Deep Concerns

Deep calibration, pricing and hedging:
for rough volatility models ...and beyond

Blanka Horvath
King's College London
Based on joint works with A. Muguruza & M. Tomas
and with C. Bayer and B. Stemper

Fourth Conference on Ambit Fields and Related Topics
4-8 August, 2019, Sandbjerg Estate, Snderborg, Denmark

The talk is based on the papers **Deep Learning Volatility**
and
Deep calibration of rough volatility models
and
Deep hedging under rough volatility

All codes are available and downloadable from github, and also from the **Rough Volatility Network** website.

Deep Learning for Pricing, Hedging and Calibration

The surge in ML and AI approaches to financial data is transforming the horizons of quantitative finance.

The future of quantitative finance ...

- ... shifting away from classical models towards ML directly applied to data.
- ... Deep networks replacing classical models?

ML is transforming the horizons of mathematical modelling

Changes induced by the increase in computing power availability

0. Black-Scholes
   (inflexible, but fully analytic solutions)
1. Currently used stochastic market models
   (few parameters, approximative solutions, often full error analysis available)
2. Data-driven DNN models
   (fully flexible: many parameters, but often "black-box" and explainability issues)

ML is transforming the horizons of mathematical modelling

Changes induced by the increase in computing power availability

0. Black-Scholes
   (inflexible, but fully analytic solutions)
1. Currently used stochastic market models
   (few parameters, approximative solutions, often full error analysis available)
2. Data-driven DNN models
   (fully flexible: many parameters, but often "black-box" and explainability issues)

$\Rightarrow$ Augmenting and extending currently prevalent stoachstic models by ML features

ML is transforming the horizons of mathematical modelling

Changes induced by the increase in computing power availability

0. Black-Scholes
   (inflexible, but fully analytic solutions)
1. Currently used stochastic market models
   (few parameters, approximative solutions, often full error analysis available)
2. Data-driven DNN models
   (fully flexible: many parameters, but often "black-box" and explainability issues)

$\Rightarrow$ Augmenting and extending currently prevalent stoachstic models by ML features

Classical models do remain without any doubt essential and necessary to develop a thorough understanding of risk scenarios

What is a good "model" in the ML era?

Beyond classical?

By the increased technological progress at hand we can readily create

1. **SL-based and controllable**: Classical models, mixture of "expert" models from existing ones (more flexibility for different regimes in a single meta-model, more parameters)

   Solving high-dimensional equations by DNN (convergence analysis available)

2. **RL-based**: Fully flexible: model is learned from data (objective, data, architecture): Deep Hedging $\Rightarrow$ The architecture should then reflect the structure of the data

ML starts changing the shape of mathematical modelling

Deep Concerns?

Explainability, Transparency, Black-Box issues, Robustness, Sensitivity to training data, Data privacy, ...Sleepwalking into an AI crisis?...

Keyword: "Continuity"
How far do our current risk management mechanisms take us?

ML starts changing the shape of mathematical modelling

Deep Concerns?

Explainability, Transparency, Black-Box issues, Robustness, Sensitivity to training data, Data privacy, ...Sleepwalking into an AI crisis?...

Keyword: "Continuity"
How far do our current risk management mechanisms take us?

Today's talk: meaningful ways to swiftly adapt some aspects of deep learning's advances into calibration practice upon currently existing libraries for risk management

Deep Learning for Option Price Approximation and Calibration

Active and rapidly developing area with a long history of contributions to NN pricing and calibration

Hutchinson et al (1994), Avellaneda-Carelli-Stella (1998), Morelli et al (2004)

Hernandez (2016), Bayer-Stemper (2018), McGhee (2018), Ferguson & Green (2018), Elouerkharaoui , Kondratyev (2017),
Spiegeleer-Madan-Reyners-Schoutens (2018),....
Working papers: Babbar-McGhee, Chuciero-Khosrawi-Teichmann

Our approach

Step 0: Choose a (rough) SV Model and Generate Data

Step 1: NN training

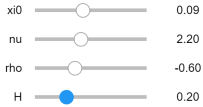
Train Neural Network to learn pricing map

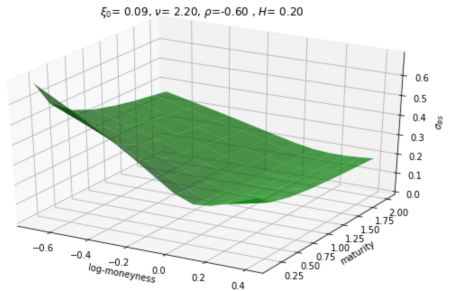
Step 2: Model calibration

Apply standard optimisers to Neural Network approximation to calibrate model to Market Data

Training the network

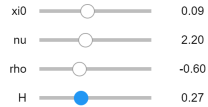
Training F is done solving $\hat{w} = \text{argmin}_{w \in \Omega} \mathcal{L}\left(\{\widetilde{f}(w, x_i)\}_{i=1}^{N}, \{\widetilde{P}(x_i)\}_{i=1}^{N}\right)$

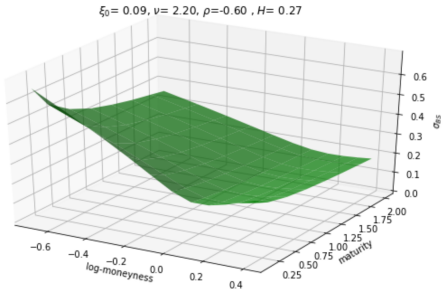




Training the network

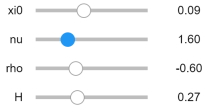
Training F is done solving $\hat{w} = \text{argmin}_{w \in \Omega} \mathcal{L}\left(\{\widetilde{f}(w, x_i)\}_{i=1}^{N}, \{\widetilde{P}(x_i)\}_{i=1}^{N}\right)$

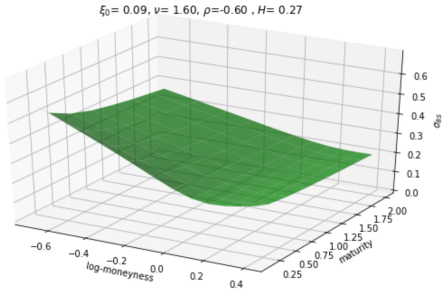




Training the network

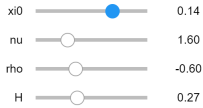
Training F is done solving $\hat{w} = \text{argmin}_{w \in \Omega} \mathcal{L}\left(\{\widetilde{f}(w, x_i)\}_{i=1}^{N}, \{\widetilde{P}(x_i)\}_{i=1}^{N}\right)$

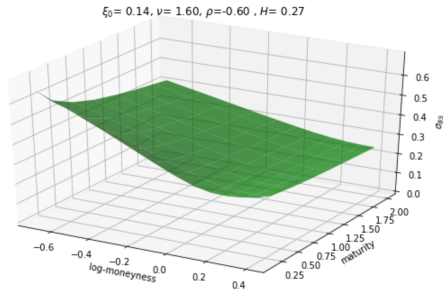




Training a NN to learn the pricing map

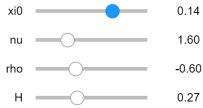
Training F is done solving $\hat{w} = \text{argmin}_{w \in \Omega} \mathcal{L}\left(\{\widetilde{f}(w, x_i)\}_{i=1}^{N}, \{\widetilde{P}(x_i)\}_{i=1}^{N}\right)$

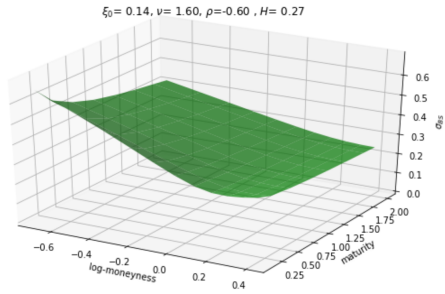




Training a NN to learn the pricing map

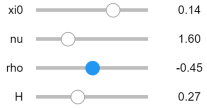
Training F is done solving $\hat{w} = \text{argmin}_{w \in \Omega} \mathcal{L}\left(\{\widetilde{f}(w, x_i)\}_{i=1}^{N}, \{\widetilde{P}(x_i)\}_{i=1}^{N}\right)$

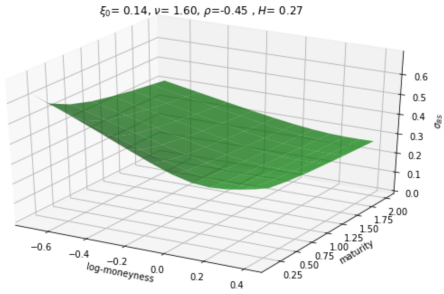




Training a NN to learn the pricing map

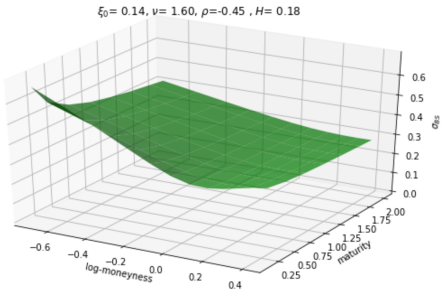
Training F is done solving $\hat{w} = \text{argmin}_{w \in \Omega} \mathcal{L}\left(\{\widetilde{f}(w, x_i)\}_{i=1}^{N}, \{\widetilde{P}(x_i)\}_{i=1}^{N}\right)$





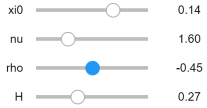
Training a NN to learn the pricing map

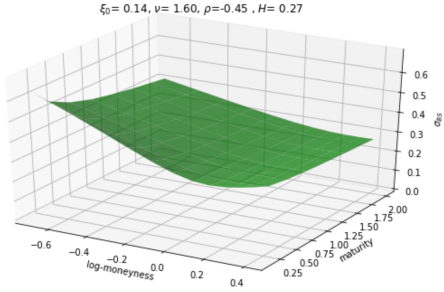
Training F is done solving $\hat{w} = \text{argmin}_{w \in \Omega} \mathcal{L}\left(\{\widetilde{f}(w, x_i)\}_{i=1}^N, \{\widetilde{P}(x_i)\}_{i=1}^N\right)$



Training a NN to learn the pricing map

Training F is done solving $\hat{w} = \text{argmin}_{w \in \Omega} \mathcal{L}\left(\{\widetilde{f}(w, x_i)\}_{i=1}^{N}, \{\widetilde{P}(x_i)\}_{i=1}^{N}\right)$





Our approach

Step 0: Choose a (rough) SV Model and Generate Data

Step 1: NN training

Train Neural Network to learn pricing map

Step 2: Model calibration

Apply standard optimisers to Neural Network approximation to calibrate model to Market Data

Our approach

Questions:

- **Why not calibrate directly to data (bypassing models)?** No feedback loops, and black-boxes: risk management, control over extreme scenarios, meaning of model parameters...
- **Why separate the price approximation and calibration tasks? Why not do it all together?** The pontentials for speedup lie mainly in the price approximation step (i). Due to the speedup, on can even use global optimizers (DE) in the calibration step (ii) to address the problem of potential multiple local minima.

Advantages of Separating Neural Network Pricing from Calibration

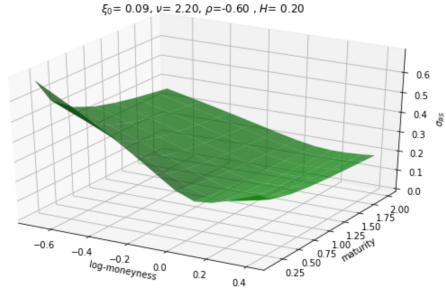
- The availability of training data for training the deep neural network is not an issue as it is synthetically generated by traditional numerical methods.
- The interpretability of model parameters $\theta \in \Theta$ remains the same as for traditional stochastic models. Traditional models can be used as a benchmark for risk management purposes. The NN is only used as a speed-up for pricing.
- By learning the pricing map from model parameters to expected payoffs we relocate the time-consuming numerical simulation and evaluation procedure into an off-line pre-preocessing. Even rough volatility models can be calibrated accurately within milliseconds.

xi0 0.09

nu 2.20

rho -0.60

H 0.20



Our Approach: Objectives

1. **accurate** calibration (i.e. close fits of models to market data, errors within a few bp) that also performs well on unseen (out of sample) data
2. **control over risk scenarios** maintaining a close link to classical models so that existing risk management libraries remain valid
3. **speedups** in on-line calibration time $\Rightarrow$ an array of accurate numerical pricing methods (Finite Elements, Monte Carlo, ...) become practically instantaneous (up to 35.000 times faster on-line calibration).

A NN Perspective on Pricing and (ϵ -) Calibration

- Let $\mathcal{M}$ denote a model (BS, Heston, SABR, Bergomi, Rough Volatility...)

A NN Perspective on Pricing and (ϵ -) Calibration

- Let $\mathcal{M}$ denote a model (BS, Heston, SABR, Bergomi, Rough Volatility...)
- $\Theta_{\mathcal{M}}$ the set of all possible parameter combinations $\theta \in \Theta$ in this model
  for example if $\mathcal{M}$=Black Scholes, then $\Theta = \{\sigma > 0\}$ and for any $\sigma > 0$

A NN Perspective on Pricing and (ϵ -) Calibration

- Let $\mathcal{M}$ denote a model (BS, Heston, SABR, Bergomi, Rough Volatility...)
- $\Theta_{\mathcal{M}}$ the set of all possible parameter combinations $\theta \in \Theta$ in this model

  for example if $\mathcal{M}$=Black Scholes, then $\Theta = \{\sigma > 0\}$ and for any $\sigma > 0$
- $P(\mathcal{M}(\theta)) =$ true no arbitrage price of an option for the chosen parameter combination θ

  for example if $\mathcal{M}$=Black Scholes, then $P(\mathcal{M}(\theta)) = P(BS(\sigma))$ is given by the Black-Scholes formula

A NN Perspective on Pricing and (ϵ -) Calibration

- Let $\mathcal{M}$ denote a model (BS, Heston, SABR, Bergomi, Rough Volatility...)
- $\Theta_{\mathcal{M}}$ the set of all possible parameter combinations $\theta \in \Theta$ in this model

  for example if $\mathcal{M}$=Black Scholes, then $\Theta = \{\sigma > 0\}$ and for any $\sigma > 0$
- $P(\mathcal{M}(\theta)) =$ true no arbitrage price of an option for the chosen parameter combination θ

  for example if $\mathcal{M}$=Black Scholes, then $P(\mathcal{M}(\theta)) = P(BS(\sigma))$ is given by the Black-Scholes formula
- $\widetilde{P}$ is the numerical approximation of the true price P

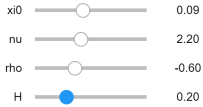
  Asymptotic approximation, or Finite differences, Monte Carlo approximation.... For most models usually this is the only thing we actually have available. Traditionally we require: $P(\mathcal{M}(\theta)) = \widetilde{P}(\mathcal{M}(\theta)) + \mathcal{O}(\epsilon)$

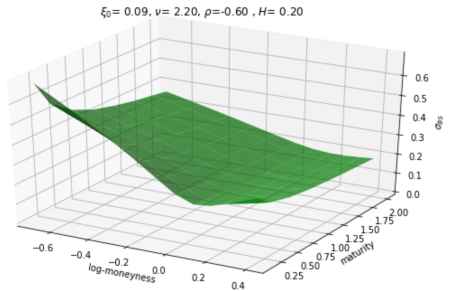
Training the network

Training F is done solving $\hat{w} = \text{argmin}_{w \in \Omega} \mathcal{L}\left(\{\widetilde{f}(w, x_i)\}_{i=1}^{N}, \{\widetilde{P}(x_i)\}_{i=1}^{N}\right)$

Training the network

Training F is done solving $\hat{w} = \text{argmin}_{w \in \Omega} \mathcal{L}\left(\{\widetilde{f}(w, x_i)\}_{i=1}^{N}, \{\widetilde{P}(x_i)\}_{i=1}^{N}\right)$





A NN Perspective on Pricing and (ϵ -) Calibration

1. **Accuracy objective**: $\widetilde{f}(\theta) = P(\mathcal{M}(\theta)) + \mathcal{O}(\epsilon) \quad$ for any $\quad \theta \in \Theta$

   We show that for any parameter combination $\theta \in \Theta$ of a stochastic model $\mathcal{M}(\Theta)$

$$\widetilde{f}(\theta) = \widetilde{P}(\mathcal{M}(\theta)) + \mathcal{O}(\epsilon) \quad \text{whenever} \quad \widetilde{P}(\mathcal{M}(\theta)) = P(\mathcal{M}(\theta)) + \mathcal{O}(\epsilon) \qquad (1)$$

That is, the network approximation $\widetilde{f}$ of the true option price P remains within the approximation precision of the numerical pricer $\widetilde{P}$.
The calibration via NN is as accurate as with traditional numerical methods.

Neural Network training setups:

- Pointwise approach (seems like the natural first approach):

$$\Theta \times [0, T] \times \mathbb{K} \longrightarrow \mathbb{R}$$

- The image based approach (faster training and generalises better):

$$\Theta \longrightarrow \mathbb{R}^{8 \times 11}$$

Pointwise learning

- Learn the map $P(\theta, T, k) = \sigma_{BS}^{\mathcal{M}(\theta)}(T, k)$ via neural network $\widetilde{F}(\theta, T, k) := F(\theta, \hat{z}, T, k)$ where

$$F^* : \Theta \times [0, T_{max}] \times [k_{min}, k_{max}] \longrightarrow \mathbb{R} \qquad (2)$$
$$(\theta, T, k) \mapsto F^*(\theta, T, k)$$

Image-based learning

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

$$F^* : \Theta \longrightarrow \mathbb{R}^{n \times m} \qquad (3)$$
$$\theta \mapsto F^*(\theta)$$

Context

Learn with the right amount of context

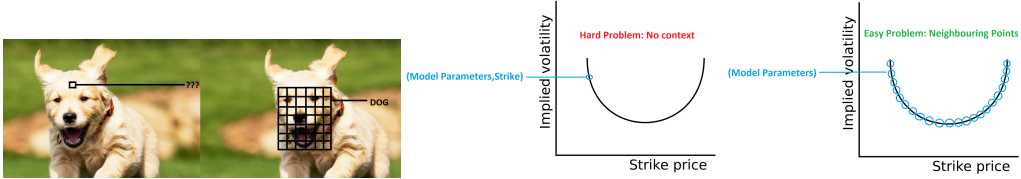
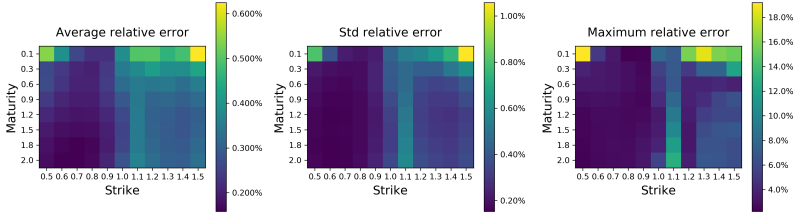
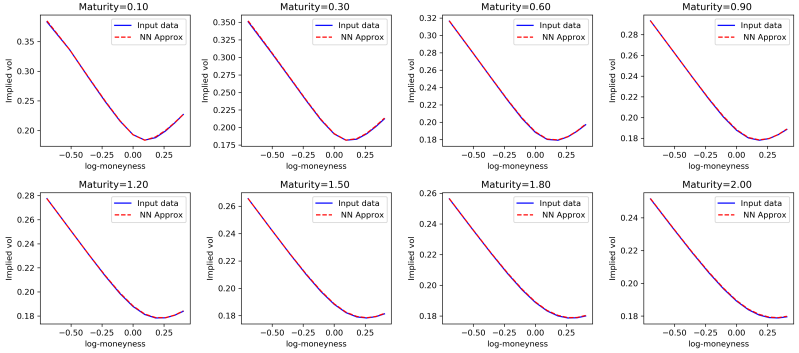


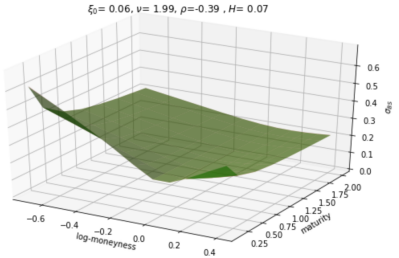
Figure 1: **Left:** Context of pixels in image-recognition problems. **Right:** Creating context in financial problems. Neighbouring points for different strikes help create context and reduce problem complexity.

Accuracy of the Neural-Network-calibrated implied volatility surface for the rBergomi model

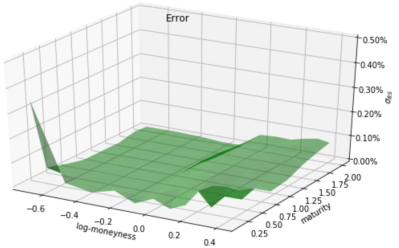


Accuracy of the Neural-Network-calibrated implied volatility surface for the rBergomi model





<matplotlib.figure.Figure at 0x2233d8c6240>



Advantages of separatig Neural Network Pricing from Calibration

1. **accurate** calibration (i.e. close fits of models to market data, errors within a few bp) that also performs well on unseen (out of sample) data
2. **control over risk scenarios** maintaining a close link to classical models so that existing risk management libraries remain valid
3. **speedups** in on-line calibration time $\Rightarrow$ an array of accurate numerical pricing methods (Finite Elements, Monte Carlo, ...) become practically instantaneous.

   By learning the pricing map from model parameters to expected payoffs we relocate the time-consuming numerical simulation and evaluation procedure into an off-line pre-preocessing. Even rough volatility models can be calibrated accurately within milliseconds.

Reminder: Rough Volatility

"**Rough volatility**" refers to the idea that sample paths of the log volatility $\log(\sigma_t)$, $t \geq 0$ are rougher than the sample paths of Brownian motion, the stock price is governed by $(S_t)_{t \geq 0}$:

$$\frac{dS_t}{S_t} = \mu_t dt + \sigma_t dB_t, \quad t \geq 0.$$

- Rough volatility models have been around since October 2014 (see the Rough Volatility website for a chronicle of developments)
- Superior to standard models in many areas: in volatility forecasting, pricing ...

Reminder: Rough Volatility

"**Rough volatility**" refers to the idea that sample paths of the log volatility $\log(\sigma_t)$, $t \geq 0$ are rougher than the sample paths of Brownian motion, the stock price is governed by $(S_t)_{t \geq 0}$:

$$\frac{dS_t}{S_t} = \mu_t dt + \sigma_t dB_t, \quad t \geq 0.$$

- Rough volatility models have been around since October 2014 (see the Rough Volatility website for a chronicle of developments)
- Superior to standard models in many areas: in volatility forecasting, pricing ...
- Relaxing the assumption of independence of volatility increments was crucial for the superior performance of rough volatility models $\Rightarrow$ but: several standard pricing methods no longer available & naive Monte Carlo methods slow
- Calibration time has been a bottleneck for rough volatility several advances have been made to pricing by [BLP '15, MP '17, HJM '17].

Back to NN solutions

1. **accurate** calibration (i.e. close fits of models to market data, errors within a few bp) that also performs well on unseen (out of sample) data
2. **control over risk scenarios** maintaining a close link to classical models so that existing risk management libraries remain valid
3. **speedups** in on-line calibration time $\Rightarrow$ an array of accurate numerical pricing methods (Monte Carlo, ...) become practically instantaneous.

   $\Rightarrow$ Opens the door for the use of (i) more robust but slower numerical pricers (Monte Carlo...) (ii) more involved models: here Rough Bergomi

3) Speedup per functional evaluation comparted to MC

	MC Pricing 1F Bergomi Full Surface	MC Pricing rBergomi Full Surface	NN Pricing Full Surface	NN Gradient Full Surface	Speed up NN vs. MC
Flat forward variance	300.000 μs	500.000 μs	$14,3$ μs	47 μs	$21.000 - 35.000$
Piecewise constant forward variancce	300.000 μs	500.000 μs	$30,9$ μs	113 μs	$9.000 - 16.000$

Step 0: Generate Data from a Model

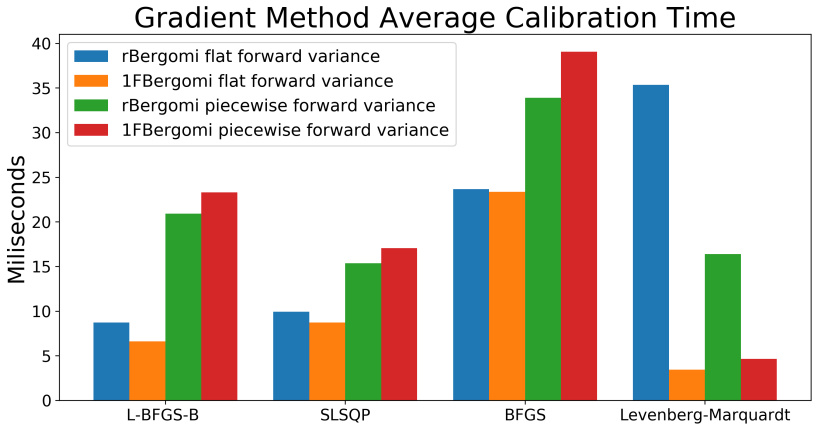
Step 1: NN training

Train Neural Network to learn pricing map

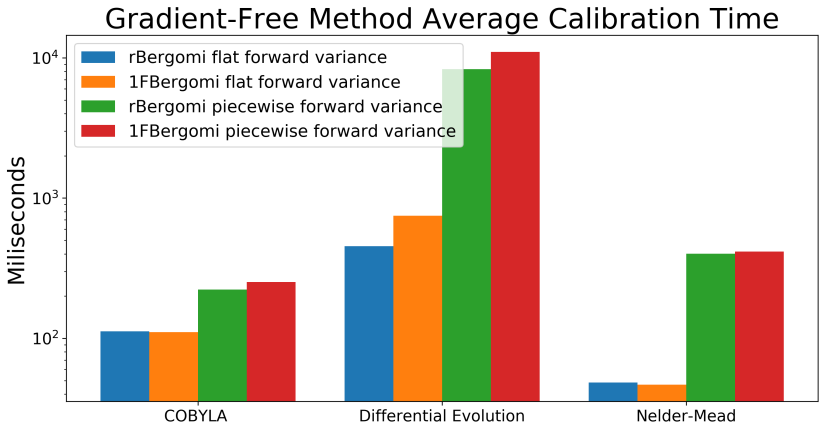
Step 2: Model calibration

Apply standard optimisers to Neural Network approximation to calibrate model to Market Data

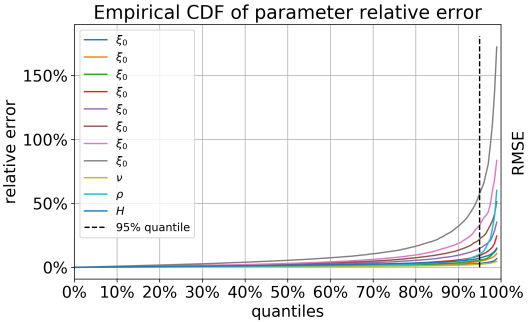
Calibration times for the Neural-Network-IVsurface

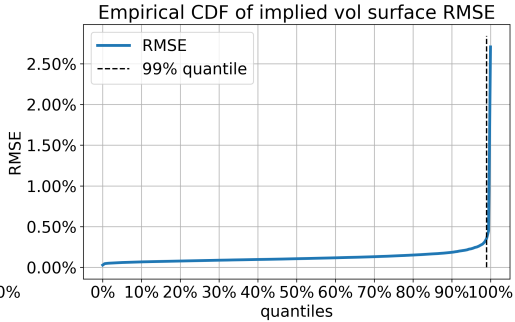


Calibration times for the Neural-Network-IVsurface

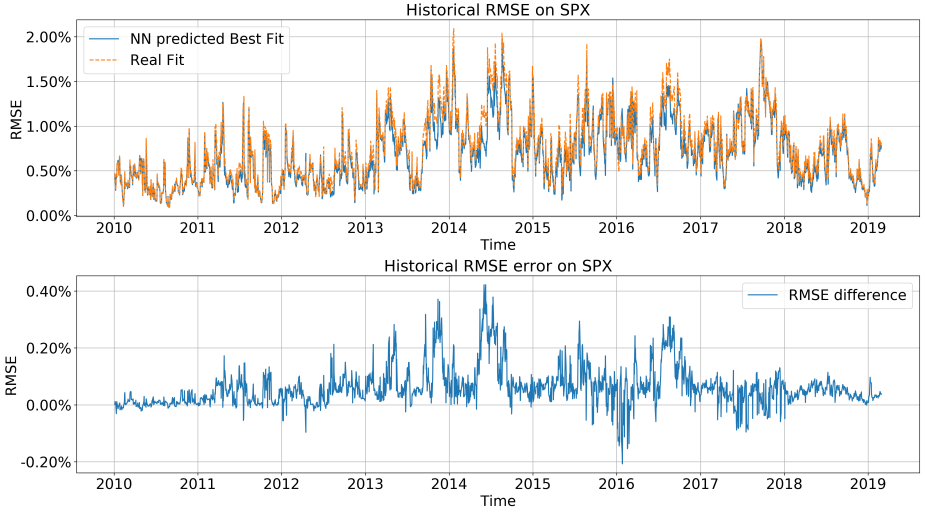


Calibration experiments: Simulated data (rBergomi)

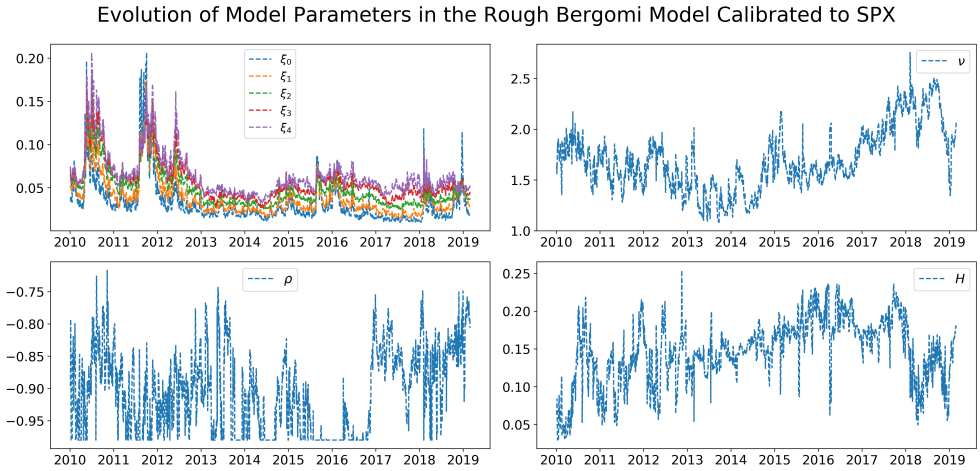




Calibration experiments: historical data (rBergomi)



Calibration experiments on historical data: Evolution of rBergomi parameters



Further Applications: Exotic Payoffs - Digital Barriers

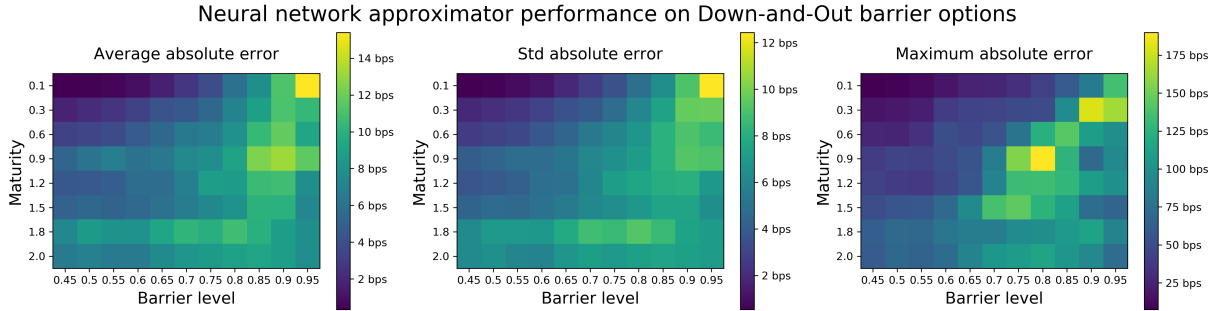
$$P^{Down-and-In}(B,T) = \mathbb{E}\left[1_{\{\tau_B \leq T\}}\right] \tag{4}$$

$$P^{Down-and-Out}(B,T) = \mathbb{E}\left[1_{\{\tau_B \geq T\}}\right] \tag{5}$$

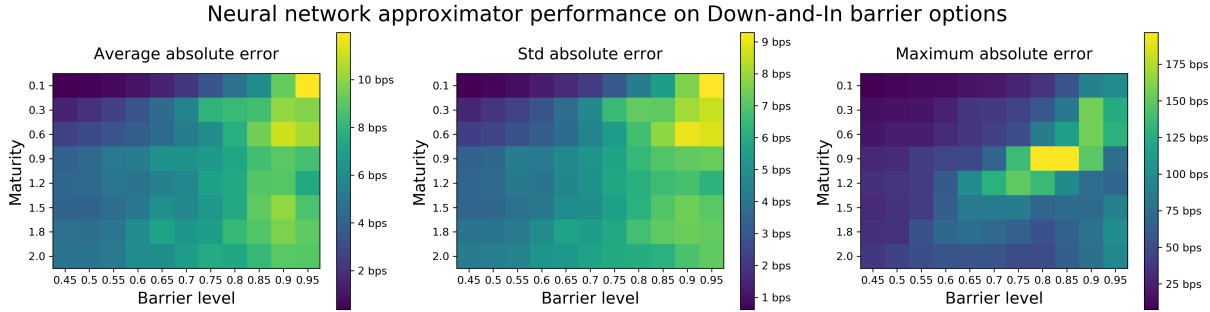
where $\tau_B = \inf_t \{S_t = B\}$. In this setting, we may easily generate a grid for barrier levels and maturities $\Delta^{Barrier} := \{B_i, T_j\}_{i=1,\, j=1}^{n,\, m}$.

- Relevant for Autocalls

Barriers I


Neural network approximator performance on Down-and-Out barrier options
Average absolute error
Std absolute error
Maximum absolute error
Maturity
Barrier level
0.1
0.3
0.6
0.9
1.2
1.5
1.8
2.0
0.45 0.5 0.55 0.6 0.65 0.7 0.75 0.8 0.85 0.9 0.95
14 bps
12 bps
10 bps
8 bps
6 bps
4 bps
2 bps
12 bps
10 bps
8 bps
6 bps
4 bps
2 bps
175 bps
150 bps
125 bps
100 bps
75 bps
50 bps
25 bps

Barrier II


Neural network approximator performance on Down-and-In barrier options
Average absolute error
Std absolute error
Maximum absolute error
Maturity
Barrier level
0.1
0.3
0.6
0.9
1.2
1.5
1.8
2.0
0.45 0.5 0.55 0.6 0.65 0.7 0.75 0.8 0.85 0.9 0.95
2 bps
4 bps
6 bps
8 bps
10 bps
1 bps
2 bps
3 bps
4 bps
5 bps
6 bps
7 bps
8 bps
9 bps
25 bps
50 bps
75 bps
100 bps
125 bps
150 bps
175 bps

Further applications: Model recognition

We see that after learning, calibrating many parameters is fast

Further applications: Model recognition

We see that after learning, calibrating many parameters is fast
$\Rightarrow$ approximate several models at the same time.

Further applications: Model recognition

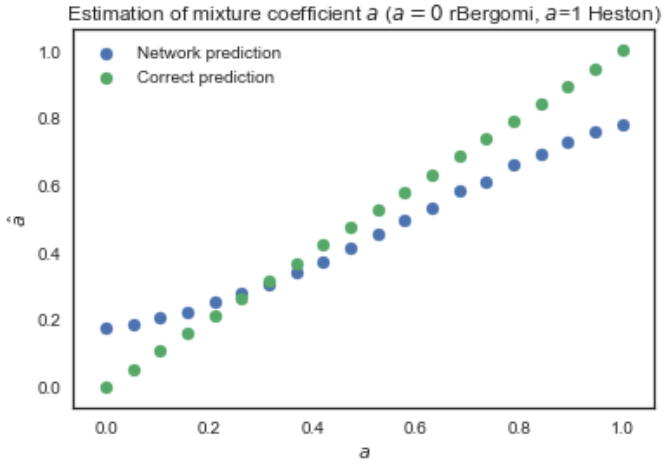
We see that after learning, calibrating many parameters is fast
$\Rightarrow$ approximate several models at the same time.
New learning procedure:

- Train the generator on several models at the same time (here Heston and rBergomi) in Monte Carlo experiments as before: New mixture parameter a

$$a \times \textbf{Heston} + (1 - a) \times \textbf{rBergomi}$$

- Calibrate mixture models $\Rightarrow$ determine the best-fit mixture of model the two models to a given data.
- Controlled experiments: train on both Bergomi and Heston $\Rightarrow$ test on data generated by Heston.

Further applications: Model recognition



Thank you for your attention!