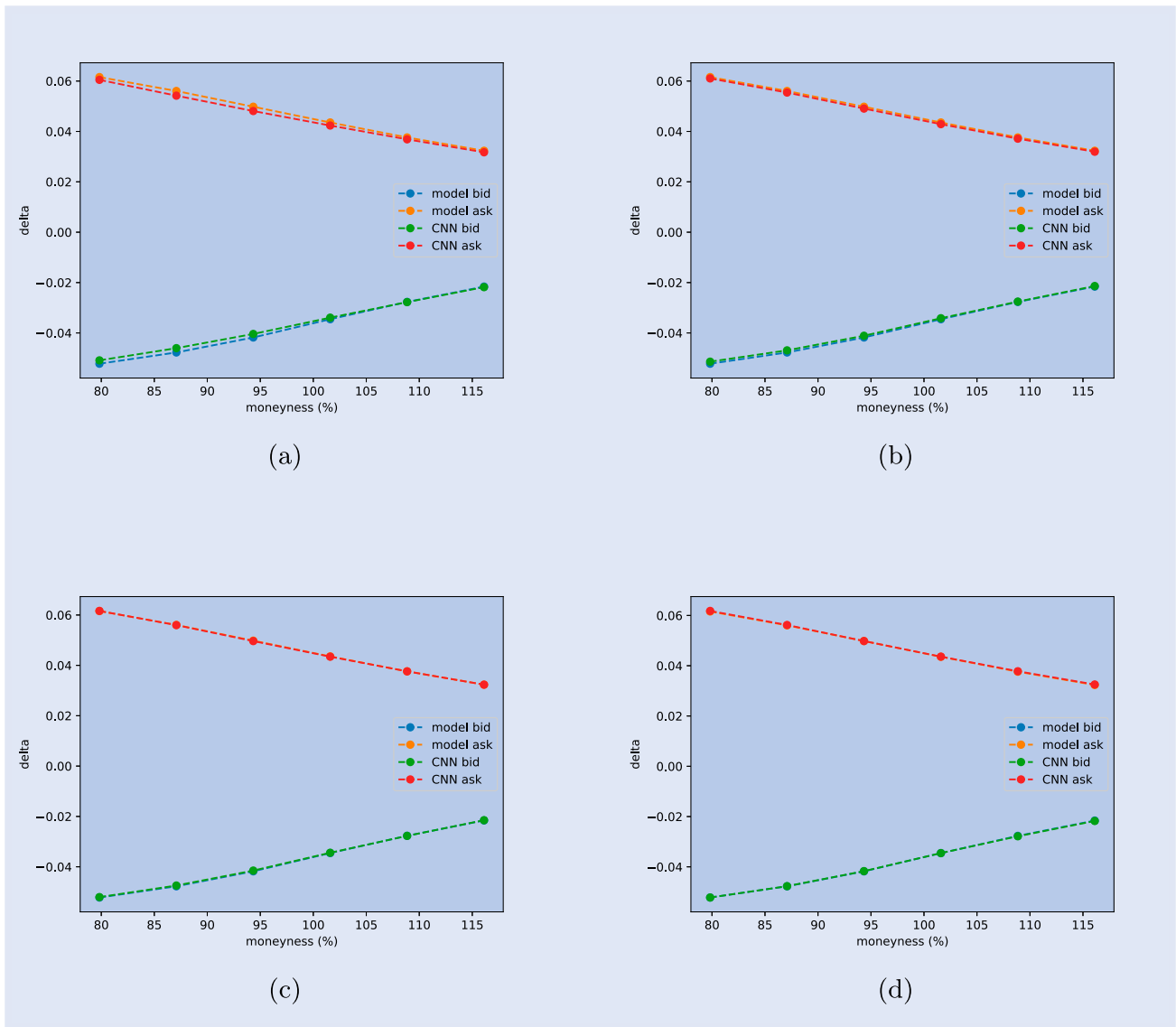


Figure 8. Replication of implied liquidity delta amounts computed using the rBergomi model by means of its CNN extension. Panel (a) corresponds to a CNN trained on an input dataset where each model parameter’s range has been discretized into a uniform grid with 5 evenly spaced points, while in panels (b), (c) and (d) this number has been increased to 10, 15 and 20, respectively. Liquidity delta amounts have been multiplied by 1%, in line with what is often done in risk monitoring, e.g. for vega calculations.

Table 5. In-sample ask calibration MSEs, RMSEs, and MAPEs given the example of section 3.2.2.

Error	# points			
	5	10	15	20
MSE	$2.07 \cdot 10^{-2}$	$2.03 \cdot 10^{-3}$	$2.16 \cdot 10^{-4}$	$1.06 \cdot 10^{-4}$
RMSE	$1.44 \cdot 10^{-1}$	$4.51 \cdot 10^{-2}$	$1.47 \cdot 10^{-2}$	$1.03 \cdot 10^{-2}$
MAPE	$1.16 \cdot 10^{-1}$	$3.36 \cdot 10^{-2}$	$1.09 \cdot 10^{-2}$	$7.68 \cdot 10^{-3}$

Note: The number of points refers to the fineness of the uniform discretization grid of each model parameter’s range.

Table 6. Out-of-sample bid calibration MSEs, RMSEs, and MAPEs given the example of section 3.2.2.

Error	# points			
	5	10	15	20
MSE	$7.37 \cdot 10^{-3}$	$7.49 \cdot 10^{-4}$	$2.86 \cdot 10^{-4}$	$1.45 \cdot 10^{-4}$
RMSE	$8.59 \cdot 10^{-2}$	$2.74 \cdot 10^{-2}$	$1.69 \cdot 10^{-2}$	$1.20 \cdot 10^{-2}$
MAPE	$7.08 \cdot 10^{-2}$	$2.11 \cdot 10^{-2}$	$1.16 \cdot 10^{-2}$	$8.83 \cdot 10^{-3}$

Note: The number of points refers to the fineness of the uniform discretization grid of each model parameter’s range.

we randomly generate the same amount of points per input parameter and calculate the MSE, RMSE and MAPE for both bid and ask prices. The procedure is repeated for the cases with 10, 15 and 20 points per model parameter’s range. We report the results in tables 6 and 7, respectively.

By comparing the error metrics considered in this section for in- (tables 4 and 5) and out-of-sample (tables 6 and 7)

CNN values we observe that in-sample and out-of-sample errors are, overall, of the same magnitude, with out-of-sample errors in some cases being smaller than their in-sample counterparts. This phenomenon is also observed in Spiegeleer *et al.* (2018). From the results it is apparent that, in relative terms (i.e. in APE terms), with 10 input points per parameter, average in-sample and out-of-sample errors are between

Table 7. Out-of-sample ask calibration MSEs, RMSEs, and MAPEs given the example of section 3.2.2.

Error	# points			
	5	10	15	20
MSE	$1.01 \cdot 10^{-2}$	$9.51 \cdot 10^{-4}$	$2.73 \cdot 10^{-4}$	$1.33 \cdot 10^{-4}$
RMSE	$1.01 \cdot 10^{-1}$	$3.08 \cdot 10^{-2}$	$1.65 \cdot 10^{-2}$	$1.15 \cdot 10^{-2}$
MAPE	$8.31 \cdot 10^{-2}$	$2.35 \cdot 10^{-2}$	$1.14 \cdot 10^{-2}$	$8.44 \cdot 10^{-3}$

Note: The number of points refers to the fineness of the uniform discretization grid of each model parameter's range.

two and three basis points, which further decrease to one basis point when 15 input points per parameter are considered. With 20 input points, all relative errors observed (both in- and out-of-sample) are below one basis point, indicating the high accuracy and robustness of the calibrated CNN.

3.3. Sensitivities: learning (the) Greek(s)

We now discuss how to calculate Greeks using CNNs. That is, the approach provided in this section shows how to calculate Greeks of bid and ask prices by training the CNN, at the same time, at ‘learning the Greeks’ as well. Also here the methodology outlined is general and, again, model-agnostic. In particular, as the research proposed in this article relates to the conic finance paradigm (Cherny and Madan 2010), we will focus our attention on the computations of liquidity Greeks (i.e. liquidity deltas in this case). We have chosen to take into account liquidity delta sensitivities only because, to the best of our knowledge, research developments in liquidity management using distorted expectations are still in an embryonic phase. Therefore, from a practical perspective being able to account for first-order sensitivities in this respect should suffice. We bring to the forefront the fact that not only can this methodology be extended to higher-order liquidity Greeks, but that the same would also for any other (potentially high-order) Greek. This can be easily achieved as soon as the training framework is rich enough. This means that the considerations and the approach described here easily allow to be generalized. However, this would make the notation glutted and saturated, which we aim to avoid for the sake of clear and concise exposition.

In sections 3.1 and 3.2, we illustrated how using CNNs requires approximating, by means in this case of a vector-valued neural network, a function such that (θ, γ, K) is mapped to $(\text{bid}_\gamma(\theta, K), \text{ask}_\gamma(\theta, K))$. However, one might want to extend the aforementioned considerations to the calculations of sensitivities as well. This will be now illustrated. Note that here, for the sake of exposition, we will show how to calculate Greeks of the first order only. The methodology can be easily extended to higher-order sensitivities as well.

Having said that, the steps are the following. Once set

$$\nabla \text{bid}_\gamma(\theta, K) := \left(\frac{\partial \text{bid}_\gamma(\theta, K)}{\partial \theta_1}, \dots, \frac{\partial \text{bid}_\gamma(\theta, K)}{\partial \theta_N}, \frac{\partial \text{bid}_\gamma(\theta, K)}{\partial \gamma} \right)$$

and, similarly,

$$\nabla \text{ask}_\gamma(\theta, K) := \left(\frac{\partial \text{ask}_\gamma(\theta, K)}{\partial \theta_1}, \dots, \frac{\partial \text{ask}_\gamma(\theta, K)}{\partial \theta_N}, \frac{\partial \text{ask}_\gamma(\theta, K)}{\partial \gamma} \right),$$

we consider the map

$$(\theta, \gamma, K) \mapsto (\text{bid}_\gamma(\theta, K), \text{ask}_\gamma(\theta, K), \nabla \text{bid}_\gamma(\theta, K), \nabla \text{ask}_\gamma(\theta, K)). \quad (30)$$

By performing the training of the CNN as illustrated in section 3.2 (in this case the derivatives above can be computed, e.g. by means of finite differences) one can then obtain a CNN that also allows to compute sensitivities, simultaneously. In the case of the (conic) rBergomi model, we provide some examples in figure 9(a–d). As the graphs illustrate, in the case of delta computations a lower number of points per model parameter's range seems enough to produce a satisfactory accuracy in contrast to the case of option prices (compare with figure 4(a–d)) and implied liquidity parameters (compare with figures 5(a) to 6(b)). The reason for this is given by the fact that, contrary to the former two cases (i.e. market values and implied distortions), here we calculate ‘differences in market values’. Therefore, if the valuation of the CNN slightly overestimates (underestimates) a specific quantity, it is natural to expect that these misalignments tend to somehow compensate each other, resulting in acceptable approximation errors for the deltas. This is evident from figure 9(a–d).

For analysis and comparison purposes, we want to further comment on the methodology. In fact, one might observe that the majority of the neuron activation functions are infinitely differentiable. To be precise, note that some activation functions are infinitely differentiable almost everywhere, such as the ReLU or the softsign activation functions (amongst others). That is, these examples of activation functions, which are very popular in practical applications, are only not differentiable at zero. For this reason, therefore, we compare the two approaches. We observe that one could compute liquidity deltas by means of the CNN using a simple finite difference approach. For this reason, to compare the two possible approaches we consider in this case the liquidity delta computed this time using the CNN where a first-order forward finite difference scheme is applied; see figure 10(a,b) for an illustration.

As figure 10(a,b) show, applying a finite difference scheme to the neural network directly does not provide results as accurate as those generated using the methodology provided in this section; see figure 9(a,b). It is clear how performing sensitivity calculations by means of finite differences requires more points in the training set to obtain the desired accuracy. The underlying reason for this phenomenon is given by the fact that if two functions are close, in some sense, their derivatives are not necessarily so. This leads to the differences observed. Therefore, the comparison performed clearly provides supporting evidence to justify the approach proposed.

An additional benefit this approach is the fact that computing Greeks by a finite difference approach requires double

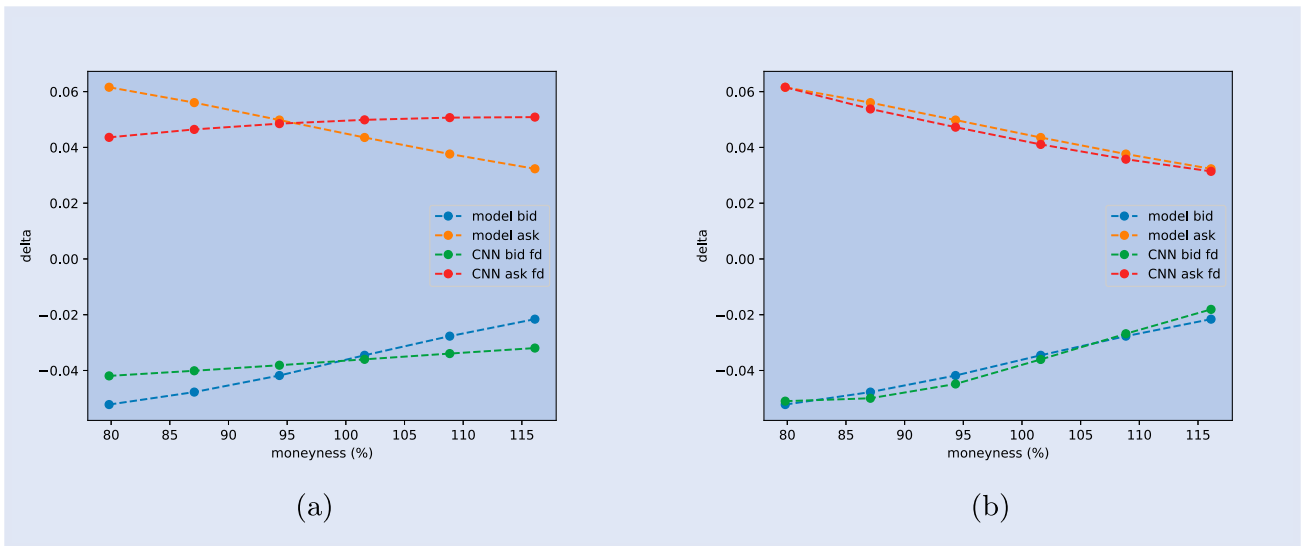


Figure 9. Attempt to replicate liquidity delta amounts computed using the rBergomi model by means of its CNN extension where finite differences are used. Panel (a) corresponds to a CNN trained on an input dataset where each model parameter’s range has been discretized into a uniform grid with 5 evenly spaced points, while in panel (b) this number has been increased to 10. In both panels (a) and (b) the label ‘model bid (ask)’ refers to the liquidity delta amounts computed using the conic rBergomi model, while the label ‘CNN bid (ask) fd’ to their counterparts calculated using the CNN where sensitivities are computed by means of (forward) finite differences. Liquidity delta amounts have been multiplied by 1%, in line with what is often done in risk monitoring, e.g. for vega calculations.

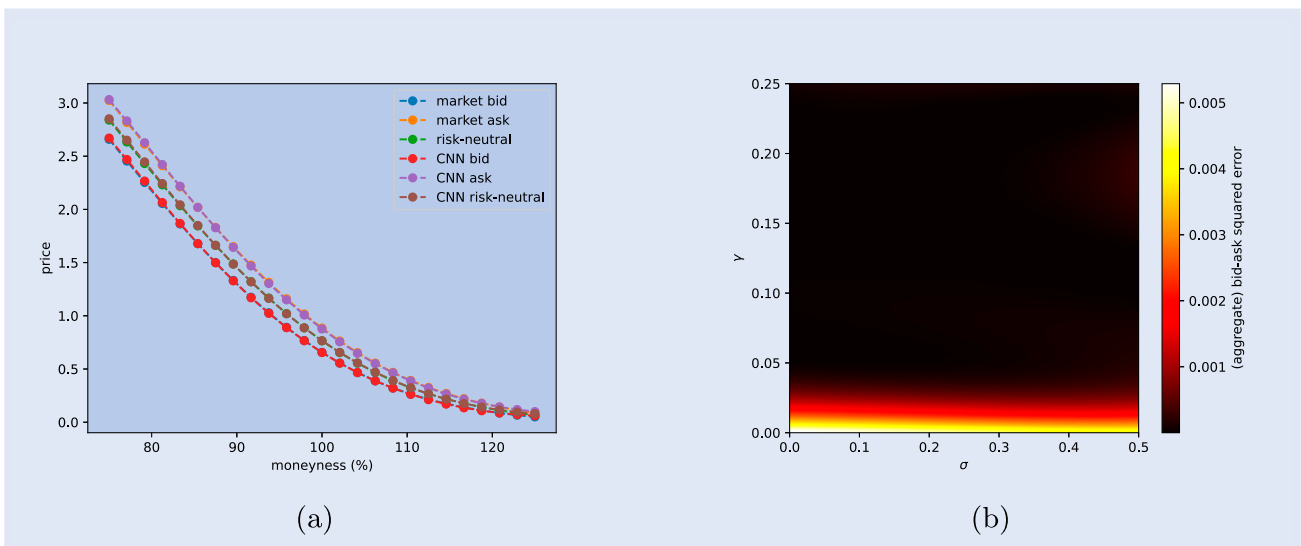


Figure 10. Replication of bid and ask (daily averaged) Asian call option prices, as well as their risk-neutral counterparts, using the Black–Scholes model and its CNN extension, panel (a), and (aggregate) bid-ask MSEs as a function of σ and γ (i.e. out-of-sample MSEs), panel (b). In panel (b), to lighter (darker) colors corresponds higher (lower) errors. Calculations based on 10 000 simulations.

(triple) the time when forward (central) differentiation is used compared to that of pricing an option. In contrast, only a marginal increase in the computational overhead is observed in the case a CNN is used, at the extra of a sensitivity-augmented data set. Notwithstanding, the augmented dataset needs to be generated once only.

We remark that, from a computational perspective, an analogous approach can be adopted. I.e. instead of constructing the multi-output as per (30), one could first train the CNN to bid and ask price, and afterward train a second neural network, to perform sensitivity analysis, by means of considering

the auxiliary map given by

$$(\theta, \gamma, K) \mapsto (\nabla \text{bid}_\gamma(\theta, K), \nabla \text{ask}_\gamma(\theta, K)). \quad (31)$$

In terms of performance, we observe that both approaches require the same amount of time to generate the training set(s). From the tests considered in this paper, we observe that training a single CNN with multi-output as per (30) is between 12% and 16% faster than training two neural networks. However, the training dataset is the same for both procedures and has to be generated independently from the training step

(a least in the case of a fixed grid). Therefore, if it is possible to run the pricing and sensitivity training tasks in parallel, then considering multiple CNNs might be more efficient than considering a single one.

3.4. Some concluding remarks on training CNNs

The numerical results provided in sections 3.2 and 3.3 have been obtained with a consumer-grade personal computing device, whose characteristics and specifications are provided in section 3.2.3. However, the infrastructure needed to facilitate the robust implementation of these techniques in a real-world environment would require (the integration of) massive parallel, concurrent and distributed calculations to achieve optimal performance and scalability (Wang *et al.* 2019, Robey and Zamora 2021, Hwu *et al.* 2022). The CNN approach developed here, as, e.g. Spiegeleer *et al.* (2018), are based on training a machine learning model on a (potentially non-uniform) multidimensional grid set a priori. The objective of this paper does not entail an examination of the hardware architecture needed for implementing CNNs on a larger scale.

Two common techniques employed to enhance the efficiency of neural network training procedures are *data pruning* (Sorscher *et al.* 2022) and *data distillation* (Hinton *et al.* 2010). In particular, the first set of strategies aims to remove ‘redundant’ elements from the dataset(s) utilized for training a given neural network model. Data pruning techniques typically involve the systematic elimination of noisy (or less informative) data elements while aiming to preserve the essential characteristics necessary for effective model learning and inference; for a survey refer to Sachdev and McAuley (2010). On the other hand, data distillation methodologies strive to generate concise yet comprehensive representations of datasets by extracting crucial information. These distilled representations are meticulously optimized to function as efficient substitutes for the original datasets in various data-driven applications, including model training, inference, and architecture exploration. For instance, knowledge distillation, a widely adopted technique, aims to transfer knowledge from a larger neural network, often referred to as the teacher network, to a smaller one, termed the student network. This process aims to enhance the performance and accuracy of the smaller model by leveraging the guidance provided by the larger network; see Vadera and Ameen (2021) for a survey.

However, these two classes of techniques are applicable up to a certain extent in the framework we have defined. This is because in our case the training dataset is not given but, instead, needs to be generated by means of a chosen pricing model. Therefore, instead of generating a full dataset as a starting point and performing selection operations on it, intuitively one might want to investigate the possibility of performing iterative training on a CNN, instead, and to stop the training routine when a certain criterion is met. In Okanovic *et al.* (2023) a novel approach is proposed, where subsets of training data are randomly sampled for each epoch of model training procedure. Inspired by this, we provide a methodology that can be considered when calibrating a CNN in real-world circumstances. We also provide an example, based

on the model of section 3.1, of how this methodology would potentially work.

In Okanovic *et al.* (2023), using notation and terminology that are appropriate for the use case considered here, N (potentially multidimensional) input data points x_1, \dots, x_N and their corresponding (potentially multidimensional) ground-truth outputs y_1, \dots, y_N are considered. In our case, the x_i 's represent model inputs and parameters, while the y_i 's bid and ask model prices. In Okanovic *et al.* (2023) it is proposed to randomly sample (with or without replacement) a certain subset of size smaller than N from $\{(x_i, y_i)\}_{i=1}^N$, and to iteratively train a neural network for a given number of rounds set a priori. This is done instead of performing the training of the neural network on $\mathcal{X} := \{(x_i, y_i)\}_{i=1}^N$ in a single pass directly. As it is clear from our CNN framework, adopting the aforementioned approach as is would first require to generate a full training dataset, and then to proceed with batch training. Conversely, one could generate small training batches on-the-fly, to be then fed to the in-training CNN. The process could then can be stopped once a given condition is met, rather than relying on a pre-specified number of training iterations. Therefore, we propose now a simple variation of the aforementioned approach that better fits our purpose. In particular, we denote with *batch_size* the number of training points to be fed to the CNN at each training step, with *max_iter* the maximum number of training iterations to be performed (to avoid endless training in the case of poor learning convergence), and with *iter* the index representing the training iteration step.

The idea is that of generating, at each training step, a random batch of size *batch_size* and to calculate the training error on that batch before feeding the former to the CNN. The choice of calculating the error given a batch before using it for training has been made to avoid the neural network computing errors on data already used for training (i.e. we are basically calculating out-of-sample errors instead of in-sample ones). If, over a defined sequence of consecutive training iterations *max_count* the training error consistently remains below a predetermined threshold or tolerance level *tol*, it may be deemed appropriate to conclude the training process of the CNN. The outlined methodology is encapsulated in Algorithm 1 through the use of pseudo-code notation.

We now provide an example of how Algorithm 1 works in practice. For the ease of implementation and data visualization, we take into account the same framework as considered in section 3.1. This choice has been made as having two unknown model parameters only (i.e. the volatility σ and the distortion parameter γ) facilitates clear graphical depiction and enhances the interpretability of the findings. We also want to point out that in the results coming from the analysis provided in section 3.2, no special computational architecture was needed, as the training set(s) could be easily generated a priori. However, should the scope of this study extend to larger-scale implementations, the adoption of distributed training setups with asynchronous data loading from multiple data generation processes would become imperative. This would be a consequence of computing large batches of bid and ask prices on-the-fly as required by Algorithm 1, which would considerably slow down the training process given the specifications of the machine used in section 3.2.

input : $batch_size, max_count, max_iter, tol$;
output: trained CNN;

```

1 initialize  $CNN_0$ ;
2  $iter \leftarrow 1$ ;
3  $count \leftarrow 0$ ;
4 while  $iter \leq max\_iter$  do
5    $\mathcal{X}_{iter} \leftarrow$  generate random batch of size  $batch\_size$ ;
6    $\epsilon_{iter} \leftarrow$  training error of  $CNN_{iter-1}$  given  $\mathcal{X}_{iter}$ ;
7   if  $\epsilon_{iter} < tol$  then
8      $count \leftarrow count + 1$ ;
9   else
10     $count \leftarrow 0$ ;
11  end
12  if  $count = max\_count$  then
13    break;
14  end
15   $CNN_{iter} \leftarrow CNN_{iter-1}$  trained on  $\mathcal{X}_{iter}$ ;
16   $iter \leftarrow iter + 1$ ;
17 end
18 return  $CNN_{iter}$ ;

```

Algorithm 1: Pseudo-code that can be used to train a CNN on random batches.

A detailed discussion of these software engineering techniques that would be needed to upscale techniques such as Algorithm 1 falls beyond the purview of this paper.

Taking into account the same Black–Scholes framework of section 3.1 on a given underlying asset with current value $Y_0 = 10$, we assume the volatility parameter to range within the interval $[0, 50\%]$ while the distortion parameter (of the Wang transform) within $[0, 25\%]$. We assume that the range of possible strike prices is $[7, 13]$. We consider an arithmetic Asian call option with daily averaging and having time to maturity $T = 1$ and use Monte Carlo simulations. We select the batch size to be equal to 125 (i.e. 5 points per parameter), consider a tolerance error equal to $5 \cdot 10^{-3}$ (i.e. half a cent), a constant learning rate of 10^{-4} , and set the convergence counter to 10^2 . After having run Algorithm 1, we depict some illustrative results in figure 11(a,b).

In particular, in figure 11(a) we have depicted, assuming the ‘real’ volatility σ and distortion parameter γ equal 25% and 12.5%, respectively, how the CNN is able to reproduce bid and ask prices, as well as their risk-neutral counterparts. Moreover, in figure 11(b) we show, after having set the strike price equal to Y_0 to reduce the problem to a two-dimensional one, the (aggregate) bid-ask square error given a deterministic grid, as a function of σ and γ (i.e. out-of-sample error). As expected, overall errors are below the threshold we set and, in particular, they are slightly higher in proximity of $\gamma = 0$ (i.e. risk-neutral area). This is because there bid and ask prices in principle coincide, leading to negligible errors by construction. Therefore the CNN tends to ‘prioritize’ error minimization outside that neighborhood. Note that, as expected, as we generate a new grid (not used for training) to calculate bid-ask squared errors between the CNN and the Monte Carlo model, the maximum error observed slightly exceeds the threshold set a priori by a marginal quantity. This

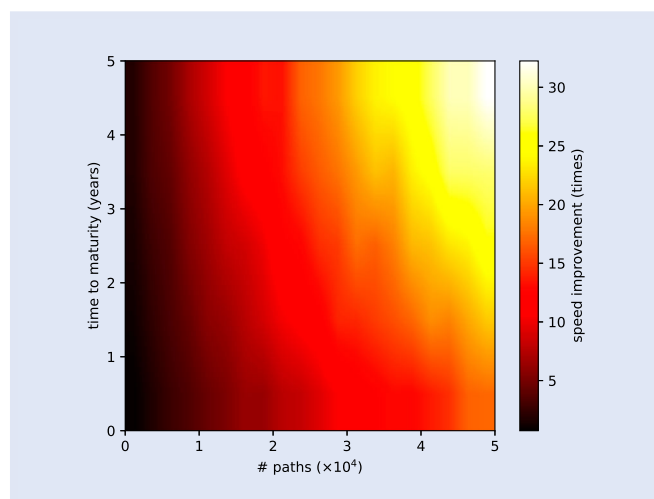


Figure 11. Average speed enhancement when pricing European Asian options (one to five years to maturity) while approximating the Black–Scholes model by means of a CNN. Lighter (darker) colors corresponds to higher (lower) speed improvements. Numbers are based on ten runs.

is expected as figure 11(b) depicts post-training out-of-sample errors.

For completeness, we report the speed improvements achieved by pricing bid and ask Asian call options (with daily averaging) using a CNN rather than Monte Carlo simulations under Black–Scholes assumptions; see figure 12. Even if the model chosen is (one of the) simplest possible, performance improvements are still noticeable.

4. Neural network-driven LSV calibration in a two-price economy

In financial mathematics LSV models have become increasingly popular to capture the complex dynamics of financial markets. LSV modeling provides an hybrid framework allowing to embed the advantages of both local and stochastic volatility, as the name obviously suggests.

Local volatility modeling makes it possible to replicate vanilla option prices by assuming that the volatility of the underlying asset is a deterministic function of both strike and time to expiry; see Dupire (1994). However, practical evidence suggests that local volatility models produce forward skews which are mostly flat. This, therefore, might result in price misestimation for some financial securities. On the other hand, stochastic volatility models, see, e.g. Heston (1993), form a class of financial models attempting to capture the erratic nature of the volatility. This is achieved by supposing that the volatility is a random process itself, varying over time. However, this does not guarantee that pure stochastic volatility models are always able to reproduce the term structure of the volatility smile accurately enough. It is worth pointing out that, despite many stochastic volatility models allow to closely replicate quoted vanilla prices in an almost indistinguishable manner, they might result in very different prices should exotic derivatives be taken into account (see Schoutens

et al. 2004). This can therefore be a non-negligible source of model risk. By combining the two aforementioned approaches at the same time, LSV models allow for a more accurate representation of forward smile risk compared to their local volatility counterparts; see Mazzon and Pascucci (2016). LSV models have been successfully used in a wide range of applications, including option pricing, hedging, and risk management. They have been particularly useful in pricing exotic options, which are often difficult to be valued accurately using more simplistic models. For this reason, LSV modeling has found applications in different asset classes such as equities and foreign exchange, to name but two.

The popularity of LSV models in the financial industry, as well as the continuously-growing interest of researchers in applications of machine learning for quantitative analysis purposes, have driven research aiming to improve volatility modeling by means of neural networks. However, to the best of our knowledge, at the time of writing, no literature investigating the applications of neural networks within LSV modeling in markets with frictions in the sense of Cherny and Madan (2010) seems to be available. Therefore, this section aims to investigate this new research direction. In particular, we consider the approach of Cuchiero et al. (2020), where it is proposed to approximate the local volatility component of a LSV model by means of a (combination of) neural network(s) and illustrate, by means of numerical examples, the benefits coming from this approach in combination with conic Monte Carlo. We recall that, in this paper, for simplicity and without loss of generality, the risk-free rate is assumed to be zero. Also, underlying prices will be always interpreted as cleansed from dividends, and all the calculations will be intended under a given pricing (i.e. risk-neutral) measure \mathbb{Q} .

4.1. Leverage function: a neural network approximation

A LSV model for an underlying process $(Y_t)_{t \geq 0}$ can be defined by means of

$$dY_t = \sigma_t L(Y_t, t) Y_t dW_t, \quad (32)$$

with $(\sigma_t)_{t \geq 0}$ a stochastic process for the volatility, and with the factor $L(\cdot, \cdot)$, often named *leverage function*, aiming to represent the local volatility component of the model. It can be shown, see Guyon and Henry-Labordère (2012), that in order to match quoted (European) call option prices, the leverage function needs to satisfy the relationship

$$L(k, t) = \frac{\sigma^{\text{Dupire}}(k, t)}{\sqrt{\mathbb{E}^{\mathbb{Q}}(\sigma_t^2 | Y_t = k)}}, \quad (33)$$

with $\sigma^{\text{Dupire}}(k, t)$ the *Dupire's local volatility function* (Dupire 1994) evaluated at the point (k, t) given by

$$\sigma^{\text{Dupire}}(k, t) := \sqrt{\frac{\frac{\partial \mathcal{C}(k, t)}{\partial t}}{\frac{1}{2} k^2 \frac{\partial^2 \mathcal{C}(k, t)}{\partial k^2}}}, \quad (34)$$

where in (34) $\mathcal{C}(k, t)$ denotes the call price of an European option with strike k expiring in t years. Equation (34) can also be expressed in terms of implied volatilities rather than option

prices. This is a convenient feature as, often, implied volatilities are used as quoting conventions. This expression can be found, for instance, in Gatheral (2006, Sec. 1).

Despite (33) provides an expression for the leverage function under LSV dynamics as per (32), in practice dealing with such an expression is a non-trivial task for a number of reasons. For instance, inside (33) the Dupire's formula (34) appears. So, unless one is able to compute the derivatives in (34) exactly, as it would be the case should algorithmic differentiation techniques be used (see Henrard 2017 for an overview), then the choice of assuming a parametric form for (34) is often the one followed in practice as proposed, e.g. in Carmona and Nadtochiy (2009). The reason why this is, in many cases, the preferred choice, is that the Dupire's formula is very sensitive to the values of the second derivative at the denominator in (34) which, due to numerical approximations, can cause severe pricing inaccuracies. Also, adopting an a-priori fully-parametric approach implies additional assumptions and model choices in the framework. Furthermore, we also highlight that the denominator in (33) cannot, in general, be computed analytically, leading to further undesirable numerical instabilities.

The approach provided in Cuchiero et al. (2020) introduces a novel idea as far as LSV modeling is concerned. That is, consider M maturities for vanilla call options on a given underlying $(Y_t)_{t \geq 0}$ denoted as $T_1 < \dots < T_M$ (with the convention that $T_0 := 0$) and assume that the leverage function (33) is of the form

$$L(k, t, \theta) := 1 + \sum_{i=1}^M \text{NN}_i(k, \theta_i) \cdot \mathbb{1}_{[T_{i-1}, T_i)}(t) \quad (35)$$

where, for every $i \in \{1, \dots, M\}$, $\text{NN}_i(\cdot, \cdot)$ denotes a family of feed-forward neural networks, dependent on a set of (to-be-calibrated) weights $\theta_i \in \Theta_i \subseteq \mathbb{R}^{l_i}$ (for some $l_i > 0$); θ denotes the collection of all the aforementioned weights, while $\mathbb{1}_{(\cdot)}$ the indicator function. In Cuchiero et al. (2020) it is proposed to approximate the leverage function by means of multiple neural networks instead of a single one (in Gierjatowicz et al. 2020, Vidales et al. 2018 a similar strategy is adopted). As outlined in Gierjatowicz et al. (2020), the choice of using multiple neural networks to be calibrated (i.e. one per maturity) is backed by efficiency-related reasons. We consider each neural network $\text{NN}_1(\cdot, \cdot), \dots, \text{NN}_M(\cdot, \cdot)$ with an architecture consisting of 4 hidden layers, each containing 64 neurons. For activation functions, we used the Leaky ReLU neuron activation with a parameter of 0.2 for the initial three layers and the Tanh activation function for the subsequent layer. The loss function used during training is the MSE. Additional information on the structure of the neural network is provided in table 8 (which are the same as those outlined in Cuchiero et al. 2020). Furthermore, we leveraged the parallel processing capabilities of an NVIDIA GeForce GTX 1080 Ti GPU to run the entire machine-learning workflow, mounted on a computing device equipped with an Intel® Xeron® Platinum 8259CL processor (2.50GHz CPU), with 2 (4) physical (logical) cores, 16GB RAM, and a 250GB hard drive. In Cuchiero et al. (2020) the reported calibration time is approximately 25 minutes. Our results are proportional, taking an average of 40 minutes. This is because the calibration time scales with

Table 8. Hyperparameter specifications for neural networks used to approximate the leverage function.

Parameter(s)	Value(s)
Hidden layers	4
Neurons (each layer)	64
Activation (first 3 layers)	Leaky ReLU (0.2)
Activation (last layer)	Tanh
Dropout rate	0.0
Batch-normalization	No
Initialization	Truncated normal
Optimizer	Adam
Learning rate	10^{-3}
Iterations per epoch	10
Batch size	10^6
Loss	Mean squared error

respect to the number of points in the volatility surface to be calibrated, as expected. Also, observe that the calibration threshold we have considered in our example is approximately half that considered in Cuchiero *et al.* (2020), i.e. 25bps versus 45bps, respectively.

4.2. Calibration

Once the functional form for the leverage function has been set as done by means of (35), one needs to set up a calibration routine for both the leverage function and the parameters relating to the stochastic volatility component of (32), i.e. those related to the process $(\sigma_t)_{t \geq 0}$. In Cuchiero *et al.* (2020) it is proposed to perform the calibration in two steps. That is, after having calibrated the ‘stand-alone’ stochastic volatility process, then the calibration of the leverage function (33) can take place with the aim of compensating for the calibration inaccuracies that the pure stochastic volatility approach produces (at least, as much as possible). Therefore, we assume that the parameters of the process $(\sigma_t)_{t \geq 0}$ are known (i.e. they have been calibrated already) and we will solely focus on the calibration of the local volatility component.

Considering the framework provided in section 4.1 given by M option maturities, we assume that, for the i th maturity T_i , J_i options are quoted. We denote their strikes and prices by $K_{i,j}$ and $C_{i,j}$, respectively ($j \in \{1, \dots, J_i\}$).[†] The corresponding model prices computed with the LSV approach described in section 4.1 will be denoted by $C_{i,j}^{\text{LSV}}(\theta_i)$. Assuming the calibration is performed in a least-squares sense, then it is necessary to solve the weighted minimization problems given by

$$\min_{\theta_i \in \Theta_i} \sum_{j=1}^{J_i} w_{i,j} (C_{i,j}^{\text{LSV}}(\theta_i) - C_{i,j})^2, \quad (36)$$

where the factors $w_{i,j}$ ’s denote some positive weights.

[†]From here onwards, for simplicity, the index i will be always assumed to range in $\{1, \dots, M\}$, while the index $j(=j(i))$ in $\{1, \dots, J_i\}$, unless otherwise stated.

4.3. Application

In this section, we provide an applicative example of how to calibrate a LSV pricing model to European option prices under risk-neutral settings. The results available here will be then the foundations for calibrating to bid and ask market prices, by means of conic Monte Carlo (see section 2.2). In particular, we employ the neural network approximation approach for the leverage function outlined in sections 4.1 and 4.2. To start with, following Cuchiero *et al.* (2020), as LSV model we chose the LSV variation of the SABR model (6) given by

$$\begin{cases} dY_t = \sigma_t Y_t L(t, Y_t) dW_t \\ d\sigma_t = \alpha \sigma_t dZ_t \\ d\langle W_t, Z_t \rangle = \rho dt \end{cases}, \quad (37)$$

where the notation used in (37) has been already outlined in sections 2.1.1 and 4.1. Choosing SABR specifications also allows to assess the advantages of considering the framework taken into account here compared to the pure conic SABR model (see section 2.1.1). Note that, in this example, we actually consider a special case of the SABR model in order to be consistent with equity volatilities, which are in general expressed in terms of lognormal ones. This means that, in practice, the parameter β is exogenously set equal to one. Therefore, given that choosing $\beta = 1$ is often referred to, in practitioner’s jargon, as considering a *stochastic lognormal* (SL) model (see also Hagan *et al.* 2002), from here onwards, to avoid misunderstandings, will use this terminology. On a side note, in order to calibrate the leverage function, we will work with log-coordinates.

For LSV calibration purposes (Cuchiero *et al.* 2020) propose to split the calibration procedure into two steps. That is, in the first part of the calibration, the goal is that of calibrating the SL parameters only. This means that, in this case, we set $L(\cdot, \cdot) \equiv 1$, and therefore consider the model as a pure stochastic volatility one. In particular, given the SL specifications considered, we aim here to find a set of SL parameters that, as good as possible, fit the ‘observable’ risk-neutral prices for all the maturities available in the dataset. Thus, once the SL parameters have been calibrated, then the local volatility component of the model will be fine-tuned at a later moment. This to compensate for the miscalibrations caused by the fact that the stand-alone SL model cannot perfectly fit, at the same time, all the options quoted for all the possible strikes and maturities. However, from a practical perspective, we propose here not to use a simulation-based approach to calibrate the SL parameters but, rather, to use the (approximated) analytical SL volatilities as per (7). The underlying reason for this choice is twofold, i.e. using analytical formulae for computing option prices under SL dynamics is not only better performing than using a Monte Carlo approach, but it also avoids introducing unnecessary simulation noise within the calibration procedure.

In symbols, following the conventions introduced in section 4.2, assuming M maturities $T_1 < \dots < T_M$ and J_i options quoted per maturity ($i \in \{1, \dots, M\}$) with strike prices $K_{i,j}$ ($j \in \{1, \dots, J_i\}$), then the calibration task for the pure SL component of the LSV model (37) (in implied volatility

terms) boils down to the minimization problem

$$\min_{\sigma_0, \alpha, \rho} \sum_{i=1}^M \sum_{j=1}^{J_i} w_{ij} \left(\sigma_j^{\text{SL}}(K_{ij}, T_i) - \sigma^{\text{market}}(K_{ij}, T_i) \right)^2, \quad (38)$$

where in (38) $\sigma^{\text{market}}(K_{ij}, T_i)$ denotes the (lognormal) market implied volatility corresponding to maturity T_i and to strike K_{ij} , while the weights w_{ij} 's are defined by means of

$$w_{ij} := 1 - \frac{|K_{ij} - K_i^{\text{ATM}}|}{K_i^{\text{ATM}}}. \quad (39)$$

In (39), K_i^{ATM} represents the at-the-money point for the i th maturity. That is, defining the weights as per (39) aims to give less and less importance to options which are far from the at-the-money point. This is sufficient, in our case, as the moneyness levels considered always allow for (39) to be positive. This choice is justified by the fact that, solely with the SL parameters, it is not possible to have a satisfactory calibration for all maturities and strikes simultaneously as already highlighted above. Therefore, here we attempt to fit the volatility surface backbone the best way possible given the set of SL parameters available. In order to achieve this we give less and less importance to the points that departure from the relevant at-the-money levels. We then attempt to fit, in a least-squares sense, those implied volatilities. In contrast, in Cuchiero *et al.* (2020) it is proposed to calibrate the SL parameters to the first maturity only. Despite the choice of how to calibrate the SL parameters remains in any case arbitrary, by experimentation it seems that attempting to fit the SL parameters to reproduce the volatility backbone is the best choice amongst the two.

Once the SL parameters have been calibrated following the methodology highlighted above, then the calibration of the local volatility component of the SL LSV model (i.e. the leverage function) can take place, at the risk-neutral level, following the procedure highlighted in section 4.2.

We now provide an example of the calibration routine just outlined. We recall that, in a two-price economy under conic finance settings, the calibration is a two-step process. That is, first the parameters describing the risk-neutral dynamics of the underlying process need to be calibrated. Afterwards, the calibration to the observed bid and ask prices can take place. In this example, as proxy for the risk-neutral prices, we consider their mid-counterparts. The results of the second step, i.e. those related to the bid-ask calibration, will be outlined in section 4.3.1.

We start by considering, under risk-neutral settings, the SL model and calibrate it for the full range of options considered utilizing (38) and (39). With the benchmark calibration now complete we proceed with the second step. That is, we calibrate the actual SL LSV model outlined in section 4.2 (under risk-neutral settings). As the final step for the risk-neutral calibration, we now add the local volatility component to the SL model and fit the model to the market data by means of approximating its local volatility component with a combination of neural networks as illustrated in section 4.1.

In order to provide a transparent analysis of the calibration results we now quantify and compare the errors obtained

Table 9. Calibrated SL parameters for each maturity.

Maturity (years)	σ_0	ρ	α
0.17	0.21	-0.81	2.50
0.33	0.21	-0.82	1.62
0.46	0.22	-0.86	1.28
0.62	0.22	-0.91	1.05
0.89	0.22	-0.97	0.81
1.04	0.75	-0.95	2.44

Table 10. Performance metrics for the SL and SL LSV models.

Error	Model	
	SL	LSV SL
MSE	$7.20 \cdot 10^{-6}$	$4.85 \cdot 10^{-7}$
RMSE	$2.68 \cdot 10^{-3}$	$6.96 \cdot 10^{-4}$
MAPE	$1.02 \cdot 10^{-2}$	$2.40 \cdot 10^{-3}$

using a pure SL calibration versus the case where a local volatility component is introduced in the model (in both approaches, i.e. SL model with and without local volatility component, $\beta = 1$, so the settings considered are consistent with each other). The results of this comparison are available in table 10. Note that it is obvious that calibrating the SL LSV model will result in a better fit compared to just calibrating the SL model. Therefore, in order to show the performance of the SL LSV model, we compare its calibration errors with those of the SL calibrated maturity by maturity. That is, in the latter case, for each maturity we compute a different set of risk-neutral SL parameters in order to have a fit, per maturity slice, that is better than that given by solely considering the SL model for all maturities simultaneously (see Appendix for some remarks concerning this choice). The calibrated SL parameters are present in table 9. We report the calibration errors in terms of MSEs, RMSEs and MAPEs in table 10.

Finally, to make the error analysis more comprehensive, we provide additional details concerning the implied volatility squared errors of the SL and the SL LSV models. The results of this analysis are presented in figure 13(a,b), respectively. The outcomes clearly illustrate how introducing a local volatility component within the SL calibration allows to obtain (squared) errors in volatility space which are one order of magnitude lower than their counterparts computed using a pure SL approach. In particular, from the empirical analysis provided it is clear that introducing the leverage function in the calibration allows to drastically reduce calibration errors. We also observe that, with the exception of a handful of points, the implied volatility squared error in the case of the LSV calibration (see figure 13(b)) is smoother than its pure SL counterpart (see figure 13(a)). And it is also worth remarking that, even when the peaks observed in figure 13(b) are taken into account, the values they correspond to are, in any case, smaller than their counterparts available in figure 13(a).

4.3.1. Results: bid-ask calibration. In this section, we present the results obtained by extending the calibration of

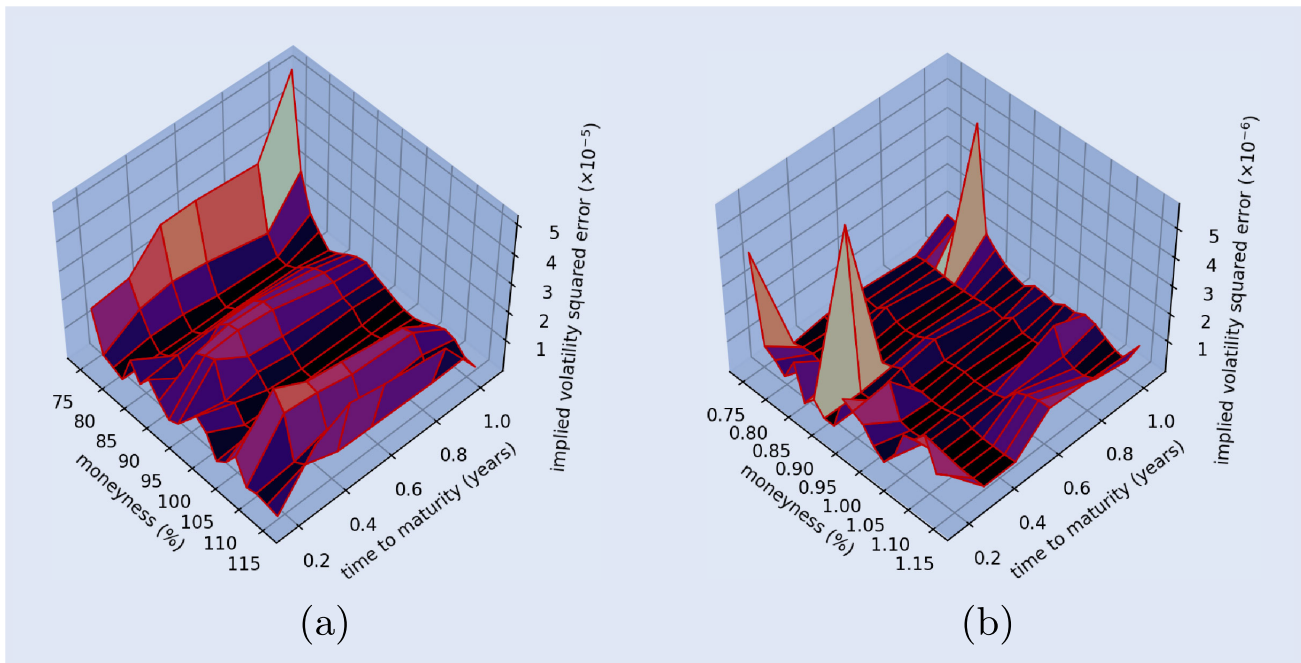


Figure 12. Surface of the squared calibration errors in implied volatility terms for the SL model (panel (a)) and for the SL LSV model (panel (b)).

the SL LSV model to a conic two-price setting. The methodology outlined in section 2.2, as well as the results obtained section 4.3, serve as foundations for this extension. The results outlined here are a continuation of those available in section 4.3. That is, after having performed the calibration of the LSV model at the risk-neutral level we can, by means of conic Monte Carlo (see section 2.2), perform a bid-ask calibration given real market data. Furthermore, we also present a selection of calibration results coming from the application of some of the hybrid families of distortion functions introduced in section 2.3. For the sake of conciseness, we exhibit here, from a graphical perspective, only the calibration results pertaining to the first and to the last maturities.

In figure 14, we present the bid and ask implied volatilities obtained following the pure conic SL calibration outlined in section 2.1.1. Additionally, for the purpose of comparison, we include the market bid and ask implied volatilities, as well as the market mid-price implied volatilities.

An examination of figure 14 reveals an often poor performance of the calibration method, which seems particularly evident for options with low moneyness levels and short maturities. The plots in figure 14 illustrate a significant bid-ask spread which the calibration fails to capture, as evidenced by the high repricing errors for certain options, where the calibrated bid implied volatility differ by approximately 800 basis points from the market bid implied volatility.

For comparison purposes, in figure A1 we now present the results obtained by calibrating the SL LSV model by means of conic Monte Carlo given the same dataset considered so far. Furthermore, in figure A1 we present bid and ask implied volatilities obtained through the SL LSV calibration using both the Minmaxvar and the Wang distortion functions. The results demonstrate a clear improvement in terms of calibration accuracy. We remark here that, in this framework, as we

are not dealing with analytical formulae, it does not make a difference, from an implementation angle, whether we use one distortion function or another. On the contrary, in the case of the conic SL model (see section 2.1.1), the choice of the Wang transform (instead of another distortion) allows to still remain in a framework where (approximated) analytical formulae can be used.

We now present, for completeness, some calibration results obtained by employing the novel t -Minmaxwang distortion introduced in section 2.3. The key distinction of this approach is the inclusion of an additional parameter, denoted as t (we use t in this context due to the interpretation of this distortion function provided in section 2.3). We also consider the Minmaxvar-Wang and the Wang-Minmaxvar distortions introduced in section 2.3. This parameter serves as a metric for selecting the appropriate distortion in each instance. This is a useful feature because, as highlighted in section 2.3, often the choice of the distortion function to be used is somewhat arbitrary. Therefore, having a simple tool allowing to discriminate between the possible choices available can often come in handy. Through the analysis of the results we find that the majority of points in the volatility surface exhibit a (close to) zero value for t , indicating a predominance of the Wang transform in terms of performance accuracy. Based on this observation, we can empirically assess that, at least for the cases considered in the examples available in the paper at hand, using the Wang transform results in lower calibration errors compared to the Minmaxvar distortion. We present in tables 11 and 12 some summary statistics concerning the bid-ask calibration errors observed using different distortions functions. From the results obtained, and as expected by construction, we observe that the t -Minmaxwang (slightly) outperforms the other choices (due to the extra degree of freedom introduced).

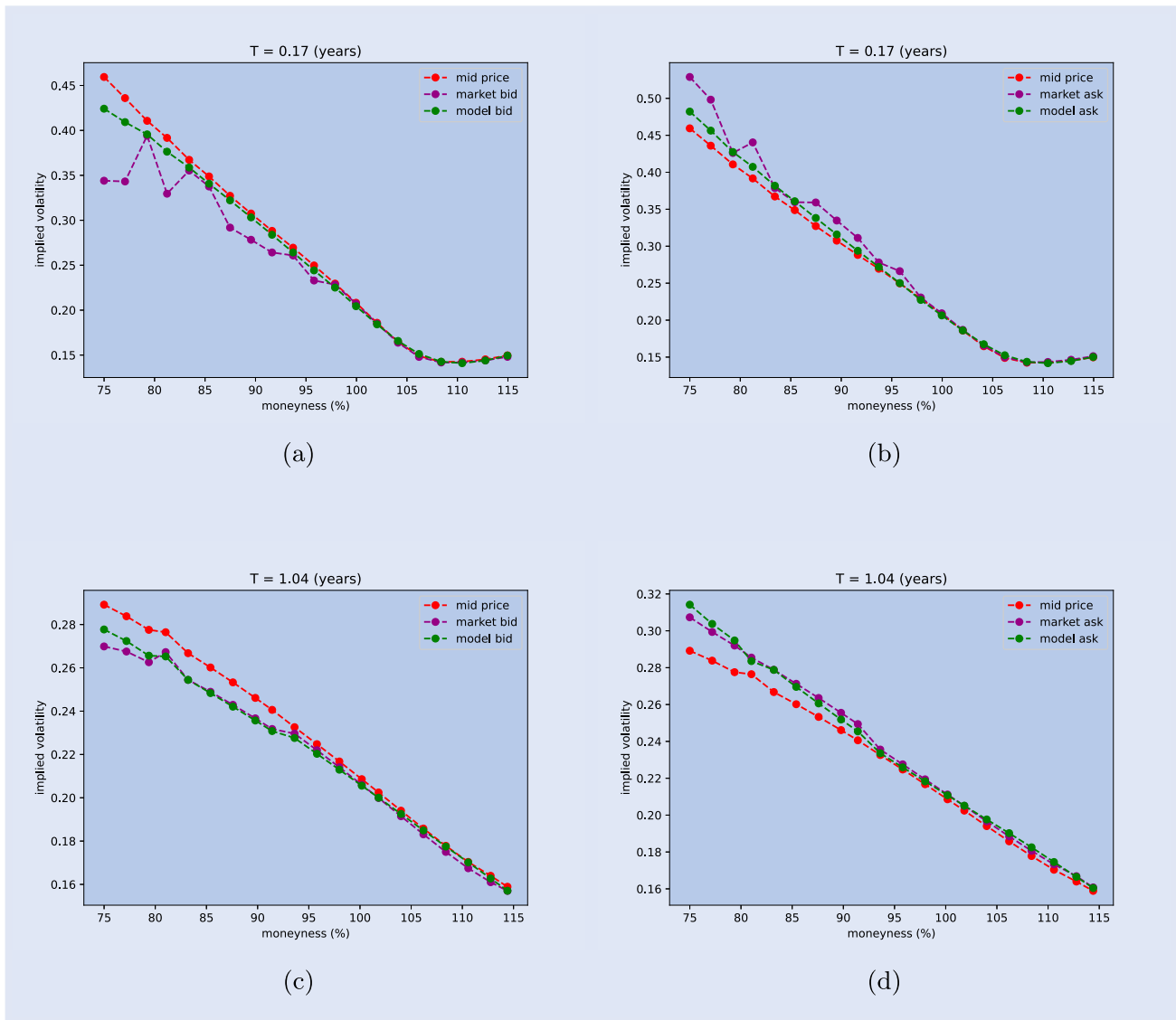


Figure 13. Calibrated bid and ask implied volatilities using the conic SL model. Panels (a) and (b) illustrate the calibrated bid and ask implied volatilities for the first maturity of the dataset considered, while (c) and (d) their counterparts in the case the last maturity is taken into account.

Table 11. Calibration bid errors per distortion function given the dataset considered.

Error	Model				
	Minmaxvar	Wang	t-MinmaxWang	Minmaxvar-Wang	Wang-Minmaxvar
MSE	$6.93 \cdot 10^{-7}$	$6.49 \cdot 10^{-7}$	$6.42 \cdot 10^{-7}$	$6.83 \cdot 10^{-7}$	$6.69 \cdot 10^{-7}$
RMSE	$8.32 \cdot 10^{-4}$	$8.05 \cdot 10^{-4}$	$8.01 \cdot 10^{-4}$	$8.26 \cdot 10^{-4}$	$8.18 \cdot 10^{-4}$
MAPE	$2.65 \cdot 10^{-3}$	$2.60 \cdot 10^{-3}$	$2.57 \cdot 10^{-3}$	$2.63 \cdot 10^{-3}$	$2.63 \cdot 10^{-3}$

Table 12. Calibration ask errors per distortion function given the dataset considered.

Error	Distortion				
	Minmaxvar	Wang	t-Minmaxwang	Minmaxvar-Wang	Wang-Minmaxvar
MSE	$4.77 \cdot 10^{-7}$	$4.62 \cdot 10^{-7}$	$4.58 \cdot 10^{-7}$	$4.71 \cdot 10^{-7}$	$4.69 \cdot 10^{-7}$
RMSE	$6.90 \cdot 10^{-4}$	$6.79 \cdot 10^{-4}$	$6.76 \cdot 10^{-4}$	$6.85 \cdot 10^{-4}$	$6.85 \cdot 10^{-4}$
MAPE	$2.22 \cdot 10^{-3}$	$2.20 \cdot 10^{-3}$	$2.17 \cdot 10^{-3}$	$2.21 \cdot 10^{-3}$	$2.21 \cdot 10^{-3}$

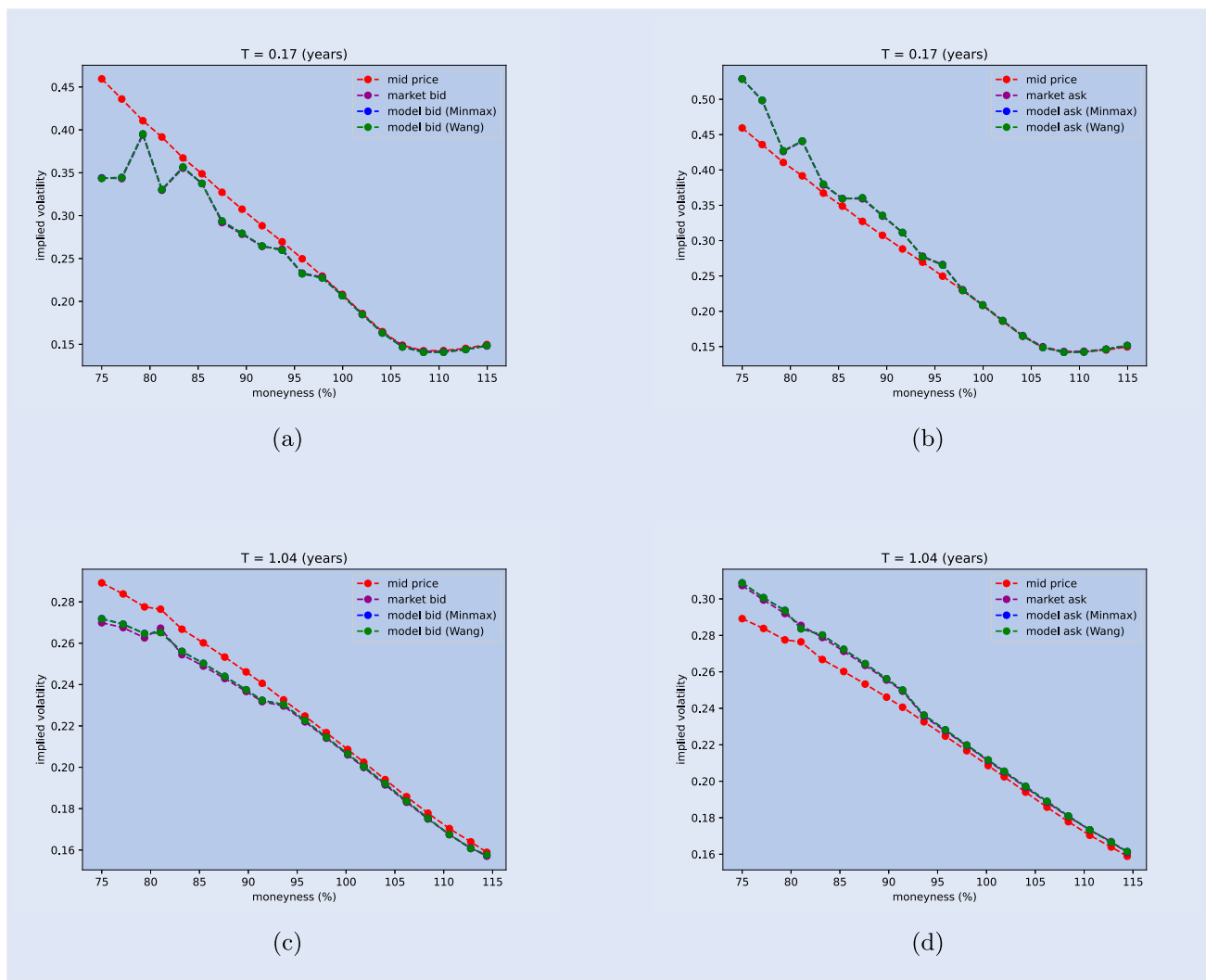


Figure 14. Calibrated bid and ask implied volatilities using the conic SL LSV model. Panels (a) and (b) illustrate the calibrated bid and ask implied volatilities for the first maturity of the dataset considered, while (c) and (d) their counterparts in the case the last maturity is taken into account.

5. Conclusion

In the article at hand we investigated new ways of combining the usage of neural networks in quantitative finance and, in particular, in the context of derivatives valuation in a two-price economy according to the paradigm of Cherny and Madan (2010). We examined two different applications of neural networks in conic financial markets. The first explores the possibility of replicating bid and ask pricing by the use of (vector-valued) neural networks, while the second aims to combine the use of neural networks with the LSV modeling framework. More precisely, (i) we provided an accurate, flexible and fast neural network-based architecture for derivative valuation in a conic economy, and (ii) we extended this approach to compute sensitivities. Moreover, in the second part (iii) we extended the work of Cuchiero *et al.* (2020) to incorporate bid-ask pricing through conic Monte Carlo simulations. To conclude, (iv) we introduced a conic version of the SABR (Hagan *et al.* 2014) model, based on the Wang distortion (Wang 2000), and (v) we provided simple techniques to generate arbitrary families of hybrid distortion functions.

Acknowledgments

We would like to thank two anonymous referees for their valuable comments and suggestions. Their detailed and insightful feedback has greatly contributed to improving the quality of our paper. The views and opinions expressed in this article are solely those of the authors and do not necessarily reflect those of their current or past employers.

Disclosure statement

No potential conflict of interest was reported by the author(s).

ORCID

Diogo Franquinho  <https://orcid.org/0009-0005-7258-5903>
Peter Spreij  <http://orcid.org/0000-0002-6416-6320>

References

- Awoyemi, J.O., Adetunmbi, A.O. and Oluwadare, S.A., Credit card fraud detection using machine learning techniques: A comparative analysis. In *2017 International Conference on Computing Networking and Informatics (ICCNi)*, pp. 1–9, 2017.
- Bayer, C., Friz, P. and Gatheral, J., Pricing under rough volatility. *Finance*, 2016, **16**(6), 887–904.
- Bennedsen, M., Lunde, A. and Pakkanen, M.S., Hybrid scheme for Brownian semistationary processes. *Finance Stoch.*, 2017, **21**(4), 931–965.
- Borchani, H., Varando, G., Bielza, C. and Larrañaga, P., A survey on multi-output regression. *WIREs Data Min. Knowl. Discov.*, 2015, **5**, 216–233.
- Buehler, H., Volatility and dividends—volatility modelling with cash dividends and simple credit risk. Working paper, 2010. Available online at: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=1141877.
- Buehler, H., Gonon, L., Teichmann, J. and Wood, B., Deep hedging. *Quant. Finance*, 2019, **19**(8), 1271–1291.
- Carmona, R. and Nadtochiy, S., Local volatility dynamic models. *Finance Stoch.*, 2009, **13**, 1–48.
- Chen, Z., Van Khoa, L.D., Teoh, E.N., Nazir, A., Karuppiah, E.K. and Lam, K.S., Machine learning techniques for anti-money laundering (AML) solutions in suspicious transaction detection: A review. *Knowl. Inf. Syst.*, 2021, **190**, 184–192.
- Cherny, A. and Madan, D.B., New measures for performance evaluation. *Rev. Financ. Stud.*, 2009, **22**(7), 2571–2606.
- Cherny, A. and Madan, D.B., Markets as a counterparty: An introduction to conic finance. *Int. J. Theor. Appl. Finance*, 2010, **13**(8), 1149–1177.
- Chopra, S., Reinforcement learning methods for conic finance. PhD thesis, University of Maryland, 2020.
- Corcuera, J.M., Guillaume, F., Madan, D.B. and Schoutens, W., Implied liquidity: Towards stochastic liquidity modeling and liquidity trading. *Int. J. Portf. Anal. Manag.*, 2012, **1**(1), 80–91.
- Cuchiero, C., Khosrawi, W. and Teichmann, J., A generative adversarial network approach to calibration of local stochastic volatility models. *Risks*, 2020, **8**(4), 101.
- Davis, J., Devos, L., Reyners, S. and Schoutens, W., Gradient boosting for quantitative finance. *J. Comput. Finance*, 2021, **21**(4), 1–40.
- De Spiegeleer, J., Madan, D.B., Reyners, S. and Schoutens, W., Machine learning for quantitative finance: Fast derivative pricing, hedging and fitting. *Quant. Finance*, 2018, **18**(10), 1635–1643.
- Denneberg, D., *Non-Additive Measure and Integral*, 1994 (Kluwer Academic Publishers: Dordrecht).
- Domashova, J. and Mikhailina, N., Usage of machine learning methods for early detection of money laundering schemes. *Procedia Comput. Sci.*, 2021, **190**, 184–192.
- Dupire, B., Pricing with a smile. *Risk*, 1994, **7**(1), 18–20.
- Ferguson, R. and Green, A., Deeply learning derivatives. Working paper, 2018. Available online at: <https://arxiv.org/abs/1802.03042>.
- Fritsch, F.N. and Carlson, R.E., Monotone piecewise cubic interpolation. *SIAM J. Numer. Anal.*, 1980, **17**(2), 238–246.
- Gatheral, J., *The Volatility Surface*, 2006 (John Wiley & Sons: Hoboken).
- Gierjatowicz, P., Sabate-Vidales, M., Šiška, D., Szpruch, L. and Žurič, Ž., Robust pricing and hedging via neural SDEs. Working paper, 2020. Available online at: <https://arxiv.org/abs/2007.04154>.
- Guyon, J. and Henry-Labordère, P., Being particular about calibration. *Risk Mag.*, 2012, **January**, 92–97.
- Hagan, P.S., Kumar, D., Lesniewski, A. and Woodward, D., Arbitrage-free SABR. *Wilmott*, 2014, **69**, 60–75.
- Hagan, P.S., Kumar, D., Lesniewski, A.S. and Woodward, D.E., Managing smile risk. *Wilmott*, 2002, **1**, 84–108.
- Haug, E.G., *The Complete Guide to Option Pricing Formulas*, 2007 (McGraw-Hill: New York).
- Henrard, M., *Algorithmic Differentiation in Finance Explained*, 2017 (Springer: Cham).
- Heston, S.L., A closed-form solution for options with stochastic volatility with applications to bond and currency options. *Rev. Financ. Stud.*, 1993, **6**(2), 327–343.
- Hinton, G.E., Vinyals, O. and Dean, J., Distilling the knowledge in a neural network. Working paper, 2010. Available online at: <https://arxiv.org/abs/1503.02531>.
- Hwu, W., Kirk, D.B. and El Hajj, I., *Programming Massively Parallel Processors: A Hands-on Approach*, 2022 (Morgan Kaufmann: Waltham).
- Johri, P., Verma, J.K. and Paul, S., *Applications of Machine Learning*, 2020 (Springer Verlag: Singapore).
- Kaelo, P. and Ali, M., Integrated crossover rules in real coded genetic algorithms. *Eur. J. Oper. Res.*, 2007, **176**(1), 60–76.
- Krauss, C., Do, X.A. and Huck, N., Deep neural networks, gradient-boosted trees, random forests: Statistical arbitrage on the S&P 500. *Eur. J. Oper. Res.*, 2017, **259**(2), 689–702.
- Liu, S., Borovkyh, A. and Grzelak, L., A neural network-based framework for financial model calibration. *J. Math. Ind.*, 2019a, **9**(9), 9.
- Liu, S., Oosterlee, C. and Bohte, S., Pricing options and computing implied volatilities using neural networks. *Risks*, 2019b, **7**(1), 16.
- Madan, D.B. and Schoutens, W., *Applied Conic Finance*, 2016 (Cambridge University Press: Cambridge).
- Madan, D.B. and Sharaiha, Y.M., Machine trading: Theory, advances, and applications. *J. Financ. Data Sci.*, 2020, **2**(3), 8–24.
- Mandelbrot, B.B. and van Ness, J.W., Fractional Brownian motions, fractional noises and applications. *SIAM Rev.*, 1968, **10**(4), 422–437.
- Mazzon, A. and Pascucci, A., The forward smile in local-stochastic volatility models. *J. Comput. Finance*, 2016, **20**(3), 1–29.
- McCrickerd, R. and Pakkanen, M.S., Turbocharging Monte Carlo pricing for the rough Bergomi model. *Quant. Finance*, 2018, **18**(11), 1877–1886.
- Michielon, M., Khedher, A. and Spreij, P., Liquidity-free implied volatilities: An approach using conic finance. *Int. J. Financ. Eng.*, 2021, **8**(4), 2150041.
- Morelli, M.J., Montagna, G., Nicrosini, O., Treccani, M., Farina, M. and Amato, P., Pricing financial derivatives with neural networks. *Phys. A Stat. Mech. Appl.*, 2004, **338**(1), 160–165.
- Moussa, K., Arbitrage-based filtering of option price data. Working paper, 2018. Available online at: <https://ssrn.com/abstract=3197284>.
- Nasse, F., Thureau, C. and Fink, G.A., Face detection using GPU-based convolutional neural networks. In *CAIP*, pp. 83–90, 2009.
- Oh, K.-S. and Jung, K., GPU implementation of neural networks. *Pattern Recognit.*, 2004, **37**(6), 1311–1314.
- Okanovic, P., Waleffe, R., Mageirakos, V., Nikolakakis, K.E., Karbasi, A., Kalogieras, D., Gürel, N.M. and Rekasinas, T., Repeated random sampling for minimizing the time-to-accuracy of learning. Working paper, 2023. Available online at: <https://arxiv.org/abs/2305.18424>.
- Overhaus, M., Bermúdez, A., Buehler, H., Ferraris, A., Jordison, C. and Lamnouar, A., *Equity Hybrid Derivatives*, 2007 (John Wiley & Sons: Hoboken).
- Press, W.H., Teukolsky, S.A., Vetterling, W.T. and Flannery, B.P., *Numerical Recipes. The Art of Scientific Computing*, 2007 (Cambridge University Press: Cambridge).
- Robey, R. and Zamora, Y., *Parallel and High Performance Computing*, 2021 (Manning: Shelter Island).
- Sachdev, N. and McAuley, J., Data distillation: A survey. Working paper, 2010. Available online at: <https://arxiv.org/abs/2301.04272>.
- Sadgali, I., Sael, N. and Benabbou, F., Performance of machine learning techniques in the detection of financial frauds. *Procedia Comput. Sci.*, 2019, **148**, 45–54.
- Schoutens, W., Simons, E. and Tistaert, J., A perfect calibration! now what? *Wilmott Mag.*, 2004, **March**, 66–78.
- Sorscher, B., Geirhos, R., Shekhar, S., Ganguli, S. and Morcos, A.S., Beyond neural scaling laws: Beating power law scaling via data pruning. In *Advances in Neural Information Processing Systems*, edited by A. H. Oh, A. Agarwal, D. Belgrave and K. Cho, volume 35, pp. 19523–19536, 2022.

- Spiegeleer, J.D., Madan, D.B., Reyners, S. and Schoutens, W., Machine learning for quantitative finance: Fast derivative pricing, hedging and fitting. *Quant. Finance*, 2018, **18**(10), 1635–1643.
- Szandafa, T., Review and comparison of commonly used activation functions for deep neural networks. In *Bio-Inspired Neurocomputing*, edited by A. K. Bhoi, P. K. Mallick, C.-M. Liu, and V. E. Balas, pp. 203–224, 2021 (Springer: Singapore).
- Vadera, S. and Ameen, S., Methods for pruning deep neural networks. Working paper, 2021. Available online at: <https://arxiv.org/abs/2011.00241>.
- van der Stoep, A.W., Grzelak, L.A. and Oosterlee, C.W., The time-dependent FX-SABR model: Efficient calibration based on effective parameters. *Int. J. Theor. Appl. Finance*, 2015, **18**(6), 1550042.
- Vidales, M.S., Siska, D. and Szpruch, L., Unbiased deep solvers for linear parametric PDEs. Working paper, 2018. Available online at <https://arxiv.org/abs/1810.05094>.
- Wang, S.S., A class of distortion operators for pricing financial and insurance risks. *J. Risk Insur.*, 2000, **67**(1), 15–36.
- Wang, Y., Wei, G. and Brooks, D., Benchmarking TPU, GPU, and CPU platforms for deep learning. Working paper, 2019. Available online at: <https://arxiv.org/abs/1907.10701>.
- Wang, Z. and Klir, G.J., *Generalized Measure Theory*, 2009 (Springer: New York).

Appendix. A remark on the calibration choice of the SL model

In terms of calibration of the SL model, see section 4.3, one could follow two approaches:

- (i) Calibrate the parameters of the SL model given all the options quoted in the market (i.e. all maturities included at the same time);
- (ii) Calibrate, for each maturity, a set of SL model parameters (as done in section 4.3) and interpolate prices in the maturity direction by, for instance, interpolate the SL parameters or the implied volatilities appropriately.

Despite approach (i) would be the theoretically-correct way of calibrating the model, it would obviously provide, per maturity slice, larger calibration errors compared to approach (ii). In figure A1 we have reported the squared errors obtained by calibrating the SL model to all the available option maturities considered in section 4.3 at the same time (i.e. by means of (i)). The resulting calibrated parameters are $\sigma_0 = 0.22$, $\rho = -0.91$ and $\alpha = 1.30$.

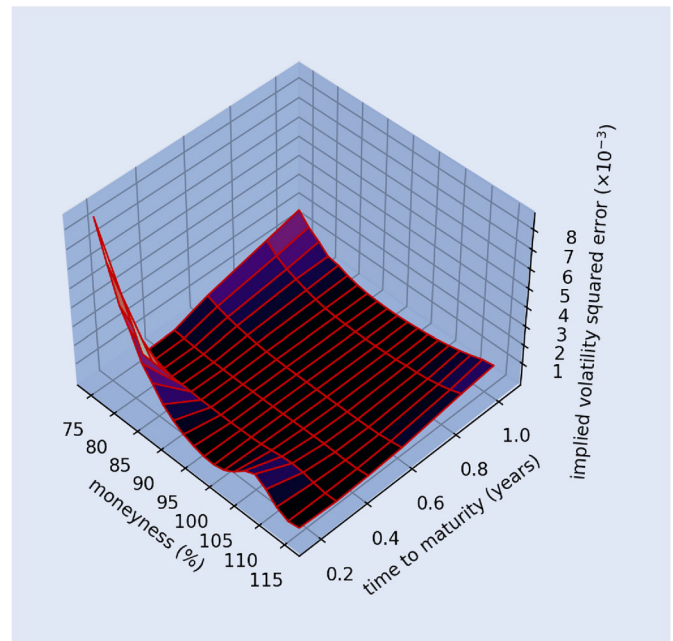


Figure A1. Squared errors in implied volatility obtained by calibrating the SL model to all maturities simultaneously for the options considered in section 4.3, i.e. approach (i).

If we compare figure A1 above with figure 13(a) (which follows approach (ii)), we note that the errors in the former are two orders of magnitude larger compared to the errors in the latter. We further report, by means of the four panels in figure A2 below, how the calibration performance of the SL model would look like when using approach (i) (cf. panels in figure 14).

The reason why we considered approach (ii) in section 4.3 to be compared with the LSV calibration is that we wanted to show that, even if model choices were made in such a way to obtain (without a local volatility component) repricing errors that are as low as possible, the approach described in section 4.3 still outperforms it. We highlight that by no means we are stating that approach (ii) is the one that is theoretically justified (despite often used in practice), but it provides a useful benchmark to be used in the tests outlined in section 4.3.

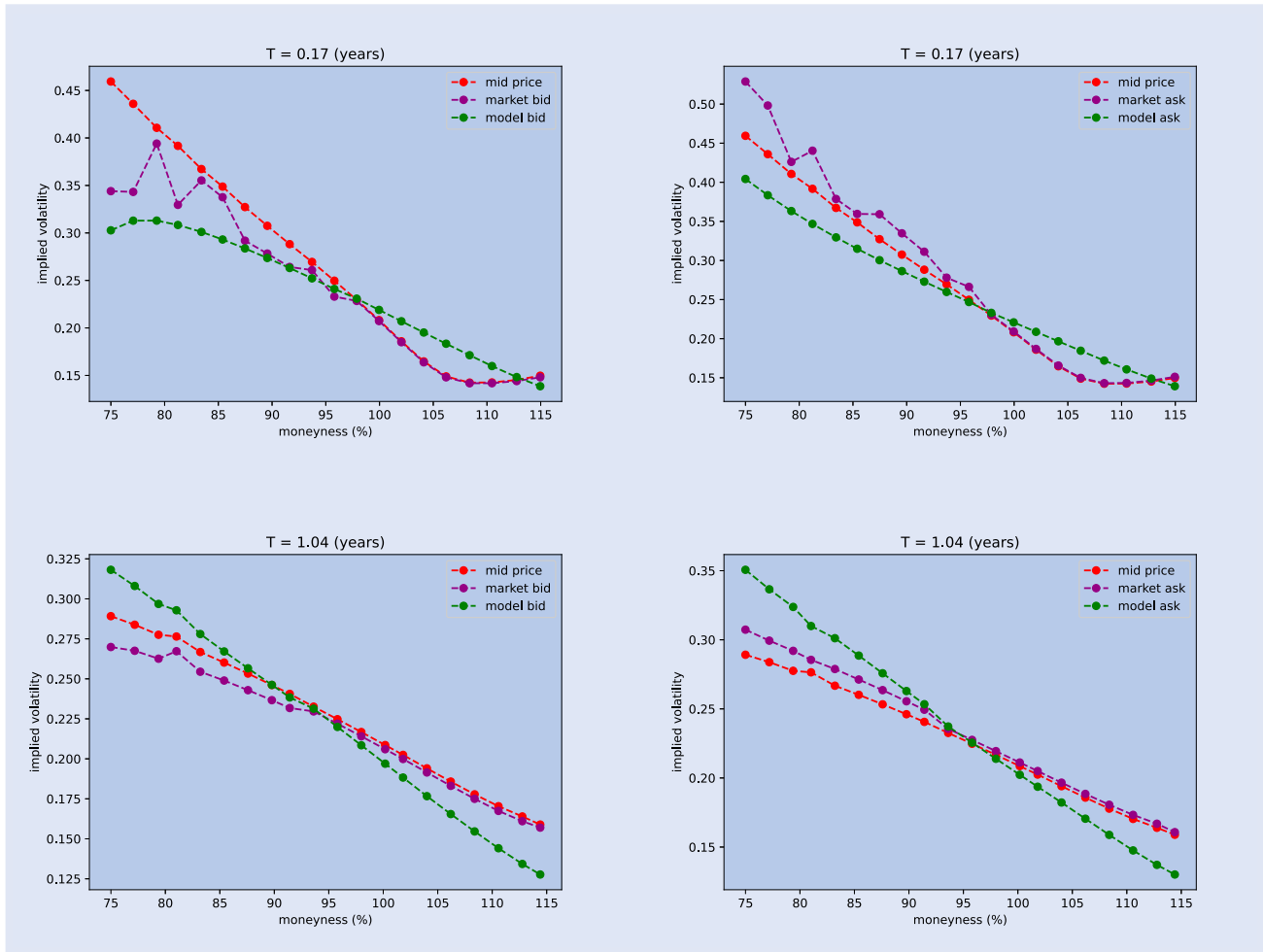


Figure A2. Calibrated bid and ask implied volatilities using the conic SL model where the risk-neutral parameters of the SL model have been calibrated as per (i) given the dataset of section 4.3. Panels (a) and (b) illustrate the calibrated bid and ask implied volatilities for the first maturity of the dataset considered, while (c) and (d) their counterparts in the case the last maturity is taken into account.