Weierstraß-Institut
für Angewandte Analysis und Stochastik
Leibniz-Institut im Forschungsverbund Berlin e. V.

Preprint ISSN 2198-5855

Deep calibration of rough stochastic volatility models

Christian Bayer[1], Benjamin Stemper[1,2]

submitted: October 19, 2018

[1] Weierstrass Institute
Mohrenstr. 39
10117 Berlin
Germany
E-Mail: christian.bayer@wias-berlin.de
benjamin.stemper@wias-berlin.de

[2] Department of Mathematics
Technical University Berlin
Strasse des 17. Juni 136
10623 Berlin
Germany
E-Mail: stemper@math.tu-berlin.de

No. 2547
Berlin 2018

2010 *Mathematics Subject Classification.* 91G20, 91G60, 91B25, 60G22, 68T20, 62F10, 62F15, 62P05, 62M45.

Key words and phrases. Rough fractional stochastic volatility, option pricing, model calibration, deep learning.

The authors thank Jim Gatheral and Peter K. Friz for some helpful discussions and suggestions and acknowledge financial support through DFG Research Grants BA5484/1 and FR2943/2.

Edited by
Weierstraß-Institut für Angewandte Analysis und Stochastik (WIAS)
Leibniz-Institut im Forschungsverbund Berlin e. V.
Mohrenstraße 39
10117 Berlin
Germany

Fax: +49 30 20372-303
E-Mail: preprint@wias-berlin.de
World Wide Web: http://www.wias-berlin.de/

Deep calibration of rough stochastic volatility models

Christian Bayer, Benjamin Stemper

Abstract

Sparked by Alòs, León und Vives (2007); Fukasawa (2011, 2017); Gatheral, Jaisson und Rosenbaum (2018), so-called *rough* stochastic volatility models such as the rough Bergomi model by Bayer, Friz und Gatheral (2016) constitute the latest evolution in option price modeling. Unlike standard bivariate diffusion models such as Heston (1993), these non-Markovian models with fractional volatility drivers allow to parsimoniously recover key stylized facts of market implied volatility surfaces such as the exploding power-law behaviour of the at-the-money volatility skew as time to maturity goes to zero. Standard model calibration routines rely on the repetitive evaluation of the map from model parameters to Black-Scholes implied volatility, rendering calibration of many (rough) stochastic volatility models prohibitively expensive since there the map can often only be approximated by costly Monte Carlo (MC) simulations (Bennedsen, Lunde & Pakkanen, 2017; McCrickerd & Pakkanen, 2018; Bayer et al., 2016; Horvath, Jacquier & Muguruza, 2017). As a remedy, we propose to combine a standard Levenberg-Marquardt calibration routine with neural network regression, replacing expensive MC simulations with cheap forward runs of a neural network trained to approximate the implied volatility map. Numerical experiments confirm the high accuracy and speed of our approach.

1 Introduction

Almost half a century after its publication, the option pricing model by Black und Scholes (1973) remains one of the most popular analytical frameworks for pricing and hedging European options in financial markets. A part of its success stems from the availability of explicit and hence instantaneously computable closed formulas for both theoretical option prices and option price sensitivities to input parameters (*Greeks*), albeit at the expense of assuming that *volatility* – the standard deviation of log returns of the underlying asset price – is deterministic and constant. Still, in financial practice, the Black-Scholes model is often considered a sophisticated transform between option prices and Black-Scholes (BS) *implied volatility (IV)* σ_{iv} where the latter is defined as the constant volatility input needed in the BS formula to match a given (market) price. It is a well-known fact that in empirical IV surfaces obtained by transforming market prices of European options to IVs, it can be observed that IVs vary across moneyness and maturities, exhibiting well-known smiles and at-the-money (ATM) skews and thereby contradicting the flat surface predicted by Black-Scholes (Figure 1). In particular, Bayer et al. (2016) report empirical at-the-money volatility skews of the form

$$\left| \frac{\partial}{\partial m} \sigma_{\text{iv}}(m, T) \right| \sim T^{-0.4}, \quad T \to 0 \tag{1}$$

for log moneyness m and time to maturity T.

While plain vanilla European Call and Put options often show enough liquidity to be marked-to-market, pricing and hedging path-dependent options (so-called *Exotics*) necessitates an option pricing model that prices European options *consistently* with respect to observed market IVs across moneyness and maturities. In other words, it should parsimoniously capture stylized facts of empirical IV surfaces. To address the shortcomings of Black-Scholes and incorporate the stochastic nature of volatility itself, popular bivariate diffusion models such as SABR (Hagan, Kumar, Lesniewski & Woodward, 2002) or

DOI 10.20347/WIAS.PREPRINT.2547 Berlin 2018

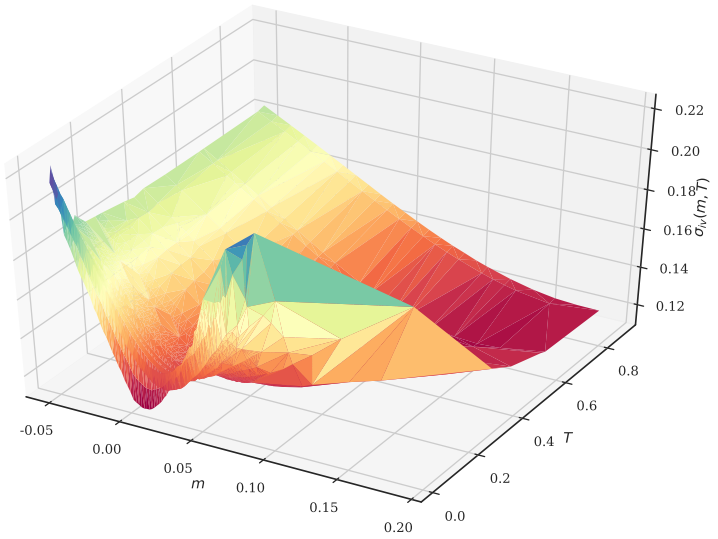


Figure 1: **SPX Market Implied Volatility surface on 15th February 2018.** IVs have been inverted from SPX Weekly European plain vanilla Call Mid prices and the interpolation is a (non-arbitrage-free) Delaunay triangulation. Axes denote log-moneyness $m = \log(K/S_0)$ for strike K and spot S_0, time to maturity T in years and market implied volatility $\sigma_{\text{iv}}(m, T)$.

the ones by Heston (1993) or Hull und White (1990) have been developed to capture *some* important stylized facts. However, according to Gatheral (2011), diffusive stochastic volatility models in general fail to recover the exploding power-law nature (1) of the volatility skew as time to maturity goes to 0 and instead predict a constant behaviour.

Sparked by the seminal work of Alòs et al. (2007); Fukasawa (2011, 2017); Gatheral et al. (2018), we have since seen a shift from classical diffusive modeling towards so-called rough stochastic volatility models. They may be defined as a class of *continuous-path* stochastic volatility models where the instantaneous volatility is driven by a stochastic process with Hölder regularity smaller than Brownian Motion, typically modeled by a fractional Brownian Motion with Hurst parameter $H < \frac{1}{2}$ (Mandelbrot & Van Ness, 1968). The evidence for this paradigm shift is by now overwhelming, both under the physical measure where time series analysis suggests that log realized volatility has Hölder regularity in the order of ≈ 0.1 (Bennedsen, Lunde & Pakkanen, 2016; Gatheral et al., 2018) and also under the pricing measure where the empirically observed power-law behaviour of the volatility skew near zero may be reproduced in the model (Alòs et al., 2007; Fukasawa, 2011, 2017; Bayer et al., 2016; Bayer, Friz, Gulisashvili, Horvath & Stemper, 2017). Serious computational and mathematical challenges arise from the non-Markovianity of fractional Brownian motion, effectively forcing researchers to resort to asymptotic expansions (Forde & Zhang, 2017; Bayer, Friz, Gulisashvili et al., 2017) in limiting regimes or (variance-reduced) Monte Carlo schemes (Bayer et al., 2016; Bennedsen et al., 2017; Bayer, Friz, Gassiat, Martin & Stemper, 2017; Horvath et al., 2017; McCrickerd & Pakkanen, 2018) to compute fair option prices.

Model calibration is the optimization procedure of finding model parameters such that the IV surface induced by the model best approximates a given market IV surface in an appropriate metric. In the absence of an analytical solution, it is standard practice to solve the arising weighted non-linear least squares problem using iterative optimizers such as Levenberg-Marquardt (LM) (Levenberg, 1944; Marquardt, 1963). However, these optimizers rely on the repetitive evaluation of the function φ from the space of model & option parameters (and external market information) to model BS implied volatility. If

each such evaluation involves a time– and/or memory–intensive operation such as a Monte Carlo simulation in the case of *rough Bergomi* (Bayer et al., 2016) or other (rough) stochastic volatility models, this makes efficient calibration prohibitively expensive.

Made possible by theoretical advancements as well as the widespread availability of cheap, high performance computing hardware, *Machine Learning* has seen a tremendous rise in popularity among academics and practitioners in recent years. Breakthroughs such as (super-) human level performance in image classification (Krizhevsky, Sutskever & Hinton, 2012; Simonyan & Zisserman, 2014; Szegedy et al., 2015) or playing the ancient Chinese board game *Go* (Silver et al., 2017) may all be attributed to the advent of *Deep Learning* (Goodfellow, Bengio & Courville, 2016). Fundamentally, its success stems from the capability of multi-layered artificial neural networks to closely approximate functions f only implicitly available through input-output pairs $\{(x_i, f(x_i))\}_{i=1}^N$, so-called *labeled data*.

The fundamental idea of this paper is to leverage this capability by training a fully-connected neural network on specifically tailored, synthetically generated training data to learn a map φ_{NN} approximating the true implied volatility map φ.

Remark 1.1. In a related but different approach, Hernandez (2017) proposes to use a neural network to learn the complete calibration routine – denoted Ψ in our notation in (6) – taking market data as inputs and returning calibrated model parameters directly. He demonstrates numerically the prowess of his approach by calibrating the popular short rate model of Hull und White (1990) to market data.

Both generating the synthetic data set as well as the actual neural network training are expensive in time and computing resource requirements, yet they only have to be performed a single time. Trained networks may then be quickly and efficiently saved, moved and deployed. The benefit of this novel approach is twofold: First, evaluations of φ_{NN} amount to cheap and almost instantaneous forward runs of a pre-trained network. Second, automatic differentiation of φ_{NN} with respect to the model parameters returns fast and accurate approximations of the Jacobians needed for the LM calibration routine. Used together, they allow for the efficient calibration of *any* (rough) stochastic volatility model including *rough Bergomi*.

To demonstrate the practical benefits of our approach numerically, we apply our machinery to Heston (1993) and *rough Bergomi* (Bayer et al., 2016) as representatives of classical and (rough) stochastic volatility models respectively. Speed-wise, no *systematic* comparison is made between the proposed neural network based approach and existing methods, yet with about 40ms per evaluation, our approach is at least competitive with existing Heston pricing methods and beats state-of-the-art rough Bergomi pricing schemes by magnitudes. Also, in both experiments, φ_{NN} exhibits small relative errors across the highly-liquid parts of the IV surface, recovering characteristic IV smiles and ATM IV skews. To quantify the uncertainty about model parameter estimates obtained by calibrating with φ_{NN}, we infer model parameters in a Bayesian spirit from (i) a synthetically generated IV surface and (ii) SPX market IV data. In both experiments, a simple (weighted) Bayesian nonlinear regression returns a (joint) posterior distribution over model parameters that (1) correctly identifies sensible model parameter regions and (2) places its peak at or close to the true (in the case of the synthetic IV) or previously reported (Bayer et al., 2016) (in the case of the SPX surface) model parameter values. Both experiments thus confirm the idea that φ_{NN} is sufficiently accurate for calibration.

This paper is organized as follows. In Section 2, we set the scene, introduce notation and revisit some important machinery that lies at the core of our proposed calibration scheme. In Section 3, we state the model calibration objective and introduce *deep calibration*, our approach of combining the established Levenberg-Marquardt calibration algorithm with neural network regression to enable the efficient calibration of (rough) stochastic volatility models. In Section 4, we outline practical intricacies of our approach, ranging from considerations related to generating synthetic, tailored *labeled data*

for training, validation and testing to tricks of the trade when training neural networks and performing hyperparameter optimization. Finally, in Section 5, we collect the results of our numerical experiments.

2 Background

We now set the scene and introduce notation. Throughout the paper, we shall be working on a filtered probability space $(\Omega, \mathcal{F}, \{\mathcal{F}_t\}_{t\geq 0}, \mathbb{P})$ satisfying the *usual conditions* and supporting two (or more) independent Brownian motions under the pricing measure $\mathbb{P}$. We consider a finite time horizon $T < \infty$ and assume the asset price process $S = (S_t)_{t\in[0,T]}$ has been without loss of generality normalized such that spot $S_0 = 1$ and risk-free rate $r = 0$. We define *moneyness* $M := K/S_0$ and *log moneyness* $m := \log(M) = \log(K)$.

2.1 Construction of a model IV surface

The concept of an *implied volatility surface* is an important idea and tool central to the theory of modern option pricing. In the introduction, we saw how such a surface arises from market prices of liquid European Call options on the S&P 500 Index *SPX* (cf. Figure 1). We now formalize the construction of such a surface from model prices. In a first step, we define the pricing function that maps model & option parameters (and possibly external market information) to the fair price of a European option at time $t = 0$.

Definition 2.1 (Pricing map)**.** Consider a (rough) stochastic volatility (market) model for an asset S with model parameters $\mu \in \mathcal{M} \subseteq \mathbb{R}^m$ and possibly incorporated market information $\xi \in \mathcal{E} \subseteq \mathbb{R}^k$. The fair price of a European Call option at time $t = 0$ is then given by

$$\mathbb{E}\left[S_T(\mu, \xi) - M\right]^+$$

where $(M, T) \in \Theta \subseteq \mathbb{R}^2$ denote moneyness and time to maturity respectively. Letting

$$\mathcal{I} := \{(\mu, \xi) \times (M, T) \mid \mu \in \mathcal{M}, \xi \in \mathcal{E}, (M, T)^T \in \Theta\} \subseteq \mathbb{R}^{m+k+2} \tag{2}$$

be the pricing input space, we then define the pricing map $P_0 : \mathcal{I} \to \mathbb{R}_+$ by

$$(\mu, \xi) \times (M, T) \mapsto \mathbb{E}\left[S_T(\mu, \xi) - M\right]^+. \tag{3}$$

Example 2.2. In the rough Bergomi model by Bayer et al. (2016), the dynamics for the asset price process S and the instantaneous variance process $v = (v_t)_{t\in[0,T]}$ are given by

$$\begin{aligned}\frac{\mathrm{d}S_t}{S_t} &= \sqrt{v_t}\mathrm{d}\left(\rho W_t + \sqrt{1-\rho^2}W_t^\perp\right)\\ v_t &= \xi_0(t)\exp\left(\eta W_t^H - \frac{1}{2}\eta^2 t^{2H}\right), \quad t \in [0,T].\end{aligned}$$

Here, $\left(W, W^\perp\right) = \left(W_t, W_t^\perp\right)_{t\in[0,T]}$ are two independent Brownian motions and $\rho \in (-1, 1)$ is a constant correlation parameter introducing the *leverage effect* – the empirically observed anti correlation between stock and volatility movements – at the driving noise level. The parameter $\eta > 0$ denotes volatility of variance and $\xi_0(t) : \mathbb{R}_+ \to \mathbb{R}_+$ given by $\xi_0(t) = \mathbb{E}(v_t), t \in [0, T]$ is a so-called *forward variance curve* which may be recovered from market information (Bayer et al., 2016). Moreover, W^H is a *Riemann-Liouville* fractional Brownian motion given by

$$W_t^H = \sqrt{2H}\int_0^t (t-s)^{H-\frac{1}{2}}\mathrm{d}W_s, \quad t \in [0,T]$$

with Hurst parameter $H \in (0,1)$. By Kolmogorov, sample paths of W^H are locally almost surely H-ε Hölder for $\varepsilon > 0$. With respect to Definition 2.1, hence $\mu = (H, \eta, \rho)$ and $\xi = \xi_0$.

Example 2.3. In the Heston model (Heston, 1993), with independent Brownian motions W and $W^\perp$ and model parameters ρ, η defined as in Example 2.2, the dynamics of the asset price S and the instantaneous variance process $v = (v_t)_{t\in[0,T]}$ starting from spot variance $v_0 > 0$ follow

$$\begin{aligned}\frac{\mathrm{d}S_t}{S_t} &= \sqrt{v_t}\mathrm{d}\left(\rho W_t + \sqrt{1-\rho^2}W_t^\perp\right)\\ \mathrm{d}v_t &= \lambda(\bar{v} - v_t)\mathrm{d}t + \eta\sqrt{v_t}\mathrm{d}W_t, \quad t \in [0,T].\end{aligned}$$

Here, $\bar{v} > 0$ is the long-run average variance and $\lambda > 0$ is the speed of mean reversion. Feller's condition $2\lambda\bar{v} > \eta^2$ ensures that $v_t > 0$ for $t \geq 0$. In this model, we thus have $\mu = (\lambda, \bar{v}, v_0, \rho, \eta)$ and no market information is incorporated into the model.

Let $\mathrm{BS}(M, T, \sigma)$ denote the Black-Scholes price of a European Call with moneyness M, time to maturity T and assumed constant volatility σ of the underlying and let $Q(M,T)$ be the corresponding market price. The BS implied volatility $\sigma_{\mathrm{iv}}(M,T)$ corresponding to $Q(M,T)$ satisfies

$$Q(M,T) - \mathrm{BS}(M,T,\sigma_{\mathrm{iv}}(M,T)) \stackrel{!}{=} 0.$$

and the map $(M,T) \mapsto \sigma_{\mathrm{iv}}(M,T)$ is called a *volatility surface*.

Definition 2.4 (IV map)**.** Let μ, ξ, M, T be defined as in Definition 2.1. The Black-Scholes IV $\sigma_{\mathrm{iv}}(\mu, \xi, M, T)$ corresponding to the theoretical model price $P_0(\mu, \xi, M, T)$ satisfies

$$P_0(\mu,\xi,M,T) - \mathrm{BS}(M,T,\sigma_{\mathrm{iv}}(\mu,\xi,M,T)) \stackrel{!}{=} 0. \tag{4}$$

The function $\varphi : \mathcal{I} \to \mathbb{R}_+$ given by

$$(\mu,\xi,M,T) \mapsto \sigma_{\mathrm{iv}}(\mu,\xi,M,T) \tag{5}$$

is what we call the *implied volatility map*.

2.2 Regression with neural networks

Given a data set $\mathcal{D} = \left\{(x_i, y_i) : x_i \in \mathbb{R}^d, y_i \in \mathbb{R}\right\}_{i=1}^n$ of variables x_i and corresponding scalar, continuous response variables y_i, the statistical procedure of estimating the relationship between these variables is commonly called *regression analysis*. Here, we will introduce neural networks and outline their prowess as a regression tool.

The atomic building block of every neural network is a *node*, a functional that performs a weighted sum of its (multi-dimensional) inputs, adds a bias term and then composes the linearity with a scalar non-linear function $\alpha : \mathbb{R} \to \mathbb{R}$ that is identical across the network. Formally, for some input $x \in \mathbb{R}^d, d \in \mathbb{N}$, the output of an individual node is given by

$$y = \alpha\left(w^T x + b\right) \in \mathbb{R}$$

where $w \in \mathbb{R}^d$ and $b \in \mathbb{R}$ are individual *weight* and *bias* terms. An *artificial neural network* is then a collection of many such nodes, grouped into non-overlapping sets called *layers* together with a rule of how the information flows between the layers.

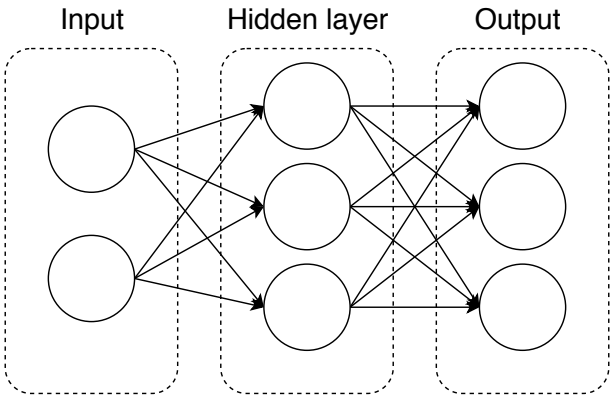


Figure 2: **Schematic of a fully-connected neural network (FCNN).** Depicted FCNN has a single *hidden layer* consisting of three neurons and may learn to represent a subset of general functions $f : \mathbb{R}^2 \to \mathbb{R}^3$. In the directed acyclic graph above, vertices denote nodes and directed edges describe the flow of information from one node to the next. If number of hidden layers higher than one (typically dozens or hundreds of layers), a neural network is considered *deep*.

Over the years different architectural styles have been developed to suit the specific needs of different domains such as speech, text or vision. Arguably the simplest neural network topology not adapted to any particular domain is that of a *fully-connected neural network* (FCNN). An FCNN consists of sequentially ordered so-called *dense layers* followed by a linear output layer. Any two nodes of a dense layer act independently of each other and do not share weights and biases. Their input is given by the output of all nodes in the previous layer – or all input features if it is the first layer – and their output serves as an input to all nodes in the following layer, see Figure 2 for a depiction of a small example.

FCNNs serve as powerful regression tools because they are able to represent large families of functions. In his *Universal Approximation Theorem*, Hornik (1991) proves that FCNNs can approximate continuous functions on $\mathbb{R}$ arbitrarily well.

Theorem 2.5 (Universal Approximation Theorem)**.** Let $N(\alpha)$ denote the space of functions that a fully connected neural network with activation function $\alpha : \mathbb{R} \to \mathbb{R}$, a single hidden layer with a finite number of neurons $l \in \mathbb{N}$ and a linear output layer can represent, i.e.

$$N(\alpha) = \left\{ f : \mathbb{R}^d \to \mathbb{R} \mid f(x) = \sum_{i=1}^{l} w_i \alpha \left(\sum_{j=1}^{d} \bar{w}_j^{(i)} x_j + b^{(i)} \right) + b_i \right.$$
$$\left. \text{for some } w, b \in \mathbb{R}^l \text{ and } \bar{w}^{(i)}, \bar{b}^{(i)} \in \mathbb{R}^d, 1 \leq i \leq l \right\}$$

where $w, b \in \mathbb{R}^l$ are weights and biases of the output layer and $\bar{w}^{(i)}, \bar{b}^{(i)} \in \mathbb{R}^d, 1 \leq i \leq l$ are the weights and biases of the l individual neurons in the hidden layer. Assuming the activation function $\alpha : \mathbb{R} \to \mathbb{R}$ is non-constant, unbounded and continuous, $N(\alpha)$ is dense in $C(X)$ for compact $X \subseteq \mathbb{R}$ in the uniform topology, i.e. for any $f \in C(X)$ and arbitrary $\varepsilon > 0$, there is $g \in N(\alpha)$ such that

$$\sup_{x \in X} |f(x) - g(x)| < \varepsilon.$$

The *Rectified Linear Unit (ReLU)* nonlinearity $\alpha : \mathbb{R} \to \mathbb{R}_+$ given by $\alpha(x) := \max(0, x)$ fulfills the conditions of being non-constant, unbounded and continuous and so in theory ReLU FCNNs allow

for approximation of continuous functions to arbitrary accuracy. However, the reason the ReLU has become a de facto standard in recent years (LeCun, Bengio & Hinton, 2015) is that in comparison to first generation nonlinearities such as the *sigmoid* or *tanh*, ReLU networks are superior in terms of their *algorithmic learnability*, see more in Section 4.

Remark 2.6. Over the years, various alternative activation functions have been proposed such as Leaky ReLU (He, Zhang, Ren & Sun, 2015), ELU (Clevert, Unterthiner & Hochreiter, 2015) or lately the SiLU (Elfwing, Uchibe & Doya, 2018; Ramachandran, Zoph & Le, 2017). To date, none of these activation functions have been shown to consistently outperform ReLUs (Ramachandran et al., 2017), so a systematic comparison of the effect of different activation functions on training results has been left for future research.

3 Calibration of option pricing models

The implied volatility map $\varphi : \mathcal{I} \to \mathbb{R}_+$ defined in (5) formalizes the influence of model parameters on an option pricing model's implied volatility surface. *Calibration* describes the procedure of tweaking model parameters to fit a model surface to an empirical IV surface obtained by transforming liquid European option market prices to Black-Scholes IVs (cf. Figure 1). A mathematically convenient approach consists of minimizing the weighted squared differences between market and model IVs of $N \in \mathbb{N}$ plain vanilla European options.

Proposition 3.1 (Calibration objective)**.** Consider a (rough) stochastic volatility model with model parameters $\mu \in \mathcal{M} \subseteq \mathbb{R}^m$ and embedded market information $\xi \in \mathcal{E} \subseteq \mathbb{R}^k$ (recall Def. 2.1). Suppose the *market* IV quotes of N European options with moneyness $M^{(i)}$ and time to maturity $T^{(i)}$ are given by

$$\mathbf{Q} := \left(Q\left(M^{(1)}, T^{(1)}\right), \ldots, Q\left(M^{(N)}, T^{(N)}\right)\right)^T \in \mathbb{R}^N$$

and analogously the *model* IV quotes of the same options under said pricing model are given by

$$\boldsymbol{\varphi}\left(\mu, \xi\right) := \left(\varphi\left(\mu, \xi, M^{(1)}, T^{(1)}\right), \ldots, \varphi\left(\mu, \xi, M^{(N)}, T^{(N)}\right)\right)^T \in \mathbb{R}^N.$$

Given market quotes $\boldsymbol{Q}$ and market information ξ, we define the residual $\boldsymbol{R}(\mu) : \mathcal{M} \to \mathbb{R}^N$ between market and model IVs by

$$\boldsymbol{R}(\mu) := \boldsymbol{\varphi}(\mu, \xi) - \mathbf{Q}$$

so that the calibration objective becomes

$$\mu^\star = \underset{\mu \in \mathcal{M}}{\arg\min} \left\| \mathbf{W}^{\frac{1}{2}} \boldsymbol{R}(\mu) \right\|_2^2 = \underset{\mu \in \mathcal{M}}{\arg\min} \left\| \mathbf{W}^{\frac{1}{2}} \left[\boldsymbol{\varphi}(\mu, \xi) - \mathbf{Q}\right] \right\|_2^2 := \Psi\left(\mathbf{W}, \xi, \boldsymbol{Q}\right) \tag{6}$$

where $\boldsymbol{W} = \operatorname{diag}\left[w_1, \ldots, w_N\right] \in \mathbb{R}^{N \times N}$ is a diagonal matrix of weights and $\|\cdot\|_2$ denotes the standard Euclidean norm.

Since $\boldsymbol{R}(\mu) : \mathcal{M} \to \mathbb{R}^N$ is non-linear in the parameters $\mu \in \mathcal{M} \subseteq \mathbb{R}^m$ and $N > m$, the optimization objective (6) is an example of an overdetermined non-linear least squares problem, usually solved numerically using iterative solvers such as the de-facto standard Levenberg-Marquardt (LM) algorithm (Levenberg, 1944; Marquardt, 1963).

Proposition 3.2 (LM calibration)**.** Suppose $\boldsymbol{R} : O \to \mathbb{R}^N$ is twice continuously differentiable on an open set $O \subseteq \mathbb{R}^m$ and $N > m$. Let $\boldsymbol{J} : O \to \mathbb{R}^{N\times m}$ denote the Jacobian of $\boldsymbol{R}$ with respect to the model parameters $\mu \in \mathbb{R}^m$, i.e. its components are given by

$$[\boldsymbol{J}_{ij}]_{\substack{1\le i\le N,\\ 1\le j\le m}} = \left[\frac{\partial \boldsymbol{R}_i(\mu)}{\partial \mu_j}\right]_{\substack{1\le i\le N,\\ 1\le j\le m}} = \left[\frac{\partial \boldsymbol{\varphi}_i(\mu,\xi)}{\partial \mu_j}\right]_{\substack{1\le i\le N,\\ 1\le j\le m}}.$$

With regards to the objective in (6), the algorithm starts with an initial parameter guess $\mu_0 \in \mathbb{R}^m$ and then at each iteration step with current parameter estimate $\mu_k \in \mathbb{R}^m, k \in \mathbb{N}$, the parameter update $\Delta_\mu \in \mathbb{R}^m$ solves

$$\left[\boldsymbol{J}(\mu_k)^T \boldsymbol{W} \boldsymbol{J}(\mu_k) + \lambda I_m\right] \Delta_\mu = \boldsymbol{J}(\mu_k)^T \boldsymbol{W} \boldsymbol{R}(\mu_k) \tag{7}$$

where $I_m \in \mathbb{R}^{m\times m}$ denotes the identity and $\lambda \in \mathbb{R}$.

It is hence necessary that the *normal equations* (7) be quickly and accurately solved for the iterative step Δ_μ. In a general (rough) stochastic volatility setting this is problematic: The true implied volatility map $\varphi : \mathcal{I} \to \mathbb{R}_+$ as well as its Jacobian $\boldsymbol{J} : O \to \mathbb{R}^{N\times m}$ are unknown in analytical form. In the absence of an analytical expression for Δ_μ, an immediate remedy is:

(I) Replace the (theoretical) true pricing map $P_0 : \mathcal{I} \to \mathbb{R}_+$ defined in (3) by an efficient numerical approximation $\tilde{P}_0 : \mathcal{I} \to \mathbb{R}_+$ such as Monte Carlo, Fourier Pricing or similar means. This gives rise to an approximate implied volatility map $\tilde{\varphi} : \mathcal{I} \to \mathbb{R}_+$.

(II) Apply finite-difference methods to $\tilde{\varphi} : \mathcal{I} \to \mathbb{R}_+$ to compute an approximate Jacobian $\tilde{\boldsymbol{J}} : O \to \mathbb{R}^{N\times m}$.

In many (rough) stochastic volatility models such as *rough Bergomi*, expensive Monte Carlo simulations have to be used to approximate the pricing map. In a common calibration scenario where the normal equations (7) have to be solved frequently, the approach outlined above thus renders calibration prohibitively expensive.

3.1 Deep calibration

In a first step, we use the approximate implied volatility map $\tilde{\varphi} : \mathcal{I} \to \mathbb{R}_+$ to synthetically generate a large and as accurate as computationally feasible set of labeled data

$$\mathcal{D} := \left\{\left(x^{(i)}, \tilde{\varphi}\left(x^{(i)}\right)\right) \mid x \in \mathcal{I}\right\}_{i=1}^{n} \in (\mathcal{I}\times\mathbb{R}_+)^n, \quad n \in \mathbb{N}.$$

Here, it is sensible to trade computational savings for an increased numerical accuracy since the expensive data generation only has to be performed once. Using the sample input-output pairs $\mathcal{D}$, a ReLU FCNN is trained to approximate $\tilde{\varphi} : \mathcal{I} \to \mathbb{R}_+$, in other words, we use a ReLU FCNN to regress response variables $\tilde{\varphi}\left(x^{(i)}\right) = \tilde{\varphi}\left(\mu^{(i)}, \xi^{(i)}, M^{(i)}, T^{(i)}\right)$ on explanatory variables $\left(\mu^{(i)}, \xi^{(i)}, M^{(i)}, T^{(i)}\right)$. We denote this function that the network is now able to represent by $\varphi_{\text{NN}} : \mathcal{I} \to \mathbb{R}_+$. With respect to the repeated solving of the normal equations (7), the benefit of this new approach is twofold:

(I) Evaluations of $\varphi_{\text{NN}} : \mathcal{I} \to \mathbb{R}_+$ amount to forward runs of a trained ReLU FCNN. Computationally, forwards runs come down to highly optimized and parallelizable matrix-matrix multiplications combined with element-wise comparison operations – recall the ReLU activation is given by $\alpha(\cdot) = \max(0, \cdot)$ – both of which are fast.

Algorithm 1: Deep calibration (LM combined with NN regression)

Input: Implied vol map $\boldsymbol{\varphi}_{\mathrm{NN}}$ and its Jacobian $\boldsymbol{J}_{\mathrm{NN}}$, market quotes $\boldsymbol{Q}$, market info ξ

Parameters: Lagrange multiplier $\lambda_0 > 0$, maximum number of iterations $n_{\max}$, minimum tolerance of step norm $\varepsilon_{\min}$, bounds $0 < \beta_0 < \beta_1 < 1$

Result: Calibrated model parameters $\mu^\star$

initialize model parameters $\mu = \mu_0$ and step counter $n = 0$;
compute $\boldsymbol{R}(\mu) = \boldsymbol{\varphi}_{\mathrm{NN}}(\mu, \xi) - \mathbf{Q}$ and $\boldsymbol{J}_{\mathrm{NN}}(\mu)$ and solve normal equations (7) for Δ_μ;
while $n < n_{max}$ *and* $\|\Delta_\mu\|_2 > \varepsilon$ **do**
  compute relative improvement $c_\mu = \frac{\|\boldsymbol{R}(\mu)\|_2 - \|\boldsymbol{R}(\mu+\Delta_\mu)\|_2}{\|\boldsymbol{R}(\mu)\|_2 - \|\boldsymbol{R}(\mu)+\boldsymbol{J}_{\mathrm{NN}}(\mu)\Delta_\mu\|_2}$ with respect to predicted improvement under linear model;
  if $c_\mu \leq \beta_0$ **then** reject Δ_μ, set $\lambda = 2\lambda$;
  if $c_\mu \geq \beta_1$ **then** accept Δ_μ, set $\mu = \mu + \Delta_\mu$ and $\lambda = \frac{1}{2}\lambda$;
  compute $\boldsymbol{R}(\mu)$ and $\boldsymbol{J}_{\mathrm{NN}}(\mu)$ and solve normal equations (7) for Δ_μ;
  set $n = n + 1$;
end

(II) In order to perform *backpropagation*, the standard training algorithm for neural networks, industrial grade machine learning software libraries such as Google Inc.'s Tensorflow (Abadi et al., 2016) ship with built-in implementations of *automatic differentiation* (Baydin, Pearlmutter, Radul & Siskind, 2015). This may easily be exploited to quickly compute approximative Jacobians $\boldsymbol{J}_{\mathrm{NN}} : O \to \mathbb{R}^{N\times m}$ accurate to machine precision.

It is also important to stress that trained networks can be efficiently stored, moved and loaded, so training results can be shared and deployed quickly.

Remark 3.3. Hernandez (2017) calibrates the Hull und White (1990) short-rate model by directly learning calibrated model parameters from market data, i.e. the total calibration routine Ψ in (6). Extending his approach to equity models necessitates a network topology that allows to learn from empirical IV point clouds. Here, adaptations of Convolutional Neural Networks (CNNs) invented for computer vision problems might be worthwhile to explore.

4 Neural network training

While theoretically easy to understand, the training of neural networks in practice often becomes a costly and most importantly time–consuming exercise full of potential pitfalls. To this end, we outline here the approach taken in this paper, briefly mentioning important *tricks of the trade* that have been utilized to facilitate or accelerate the training of the ReLU FCNN networks.

4.1 Generation of synthetic labeled data

The ability of a neural network to learn the implied volatility map φ to a high degree of accuracy critically hinges upon the provision of a large and accurate labeled data set

$$\mathcal{D} = \left\{ \left(\mu^{(i)}, \xi^{(i)}, M^{(i)}, T^{(i)}, \tilde{\varphi}\left(\mu^{(i)}, \xi^{(i)}, M^{(i)}, T^{(i)}\right)\right) \right\}_{i=1}^{n} \in (\mathcal{I} \times \mathbb{R}_+)^n, \quad n \in \mathbb{N}.$$

Table 1: Marginal priors of model parameters μ for synthetically generating $\mathcal{D}$. The continuous uniform distribution on the interval bounded by $a_i, b_i \in \mathbb{R}$ is denoted by $\mathcal{U}[a_i, b_i]$ and $\mathcal{N}_{\text{trunc}}[a_i, b_i, \lambda, \sigma]$ stands for the normal distribution with mean $\lambda \in \mathbb{R}$ and standard deviation $\sigma \in \mathbb{R}_+$, truncated to the interval $[a_i, b_i]$ with $a_i, b_i \in \mathbb{R}$.

Heston		rough Bergomi	
Parameter	Marginal	Parameter	Marginal
η	$\mathcal{U}[0,5]$	η	$\mathcal{N}_{\text{trunc}}[1,4,2.5,0.5]$
ρ	$\mathcal{U}[-1,0]$	ρ	$\mathcal{N}_{\text{trunc}}[-1,-0.5,-0.95,0.2]$
λ	$\mathcal{U}[0,10]$	H	$\mathcal{N}_{\text{trunc}}[0.01,0.5,0.07,0.05]$
$\bar{v}$	$\mathcal{U}[0,1]$	v_0	$\mathcal{N}_{\text{trunc}}[0.05,1,0.3,0.1]^2$
v_0	$\mathcal{U}[0,1]$		

Knowledge of the parametric dependence structure $\tilde{\varphi} : \mathcal{I} \to \mathbb{R}_+$ between inputs and corresponding outputs allows us to address these requirements adequately. First, trading computational savings for increased numerical accuracy, we ensure that $\|\tilde{\varphi} - \varphi\|_\infty < \epsilon$ for ϵ small. Second, we can sample an arbitrarily large set of labeled data $\mathcal{D}$, allowing the network to learn the underlying dependence structure $\tilde{\varphi}$ – rather than noise present in the training set – and generalize well to unseen test data. In the numerical tests in Section 5, we draw $n = |\mathcal{D}| = 10^6$ iid sample inputs from a to be specified sampling distribution $\mathcal{G}$ on $\mathcal{I}$ and compute the corresponding outputs as follows: For Heston, we use the Fourier pricing method implemented in the open-source quantitative finance library *QuantLib* (Ametrano et al., 2015) which makes use of the well-known fact that the characteristic function of the log asset price is known. For *rough Bergomi*, we use a self-coded, parallelized implementation of a slightly improved version of the Monte Carlo scheme proposed by McCrickerd und Pakkanen (2018). Black-Scholes IVs are inverted from option prices using a publicly available implementation of the implied volatility solver by Jäckel (2015). The full dataset $\mathcal{D}$ is then randomly shuffled and partitioned into training, validation and test sets $\mathcal{D}_{train}, \mathcal{D}_{valid}$ and $\mathcal{D}_{test}$ of sizes $n_{\text{train}}, n_{\text{valid}}$ and n_{test} respectively.[1]

An important advantage of being able to synthetically generate labeled data is the freedom in choosing the sampling distribution $\mathcal{G}$ on $\mathcal{I}$. Prior to calibration, little is known about the interplay of model parameters and particular model parameter regions of highest interest to be learned accurately. Consequently, we assume zero prior knowledge of the (joint) relevance of model parameters μ in the Heston experiment in Section 5. An ad-hoc approach is to sample individual model parameters independently of each other from the uniformly continuous marginal distributions collected in Table 1. A similar reasoning also applies in the rough Bergomi experiment in Section 5, except that here we do assume some prior marginal distributional knowledge and use truncated normal marginals instead of uniform marginals.

On the other hand, it is reasonable to increase the number of samples in option parameter regions with high liquidity since these are given more weight by the calibration objective (6) and as such require to be more accurate. To that end, we postulate a joint distribution of moneyness and time to maturity based on liquidity and estimate it using a weighted Gaussian kernel density estimation (wKDE) (Scott, 2015): Let L_i denote the market liquidity of an option $i, i \in \mathbb{N}$, with time to maturity $T^{(i)}$ and moneyness $M^{(i)}$. We proxy liquidities by inverse bid-ask spreads of traded European Call Options on SPX and then run a wKDE on samples $\left\{\left(M^{(i)}, T^{(i)}\right)\right\}_{i=1}^n$ with weights $\{L_i\}_{i=1}^n$ and a

[1] The code has been made available at https://github.com/roughstochvol.

smoothing bandwidth. In a similar vein, one may also derive a multivariate distribution $\mathcal{K}_\xi$ of external market data $\xi \in \mathbb{R}^k$.

With regards to the individual marginals collected in Table 1, the sampling distribution $\mathcal{G}_{\text{Heston}}$ on $\mathcal{I} \subseteq \mathbb{R}^{m+k+2}$ is given by

$$\mathcal{G}_{\text{Heston}} := \mathcal{U}^{\otimes m}[a_i, b_i] \otimes \mathcal{K}_\xi \otimes \mathcal{K}_{(M,T)} \tag{8}$$

and analogously for the rough Bergomi model, we have

$$\mathcal{G}_{\text{rBergomi}} := \mathcal{N}_{\text{trunc}}^{\otimes m}[a_i, b_i, \lambda_i, \sigma_i] \otimes \mathcal{K}_\xi \otimes \mathcal{K}_{(M,T)}. \tag{9}$$

4.2 Backpropagation and hyper parameter optimization

Consider a ReLU FCNN with $L \in \mathbb{N}$ hidden layers as described in Section 2.2. Let $n_l, 1 \leq l \leq L$ denote the number of nodes of the hidden layers and $\mathcal{S}_{h_{\text{model}}}$ the function space spanned by such a network with model hyper parameters $h_{\text{model}} = (L, n_1, \ldots, n_L)$. Let $X : \Omega \to \mathcal{I}$ denote a random input and consider h_{model} fixed. Then the fundamental objective of neural network training is to learn a function that minimizes the generalization error:

$$f^\star_{h_{\text{model}}} = \underset{f_{h_{\text{model}}} \in \mathcal{S}_{h_{\text{model}}}}{\arg\min} \; \| f_{h_{\text{model}}}(X) - \tilde{\varphi}(X) \|^2_{L^2(\Omega)}, \quad X \sim \mathcal{G} \tag{10}$$

where $\mathcal{G} \in \{\mathcal{G}_{\text{Heston}}, \mathcal{G}_{\text{rBergomi}}\}$, depending on experiment. In many calibration scenarios, $\tilde{\varphi}$ is a Monte-Carlo approximation to φ, so $\tilde{\varphi}(\cdot) = \varphi(\cdot) + \varepsilon$ for ε some homoskedastic error with $\mathbb{E}(\varepsilon) = 0$ and $\text{Var}(\varepsilon) = \sigma^2 > 0$. The MSE loss in (10) admits the well-known bias-variance decomposition

$$\| f_{h_{\text{model}}}(X) - \tilde{\varphi}(X) \|^2_{L^2(\Omega)} = \left(\mathbb{E}\left[f_{h_{\text{model}}}(X) - \varphi(X) \right] \right)^2 + \text{Var}\left[f_{h_{\text{model}}}(X) \right] + \sigma^2 \tag{11}$$

where in addition to a bias and variance term we also have the variance of the sample error, the irreducible error. The empirical analogue to (10) relevant for practical training is given by

$$f^\star_{h_{\text{model}}} \approx \underset{f_{h_{\text{model}}} \in \mathcal{S}_{h_{\text{model}}}}{\arg\min} \; \frac{1}{n_{\text{valid}}} \sum_{i=1}^{n_{\text{valid}}} \left[f_{h_{\text{model}}}\left(x^{(i)}\right) - \tilde{\varphi}\left(x^{(i)}\right) \right]^2 \tag{12}$$

where $\left(x^{(i)}, \tilde{\varphi}\left(x^{(i)}\right)\right) \in \mathcal{D}_{\text{valid}}$. The optimization in the function space $\mathcal{S}_{h_{\text{model}}}$ corresponds to a high-dimensional nonlinear optimization in the space of network weights and biases, similarly to (6) typically addressed by gradient-based schemes. *Backpropagation* (Goodfellow et al., 2016), a specific form of *reverse-mode automatic differentiation* (Baydin et al., 2015) in the context of neural networks, prevails as the go-to approach to iteratively compute gradients of the empirical MSE loss with respect to weights and biases of all nodes in the network. The gradients are then often used in the well-known *Mini-Batch Gradient Descent* (Goodfellow et al., 2016) optimization algorithm, a variant of which called *Adam* (Kingma & Ba, 2014) we use in our experiments. *Adam* incorporates momentum to prevent the well-known zigzagging of *Gradient Descent* in long and sharp valleys of the error surface and adaptively modifies a given global step size for each component of the gradient individually to speed up the optimization process. It in turn has its own optimization hyper parameters $h_{\text{opt}} = (\delta, \beta)$ where δ denotes the mentioned global learning rate and β denotes the mini-batch size used. In the following, we denote the learning algorithm *Adam* mapping training data $\mathcal{D}_{\text{train}}$ to a local minimizer $f^\star_{h_{\text{model}}}$ of (12) by $\mathcal{A}_{h_{\text{opt}}} : \mathcal{I}^n \to \mathcal{S}_{h_{\text{model}}}$.

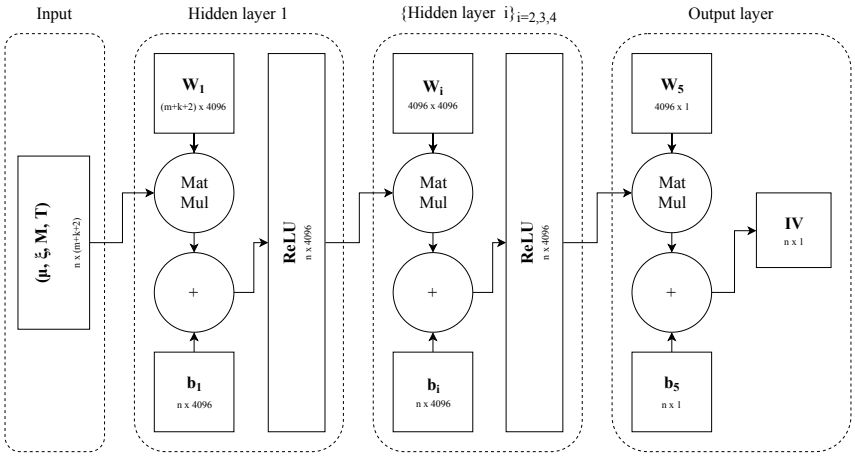


Figure 3: **Schematic of IV ReLU FCNN**. Depiction of 4-layer ReLU FCNN used to learn IV maps. It consists of $2^{12} = 4096$ nodes at each hidden layer. Note the output layer is a linear layer with no activation function. Rectangles denote tensors and circles denote operations, *MatMul* is matrix multiplication. The number of parallel IV calculations is given by $n \in \mathbb{N}$.

Up to know, we treated the hyper parameters $(h_{\text{opt}}, h_{\text{model}})$ as fixed whereas in reality they may be varied and have a crucial influence on the training outcome. Indeed, for all other variables besides $(h_{\text{opt}}, h_{\text{model}})$ fixed, let us define a hyper parameter response function $\mathcal{H}$ by

$$\mathcal{H}(h_{\text{opt}}, h_{\text{model}}) := \frac{1}{n_{\text{valid}}} \sum_{i=1}^{n_{\text{valid}}} \left[\left[\mathcal{A}_{h_{\text{opt}}} \left(\mathcal{D}_{\text{train}} \right) \right]_{h_{\text{model}}} \left(x^{(i)} \right) - \tilde{\varphi} \left(x^{(i)} \right) \right]^2 . \tag{13}$$

In practice, it then turns out the real challenge in training neural networks to high accuracy lies in the additional (outer) optimization over hyper parameters:

$$(h^\star_{\text{opt}}, h^\star_{\text{model}}) = \underset{(h_{\text{opt}}, h_{\text{model}})}{\arg\min}\, \mathcal{H}(h_{\text{opt}}, h_{\text{model}}). \tag{14}$$

The scope of effect of hyper parameters h_{model} and h_{opt} does not overlap: The former determines the capacity of $\mathcal{S}_{h_{\text{model}}}$, the latter governs which local minimizer $f^\star_{h_{\text{model}}}$ the optimization algorithm $\mathcal{A}_{h_{\text{opt}}}$ converges to and the speed with which this happens. This allows us to treat their optimization separately. A coarse grid search reveals that adding additional layers beyond 4 hidden layers does not consistently reduce errors on the validation set. Rather, networks become harder to train as evidenced by errors fluctuating more wildly on the validation set. We suspect this is a consequence of what Ioffe und Szegedy (2015) call *internal covariate shift*: First-order methods such as Gradient Descent are blind to changes in the weights and biases of the layers feeding into a given layer and so with deeper networks the propagating and magnifying effects of changes in one layer to subsequent layers worsen and slow down the training. On the other hand, our locally available compute resources max out at $4096 = 2^{12}$ nodes per layers, so we fix $h_{\text{model}} = (4) \times (4096)^4$. Each evaluation of the hyper parameter response function $\mathcal{H}$ in (14) requires a ReLU FCNN to be fully trained from scratch which is a very costly operation in terms of time and computing resources. Moreover, gradient-based optimization approaches are ruled out by the fact that gradients of $\mathcal{H}$ with respect to h_{opt} are unavailable (after all, batch sizes are discrete). In our experiments, we explore the use of Gaussian Regression (Rasmussen & Williams, 2006; Snoek, Larochelle & Adams, 2012) which is an adaptive gradient-free minimization algorithm. Postulating a surrogate Gaussian model for $\mathcal{H}$, it takes existing function evaluations into account and – balancing exploitation and exploration – iteratively proposes the next most

promising candidate input in terms of information gain. As is common in applied sciences, we use a Matérn Kernel for the covariance function of the Gaussian model and the Lower Confidence Bound (LCB) acquisition function.

4.2.1 Tricks of the trade

Feature scaling or **preconditioning** is a standard preprocessing technique applied to input data of optimization algorithms in order to improve the speed of optimization. After the data set $\mathcal{D}$ has been partitioned into training, validation and test sets, we compute the sample mean $\bar{x}_{\text{train}} \in \mathbb{R}^{m+k+2}$ of the inputs across the training set and the corresponding sample standard deviation $s_{\text{train}} \in \mathbb{R}^{m+k+2}$. For each input $x^{(i)} \in \mathcal{I}$ from $\mathcal{D}$, its standardized version $\hat{x}^{(i)}$ is given by

$$\hat{x}^{(i)} := \frac{x^{(i)} - \bar{x}_{\text{train}}}{s_{\text{train}}}, \quad 1 \leq i \leq n.$$

where the operations are defined componentwise. We then use these standardized inputs $\hat{x}^{(i)}$ – which have zero offset and unit scale – for training and prediction. It is important to stress that *all* n inputs from the complete set $\mathcal{D}$ are standardized using the *training* mean and standard deviation, including those of the validation and test sets.

Weight initialization is an important precursor to the iterative optimization process of *Adam*. Initialization is a delicate task that may speed up or hamper the training process all together: If within (but not necessarily across) all layers, weights and biases of all nodes are identical, then the same is true for their outputs and the partial derivative of the loss with respect to their weights and biases, impeding any learning on the part of the optimizer. To *break the symmetry*, it is standard procedure to draw weights from a symmetric probability distribution centered at zero. Suppose $w_{ij}^{(l)}$ denotes the weight of node $i, 1 \leq i \leq n_l$ in layer $1 \leq l \leq L$ being multiplied with the output of node $j, 1 \leq j \leq n_{l-1}$ in layer $l-1$ and n_0 denotes the number of network inputs. He et al. (2015) suggest the weights and biases be independently drawn as follows

$$w_{ij}^{(l)} \sim \mathcal{N}\left(0, \frac{2}{n_{l-1}}\right), \quad b_i^{(l)} = 0.$$

Adapting an argument by Glorot und Bengio (2010) for linear layers to ReLU networks, they can show that – under some assumptions – this ensures that, at least at initialization, input signals and gradients do not get magnified exponentially during forward or backward passes of backpropagation.

Regularization in the context of regression – be it deterministic in the case of L^2 or L^1 or stochastic in the form of Dropout (Srivastava, Hinton, Krizhevsky, Sutskever & Salakhutdinov, 2014) – describes a set of techniques aimed at modifying a training algorithm so as to reduce overfitting on the training set. With regards to (11), the conceptual idea is that a modified optimizer allows to trade an increased bias of the estimator for an over proportional decrease in its variance, effectively reducing the MSE overall. In our experiments, we only regularize in time in the form of *early stopping*: While optimizing the weights and biases on the training set, we periodically check the performance on the validation set and save the model if a new minimum error is reached. When the error on the validation set begins to stall, training is stopped.

Batch normalization (BN) devised by Ioffe und Szegedy (2015) is very popular technique to facilitate and accelerate the training of deeper networks by addressing the mentioned *internal covariate shift*. It alters a network's topology by inserting normalization operations between linearities and non-linearities of each dense layer, effectively reducing the dependence of each node's input on the weights

Table 2: Reference model parameters $\mu^\dagger$ for Heston and rough Bergomi. Obtained from (Gatheral, 2011) and (Bayer et al., 2016) respectively.

Heston		rough Bergomi	
Parameter	Value	Parameter	Value
η	0.3877	η	1.9
ρ	-0.7165	ρ	-0.9
λ	1.3253	H	0.07
$\bar{v}$	0.0354	v_0	0.01
v_0	0.0174		

and biases of all nodes in previous layers. Our numerical experiments confirm a strongly regularizing effect of BN as is well-known in the literature, reducing the expressiveness of our networks considerably and hence leading to worse performance. Despite its success in allowing to train deeper networks, we hence decided to turn it off.

5 Numerical experiments

Here, we examine the performance of our approach by applying it to the option pricing models recalled in Section 2: First, we consider the Heston model as a test case and then the rough Bergomi model as a representative from the class of *rough* stochastic volatility models. Specifically, we look at the speed and accuracy of the learned implied volatility map $\varphi_{\text{NN}} : \mathcal{I} \to \mathbb{R}_+$. A systematic comparison of performance metrics between existing methods and our approach has been left for future research.

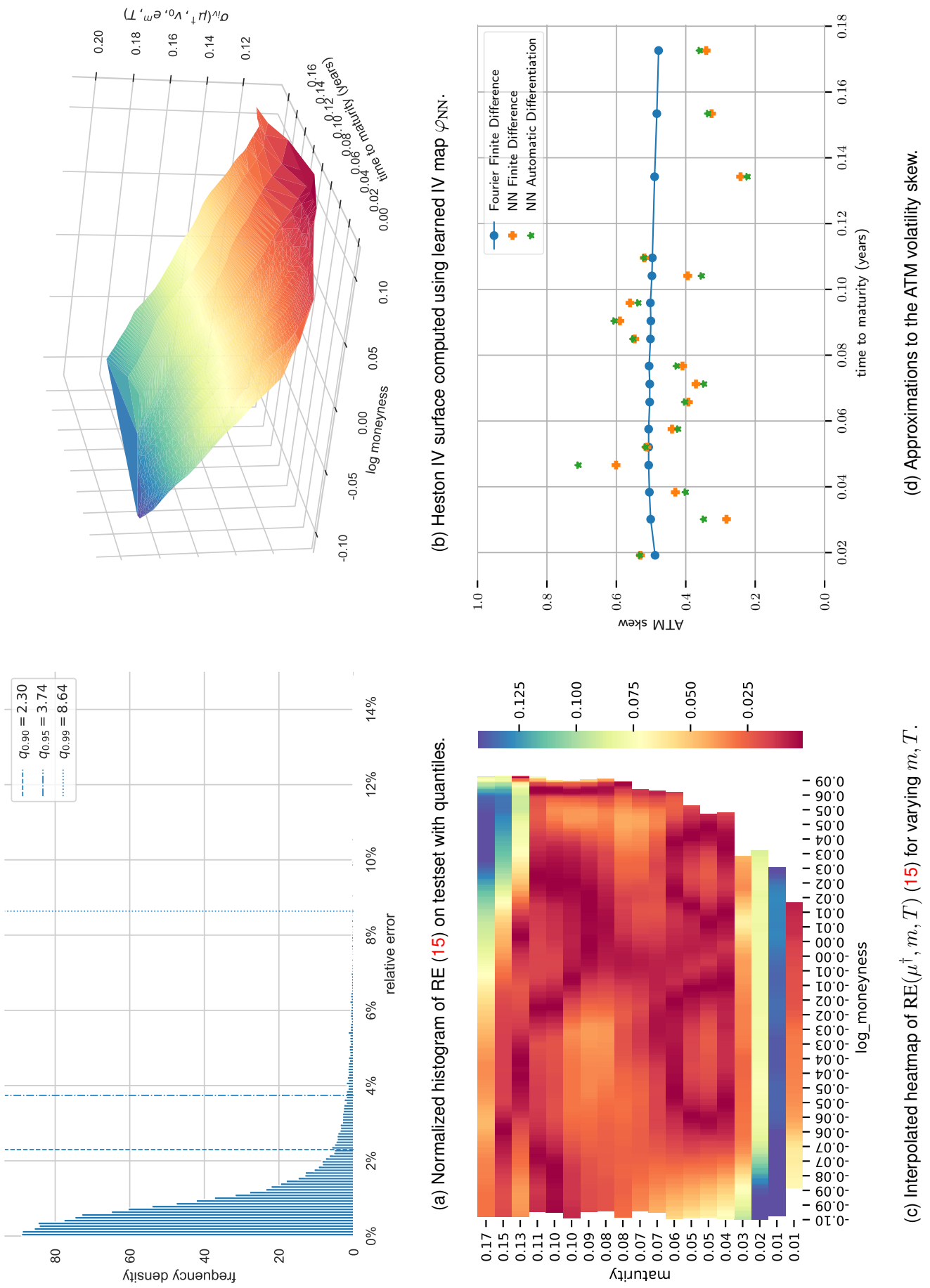
The Gaussian hyper parameter optimization and individual network training runs are performed on a local CPU-only compute server. Unless otherwise stated, all computations and performance measures referenced in this section are performed on a standard early 2015 Apple Mac Book with a 2.9 GHz Intel Core i5 CPU with no GPU used.

5.1 The Heston model

Following the approach outlined in Section 4.1, we estimate $\mathcal{K}_{(M,T)}$ using SPX Option Price data[2] from 15th February 2018. Empirically, we observe that a majority of the liquidity as proxied by inverse bid-ask spreads is concentrated in the small region given by $-0.1 \leq m \leq 0.28$ and $\frac{1}{365} \leq T \leq 0.2$ which is why for this test case we exclusively learn the IV map on this bounded domain. The size of the labeled set data $\mathcal{D}$ is $n = 990000$ of which we allocate $n_{\text{train}} = 900000$ samples to the training set and $n_{\text{valid}} = n_{\text{test}} = 45000$ to test and validation sets.

Single evaluations of the learned implied volatility map $\varphi_{\text{NN}} : \mathcal{I} \to \mathbb{R}_+$ and the associated Jacobian $\boldsymbol{J}_{\text{NN}} : O \to \mathbb{R}^{N\times m}$ are extremely fast with about 36ms on average to compute both together, making this neural network based approach at least competitive with existing Fourier-based schemes.

[2] Option prices for SPX Weeklys can be retrieved from a publicly available database at www.cboe.com/DelayedQuote/QuoteTableDownload.aspx.



(a) Normalized histogram of RE (15) on testset with quantiles.

(b) Heston IV surface computed using learned IV map φ_{NN}.

(c) Interpolated heatmap of $\text{RE}(\mu^\dagger, m, T)$ (15) for varying m, T.

(d) Approximations to the ATM volatility skew.

Figure 4: **Accuracy of the IV map φ_{NN} as learned by the Heston ReLU FCNN.**

To determine the accuracy of φ_{NN}, we define

$$\text{RE}(\mu, m, T) := \frac{|\varphi_{\text{NN}}(\mu, v_0, e^m, T) - \tilde{\varphi}(\mu, v_0, e^m, T)|}{\tilde{\varphi}(\mu, v_0, e^m, T)} \tag{15}$$

to be the relative error of the output of φ_{NN} with respect to that of a Fourier-based reference map $\tilde{\varphi}$ for model parameters μ, option parameters (m, T) and fixed spot variance v_0. Figure 4a shows a normalized histogram of relative errors on the test set where μ and (M, T) are allowed to vary across samples, demonstrating that empirically, φ_{NN} approximates $\tilde{\varphi}$ with a high degree of accuracy. In typical pricing or calibration scenarios, we are interested in the accuracy of φ_{NN} for some fixed model parameters μ which is why in Figures 4b, 4c and 4d, we fix $\mu = \mu^\dagger$ with $\mu^\dagger$ the reference model parameters in Table 2. In Figure 4b, we compute an IV point cloud using φ_{NN}, interpolate it using a (not necessarily arbitrage-free) Delaunay triangulation and recover a characteristic Heston-like model IV surface. Indeed, as the heatmap of interpolated relative errors in Figure 4c shows, these are small across most of the IV surface, with increased relative errors only for times on the short and long end which may be attributed to less training because of less liquidity. Finally, in Figure 4d, we plot three different approximations to the Heston ATM volatility skew for small times: A reference skew in blue obtained by a finite difference approximation using $\tilde{\varphi}$, another skew in orange obtained by the same method but applied to φ_{NN} and finally the exact ATM skew of φ_{NN} in green, available by automatic differentiation. As is to be expected, φ_{NN} recovers the characteristic flat behaviour for short times, the general drawback of bivariate diffusion models such as Heston.

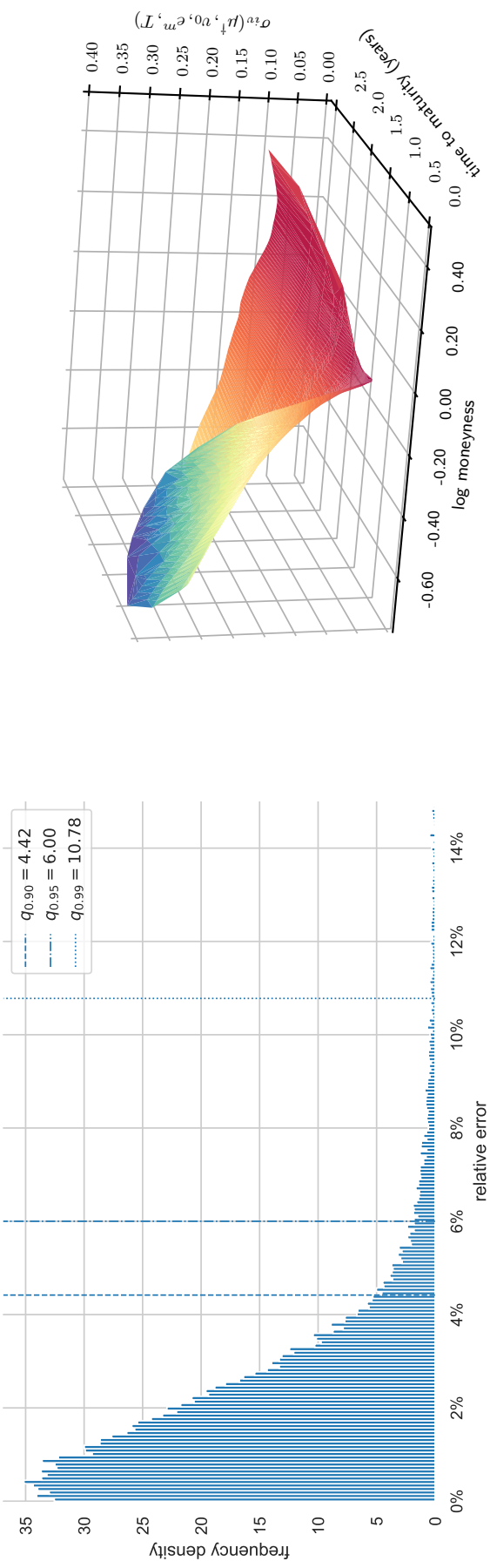
5.2 The rough Bergomi model

For simplicity, we consider the rough Bergomi model as introduced in Example 2.2 with a flat forward variance curve $\xi_0(t) = v_0 \in \mathbb{R}_+$ for $t \geq 0$. For the remainder of this work, we shall consider v_0 an additional model parameter. Again, following the approach outlined in Section 4.1, we estimate $\mathcal{K}_{(M,T)}$ using SPX Option Price data, this time from 19th May 2017[3]. We do not restrict the option parameter region considered and learn the whole surface with parameter bounds given by $-3.163 \leq m \leq 0.391$ and $0.008 \leq T \leq 2.589$. Of the one million synthetic data pairs sampled, 90% are allocated to the training set and 5% to validation and test sets respectively.

Recall that in this experiment we use the same network topology as in the Heston example. As is to be expected, the speed of single evaluations of the learned rough Bergomi IV map $\varphi_{\text{NN}} : \mathcal{I} \to \mathbb{R}_+$ and the associated Jacobian $\boldsymbol{J}_{\text{NN}} : O \to \mathbb{R}^{N \times m}$ are hence of the same order with about 36ms to compute both objects together, beating state of the art methods by magnitudes. Intuitively, the non-Markovian nature of rough Bergomi manifests itself in an increased model complexity and so it is unsurprising that the general accuracy of the rough Bergomi IV map φ_{NN} on the rough Bergomi test set is lower than its counterpart on the Heston test set (cf. 5a). On the other hand, for fixed model parameters $\mu = \mu^\dagger$ (cf. Table 2), the implied volatility map φ_{NN} recovers the characteristic rough Bergomi model IV surface (Figure 5b) with low relative error across most of the liquid parts of the IV surface (Figure 5c). It also exhibits the striking power law behaviour of the ATM volatility skew near zero (Figure 5d).

On the contrary, measuring the accuracy of the neural-network enhanced Levenberg-Marquardt scheme introduced in Section 3.1 is not a straightforward task. To see why, consider the small-time asymptotic formula for the BS implied volatility σ_{iv} of rough stochastic volatility models as derived by Bayer, Friz,

[3] Thanks to Jim Gatheral for providing us with this data set.



(a) Normalized histogram of RE (15) on testset with quantiles.

(b) Rough Bergomi IV surface computed using learned IV map φ_{NN}.

(c) Interpolated heatmap of $\text{RE}(\mu^\dagger, m, T)$ (15) for varying m, T.

(d) Approximations to the ATM volatility skew.

Figure 5: **Accuracy of the IV map φ_{NN} as learned by the rough Bergomi ReLU FCNN.**

Gulisashvili et al. (2017). With scaling parameter $\beta < \frac{2}{3}H$, their expansion applied to our setting yields

$$\sigma_{\text{iv}}(e^{k_t}, t) = \sqrt{v_0} + \frac{1}{2}\rho\eta\, C(H) k t^{\beta} + \mathcal{O}(t) \tag{16}$$

for small times $t \to 0$, time-scaled log moneyness $k_t = k t^{\frac{1}{2}-H+\beta}$ and constant $C(H)$ depending on H. Hence, at least for small times, all three model parameters enter multiplicatively either directly (ρ and η) or indirectly (H) into the second term in (16) which corrects the crude estimate given by spot volatility. A decrease in $|\rho|$ could hence for example be offset by an adequate increase in η and still yield the same IV. Mathematically speaking, for fixed moneyness and time to maturity, it is thus to be expected that the map φ_{NN} is non-injective in its model parameters on large parts of its model parameter input domain. Quantifying the accuracy of the deep calibration scheme by computing any form of distance between true and calibrated model parameters in model parameter space is hence nonsensical.

5.2.1 Bayesian parameter inference

Intuitively, we are interested in *quantifying the uncertainty* about model parameter estimates obtained by calibrating with the approximative IV map φ_{NN}. To this end, we switch to a Bayesian viewpoint and treat model parameters μ as random variables. The fundamental idea behind Bayesian parameter inference is to update prior beliefs $p(\mu)$ formalised in (9) with the likelihood $p(\boldsymbol{y} \mid \mu)$ of observing a given IV point cloud $\boldsymbol{y} \in \mathbb{R}^N$ to deduce a posterior (joint) distribution $p(\mu \mid \boldsymbol{y})$ over model parameters μ.

Formally, for pairs $\left(M^{(i)}, T^{(i)}\right)$ of moneyness & time to maturity, let an IV point cloud to calibrate against be given by

$$\boldsymbol{y} = \left[y_1\left(M^{(1)}, T^{(1)}\right), \ldots, y_N\left(M^{(N)}, T^{(N)}\right)\right]^T \in \mathbb{R}^N$$

and analogously, collect model IVs for model parameters μ as follows

$$\boldsymbol{\varphi}_{\text{NN}}(\mu) = \left[\varphi_{\text{NN}}\left(\mu, M^{(1)}, T^{(1)}\right), \ldots, \varphi_{\text{NN}}\left(\mu, M^{(N)}, T^{(N)}\right)\right]^T \in \mathbb{R}^N.$$

We perform a liquidity-weighted nonlinear Bayes regression. Mathematically, for heteroskedastic sample errors $\sigma_i > 0, i = 1, \ldots, N$, we postulate

$$\boldsymbol{y} = \boldsymbol{\varphi}_{\text{NN}}(\mu) + \boldsymbol{\varepsilon}, \quad \boldsymbol{\varepsilon} \sim \mathcal{N}\left(0, \text{diag}[\sigma_1^2, \ldots, \sigma_N^2]\right)$$

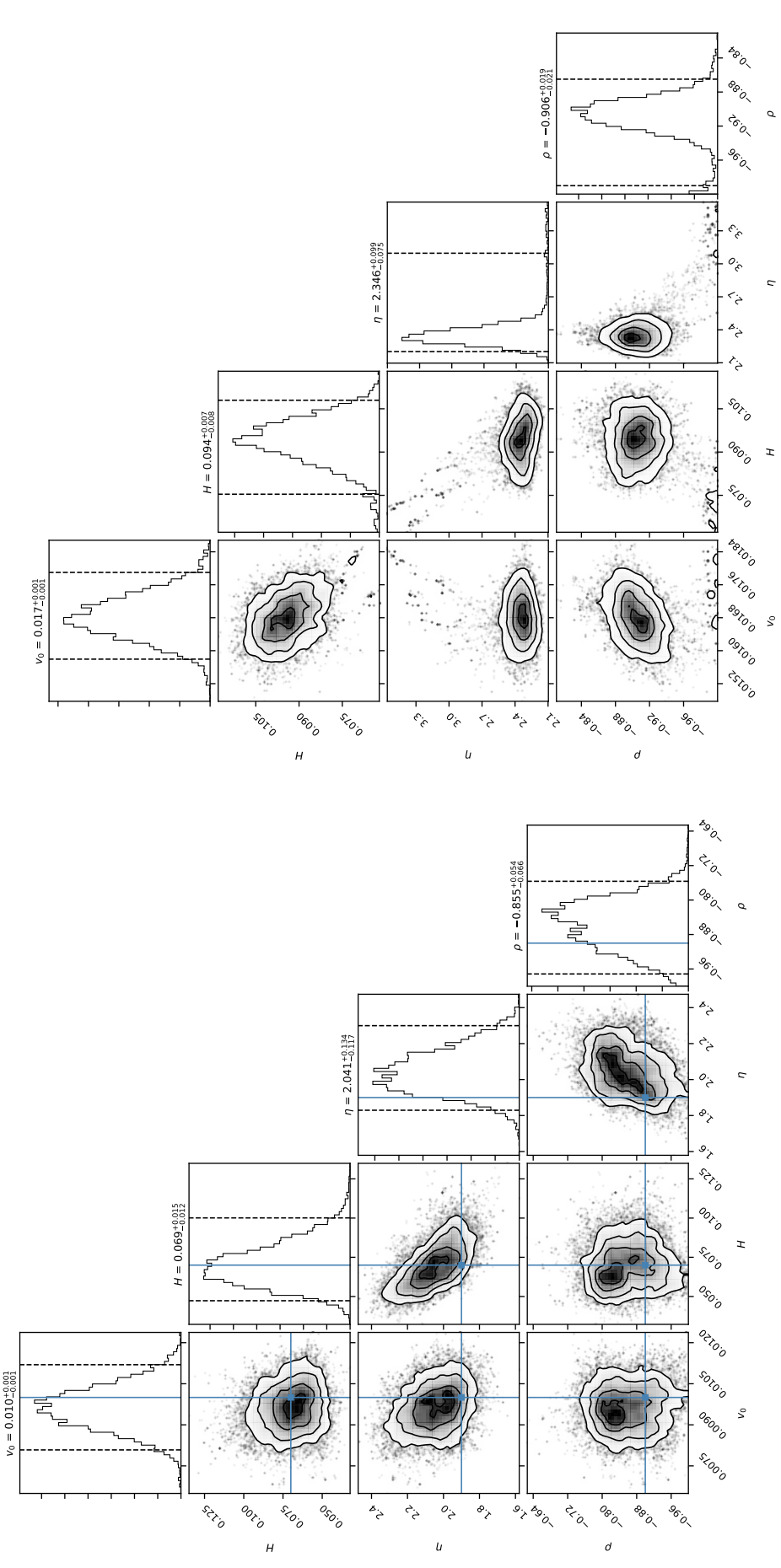
so that for some diagonal weight matrix $\boldsymbol{W} = \text{diag}\left[w_1, \ldots, w_N\right] \in \mathbb{R}^{N\times N}$, the liquidity-weighted residuals are distributed as follows

$$\boldsymbol{W}^{\frac{1}{2}}\left[\boldsymbol{y} - \boldsymbol{\varphi}_{\text{NN}}(\mu)\right] \sim \mathcal{N}\left(0, \text{diag}[w_1\sigma_1^2, \ldots, w_N\sigma_N^2]\right).$$

In other words, we assume that the joint likelihood $p\left(\boldsymbol{y} \mid \mu\right)$ of observing data $\boldsymbol{y}$ is given by a multivariate normal. In absence of an analytical expression for the posterior (joint) probability $p(\mu|\boldsymbol{y}) \propto p(\boldsymbol{y}|\mu)p(\mu)$, we approximate it numerically using MCMC techniques (Foreman-Mackey, Hogg, Lang & Goodman, 2013) and plot the one- and two-dimensional projections of the four-dimensional posterior by means of an MCMC plotting library (Foreman-Mackey, 2016).

We perform two experiments. First, fixing $\mu = \mu^\dagger$, we generate a synthetic IV point cloud

$$\boldsymbol{y}_{\text{synth}} = \left[\tilde{\varphi}\left(\mu^\dagger, M^{(1)}, T^{(1)}\right), \ldots, \tilde{\varphi}\left(\mu^\dagger, M^{(N)}, T^{(N)}\right)\right] \in \mathbb{R}^N$$



(a) Bayes calibration against synthetic IV surface computed for model parameters $\mu^\dagger$. (b) Liquidity-weighted Bayes calibration against SPX market IV surface from 19th May 2017. Liquidity proxies given by inverse bid-ask-spreads. Solid vertical blue lines indicate true parameter values.

Figure 6: **1d- and 2d-projections of 4d Bayesian posterior over rBergomi model parameters.** On diagonal, univariate histograms of model parameters with titles stating median $\pm$ the delta to 2.5% and 97.5% quantiles. Dashed vertical lines indicate those quantiles. Off-diagonal, 2d histograms of 2d projections of MCMC samples together with isocontours from 2d Gaussian KDE.

using the reference method $\tilde{\varphi}$. Next, we perform a non-weighted Bayesian calibration against the synthetic surface and collect the numerical results in Figure 6a. If the map φ_{NN} is sufficiently accurate for calibration, the computed posterior should attribute a large probability mass around $\mu^\dagger$. The results in Figure 6a are quite striking in several ways: (1) From the univariate histograms on the diagonal it is clear that the calibration routine has identified sensible model parameter regions covering the true values. (2) Histograms are unimodal and its peaks close or identical to the true parameters. (3) The isocontours of the 2d Gaussian KDE in the off-diagonal pair plots for (η, H) and (η, ρ) show exactly the behaviour expected from the reasoning in the last section: Since increases or decreases in one of η, H or ρ can be offset by adequate changes in the others with no impact on the calculated IV, the Bayes posterior cannot discriminate between such parameter configurations and places equal probability on both combinations. This can be seen by the diagonal elliptic probability level sets.

In a second experiment, we want to check whether the inaccuracy of φ_{NN} allows for a successful calibration against market data. To this end, we perform a liquidity-weighted Bayesian regression against SPX IVs from 19th May 2017. For bid and ask IVs $a_i > 0$ and $b_i > 0$ respectively, we proxy the IV of the mid price by $m_i := \frac{a_i+b_i}{2}$. With spread defined by $s_i = a_i - b_i \geq 0$, all options with $s_i/m_i \geq 5\%$ are removed because of too little liquidity. Weights are chosen to be $w_i = \frac{m_i}{a_i - m_i} \geq 0$, effectively taking inverse bid-ask spreads as a proxy for liquidity. Finally, σ_i are proxied by a fractional of the spread s_i. The numerical results in Figure 6b further confirm the accuracy of φ_{NN}: (1) As can be seen on the univariate histograms on the diagonal, the Bayes calibration has again identified sensible model parameter regions in line with what is to expected. (2) Said histograms are again unimodal with peaks at or close to values previously reported by Bayer et al. (2016). (3) Quite strikingly, at a first glance, the effect of the diagonal probability level sets in the off-diagonal plots as documented in Figure 6a cannot be confirmed here. However, the scatter plots in the diagrams do reveal some remnants of that phenomenon.

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., ... Isard, M. (2016). Tensorflow: a system for large-scale machine learning. In *OSDI* (Bd. 16, S. 265–283).

Alòs, E., León, J. A. & Vives, J. (2007). On the short-time behavior of the implied volatility for jump-diffusion models with stochastic volatility. *Finance and Stochastics*, *11* (4), 571–589.

Ametrano, F., Ballabio, L., Bianchetti, M., Césaré, N., Eddelbuettel, D., Firth, N., ... Marchioro, M. (2015). *Quantlib: A free/open-source library for quantitative finance.*

Baydin, A. G., Pearlmutter, B. A., Radul, A. A. & Siskind, J. M. (2015). *Automatic Differentiation in Machine Learning: a Survey.*

Bayer, C., Friz, P. K., Gassiat, P., Martin, J. & Stemper, B. (2017). A regularity structure for rough volatility. *ArXiv e-prints*.

Bayer, C., Friz, P. K. & Gatheral, J. (2016). Pricing under rough volatility. *Quantitative Finance*, *16* (6), 887-904.

Bayer, C., Friz, P. K., Gulisashvili, A., Horvath, B. & Stemper, B. (2017). Short-time near-the-money skew in rough fractional volatility models. *ArXiv e-prints*.

Bennedsen, M., Lunde, A. & Pakkanen, M. S. (2016). Decoupling the short- and long-term behavior of stochastic volatility. *ArXiv e-prints*.

Bennedsen, M., Lunde, A. & Pakkanen, M. S. (2017). Hybrid scheme for Brownian semistationary processes. *Finance and Stochastics*, *21* (4), 931–965.

Black, F. & Scholes, M. (1973). The Pricing of Options and Corporate Liabilities. *Journal of Political Economy*, *81* (3), 637-654.

Clevert, D.-A., Unterthiner, T. & Hochreiter, S. (2015). Fast and accurate deep network learning by exponential linear units (elus). *ArXiv e-prints*.

Elfwing, S., Uchibe, E. & Doya, K. (2018). Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Networks*.

Forde, M. & Zhang, H. (2017). Asymptotics for rough stochastic volatility models. *SIAM Journal on Financial Mathematics*, *8* (1), 114-145.

Foreman-Mackey, D. (2016). corner. py: Scatterplot matrices in python. *The Journal of Open Source Software*, *1*.

Foreman-Mackey, D., Hogg, D. W., Lang, D. & Goodman, J. (2013). emcee: the MCMC hammer. *Publications of the Astronomical Society of the Pacific*, *125* (925), 306.

Fukasawa, M. (2011). Asymptotic analysis for stochastic volatility: martingale expansion. *Finance and Stochastics*, *15* (4), 635–654.

Fukasawa, M. (2017). Short-time at-the-money skew and rough fractional volatility. *Quantitative Finance*, *17* (2), 189-198.

Gatheral, J. (2011). *The volatility surface: a practitioner's guide*. John Wiley & Sons.

Gatheral, J., Jaisson, T. & Rosenbaum, M. (2018). Volatility is rough. *Quantitative Finance*, 1–17.

Glorot, X. & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics* (S. 249–256).

Goodfellow, I., Bengio, Y. & Courville, A. (2016). *Deep learning* (Bd. 1). MIT Press Cambridge.

Hagan, P. S., Kumar, D., Lesniewski, A. S. & Woodward, D. E. (2002). Managing smile risk. *The Best of Wilmott*, *1*, 249–296.

He, K., Zhang, X., Ren, S. & Sun, J. (2015). Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In *Proceedings of the IEEE International Conference on Computer Vision* (S. 1026–1034).

Hernandez, A. (2017). Model calibration with neural networks. *Risk*.

Heston, S. L. (1993). A closed-form solution for options with stochastic volatility with applications to bond and currency options. *The Review of Financial Studies*, *6* (2), 327–343.

Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural networks*, *4* (2), 251–257.

Horvath, B., Jacquier, A. & Muguruza, A. (2017). Functional central limit theorems for rough volatility. *ArXiv e-prints*.

Hull, J. & White, A. (1990). Pricing interest-rate-derivative securities. *The Review of Financial Studies*, *3* (4), 573–592.

Ioffe, S. & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *ArXiv e-prints*.

Jäckel, P. (2015). Let's be rational. *Wilmott*, *2015* (75), 40–53.

Kingma, D. P. & Ba, J. (2014). Adam: A method for stochastic optimization. *ArXiv e-prints*.

Krizhevsky, A., Sutskever, I. & Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems* (S. 1097–1105).

LeCun, Y., Bengio, Y. & Hinton, G. (2015). Deep learning. *Nature*, *521* (7553), 436.

Levenberg, K. (1944). A method for the solution of certain non-linear problems in least squares. *Quarterly of Applied Mathematics*, *2* (2), 164–168.

Mandelbrot, B. B. & Van Ness, J. W. (1968). Fractional Brownian motions, fractional noises and applications. *SIAM Review*, *10* (4), 422–437.

Marquardt, D. W. (1963). An algorithm for least-squares estimation of nonlinear parameters. *Journal of the Society for Industrial and Applied Mathematics*, *11* (2), 431–441.

McCrickerd, R. & Pakkanen, M. S. (2018). Turbocharging Monte Carlo pricing for the rough Bergomi model. *Quantitative Finance*, 1–10.

Ramachandran, P., Zoph, B. & Le, Q. V. (2017). Searching for activation functions. *ArXiv e-prints*.

Rasmussen, C. E. & Williams, C. K. (2006). *Gaussian process for machine learning*. MIT Press.

Scott, D. W. (2015). *Multivariate density estimation: theory, practice, and visualization*. John Wiley & Sons.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., ... Bolton, A. (2017). Mastering the Game of Go without Human Knowledge. *Nature*, *550* (7676), 354.

Simonyan, K. & Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. *ArXiv e-prints*.

Snoek, J., Larochelle, H. & Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems* (S. 2951–2959).

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, *15* (1), 1929–1958.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S. & Anguelov, D. (2015). Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (S. 1–9).