

Deep Learning Volatility

A deep neural network perspective on pricing and calibration in (rough) volatility models*

Blanka Horvath

Department of Mathematics, King's College London

blanka.horvath@kcl.ac.uk, bhorvath@turing.ac.uk

Aitor Muguruza

Department of Mathematics, Imperial College London & NATIXIS

aitor.muguruza-gonzalez15@imperial.ac.uk

Mehdi Tomas

CMAP & LadHyx, École Polytechnique

mehdi.tomas@polytechnique.edu

August 26, 2019

Abstract

We present a neural network based calibration method that performs the calibration task within a few milliseconds for the full implied volatility surface. The framework is consistently applicable throughout a range of volatility models—including the rough volatility family—and a range of derivative contracts. The aim of neural networks in this work is an off-line approximation of complex pricing functions, which are difficult to represent or time-consuming to evaluate by other means. We highlight how this perspective opens new horizons for quantitative modelling: The calibration bottleneck posed by a slow pricing of derivative contracts is lifted. This brings several numerical pricers and model families (such as rough volatility models) within the scope of applicability in industry practice. The form in which information from available data is extracted and stored influences network performance. This approach is inspired by representing the implied volatility and option prices as a collection of pixels. In a number of applications we demonstrate the prowess of this modelling approach regarding accuracy, speed, robustness and generality and also its potentials towards model recognition.

2010 Mathematics Subject Classification: 60G15, 60G22, 91G20, 91G60, 91B25

Keywords: Rough volatility, volatility modelling, Volterra process, machine learning, accurate price approximation, calibration, model assessment, Monte Carlo

*The authors are grateful to Jim Gatheral, Ben Wood, Antoine Savine and Ryan McCrickerd for stimulating discussions. MT conducted research within the Econophysique et Systèmes Complexes under the aegis of the Fondation du Risque, a joint initiative by the Fondation de l'École Polytechnique, l'École Polytechnique, Capital Fund Management. MT also gratefully acknowledges the financial support of the ERC 679836 Staquamof and the Chair Analytics and Models for Regulation.

Contents

1	Introduction	3
2	A neural network perspective on model calibration	5
2.1	A brief reminder of some (rough) models considered	5
2.2	Calibration bottlenecks in volatility modelling and deep calibration	7
2.3	Challenges in neural network approximations of pricing functionals	8
2.4	Motivations for our choice of training setup and features of neural networks as approximators of pricing functionals	9
2.4.1	Reasons for the choice of grid-based implicit training	10
2.4.2	Some relevant properties of deep neural networks as functional approximators	10
3	Pricing and calibration with neural networks: Optimising network and training	12
3.1	The objective function	13
3.1.1	For vanillas	13
3.1.2	Some exotic payoffs	14
3.2	Network architecture and training	14
3.2.1	Network architecture of the implied volatility map approximation	15
3.2.2	Training of the approximation network	16
3.3	The calibration step	16
4	Numerical experiments	17
4.1	Numerical accuracy and speed of the price approximation for vanillas	18
4.1.1	Neural network price approximation in (rough) Bergomi models with piecewise constant forward variance curve	19
4.2	Calibration speed and accuracy for implied volatility surfaces	21
4.2.1	A calibration experiment with simulated data in (rough) Bergomi models with piecewise constant forward variances	22
4.2.2	Calibration in the rough Bergomi model with historical data	23
4.3	Numerical experiments with barrier options in the rough Bergomi model	26
5	Conclusions and outlook: “best-fit” models	26

1 Introduction

Approximation methods for option prices came in all shapes and forms in the past decades and they have been extensively studied in the literature and well-understood by risk managers. Clearly, the applicability of any given option pricing method (Fourier pricing, PDE methods, asymptotic methods, Monte Carlo, ...etc.) depends on the regularity properties of the particular stochastic model at hand. Therefore, tractability of stochastic models has been one of the most decisive qualities in determining their popularity. In fact it is often a more important quality than the modelling accuracy itself: It was the (almost instantaneous) SABR asymptotic formula that helped SABR become the benchmark model in fixed income desks, and similarly the convenience of Fourier pricing is largely responsible for the popularity of the Heston model, despite the well-known hiccups of these models. Needless to say that it is the very same reason (the concise Black Scholes formula) that still makes the Black-Scholes model attractive for calculations even after many generations of more realistic and more accurate stochastic market models have been developed. On the other end of the spectrum are rough volatility models, for which (despite a plethora of modelling advantages, see [7, 23, 28] to name a few) the necessity to rely on relatively slow Monte Carlo based pricing methods creates a major bottleneck in calibration, which has proven to be a main limiting factor with respect to industrial applications. This dichotomy can become a headache in situations when we have to weigh up the objectives of accurate pricing vs. fast calibration against one another in the choice of our pricing model. In this work we explore the possibilities provided by the availability of an algorithm that –for a choice of model parameters– directly outputs the corresponding vanilla option prices (as the Black-Scholes formula does) for a large range of maturities and strikes of a given model.

In fact, the idea of mapping model parameters to shapes of the implied volatility surface directly is not new. The stochastic volatility inspired SSVI, eSSVI surfaces (see [27, 29, 35]) do just that: A given set of parameters is translated directly to different shapes of (arbitrage-free) implied volatility surfaces, bypassing the step of specifying any stochastic dynamics for the underlying asset. For stochastic models that admit asymptotic expansions, such direct mappings from model parameters to (approximations of) the implied volatility surface in certain asymptotic regimes can be obtained (one example is the famous SABR formula). Such asymptotic formulae are typically limited to certain asymptotic regimes along the surface by their very nature. Complementary to asymptotic expansions we explore here a direct (approximative) mapping from different parameter combinations of stochastic models to different shapes of implied volatility surface for intermediate regimes. Its appeal is that it combines the advantages of direct parametric volatility surfaces (of the SSVI family) with the possibility to link volatility surfaces to the stochastic dynamics of the underlying asset.

In this paper we apply deep neural networks (merely) as powerful high-dimensional functional approximators to approximate the multidimensional pricing functionals from model parameters to option prices. The advantage of doing so via deep neural networks over standard (fixed-basis) functional approximations is that deep neural networks are agnostic to the approximation basis [32]. This makes them robustly applicable to several stochastic models consistently. Our objective in doing so is to move the (often time-consuming) numerical approximation of the pricing functional into an off-line preprocessing step. This preprocessing amounts to storing the approximative direct pricing functional in form of the network weights after a supervised training procedure: Using available numerical approximations of option prices as ground truth (in a stochastic model of our

choice), we train a neural network to learn an accurate approximation of the pricing functional. After training, the network outputs—for any choice of model parameters—the corresponding implied volatilities within milliseconds for a large range of maturities and strikes along the whole surface. Furthermore, we show that this procedure generalises well for unseen parameter combinations: the accuracy of price approximation of our neural network pricing functional on out-of-sample data is within the same range as the accuracy of the original numerical approximation used for training. The accuracy of this direct pricing map is demonstrated in our numerical experiments.

One of the striking advantages of this approach is that it speeds up the (on-line) calibration Rough Volatility models to the realm of just a few milliseconds. There have been several recent contributions on neural network calibrations of stochastic models [9, 13, 36, 49, 17]. Clearly, much depends on the finesse of the particular network design with respect to the performance of these networks. One contribution of this paper is to achieve a fast and accurate calibration of the rough Bergomi model of [7] with a general forward variance curve (approximated by piecewise constant function). To demonstrate this, we first perform calibration experiments on simulated data and show calibration accuracy in controlled experiments. To demonstrate the speed and prowess of the approach we then calibrate the rough Bergomi model to historical data and display the evolution of parameters on a dataset consisting of 10 years of SPX data.

Another advantage of our modelling choice is that by its very design it can be applied to portfolios including multiple strikes and maturities at the same time which is the first step towards their application as hedging instruments. See for example Buehler et al. [13] a motivation.

The paper is organised as follows: In Section 2 we present a neural network perspective on model calibration and recall stochastic models that are considered in later sections. In this section we also formalise our objectives about the accuracy and speed of neural network approximation of pricing functionals and the basic ideas of our training. Section 2.4.2 recalls some background on neural networks as functional approximators and some aspects of neural network training that influenced the setup of our network architecture and training design. In Section 3.2 we describe the network architecture and training of the price approximation network such as the calibration methods we consider. In Section 4 we present numerical experiments of price approximations of vanilla and some exotic options, calibration to synthetic data and to historical data. Section 5 points to further potential applications and outlook to future work.

Numerical experiments and codes are provided on GitHub: [NN-StochVol-Calibrations](#), where an accessible code demo of our results can be downloaded. We also created a library of stochastic models where this approach is demonstrated to work well.

2 A neural network perspective on model calibration

In plain words, any calibration procedure is meant to fix the model parameters such that the model is as close as possible to the observed reality. In a financial context, our model represents the underlying (stocks, indices, volatility, etc.) and we are interested in calibrating the model to available market prices of financial contracts based on this underlying.

Let us first formalise this by setting the notation $\mathcal{M} := \mathcal{M}(\theta)_{\theta \in \Theta}$ which represents an abstract model with parameters θ in the set $\Theta \subset \mathbb{R}^n$, for some $n \in \mathbb{N}$. Thus the model $\mathcal{M}(\theta)$ (stochastic or parametric) and the corresponding prices of financial contracts are fully specified by the choice of the parameter combination $\theta \in \Theta$. Furthermore, we introduce a pricing map $P : \mathcal{M}(\theta, \zeta) \rightarrow \mathbb{R}^m$, where $\zeta : (C(\mathbb{R}) \rightarrow \mathbb{R}^m)$, $m \in \mathbb{N}$ denote the financial products we aim to price, such as vanilla options for (a set of) given maturities and strikes. Let us denote the observed market data corresponding to the contracts, by $\mathcal{P}^{MKT}(\zeta) \in \mathbb{R}^m$, $m \in \mathbb{N}$.

Parameter Calibration: The parameter configuration $\hat{\theta}$ solves an (ideal) δ -calibration problem for a model $\mathcal{M}(\Theta)$ for the conditions $\mathcal{P}^{MKT}(\zeta)$ if

$$\hat{\theta} = \operatorname{argmin}_{\theta \in \Theta} \delta(P(\mathcal{M}(\theta), \zeta), \mathcal{P}^{MKT}(\zeta)) \quad (1)$$

where $\delta(\cdot, \cdot)$ is a suitable choice of metric for the financial contract ζ at hand.

For most financial models however (1) represents an idealised form of the calibration problem as in practice there rarely exists an analytical formula for the option price $P(\mathcal{M}(\theta), \zeta)$ and for the vast majority of financial models it needs to be computed by some numerical approximation scheme.

Approximate Parameter Calibration We say that the parameter configuration $\hat{\theta} \in \Theta$ solves an *approximate δ -calibration* problem for the model $\mathcal{M}(\Theta)$ for the conditions $\mathcal{P}^{MKT}(\zeta)$ if

$$\hat{\theta} = \operatorname{argmin}_{\theta \in \Theta} \delta(\tilde{P}(\mathcal{M}(\theta), \zeta), \mathcal{P}^{MKT}(\zeta)) \quad (2)$$

where $\delta(\cdot, \cdot)$ is a suitably chosen metric and \tilde{P} is a numerical approximation of the pricing map P .

In the remainder of this paper it is this second type of calibration problem that we will be concerned with: In our numerical experiments (Section 4) we consider the numerical approximation \tilde{P} of the pricing map P as the benchmark (available truth) for generating synthetic training samples in the training a neural network to approximate pricing maps. Clearly, the better the original numerical approximations, the better the network approximation will be. In a separate work we will illuminate this perspective with a Bayesian analysis of the calibration procedure.

2.1 A brief reminder of some (rough) models considered

We would like to emphasize that our methodology can in principle be applied to any (classical or rough) volatility model. From the classical Black Scholes or Heston models to the rough Bergomi model of [7], also to large class of rough volatility models (see Horvath, Jacquier and Muguruza [41] for a general setup). In fact the methodology is not limited to stochastic models, also parametric models of implied volatility could be used for generating training samples of abstract models, but we have not pursued this direction further.

The Rough Bergomi model

In the abstract model framework, the rough Bergomi model is represented by $\mathcal{M}^{rBergomi}(\Theta^{rBergomi})$, with parameters $\theta = (\xi_0, \nu, \rho, H) \in \Theta^{rBergomi}$. On a given filtered probability space $(\Omega, \mathcal{F}, (\mathcal{F}_t)_{t \geq 0}, \mathbb{P})$ the model corresponds to the following system

$$\begin{aligned} dX_t &= -\frac{1}{2}V_t dt + \sqrt{V_t} dW_t, \quad \text{for } t > 0, \quad X_0 = 0, \\ V_t &= \xi_0(t)\mathcal{E}\left(\sqrt{2H\nu} \int_0^t (t-s)^{H-1/2} dZ_s\right), \quad \text{for } t > 0, \quad V_0 = v_0 > 0 \end{aligned} \tag{3}$$

where $H \in (0, 1)$ denotes the Hurst parameter, $\nu > 0$, $\mathcal{E}(\cdot)$ the stochastic exponential [21], and $\xi_0(\cdot) > 0$ denotes the initial forward variance curve (see [11, Section 6]), and W and Z are correlated standard Brownian motions with correlation parameter $\rho \in [-1, 1]$. To fit the model parameters into our abstract model framework $\Theta^{rBergomi} \subset \mathbb{R}^n$ for some $n \in \mathbb{N}$, the initial forward variance curve $\xi_0(\cdot) > 0$ is approximated by a piecewise constant function in our numerical experiments in Sections 4.1.1, and 4.2.1. We refer the reader to Horvath, Jacquier and Muguruza [41] for one general setting of rough volatility models and their numerical simulation.

The Heston model

The Heston model, appearing in our numerical experiments of Section 5 is described by the system

$$\begin{aligned} dS_t &= \sqrt{V_t} S_t dW_t \quad \text{for } t > 0, \quad S_0 = s_0 \\ dV_t &= a(b - V_t)dt + v\sqrt{V_t} dZ_t \quad \text{for } t > 0, \quad V_0 = v_0 \end{aligned} \tag{4}$$

with W and Z Brownian motions with correlation parameter $\rho \in [-1, 1]$, $a, b, v > 0$ and $2ab > v^2$. In our framework it is denoted by $\mathcal{M}^{Heston}(\theta)$ with $\theta = (a, b, v, \rho) \in \Theta^{Heston} \subset \mathbb{R}^4$. The Heston model is considered in our numerical experiments in Section 5. It was also considered by [9, 18] in different neural network contexts.

The Bergomi model

In the general n -factor Bergomi model, the volatility is expressed as

$$V_t = \xi_0(t)\mathcal{E}\left(\eta_i \sum_{i=1}^n \int_0^t \exp(-\kappa_i(t-s)) dW_s^i\right) \quad \text{for } t > 0, \quad V_0 = v_0 > 0, \tag{5}$$

where $\eta_1, \dots, \eta_n > 0$ and (W^1, \dots, W^n) is an n -dimensional correlated Brownian motion, $\mathcal{E}(\cdot)$ the stochastic exponential [21], and $\xi_0(\cdot) > 0$ denotes the initial forward variance curve, see [11, Section 6] for details. In this work we consider the Bergomi model for $n = 1, 2$ in Section 4. Henceforth, $\mathcal{M}^{1FBergomi}(\xi_0, \beta, \eta, \rho)$ represents the 1 Factor Bergomi model, corresponding to the following dynamics:

$$\begin{aligned} dX_t &= -\frac{1}{2}V_t dt + \sqrt{V_t} dW_t \quad \text{for } t > 0, \quad X_0 = 0 \\ V_t &= \xi_0(t)\mathcal{E}\left(\eta \int_0^t \exp(-\beta(t-s)) dZ_s\right) \quad \text{for } t > 0, \quad V_0 = v_0 > 0, \end{aligned} \tag{6}$$

where $\nu > 0$, and W and Z are correlated standard Brownian motions with correlation parameter $\rho \in [-1, 1]$. To fit the model parameters into our abstract model framework $\Theta^{1FBergomi} \subset \mathbb{R}^n$, for some $n \in \mathbb{N}$, the initial forward variance curve $\xi_0(\cdot) > 0$ is approximated in our numerical experiments by a piecewise constant function in Sections 4.1.1, and 4.2.1.

The SABR model

The stochastic alpha beta rho model of Hagan et al. [33, 34] is denoted in our setting as $\mathcal{M}^{SABR}(\alpha, \beta, \rho)$ and is defined as

$$\begin{aligned} dS_t &= V_t S_t^\beta dW_t \quad \text{for } t > 0, \quad S_0 = s_0, \\ dV_t &= \alpha V_t dZ_t \quad \text{for } t > 0, \quad V_0 = v_0 \end{aligned} \tag{7}$$

where $v_0, s_0, \alpha > 0$ and $\beta \in [0, 1]$. The SABR model is considered by McGhee in [55] in a neural network context (see also Section 2.3).

2.2 Calibration bottlenecks in volatility modelling and deep calibration

Whenever for a stochastic volatility model the numerical approximate calibration procedures (2) are computationally slow, a bottleneck in calibration time can deem the model of limited applicability for industrial production irrespective of other desirable features the model might have. This is the case in particular for the family rough volatility models, where the rough fractional Brownian motion in the volatility dynamics rules out usual Markovian pricing methods such as finite differences. So far such calibration bottlenecks have been a major limiting factor for the class of rough volatility models, whose overwhelming modelling advantages have been explored and highlighted in rapidly expanding number of academic articles [1, 2, 7, 6, 8, 10, 22, 26, 28, 44, 40, 45] in the past years. Other examples include models with delicate degeneracies (such as the SABR model around zero forward) which for a precise computation of arbitrage-free prices require time consuming numerical pricing methods such as Finite Element Methods [42], Monte Carlo [15, 52] or the evaluation of multiple integrals [3].

Contrary to Hernandez's [36] pioneering work, where he develops a direct calibration via NN, we set up and advocate a two step calibration approach.

Two Step Approach (i) Learn a model and (ii) Calibrate to data: One separates the calibration procedure described in (2) (resp. (2)) into two parts: (i) We first learn (approximate) the pricing map by a neural network that maps parameters of a stochastic model to pricing functions (or implied volatilities (cf. section (2.1) and we store this map during an off-line training procedure. In a second step (ii) we calibrate (on-line) the now deterministic approximative learned price map, which speeds up the on-line calibration by orders of magnitude. To formalise the two step approach, we write for a payoff ζ and a model \mathcal{M} with parameters $\theta \in \Theta$

$$\text{(i) Learn: } \tilde{F}(\theta, \zeta) = \tilde{P}(\mathcal{M}(\theta, \zeta)) \quad \text{(ii) Calibrate: } \hat{\theta} = \underset{\theta \in \Theta}{\operatorname{argmin}} \delta(\tilde{F}(\theta, \zeta), \mathcal{P}^{MKT}(\zeta)). \tag{8}$$

Note that in part (ii) of (8) we essentially replaced $\tilde{P}(\mathcal{M}(\theta, \zeta))$ in equation (2) by its learned (deterministic) counterpart $\tilde{F}(\theta, \zeta)$ (which will be a Neural Network see Section 3.2) from (i). Therefore, this second calibration is—by its deterministic nature—considerably faster than calibration of all those traditional stochastic models, which involve numerical simulation of the expected

payoff $P(\mathcal{M}(\theta, \zeta)) = \mathbb{E}[\zeta(X(\theta))]$ for some underlying stochastic process X^θ . The first part **(i)** in (8) denotes an approximation of the pricing map through a neural network, which is calibrated in a supervised training procedure using the original (possibly slow) numerical pricing maps for training (see sections 3.2 and 4 for details in specific examples).

In the following sections we elaborate on the objectives and advantages of this two step calibration approach and present examples of neural network architectures, precise numerical recipes and training procedures to apply the two step calibration approach to a family of stochastic volatility models. We also present some numerical experiments (corresponding codes are available on GitHub: NN-StochVol-Calibrations) and report on learning errors and on calibration times.

2.3 Challenges in neural network approximations of pricing functionals

In general problem (1) and henceforth (2) is solved using suitable numerical optimisation techniques such as gradient descent [32], specific methods for certain metrics (such as Lavenberg-Marquadt [53] for L^2), neural networks, or tailor-made methods to the complexity of the optimisation problem and objective function at hand¹. But irrespective of their level of sophistication all optimisers for calibration share a common property: repeated (iterative) evaluation of the pricing map $\theta \mapsto P(\mathcal{M}(\theta), \zeta)$ (resp. an approximation \tilde{P} thereof) on each instance θ of consecutive parameter combinations until a sufficiently small distance $\delta(\tilde{P}(\mathcal{M}(\theta), \zeta), \mathcal{P}^{MKT}(\zeta))$ between model prices and observed prices is achieved. Consequently, the pricing map is arguably the computational cornerstone of a calibration algorithm. Main differences between specific calibration algorithms effectively lie in the way the specific choice of evaluated parameter combinations $\{\theta_1, \theta_2 \dots\}$ are determined, which hence determines the total number N of functional evaluations of the pricing function $(P(\mathcal{M}(\theta_i), \zeta))_{i=1\dots N}$ used in the calibration until the desired precision $\delta(\tilde{P}(\mathcal{M}(\hat{\theta}), \zeta), \mathcal{P}^{MKT}(\zeta))$ is achieved. In case the pricing map

$$\begin{aligned} P(\mathcal{M}(\cdot), \zeta) : \Theta &\longrightarrow P(\mathcal{M}) \\ \theta &\mapsto P(\mathcal{M}(\theta), \zeta) \end{aligned}$$

involved in (1) is available in closed form, and can be evaluated instantaneously, the calibration (2) is fast even if a high number N of functional evaluations is used. If the pricing map is approximated numerically, calibration time depends strongly on the time needed to generate a functional evaluation of the numerical approximation

$$\theta_i \mapsto \tilde{P}(\mathcal{M}(\theta_i), \zeta), \quad \theta_i \in \{\theta_1, \dots, \theta_N\} \tag{9}$$

at each iteration $i = 1, \dots, N$ of the calibration procedure. Slow functional evaluations potentially cause substantial bottlenecks in calibration time. This is where we see the most powerful use of the prowess of neural network approximation:

A neural network is constructed to replace in **(i)** of (8) the pricing map, that is to approximate (for a given financial contract ζ) the pricing map from the full set² of model parameters Θ of the model to the corresponding prices $P(\mathcal{M}(\theta, \zeta))$. The *first challenge* for the neural network approximator of pricing functionals is to speed up this process and enable us to obtain *faster functional evaluations*

¹For details and an overview on calibration methods see [32].

²Note that the set $\theta_1, \dots, \theta_N$ in (9) is extended to the full set of possible parameter combinations Θ in (10).

and thereby lift the bottleneck of calibration. The *second challenge* is to do so with an accuracy that remains within the error bounds of the original numerical pricing discretisation:

$$\begin{aligned}\tilde{F} : \Theta &\longrightarrow \tilde{P}(\mathcal{M}) \\ \theta &\mapsto \tilde{F}(\theta, \zeta)\end{aligned}\tag{10}$$

More precisely (motivated by (2)), for any parameter combination $\theta \in \Theta$ we aim to approximate the numerical approximation \tilde{P} of the true option price P with the neural network \tilde{F} up to the same order of precision $\epsilon > 0$ up to which \tilde{P} approximates P . That is, for any $\theta \in \Theta$

$$\tilde{F}(\theta) = P(\mathcal{M}(\theta), \zeta) + \mathcal{O}(\epsilon) \quad \text{whenever} \quad \tilde{P}(\mathcal{M}(\theta), \zeta) = P(\mathcal{M}(\theta), \zeta) + \mathcal{O}(\epsilon).$$

Therefore, our training objective is

$$\tilde{F}(\theta) = \tilde{P}(\mathcal{M}(\theta), \zeta) + \mathcal{O}(\epsilon).\tag{11}$$

where \tilde{P} is the available numerical approximation of the pricing function, which is considered as ground truth. In our numerical experiments in Section 4 we demonstrate that our approximation network achieves this approximation accuracy and yields a substantial speedup in terms of functional evaluations.

2.4 Motivations for our choice of training setup and features of neural networks as approximators of pricing functionals

There are several advantages of separating the tasks of pricing and calibration which we address in full detail in a separate work. Here we recall some of the most convincing reasons to do so. Above all, the most appealing reason is that it allows us to build upon the knowledge we have gained about the models in the past decades, which is of crucial importance from a risk management perspective. By its very design, deep learning the *price approximation* (i) combined with (ii) deterministic calibration does not cause more headache to risk managers and regulators than the corresponding stochastic models do. Designing the training as described above demonstrates how deep learning techniques can successfully extend the toolbox of financial engineering, without making compromises on any of our objectives.

1. The knowledge gathered in many years of experience with traditional models remains useful and risk management libraries of models remain valid. The neural network is only used as a computational enhancement of models.
2. The availability of training data for training the deep neural network does not cause any constraints as it is synthetically generated by traditional numerical methods.
3. This can be extended beyond the models presented in this work: Whenever a consistent numerical pricer exists for a model, it can be approximated and replaced by a deep neural network that provides fast numerical evaluations of the pricing map.

Here, we identify the grid-based approach as our choice of training. Though a thorough analysis of the best training approaches is subject to further research, we have good reason to believe that the grid-based approach provides a powerful and robust methodology for training:

2.4.1 Reasons for the choice of grid-based implicit training

In the grid-based approach we evaluate the values of implied volatility surface along 8×11 gridpoints with 80,000 different parameter combinations we effectively evaluate the "fit" of the surface to numerically generated ones across the same number of points. By moving the evaluation of the implied volatilities into the objective function we improve the learning in many aspects:

- The first advantage of implicit training is that it efficiently exploits the structure of the data. Updates in neighbouring volatility points σ_{n-1} and σ_n can be incorporated in the learning process. If the output is a full grid as in (16) this effect is further enhanced. Updates of the network on each gridpoint also imply additional information for updates of the network on neighbouring gridpoints. One can say that we regard the implied volatility surface as an image with a given number of pixels.
- A further advantage of the image based implicit training is, that by evaluating the objective function on a larger set of (grid) points, injectivity of the mapping can be more easily guaranteed than in the pointwise training: Two distinct parameter combinations are less likely to yield the same value across a set of gridpoints, then if evaluated only on a single point.
- We do not limit ourselves to one specific grid on the implied volatility surface. We store the generated 60,000 sample paths for the training data and chose a set of maturities (here 8) and strikes (here 11) to evaluate prices corresponding to these paths. But we can easily add and evaluate additional maturities and strikes to the same set of paths. Note in particular that in this training design we can refine the grid on the implied volatility surface without increasing the number of training samples needed and without significantly increasing the computational time for training as the portfolio of vanilla options on the same underlying grows with different strikes and maturities.

2.4.2 Some relevant properties of deep neural networks as functional approximators

Deep feed forward³ neural networks are the most basic deep neural networks, originally designed to approximate some function F^* , which is not available in closed form but only through sample pairs of given input data x and output data $y = F^*(x)$. In a nutshell, a feed forward network defines a mapping $y = F(x, w)$ and the training determines (calibrates) the optimal values of network parameters \hat{w} that result in the best function approximation⁴ $F^*(\cdot) \approx F(\cdot, \hat{w})$ of the unknown function $F^*(\cdot)$ for the given pairs of input and output data (x, y) , cf. [32, Chapter 6].

To formalise this, we introduce some notation and recall some basic definitions and principles of function approximation via (feedforward) neural networks:

Definition 1 (Neural network). Let $L \in \mathbb{N}$ and the tuple $(N_1, N_2 \dots, N_L) \in \mathbb{N}^L$ denote the number of layers (depth) and the number of nodes (neurons) on each layer respectively. Furthermore, we introduce the affine functions

$$\begin{aligned} w^l : \mathbb{R}^{N_l} &\longrightarrow \mathbb{R}^{N_{l+1}} \text{ for } 1 \leq l \leq L-1 \\ x &\mapsto A^{l+1}x + b^{l+1} \end{aligned} \tag{12}$$

³The network is called feed forward if there are no feedback connections in which outputs of the model are fed back into itself.

⁴In our case y is a 8×11 -point grid on the implied volatility surface and x are model parameters $\theta \in \Theta$, for details see Section 3.

acting between layers for some $A^{l+1} \in \mathbb{R}^{N_{l+1} \times N_l}$. The vector $b^{l+1} \in \mathbb{R}^{N_{l+1}}$ denotes the *bias term* and each entry $A_{(i,j)}^{l+1}$ denotes the *weight* connecting node $i \in N_l$ of layer l with node $j \in N_{l+1}$ of layer $l + 1$. For the collection of affine functions of the form (12) on each layer we fix the notation $w = (w^1, \dots, w^L)$. We call the tuple w the *network weights* for any such collection of affine functions. Then a Neural Network $F(w, \cdot) : \mathbb{R}^{N_0} \rightarrow \mathbb{R}^{N_L}$ is defined as the composition:

$$F := F_L \circ \cdots \circ F_1 \quad (13)$$

where each component is of the form $F_l := \sigma_l \circ W^l$. The function $\sigma_l : \mathbb{R} \rightarrow \mathbb{R}$ is referred to as the *activation function*. It is typically nonlinear and applied component wise on the outputs of the affine function W^l . The first and last layers, F_1 and F_L , are the *input* and *output* layers. Layers in between, $F_2 \cdots F_{L-1}$, are called *hidden layers*.

The following central result of Hornik justifies the use of neural networks as approximators for multivariate functions and their derivatives.

Theorem 1 (Universal approximation theorem (Hornik, Stinchcombe and White [38])). *Let $\mathcal{NN}_{d_0, d_1}^\sigma$ be the set of neural networks with activation function $\sigma : \mathbb{R} \mapsto \mathbb{R}$, input dimension $d_0 \in \mathbb{N}$ and output dimension $d_1 \in \mathbb{N}$. Then, if σ is continuous and non-constant, $\mathcal{NN}_{d_0, 1}^\sigma$ is dense in $L^p(\mu)$ for all finite measures μ .*

There is a rapidly growing literature on approximation results with neural networks, see [37, 39, 56, 65] and the references therein. Among these we would like to single out one particular result:

Theorem 2 (Universal approximation theorem for derivatives (Hornik, Stinchcombe and White [39])). *Let $F^* \in C^n$ and $F : \mathbb{R}^{d_0} \rightarrow \mathbb{R}$ and $\mathcal{NN}_{d_0, 1}^\sigma$ be the set of single-layer neural networks with activation function $\sigma : \mathbb{R} \mapsto \mathbb{R}$, input dimension $d_0 \in \mathbb{N}$ and output dimension 1. Then, if the (non-constant) activation function is $\sigma \in C^n(\mathbb{R})$, then $\mathcal{NN}_{d_0, 1}^\sigma$ arbitrarily approximates f and all its derivatives up to order n .*

Remark 1. Theorem 2 highlights that the smoothness properties of the activation function are of significant importance in the approximation of derivatives of the target function F^* . In particular, to guarantee the convergence of l -th order derivatives of the target function, we choose an activation function $\sigma \in C^l(\mathbb{R})$. Note that the ReLu activation function, $\sigma_{ReLU}(x) = (x)^+$ is not in $C^l(\mathbb{R})$ for any $l > 0$, while $\sigma_{Elu}(x) = \alpha(e^x - 1)$ is smooth.

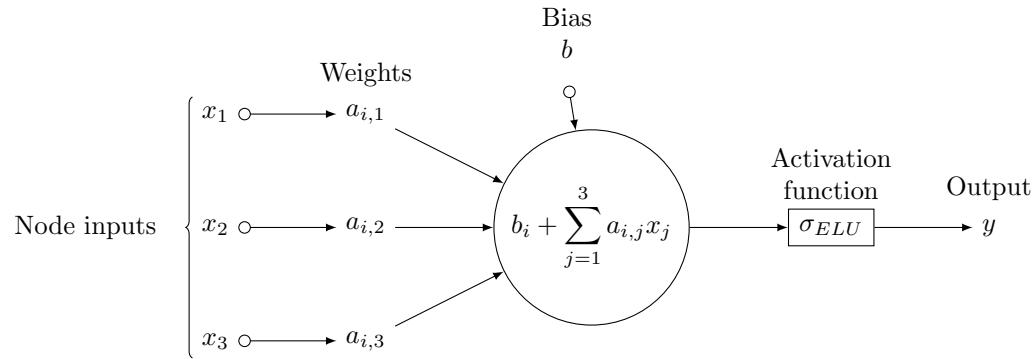


Figure 1: In detail neuron behaviour

The following Theorem provides theoretical bounds for the above rule of thumb and establishes a connection between the number of nodes in a network and the number of training samples needed to train it.

Theorem 3 (Estimation bounds for Neural Networks (Barron [5])). *Let $\mathcal{NN}_{d_0, d_1}^\sigma$ be the set of single-layer neural networks with Sigmoid activation function $\sigma(x) = \frac{e^x}{e^x + 1}$, input dimension $d_0 \in \mathbb{N}$ and output dimension $d_1 \in \mathbb{N}$. Then:*

$$\mathbb{E}\|F^* - \hat{F}\|_2^2 \leq \mathcal{O}\left(\frac{C_f^2}{n}\right) + \mathcal{O}\left(\frac{nd_0}{N} \log N\right)$$

where n is the number of nodes, N is the training set size and C_{F^*} is the first absolut moment of the Fourier magnitude distribution of F^* .

Remark 2. Barron's [5] insightful result gives a rather explicit decomposition of the error in terms of bias (model complexity) and variance:

- $\mathcal{O}\left(\frac{C_{F^*}^2}{n}\right)$ represents the model complexity, i.e. the larger n (number of nodes) the smaller the error
- $\mathcal{O}\left(\frac{nd_0}{N} \log N\right)$ represents the variance, i.e. a large n must be compensated with a large training set N in order to avoid overfitting.

Finally, we motivate the use of multi layer networks and the choice of network depth. Even though a single layer might theoretically suffice to arbitrarily approximate any continuous function, in practice the use of multiple layers dramatically improves the approximation capacities of network. We informally recall the following Theorem due to Eldan and Shamir [20] and refer the reader to the original paper for details.

Theorem 4 (Power of depth of Neural Networks (Eldan and Shamir [20])). *There exists a simple (approximately radial) function on \mathbb{R}^d , expressible by a small 3-layer feedforward neural networks, which cannot be approximated by any 2-layer network, to more than a certain constant accuracy, unless its width is exponential in the dimension.*

Remark 3. In spite of the specific framework by Eldan and Shamir [20] being restrictive, it provides a theoretical justification to the power of “deep” neural networks (multiple layers) against “shallower” networks (i.e. few layers) as in [55] with a larger number of neurons. On the other hand, multiple findings indicate [9, 47] that adding hidden layers beyond 4 hidden layers does not significantly improve network performance.

3 Pricing and calibration with neural networks: Optimising network and training

In this section we compare different objective functions (direct calibration to data to an image-based implicit learning approach) and motivate our choice of image-based objective function. We give details about network architectures for the approximation network and compare different optimisers for the calibration step.

3.1 The objective function

1. Learn the map $F^*(\theta) = \{P^{\mathcal{M}(\theta)}(\zeta_i)\}_{i=1}^n$ via neural network, where $\{\zeta_i\}_{i=1,\dots,n}$ represents the exotic product attributes (such as maturity, strike or barrier level) on a prespecified grid with size n .

$$\hat{w} = \operatorname{argmin}_{w \in \mathbb{R}^n} \sum_{u=1}^{N_{Train}} \sum_{i=1}^n (F(\theta_u, w)_i - F^*(\theta_u)_i)^2.$$

2. Solve

$$\hat{\theta} := \operatorname{argmin}_{\theta \in \Theta} \sum_{i=1}^n (\tilde{F}(\theta)_i - P^{MKT}(\zeta_i))^2. \quad (14)$$

3.1.1 For vanillas

As in many academic and industry research papers, we pursue the calibration of vanilla contracts via approximation of the implied volatility surface⁵.

We take this idea further and design an implicit form of the pricing map that is based on storing the implied volatility surface as an image given by a grid of "pixels". This image-based representation has a formative contribution in the performance of the network we present in Section 4. We present our contribution here; Let us denote by $\Delta := \{k_i, T_j\}_{i=1}^n, \{j\}_{j=1}^m$ a fixed grid of strikes and maturities, then we propose the following two step approach:

1. Learn the map $F^*(\theta) = \{\sigma_{BS}^{\mathcal{M}(\theta)}(T_i, k_j)\}_{i=1}^n, \{j\}_{j=1}^m$ via neural network $\tilde{F}(\theta) := F(\theta, \hat{w})$ where

$$\begin{aligned} F^* : \Theta &\longrightarrow \mathbb{R}^{n \times m} \\ \theta &\mapsto F^*(\theta) \end{aligned} \quad (16)$$

where the input is a parameter combination $\theta \in \Theta$ of the stochastic model $\mathcal{M}(\Theta)$ and the output is a $n \times m$ grid on the implied volatility surface $\{\sigma_{BS}^{\mathcal{M}(\theta)}(T_i, k_j)\}_{i=1}^n, \{j\}_{j=1}^m$ where $n, m \in \mathbb{N}$ are chosen appropriately (see Section 3.2). Then,

$$\hat{w} = \operatorname{argmin}_{w \in \mathbb{R}^n} \sum_{u=1}^{N_{Train}} \sum_{i=1}^n \sum_{j=1}^m (F(\theta_u, w)_{ij} - F^*(\theta_u)_{ij})^2.$$

⁵For sake of completeness we introduce the Black-Scholes Call pricing function in terms of log-strike k , initial spot S_0 , maturity T and volatility σ :

$$BS(\sigma, S_0, k, T) := S_0 \mathcal{N}(d_+) - K \mathcal{N}(d_-), \quad d_{\pm} := \frac{\log(S_0) - k}{\sqrt{T}\sigma} \pm \frac{\sqrt{T}\sigma}{2},$$

where $\mathcal{N}(\cdot)$ denotes the Gaussian cumulative distribution function. The implied volatility induced by a Call option pricing function $P(K, T)$ is then given by the unique solution $\sigma_{BS}(k, T)$ of the following equation

$$BS(\sigma_{BS}(k, T), S_0, k, T) = P(k, T).$$

Precisely, we seek to solve the following calibration problem

$$\hat{\theta} := \operatorname{argmin}_{\theta \in \Theta} d(\Sigma_{BS}^{\mathcal{M}(\theta)}, \Sigma_{BS}^{MKT}) \quad (15)$$

where $\Sigma_{BS}^{\mathcal{M}(\theta)} := \{\sigma_{BS}^{\mathcal{M}(\theta)}(k_i, T_j)\}_{i=1,\dots,n, j=1,\dots,m}$ represents the set of implied volatilities generated by the model pricing function $P(\mathcal{M}(\theta), k, T)$ and $\Sigma_{BS}^{MKT} := \{\sigma_{BS}^{MKT}(k_i, T_j)\}_{i=1,\dots,n, j=1,\dots,m}$ are the corresponding market implied volatilities, for some metric $d : \mathbb{R}^{n \times m} \times \mathbb{R}^{n \times m} \rightarrow \mathbb{R}^+$.

2. Solve

$$\hat{\theta} := \operatorname{argmin}_{\theta \in \Theta} \sum_{i=1}^n \sum_{j=1}^m (\tilde{F}(\theta)_{ij} - \sigma_{BS}^{MKT}(T_i, k_j))^2.$$

Remark 4. Notice that $\hat{w}(\Delta)$ depends on Δ implicitly, consequently so does $\tilde{F}(\theta) = F(\theta, \hat{w}(\Delta))$ (hence the name implicit learning). This setting is similar to that of image recognition and exploits the structure of the data to reduce the complexity of the Network (see Section 4 for details).

Remark 5. In our experiments we chose $n = 8$ and $m = 11$. At first, a criticism of mapping (16) might be the inability to extrapolate/interpolate between maturities/strikes outside the grid Δ . However, one is free to choose the grids Δ as fine as needed. In addition, one may use standard (arbitrage free) uni/bi-variate splines techniques to extrapolate/interpolate across strikes and maturities, as with traditional market data observable only at discrete points.

Figure 2: Volatility surface generated by the neural network approximator and the corresponding original counterpart on a grid given by 8 maturities and 11 strikes.

3.1.2 Some exotic payoffs

Our framework extends to a number of exotic products such as: Digital barriers, no-touch (or double no-touch) barrier, cliques or autocallables.

We present some numerical experiments in Section 4.3, to demonstrate the pricing of digital barrier options. More precisely, in Section 4.3 we consider down-and-in such as down-and-out digital barrier options, the main building blocks of many Autocallable products. For a barrier level $B < S_0$ and maturity T the payoff is given by:

$$P^{Down-and-In}(B, T) = \mathbb{E} [\mathbf{1}_{\{\tau_B \leq T\}}] \quad (17)$$

$$P^{Down-and-Out}(B, T) = \mathbb{E} [\mathbf{1}_{\{\tau_B \geq T\}}] \quad (18)$$

where $\tau_B = \inf_t \{S_t = B\}$. In this setting, we may easily generate a grid for barrier levels and maturities $\Delta^{Barrier} := \{B_i, T_j\}_{i=1, j=1}^{n, m}$ that we can fit in the objective function specified in (14)

3.2 Network architecture and training

Motivated by the above analysis, we choose to set up the calibration in the implicit two-step approach. This involves a separation of the calibration procedure into (i) “Deep approximation” an approximation network with an implicit training and (ii) “Calibration” a calibration layer on top. We first start by describing the approximation network in the implicit image-based training and discuss the calibration in Section 3.3 below. In addition, we will highlight specific techniques that contribute to the robustness and efficiency of our design.

3.2.1 Network architecture of the implied volatility map approximation

Here we motivate our choice of network architecture for the following numerical experiments which were inspired by the analysis in the previous sections. Our network architecture is summarised in the graph 3.2.1 below.

1. A fully connected feed forward neural network with 4 hidden layers (due to Theorem 4) and 30 nodes on each layers (see Figure 3.2.1 for a detailed representation)
2. Input dimension = n , number of model parameters
3. Output dimension = 11 strikes \times 8 maturities for this experiment, but this choice of grid can be enriched or modified.
4. The four inner layers have 30 nodes each, which adding the corresponding biases results on a number
$$(n + 1) \times 30 + 4 \times (1 + 30) \times 30 + (30 + 1) \times 88 = 30n + 6478$$
of network parameters to calibrate (see Section 2.4.2 for details).
5. Motivated by Theorem 2 we choose the Elu $\sigma_{Elu} = \alpha(e^x - 1)$ activation function for the network.

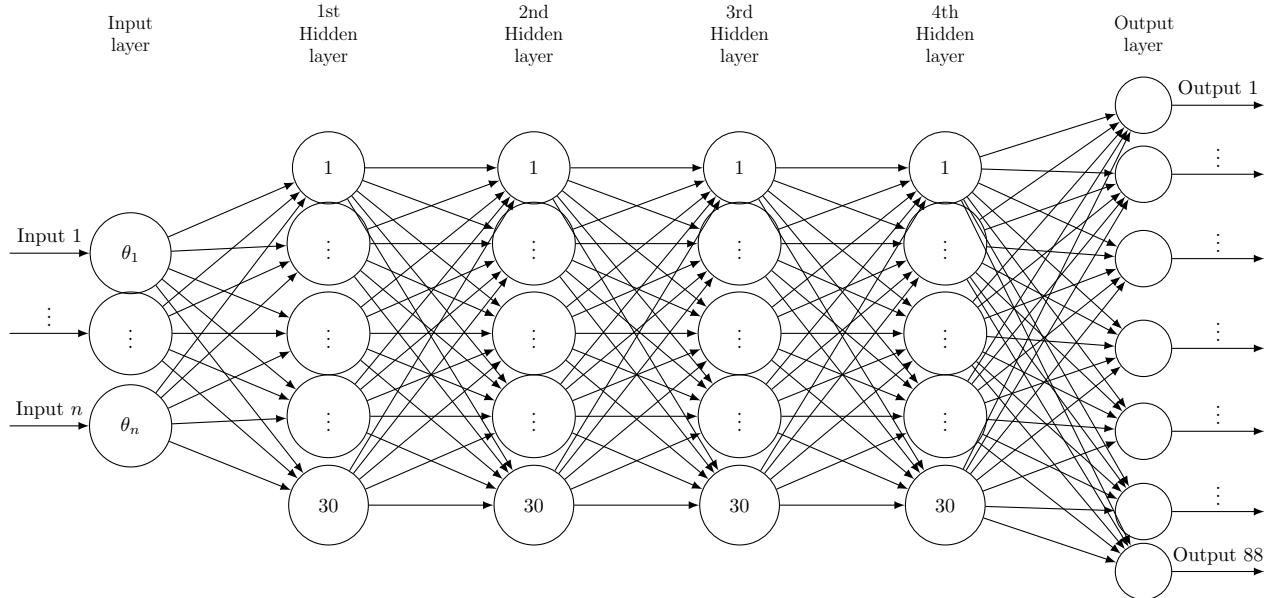


Figure 3: Our neural network architecture with 4 hidden layers and 30 neurons on each hidden layer, with the model parameters of the respective model on the input layer and with the 8×11 implied volatility grid on the output layer.

3.2.2 Training of the approximation network

We follow the common features of optimization techniques and choose mini-batches, as described in Goodfellow, Bengio and Courville [32]. Typical batch size values range from around 10 to 100. In our case we started with small batch sizes and increased the batch size until training performance consistently reached a plateau. Finally, we chose batch sizes of 32, as performance is similar for batch sizes above this level, and larger batch sizes increase computation time by computing a larger number of gradients at a time.

In our training design, we use a number of regularisation techniques to speed up convergence of the training, to avoid overfitting and improve the network performance.

1) Early stopping: We choose the number of epochs as 200 and stop updating network parameters if the error has not improved in the test set for 25 steps.

2) Normalisation of model parameters: Usually, model parameters are restricted to a given domain i.e. $\theta \in [\theta_{\min}, \theta_{\max}]$. Then, we perform the following normalisation transform:

$$\frac{2\theta - (\theta_{\max} + \theta_{\min})}{\theta_{\max} - \theta_{\min}} \in [-1, 1].$$

3) Normalisation of implied volatilities: The normalisation of implied volatilities is a more delicate matter, since $\sigma_{BS}(T, k, \theta^{train}) \in [0, \infty)$, for each T and k . Therefore, we choose to normalise the surface subtracting the sample empirical mean and dividing by the sample standard deviation.

3.3 The calibration step

Once the pricing map approximation \tilde{F} for the implied volatility is found, only the calibration step in (2) is left to solve. In general, for financial models the pricing map F^* is assumed to be smooth (at least C^1 differentiable) with respect to all its input parameters θ .

Gradient-based optimizers

A standard necessary first order condition for optimality in (2) is that

$$\nabla^\theta \delta \left(\tilde{F}(\mathcal{M}(\theta), \zeta), \mathcal{P}^{MKT}(\zeta) \right) = 0, \quad (19)$$

provided that the objective function is smooth. Then, a natural update rule is to move along the gradient via Gradient Descent i.e.

$$\theta_{i+1} = \theta_i - \lambda \nabla^\theta \delta \left(\tilde{F}(\mathcal{M}(\theta_i), \zeta), \mathcal{P}^{MKT}(\zeta) \right), \quad \lambda > 0. \quad (20)$$

A common feature of gradient based optimization methods building on (20) is the use of the gradient $\nabla^\theta \delta \left(\tilde{F}(\mathcal{M}(\theta), \zeta), \mathcal{P}^{MKT}(\zeta) \right)$, hence its correct and precise computation is crucial for subsequent success. Examples of such algorithms, are Levenberg-Marquardt [53, 54], Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm [58], L-BFGS-B [70] and SLSQP [50]. The main

advantage of the aforementioned methods is the quick convergence towards condition (19). However, (19) only gives necessary and not sufficient conditions for optimality, hence special care must be taken with non-convex problems.

Remark 6. Notably, making use of Theorem 2 we use a smooth activation functions in order to guarantee $\nabla^\theta \tilde{P} \approx \nabla^\theta \tilde{F}$

Gradient-free optimizers

Gradient-free optimization algorithms are gaining popularity due to the increasing number of high dimensional nonlinear, non-differentiable and/or non-convex problems flourishing in many scientific fields such as biology, physics or engineering. As the name suggests, gradient-free algorithms make no C^1 assumption on the objective function. Perhaps, the most well known example is the Simplex based Nelder-Mead [57] algorithm. However, there are many other methods such as COBYLA [60] or Differential Evolution [68] and we refer the reader to [61] for an excellent review on gradient-free methods. The main advantage of these methods is the ability to find global solutions in (2) regardless of the objective function. In contrast, the main drawback is a higher computational cost compared to gradient methods.

To conclude, we summarise the advantages of each approach in Table 1.

	Gradient-based	Gradient-free
Convergence Speed	Very Fast	Slow
Global Solution	Depends on problem	Always
Smooth activation function needed	Yes to apply Theorem 2	No
Accurate gradient approximation needed	Yes	No

Table 1: Comparison of Gradient vs. Gradient-free methods.

4 Numerical experiments

In our numerical experiments we demonstrate that the accuracy of the approximation network indeed remains within the accuracy of the Monte Carlo error bounds and proclaimed in the introductory sections' objectives. For this we first compute the benchmark Monte Carlo errors in Figures 4-5 and compare this with the neural network approximation errors in Figures 6 and 7. For this separation into steps (i) and (ii) to be computationally meaningful, the neural network approximation has to be a reasonably accurate approximation of the true pricing functionals and each functional evaluation (i.e. evaluation an option price for a given price and maturity) should have a considerable speed-up in comparison to the original numerical method. In this section we demonstrate that our network achieves both of these goals.

4.1 Numerical accuracy and speed of the price approximation for vanillas

As mentioned in Section 2 one crucial difference that sets apart this work from direct neural network approaches, as pioneered by Hernandez [36], is the separation of (i) the implied volatility approximation function, mapping from parameters of the stochastic volatility model to the implied volatility surface—thereby bypassing the need for expensive Monte-Carlo simulations—and (ii) the calibration procedure, which (after this separation) becomes a simple deterministic optimisation problem. As outlined in Section 2.3 our aim for the Step **(i)** in the two-step training approach is to achieve a considerable speedup per functional evaluation of option prices while maintaining the numerical accuracy of the original pricer. Here we demonstrate how our NN training for Step **(i)** achieves these goals outlined in Section 2.3:

1. Approximation accuracy: here we compare the error of the approximation network error to the error of Monte Carlo evaluations. We compute Monte Carlo prices with 60,000 paths as reference at the nodes where we compute the implied volatility grid using Algorithm 3.5 in Horvath, Jacquier and Muguruza [41]. In Figures 4 and 5 the approximation accuracy of the Monte Carlo method for the full implied volatility surface is computed using pointwise relative error with respect to the 95% Monte Carlo confidence interval. Figures 6 and 7 demonstrate that the same approximation accuracy for the neural network is achieved as for the Monte Carlo approximation (i.e. within a few basis points). For reference, the spread on options is around 0.2% in implied volatility terms for the most liquid and those below a year. This translates into 1% relative error for a implied volatility of 20%.
2. Approximation speed: Table 2 shows the CPU computation time per functional evaluation of a full surface under two different models; rBergomi 3 and 1 Factor Bergomi 6 (for a reminder see Section 4.1.1 for details).

	MC Pricing 1F Bergomi Full Surface	MC Pricing rBergomi Full Surface	NN Pricing Full Surface	NN Gradient Full Surface	Speed up NN vs. MC
Piecewise constant forward variance	300,000 μ s	500,000 μ s	30.9 μ s	113 μ s	9,000 – 16,000

Table 2: Computational time of pricing map (entire implied volatility surface) and gradients via Neural Network approximation and Monte Carlo (MC). If the forward variance curve is a constant value, then the speed-up is even more pronounced

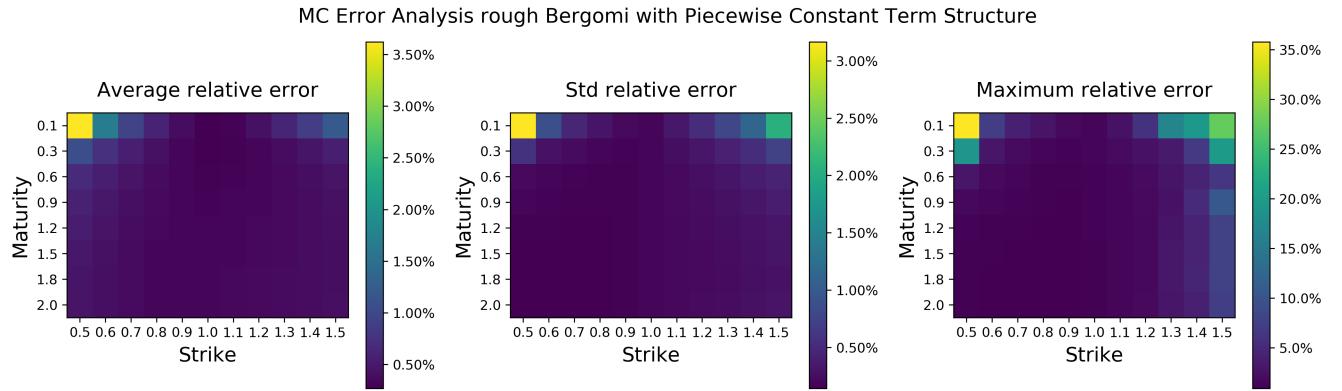


Figure 4: As benchmark we recall average relative errors of Monte Carlo prices computed across 80,000 random parameter combinations of the Rough Bergomi model. Relative errors are given in terms of Average-Standard Deviation-Maximum (Left-Middle-Right) on implied volatility surfaces in the Rough Bergomi model, computed using 95% confidence intervals.

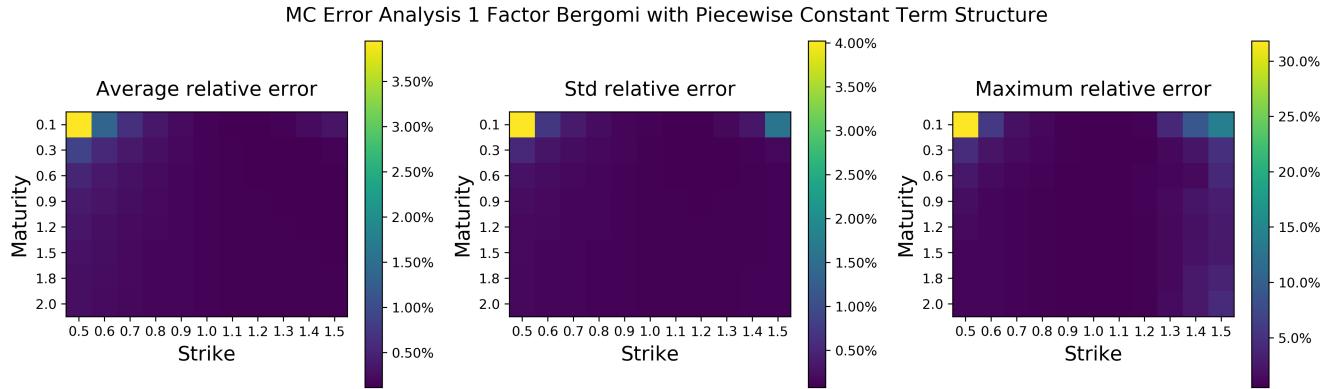


Figure 5: As benchmark we recall average relative errors of Monte Carlo prices computed across 80,000 random parameter combinations of the 1 Factor Bergomi model. Relative errors are given in terms of Average-Standard Deviation-Maximum (Left-Middle-Right) on implied volatility surfaces in the 1 Factor Bergomi model, computed using 95% confidence intervals.

4.1.1 Neural network price approximation in (rough) Bergomi models with piecewise constant forward variance curve

We consider a piecewise constant forward variance curve $\xi_0(t) = \sum_{i=1}^n \xi_i \mathbf{1}_{\{t_{i-1} < t < t_i\}}$ where $t_0 = 0 < t_1 < \dots < t_n$ and $\{t_i\}_i = 1, \dots, n$ are the option maturity dates ($n = 8$ in our case). This is the modelling approach suggested by Bergomi [11]. We will consider again the rough Bergomi 3 and 1

Factor Bergomi models 6

- Normalized parameters as input and normalised implied volatilities as output
- 4 hidden layers with 30 neurons and *Elu* activation function
- Output layer with *Linear* activation function
- Total number of parameters: 6808
- Train Set: 68,000 and Test Set: 12,000
- Rough Bergomi sample: $(\xi_0, \nu, \rho, H) \in \mathcal{U}[0.01, 0.16]^8 \times \mathcal{U}[0.5, 4.0] \times \mathcal{U}[-0.95, -0.1] \times \mathcal{U}[0.025, 0.5]$
- 1 Factor Bergomi sample: $(\xi_0, \nu, \rho, \beta) \in \mathcal{U}[0.01, 0.16]^8 \times \mathcal{U}[0.5, 4.0] \times \mathcal{U}[-0.95, -0.1] \times \mathcal{U}[0, 10]$
- strikes = {0.5, 0.6, 0.7, 0.8, 0.9, 1, 1.1, 1.2, 1.3, 1.4, 1.5}
- maturities = {0.1, 0.3, 0.6, 0.9, 1.2, 1.5, 1.8, 2.0}
- Training data samples of Input-Output pairs are computed using Algorithm 3.5 in Horvath, Jacquier and Muguruza [41] with 60,000 sample paths and the spot martingale condition i.e. $\mathbb{E}[S_t] = S_0$, $t \geq 0$ as control variate.

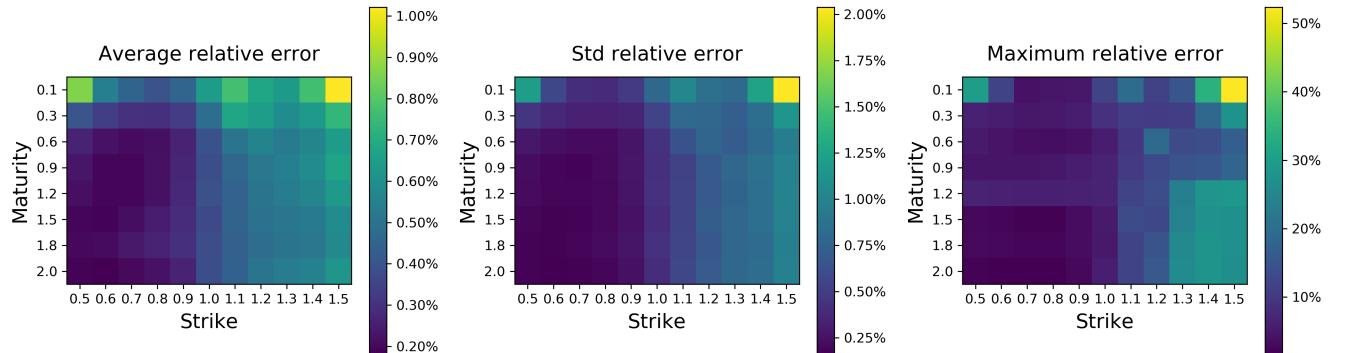


Figure 6: We compare surface relative errors of the neural network approximator against the Monte Carlo benchmark across all training data (68,000 random parameter combinations) in the rough Bergomi model. Relative errors are given in terms of Average-Standard Deviation-Maximum (Left-Middle-Right).

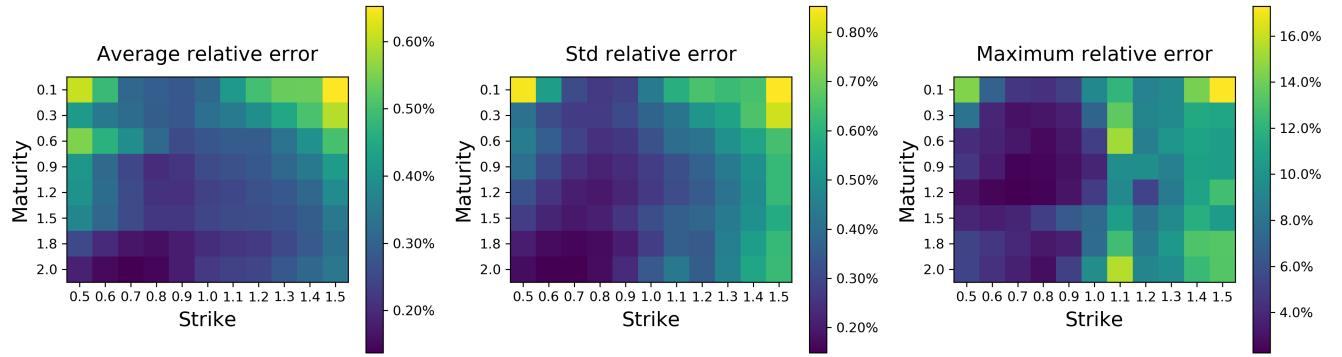


Figure 7: We compare surface relative errors of the neural network approximator against the Monte Carlo benchmark across all training data (68,000 random parameter combinations) in the 1 Factor Bergomi model. Relative errors are given in terms of Average–Standard Deviation–Maximum (Left–Middle–Right).

Figures 6 and 7 show that the average (across all parameter combinations) relative error between neural network and Monte Carlo approximations is far less than 0.5% consistently (left image in Figures 6 and 7) with a standard deviation of less than 1% (middle image in Figures 6 and 7). The maximum relative error goes as far as 25%. We conclude that the methodology generalises adequately to the case of non-constant forward variances, by showing the same error behaviour.

4.2 Calibration speed and accuracy for implied volatility surfaces

Figure 8 reports average calibration times on test set for different parameter combinations on each of the models analysed. We conclude that gradient-based optimizers outperform (in terms of speed) gradient-free ones. Moreover, in Figure 8 one observes that computational times in gradient-free methods are heavily affected by the dimension of the parameter space, i.e. flat forward variances are much quicker to calibrate than piecewise constant ones. We find that Lavenberg-Marquardt is the most balanced optimizer in terms of speed/convergence and we choose to perform further experiments with this optimizer. The reader is encouraged to keep in mind that a wide range of optimizers is available for the calibration and the optimal selection of one is left for future research.

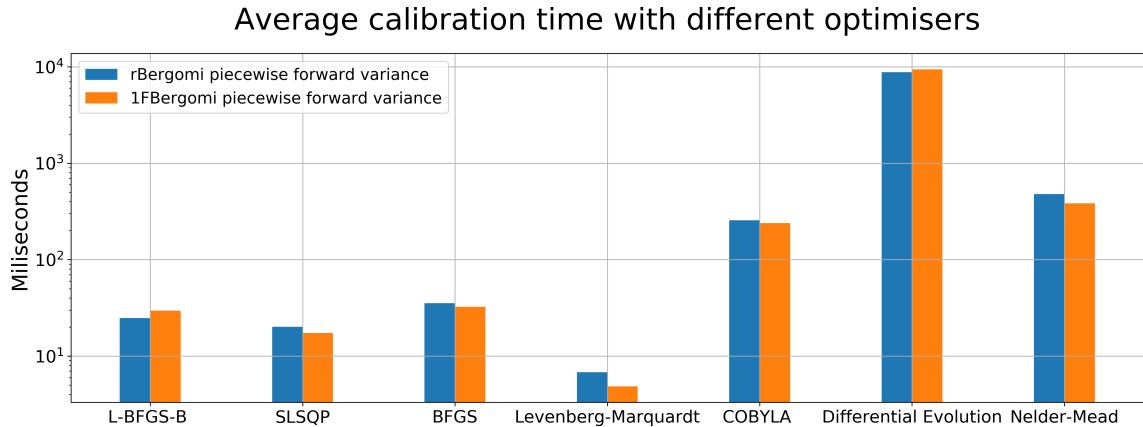


Figure 8: Average calibrations times for all models using a range of optimizers.

In order to assess the accuracy, we report the calibrated model parameters $\hat{\theta}$ compared to the synthetically generated data with the set of parameters $\bar{\theta}$ that was chosen for the generation of our synthetic data. We measure the accuracy of the calibration via parameter relative error i.e.

$$E_R(\hat{\theta}) = \frac{|\hat{\theta} - \bar{\theta}|}{|\bar{\theta}|}$$

as well as the root mean square error (RMSE) with respect to the original surface i.e.

$$\text{RMSE}(\hat{\theta}) = \sqrt{\sum_{i=1}^n \sum_{j=1}^m (\tilde{F}(\hat{\theta})_{ij} - \sigma_{BS}^{MKT}(T_i, k_j))^2}.$$

Therefore, on one hand a measure of good calibration is a small RMSE. On the other hand, a measure of parameter sensitivity on a given model is the combined result of RMSE and parameter relative error.

4.2.1 A calibration experiment with simulated data in (rough) Bergomi models with piecewise constant forward variances

We consider the rough Bergomi model (3) and the Bergomi model (6) with a piecewise constant term-structure of forward variances. Figures 9 and 10 show that the 99% quantile of the RMSE is below 1% and shows that the Neural Network approach generalises properly to the piecewise constant forward variance. Again, we find that the largest relative errors per parameter are concentrated around 0, consequence of using the relative error as measure. This suggests a successful generalisation to general forward variances, which to our knowledge has not been addressed before by means of neural networks or machine learning techniques.

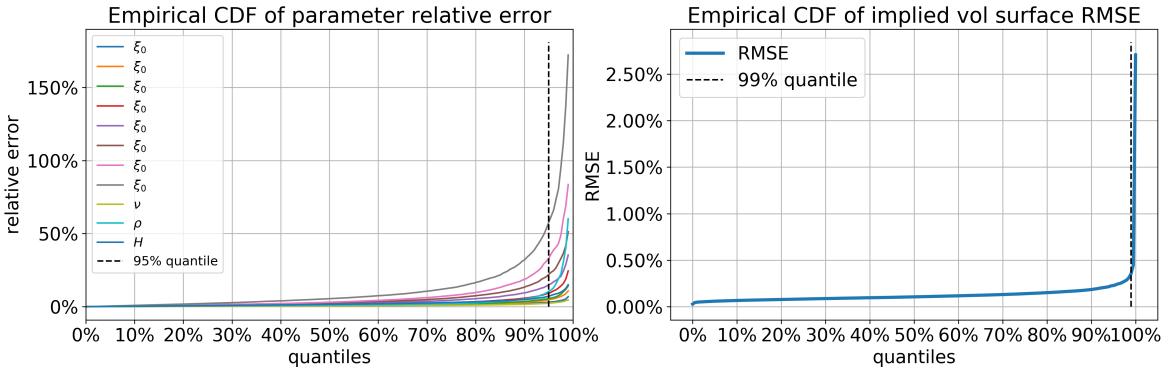


Figure 9: Cumulative Distribution Function (CDF) of Rough Bergomi parameter relative errors (left) and RMSE (right) after Levengerg-Marquardt calibration across test set random parameter combinations.

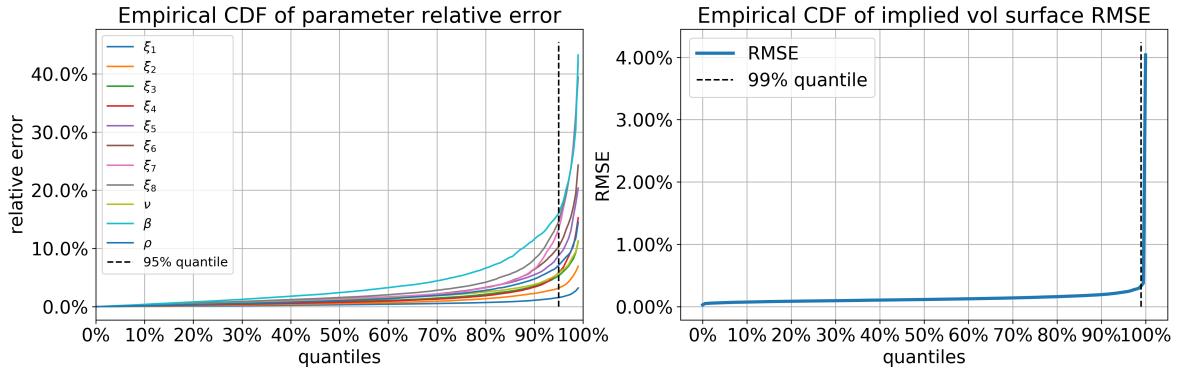


Figure 10: Cumulative Distribution Function (CDF) of 1 Factor Bergomi parameter relative errors (left) and RMSE (right) after Levengerg-Marquardt calibration across test set random parameter combinations.

4.2.2 Calibration in the rough Bergomi model with historical data

As previously mentioned, the natural use of neural network approximators is the model calibration to historical data. We discussed that as long as the approximation is accurate, the calibration task should be performed within the given tolerance. Furthermore, one should expect such tolerance to be aligned with the neural network accuracy obtained in both training and test sets.

In this section we will perform a historical calibration using the neural network approximation and compare it with that of the brute force monte carlo calibration. Precisely we seek to solve the

following optimisation problem for the rough Bergomi model

$$\hat{\theta}^{rBergomi} := \operatorname{argmin}_{\theta^{rBergomi} \in \Theta^{rBergomi}} \sum_{i=1}^5 \sum_{j=1}^9 (\tilde{F}(\theta)_{ij} - \sigma_{BS}^{MKT}(T_i, k_j))^2.$$

where $\theta^{rBergomi} = (\xi_1, \xi_2, \xi_3, \xi_4, \xi_5, \nu, \rho, H)$ and $\Theta^{rBergomi} = [0.01, 0.25]^5 \times [0.5, 4] \times [-1, 0] \times [0.025, 0.5]$. As for the time grid we choose

$$(T_1, T_2, T_3, T_4, T_5) := \frac{1}{12} \times (1, 3, 6, 9, 12)$$

and for the strike grid

$$k_i := 0.85 + (i - 1) \times 0.05 \quad \text{for } i = 1, \dots, 9.$$

We consider SPX market smiles between 01/01/2010 and 18/03/2019 on the pre-specified time and strike grid. Figure 11 shows the historical evolution of rough Bergomi parameters calibrated to SPX using the neural network price. In particular we note that $H < \frac{1}{2}$ as previously discussed in many academic papers [1, 2, 7, 6, 8, 10, 22, 26, 28, 44, 40, 45], moreover we may confirm that under \mathbb{Q} , $H \in [0.1, 0.15]$ as found in Gatheral, Jaisson and Rosenbaum [28] under \mathbb{P} . Figure 12, benchmarks the NN optimal fit using Levenberg-Marquardt and Differential Evolution against a brute force MC calibration via Levenberg-Marquardt. Again, we find that the discrepancy between both is below 0.2% most of the time and conclude that the Differential Evolution algorithm does outperform the Levenberg-Marquardt. This in turn, suggests that the neural network might not be precise enough on first order derivatives. This observation, is left as an open question for further research. Perhaps surprisingly, we sometimes obtain a better fit using the neural network than the MC pricing itself. This could be caused by the fact that gradients in the neural network are exact, whereas when using MC brute force calibration we resort to finite differences to approximate gradients.

Evolution of Model Parameters in the Rough Bergomi Model Calibrated to SPX

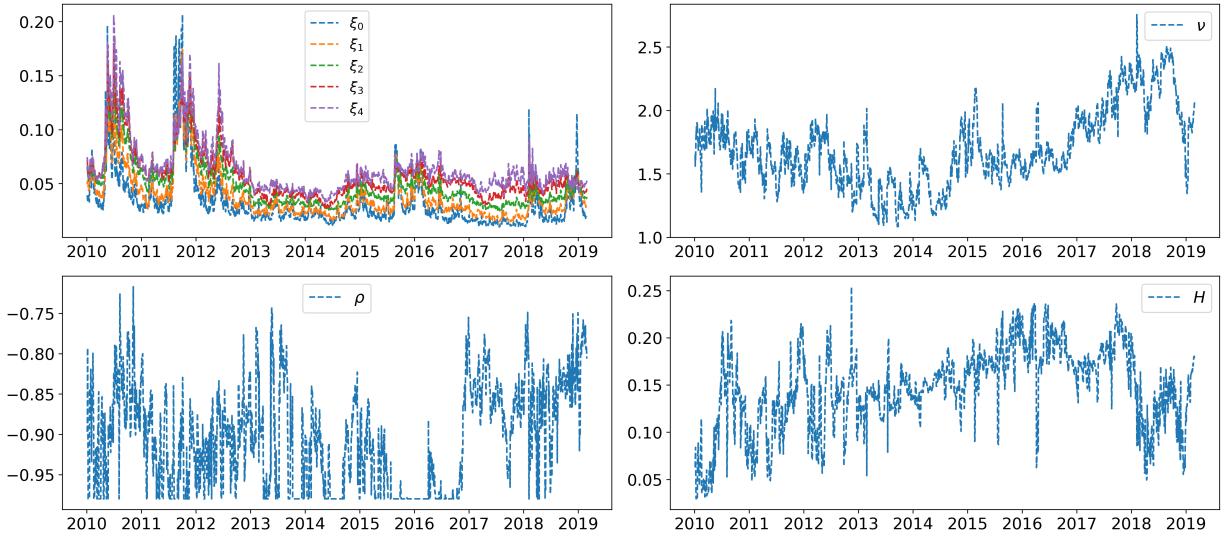


Figure 11: Historical Evolution of parameters in the rough Bergomi model with a piecewise constant forward variance term structure calibrated on SPX

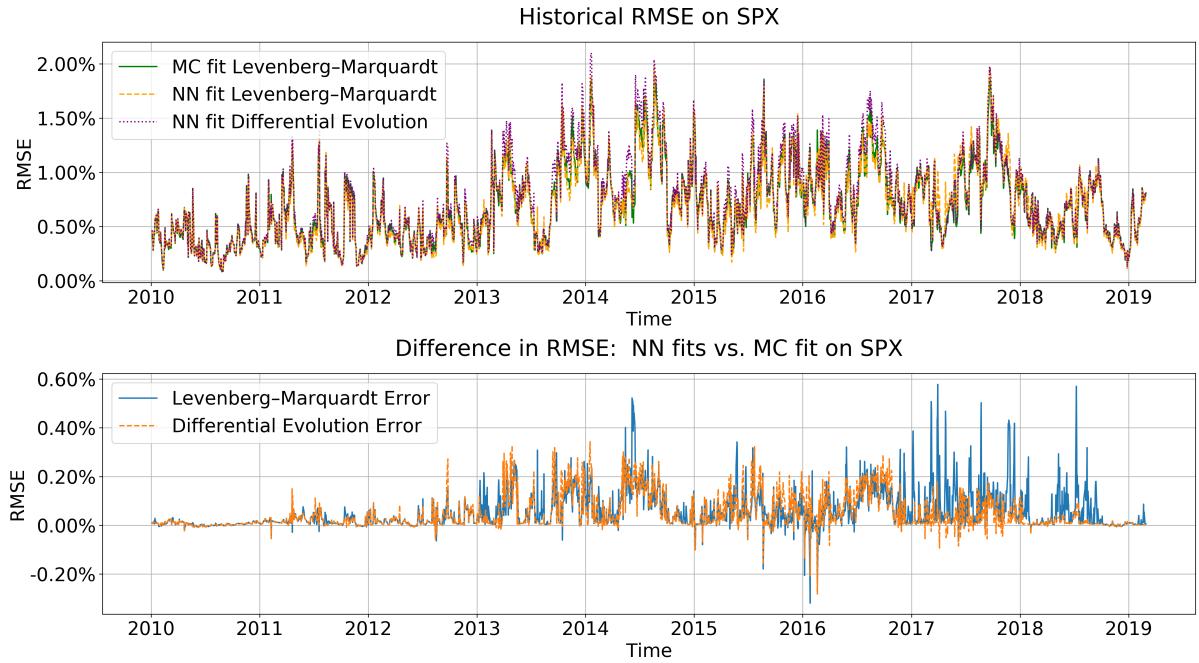


Figure 12: The image above compares historical RMSE obtained by the neural network best fit via Levenberg-Marquardt (dashed orange line) and Differential Evolution (dotted purple line) against the brute force MC calibration (green line) via Levenberg-Marquardt. Picture below shows the difference against MC brute force calibration.

4.3 Numerical experiments with barrier options in the rough Bergomi model

In this section we show that our methodology can be easily extended to exotic options. To do so we test our image-based approach on digital barrier options. We follow the same architecture and experimental design described in Section 4.1 for the rough Bergomi model. As described in Section 3.1.2 we adapt the objective function to the payoffs given in (17) and (18) and replace the strike grid by a barrier level grid. Figure 13 confirm the accuracy of the neural network approximation with average absolute errors of less than 10bps with standard deviation of 10bps.

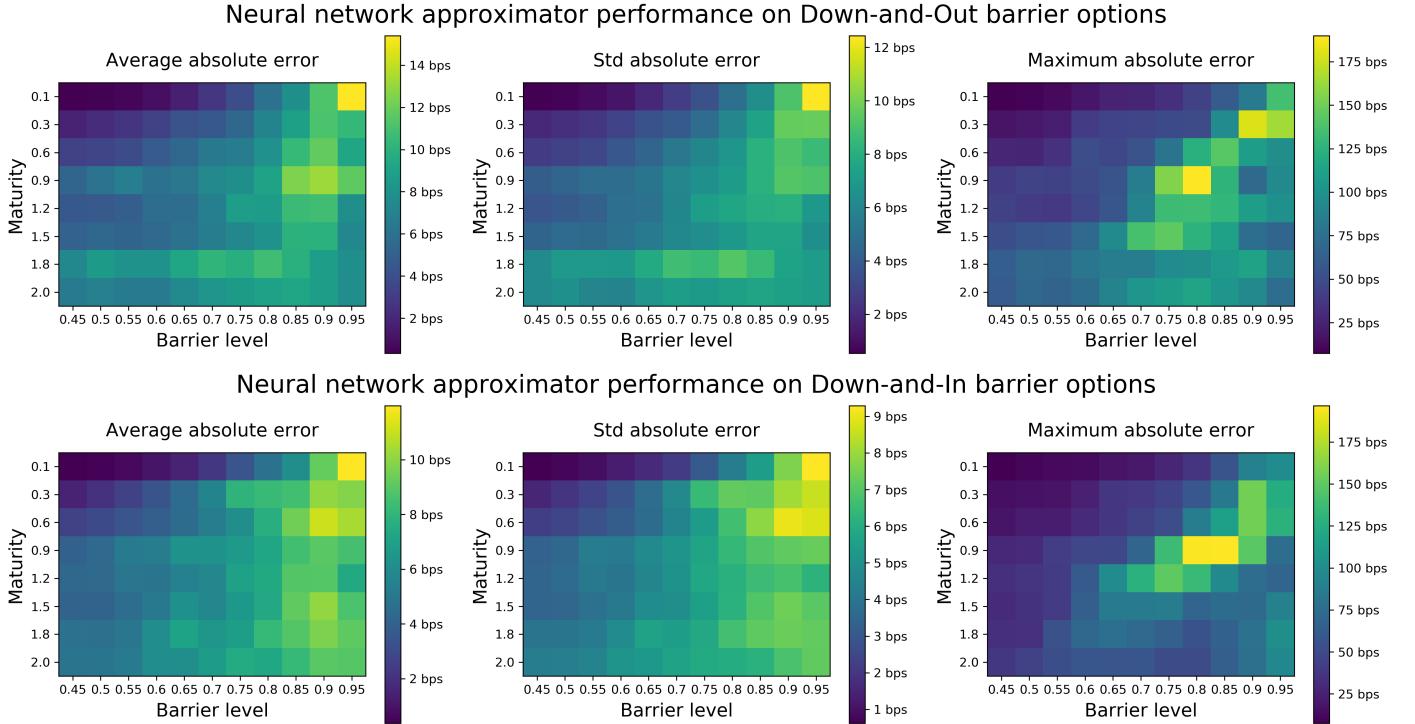


Figure 13: Picture above: Down-and-Out neural network absolute error analysis on test set. Picture below: Down-and-In neural network absolute error analysis on test set

5 Conclusions and outlook: “best-fit” models

To sum up, neural networks have the potential to efficiently approximate complex functions, which are difficult to represent and time-consuming to evaluate by other means. Using deep neural networks, as we will do here, to approximate the pricing map (or equivalently the implied volatility mapping) from parameters of traditional models to shapes of the implied volatility surface represented by grid of implied volatility values speeds up each functional evaluation, while maintaining

control over reliability and interpretability of network outputs. The implicit grid based approach that we advocate here, also allows further applications that opens up further landscapes for financial modelling.

Potential applications and outlook towards mixture of “expert” models: In the previous sections we set up a powerful approximation method to closely approximate implied volatilities under different stochastic models and highlighted that the choice of the objective function (evaluation of the surface on a grid, inspired by pixels of an image) was crucial for the performance of the network. Now we are interested in the inverse task and ask whether a neural network—trained by this objective function to multiple stochastic models simultaneously—can identify which stochastic model a given set of data comes from. By doing so, potential applications we have in mind are twofold:

- (1) Ultimately we are interested in which model (or what mixture of existing stochastic models) best describes the market.
- (2) From a more academic and less practical perspective, we are interested whether and to what extent is it possible to “translate” parameters of one stochastic model to parameters of another.

We conduct a further, preliminary experiment as a proof of concept in the classification setting. We train a further neural network to identify which of three given stochastic volatility model generated a given implied volatility surface.

Training procedure: Implied volatility surfaces in this experiment were generated by the Heston, Bergomi and rough Bergomi models (see Section 2.1 for a reminder). For each volatility surface, a “flag” was assigned corresponding to the model (eg: 1 for Heston, 2 for Bergomi and 3 for rough Bergomi). The training set thus consists of surfaces of the form: $(\Sigma_{BS}^{\mathcal{M}(\theta)}, I)$, where \mathcal{M} is one of the three models $\mathcal{M}^{\text{Heston}}$, $\mathcal{M}^{\text{Bergomi}}$, $\mathcal{M}^{\text{rBergomi}}$, θ an admissible combination of parameters for that model (thus in Θ^{Heston} , Θ^{Bergomi} or Θ^{rBergomi}) and I the flag identifying the model which generated the surface ($I = 1$ if $\mathcal{M} = \mathcal{M}^{\text{Heston}}$, $I = 2$ if $\mathcal{M} = \mathcal{M}^{\text{Bergomi}}$ and $I = 3$ if $\mathcal{M} = \mathcal{M}^{\text{rBergomi}}$).

We define a mixture of these surfaces as $\Sigma^{\mathcal{M}^{\text{Mixture}((a,b,c))}} := a\Sigma^{\mathcal{M}^{\text{Heston}}} + b\Sigma^{\mathcal{M}^{\text{Bergomi}}} + c\Sigma^{\mathcal{M}^{\text{rough Bergomi}}}$, where $a, b, c \geq 0$ and $a + b + c = 1$. So far the training is suitable for recognition of a single model surface (either $a = 0, b = 0, c = 1$, $a = 0, b = 1, c = 0$ or $a = 1, b = 0, c = 0$). To generalise this to mixtures, we randomly select surfaces (one from each model) and compute the mixture surface $\Sigma^{\mathcal{M}^{\text{Mixture}((a,b,c))}} = a\Sigma^{\mathcal{M}^{\text{Heston}}} + b\Sigma^{\mathcal{M}^{\text{Bergomi}}} + c\Sigma^{\mathcal{M}^{\text{rough Bergomi}}}$. The corresponding probabilities are $(a, b, c = 1 - a - b)$.

Network Architecture: The classifier is a small, fully connected feedforward network for the same reasons as those outlined in section 3.2. The network is composed of 2 hidden layers (of 100 and 50 output nodes respectively) with exponentially linear activation functions and an output layer with a softmax activation function. Thus, the output of the network represents the probabilities of a given surface belonging to a particular model. We used stochastic gradient descent with 20 epochs to minimize cross-entropy (the cross-entropy of two discrete distributions (p, q) with K possible distinct values is $H(p, q) := -\sum_{1 \leq i \leq K} p_i \log q_i$).

A numerical experiment on model recognition: We report on one of many experiments here as a proof of concept: We test the method on mixtures of rough Bergomi and Heston surfaces

(hence setting $b = 0$ in the training). To vary the type of mixtures generated, we chose $a \in \{0, 0.1, \dots, 0.9, 1\}$. For each a , the mixture surface is computed as the convex combination of a randomly chosen surface from the rough Bergomi and the Heston model, and repeated 20 times. The training set has 320,000 surfaces. To further test the robustness of the model, validation surfaces were generated using a finer grid of mixture parameters: $a \in \{0, 0.05, \dots, 0.95, 1\}$. In total, the validation set is made up of 105,000 surfaces.

We report the classifiers' effectiveness and comment on the results in Figure 5.

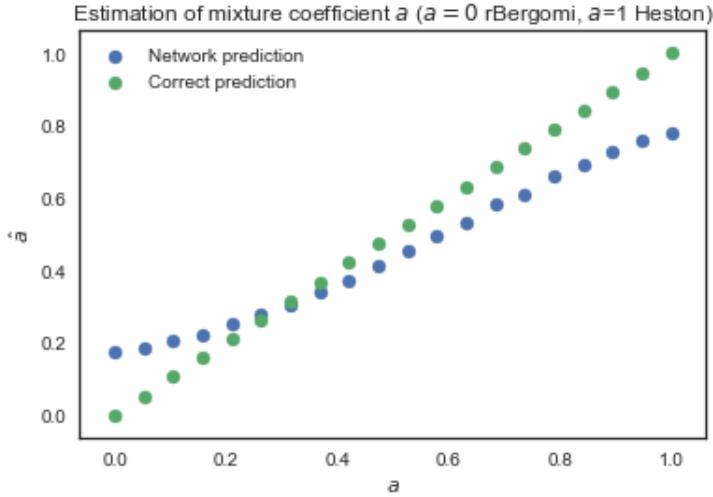


Figure 14: Error of the neural network classifier depending on the mixture coefficient a . Each point of the plot corresponds to the average estimated coefficient by the neural network for all mixture surfaces with a given a . For example, for $a = 0$, the surfaces are generated from the rough Bergomi model. For each parameter combination from those surfaces, we compute the predicted mixture coefficient and average all of them over the validation set to report \hat{a} . The network never sets the mixture coefficient very close to 1 or 0, attributing the surface to one specific model. This may be explained using Bayesian reasoning.

References

- [1] E. Alòs, D. García-Lorite and A. Muguruza. On smile properties of volatility derivatives and exotic products: understanding the VIX skew. arXiv:1808.03610, 2018.
- [2] E. Alòs, J. León and J. Vives. On the short-time behavior of the implied volatility for jump-diffusion models with stochastic volatility. *Finance and Stochastics*, 11(4), 571-589, 2007.
- [3] A. Antonov, M. Konikov and M. Spector. SABR spreads its wings. Risk (August issue), pp. 58-63, 2013.
- [4] M. Avellaneda, A. Carelli and F. Stella. Following the Bayes path to option pricing. *Journal of Computational Intelligence in Finance*, 1998.

- [5] A.R. Barron. Approximation and estimation bounds for artificial neural networks. *Machine Learning*. Vol.14:1,1994 Pages 115-133 .
- [6] C. Bayer, P. Friz, P. Gassiat, J. Martin and B. Stemper. A regularity structure for rough volatility. arXiv:1710.07481, 2017.
- [7] C. Bayer, P. Friz and J. Gatheral. Pricing under rough volatility. *Quantitative Finance*, 16(6): 1-18, 2015.
- [8] C. Bayer, P. Friz, A. Gulisashvili, B. Horvath and B. Stemper. Short-time near the money skew in rough fractional stochastic volatility models. arXiv:1703.05132, 2017.
- [9] C. Bayer and B. Stemper. Deep calibration of rough stochastic volatility models. *Preprint*, arXiv:1810.03399
- [10] M. Bennedsen, A. Lunde and M.S. Pakkanen. Hybrid scheme for Brownian semistationary processes. *Finance and Stochastics*, 21(4): 931-965, 2017.
- [11] L. Bergomi. Stochastic Volatility Modeling. Chapman & Hall/CRC financial mathematical series. *Chapman & Hall/CRC*, 2015.
- [12] A. Broström and R. Kristiansson. Exotic Derivatives and Deep Learning. *Unpublished Thesis*, KTH Royal Institute of Technology, School of Engineering Sciences, Stockholm, Sweden, 2018.
- [13] H. Buehler, L. Gonon, J. Teichmann and B. Wood. Deep Hedging. *Preprint*, arXiv:1802.03042, 2018.
- [14] J. Cao, J. Chen and J.C. Hull. A Neural Network Approach to Understanding Implied Volatility Movements. SSRN:3288067, 2018.
- [15] B. Chen, C. W. osterlee and H. Van Der Weide. Efficient unbiased simulation scheme for the SABR stochastic volatility model, 2011.
- [16] D. Clevert, T. Unterthiner, S. Hochreiter. Fast and accurate deep network learning by exponential linear units (ELUs). *Preprint*, arXiv:1511.07289, 2015.
- [17] J. De Spiegeleer, D. Madan, S. Reyners and W. Schoutens. Machine learning for quantitative finance: Fast derivative pricing, hedging and fitting. SSRN:3191050, 2018.
- [18] G. Dimitroff, D. Röder and C. P. Fries. Volatility model calibration with convolutional neural networks. *Preprint*, SSRN:3252432, 2018.
- [19] M. Duembgen and L.C.G. Rogers. Estimate Nothing. *Quantitative Finance*, 14(12), pp.2065-2072, 2014.
- [20] R. Eldan and O. Shamir. The power of depth for feedforward neural neworks. *JMLR: Workshop and Conference Proceedings* Vol 49:1-34, 2016.
- [21] C. Doléans-Dade. Quelques applications de la formule de changement de variables pour les semimartingales. *Z. Wahrscheinlichkeitstheorie verwandte Gebiete*, Vol 16: 181-194, 1970.
- [22] O. El Euch and M. Rosenbaum. The characteristic function of rough Heston models. To appear in *Mathematical Finance*.

- [23] O. El Euch and M. Rosenbaum. Perfect hedging in rough Heston models, to appear in *The Annals of Applied Probability*, 2018.
- [24] J. Friedman, R. Tibshirani and T. Hastie. *The Elements of Statistical Learning*. Springer New York Inc, 2001.
- [25] R. Ferguson and A. Green. Deeply learning derivatives. Preprint arXiv:1809.02233, 2018.
- [26] M. Fukasawa. Asymptotic analysis for stochastic volatility: martingale expansion. *Finance and Stochastics*, 15: 635-654, 2011.
- [27] J. Gatheral. A parsimonious arbitrage-free implied volatility parameterization with application to the valuation of volatility derivatives, *Presentation at Global Derivatives*, 2004.
- [28] J. Gatheral, T. Jaisson and M. Rosenbaum. Volatility is rough. *Quantitative Finance*, 18(6): 933-949, 2018.
- [29] J. Gatheral, A. Jacquier. Arbitrage-free SVI volatility surfaces. *Quantitative Finance*, 14(1): 59-71, 2014
- [30] A. Gulisashvili, B. Horvath and A. Jacquier. Mass at zero in the uncorrelated SABR model. *Quantitative Finance*, 18(10): 1753-1765, 2018.
- [31] A. Gulisashvili, B. Horvath and A. Jacquier. On the probability of hitting the boundary for Brownian motions on the SABR plane. *Electronic Communications in Probability*, 21(75): 1-13, 2016.
- [32] I. Goodfellow, Y. Bengio and A. Courville. *Deep Learning*. MIT Press, 2016.
- [33] P. Hagan, D. Kumar, A. Lesniewski, and D. Woodward. Managing smile risk. *Wilmott Magazine*, September issue: 84-108, 2002.
- [34] P. Hagan, A. Lesniewski, and D. Woodward. Probability distribution in the SABR model of stochastic volatility. Large Deviations and Asymptotic Methods in Finance, *Springer Proceedings in Mathematics and Statistics*, 110, 2015.
- [35] S. Hendriks, C. Martini The Extended SSVI Volatility Surface. *Preprint*, SSRN:2971502, 2017.
- [36] A. Hernandez. Model calibration with neural networks. *Risk*, 2017.
- [37] K. Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251-257, 1991.
- [38] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359-366, 1989.
- [39] K. Hornik, M. Stinchcombe and H. White. Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks. *Neural Networks*. Vol. 3:11, 1990.
- [40] B. Horvath, A. Jacquier and C. Lacombe. Asymptotic behaviour of randomised fractional volatility models. arXiv:1708.01121, 2017.

- [41] B. Horvath, A. Jacquier and A. Muguruza. Functional central limit theorems for rough volatility. arXiv:1711.03078, 2017.
- [42] B. Horvath, O. Reichmann. Dirichlet Forms and Finite Element Methods for the SABR Model. *SIAM Journal on Financial Mathematics*, p. 716-754(2), May 2018.
Hutchinson, James M., Andrew W. Lo, and Tomaso Poggio. "A nonparametric approach to pricing and hedging derivative securities via learning networks." *The Journal of Finance* 49.3 (1994): 851-889.
- [43] J.M. Hutchinson, A.W. Lo, T. Poggio. A nonparametric approach to pricing and hedging derivative securities via learning networks. *The Journal of Finance*, 49 p. 851-889(3), 1994.
- [44] A. Jacquier, C. Martini and A. Muguruza. On VIX futures in the rough Bergomi model. *Quantitative Finance*, 18(1): 45-61, 2018.
- [45] A. Jacquier, M. Pakkanen and H. Stone. Pathwise large deviations for the rough Bergomi model. *Journal of Applied Probability*, 55(4): pp.1078-1092, 2018.
- [46] A. Jentzen, B. Kuckuck, A. Neufeld, P. von Wurstemberger. Strong error analysis for stochastic gradient descent optimization algorithms, Preprint arXiv:1801.09324 ,2018.
- [47] S. Ioffe and C. Szegedy. Batch normalisation: Accelerating deep network training by reducing internal covariate shift. *Preprint*, arXiv:1502.03167, 2015.
- [48] D.P. Kingman and J. Ba, Adam: A Method for Stochastic Optimization. *Conference paper*, 3rd International Conference for Learning Representations, 2015.
- [49] A. Kondratyev. Learning curve dynamics with artificial neural networks. *Preprint*, SSRN:3041232, 2018.
- [50] D. Kraft. A Software Package for Sequential Quadratic Programming. *DFVLR-FB* pp.88-28, 1988.
- [51] G. Kutyniok, H. Bölcskei, P. Grohs and P. Petersen, Optimal approximation with sparsely connected deep neural networks, *Preprint* arXiv:1705.01714, 2017.
- [52] A. Leitao Rodriguez, L.A. Grzelak and C.W. Oosterlee. On an efficient multiple time step Monte Carlo simulation of the SABR model. *Quantitative Finance*, 17(10), pp.1549-1565, 2017.
- [53] K. Levenberg. A Method for the Solution of Certain Non-Linear Problems in Least Squares. *Quarterly of Applied Mathematics*. 2: pp. 164-168, 1944.
- [54] D. Marquardt. An Algorithm for Least-Squares Estimation of Nonlinear Parameters. *SIAM Journal on Applied Mathematics*. 11 (2): pp. 431-441,1963.
- [55] W. A. McGhee. An artificial neural network representation of the SABR stochastic volatility model. *Preprint*, SSRN:3288882, 2018.
- [56] H.N. Mhaskar. Approximation properties of a multilayered feedforward artificial neural network. *Advances in Computational Mathematics*, 1(1): 61-80, 1993.

- [57] J. A. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*. 7: pp. 308-313, 1965.
- [58] J. Nocedal and S. Wright. Numerical Optimization. Springer Series in Operations Research and Financial Engineering. *Springer-Verlag New York*, 2006.
- [59] D. Pedamonti. Comparison of non-linear activation functions for deep neural networks on MNIST classification task. *Preprint*, arXiv:1804.02763
- [60] M. J. D. Powell. A direct search optimization method that models the objective and constraint functions by linear interpolation. Advances in Optimization and Numerical Analysis, eds. S. Gomez and J-P. Hennart, *Kluwer Academic (Dordrecht)*, pp. 51-67, 1994.
- [61] L. M. Rios and N. V. Sahinidis. Derivative-free optimization: a review of algorithms and comparison of software implementations. *Journal of Global Optimization*. Vol. 56: 3 pp. 1247-1293, 2013.
- [62] M. Sabate Vidales, D. Siska and L. Szpruch. Martingale Functional Control variates via Deep Learning *Preprint* 1810.05094, 2018.
- [63] N. Ruiz, S. Schulter and M. Chandraker, Learning to simulate, *Preprint* arXiv:1810.02513, 2018.
- [64] L. Setayeshgar and H. Wang. Large deviations for a feed-forward network, *Advances in Applied Probability*, 43: 2, pp. 545571, 2011.
- [65] U. Shaham, A. Cloninger, and R. R. Coifman. Provable approximation properties for deep neural networks. *Appl. Comput. Harmon. Anal.*, 44(3): 537-557, 2018.
- [66] J. Sirignano and K. Spiliopoulos. Stochastic Gradient Descent in Continuous Time: A Central Limit Theorem. *SIAM J. Finan. Math.*, 8(1), pp. 933-961, 2017.
- [67] H. Stone. Calibrating rough volatility models: a convolutional neural network approach. *Preprint*, arXiv:1812.05315, 2018.
- [68] R. Storn and K. Price. A Simple and Efficient Heuristic for global Optimization over Continuous Spaces. *Journal of Global Optimization*. Vol.11:4, pp341-359, 1997.
- [69] V. N. Vapnik. Statistical Learning Theory. *Wiley-Interscience*, 1998.
- [70] C. Zhu, R. H. Byrd and J. Nocedal. L-BFGS-B: Algorithm 778: L-BFGS-B, FORTRAN routines for large scale bound constrained optimization. *ACM Transactions on Mathematical Software*, 23: 4, pp. 550-560, 1997.