Deep learning rough volatility and deep calibration of the rough Bergomi model

Bryan Delamour, MASEF

October 1, 2021

1 INTRODUCTION

Rough stochastic volatility models have proven to provide a more accurate representation of implied volatility surfaces than simpler models. However, for some of these models such as the rough Bergomi model, introduced by Bayer, Friz and Gatheral [1], in the absence of analytical formula to price European options, computationally costly Monte Carlo simulation is the only way to generate implied volatility surfaces, making fast calibration impossible. In this sense, Bayer, Horvath, Muguruza, Stemper and Tomas in [2] proposed to replace the Monte Carlo pricing by a neural network, reducing the expensive simulations to a simple forward pass in the network.

First, our objective is to reproduce the results in [2]. We will share some insights on the methodology used, and finally we will try to identify new potential areas of interest on the subject.

To generate rough Bergomi paths, we will implement the Hybrid Scheme, introduced by Bennedsen, Lunde, Pakkanen [3], as it is the most effective algorithm to generate fractional Brownian motions in terms of complexity. We will price options using the turbo charging Monte Carlo mixed estimator, McCrickerd, Pakkanen [4], as it proved to be more precise for implied volatility estimation. Implied volatilities will be inverted from option prices using a publicly available implementation of the implied volatility solver by Jäckel [5].

We will fix a strikes/maturities grid as in [2] and generate samples of implied volatility surfaces associated with rough Bergomi parameters. With the data generated, we will train a neural network, implemented using the Pytorch library, to learn the map from rough Bergomi parameters to implied volatility surfaces. After this, we will use a convolutional neural network to learn the inverse map from implied volatility surfaces to rough Bergomi parameters.

The code is written in Python, we will provide all notebooks, html files, and the data used in this project. Also, we would like to make the subject accessible for non familiar readers, therefore, we will try to keep as much as possible the same notations used in the references, in order to make it easier for the reader to verify the sources of our statements.

Principal references:

[2] On deep calibration of (rough) stochastic volatility models, Bayer, Horvath, Muguruza, Stemper and Tomas (2019)

[2.1] Deep calibration of rough stochastic volatility models, Bayer, Stemper (2018)

[2.2] Deep Learning Volatility, Horvath, Muguruza, and Tomas (2019)

[3] Hybrid scheme for Brownian semistationary processes, Bennedsen, Lunde, Pakkanen (2017)

[4] Turbocharging Monte Carlo pricing for the rough Bergomi model, McCrickerd, Pakkanen (2018)

Contents

List of Figures

2 Rough Volatility

2.1 Fractionnal Brownian motion

2.1.1 Definition

The fractional Brownian motion B_H, is a generalization of the Brownian motion. The increments of the fBm are not necessarily independent. It is a continuous-time Gaussian process on [0, T], from zero, with expectation zero for all t in [0, T], its covariance function is defined as :

$$\Gamma(t,s) = \mathrm{Cov}\left(B_H(t), B_H(s)\right) = E\left[B_H(t)B_H(s)\right] = \frac{1}{2}\left(|t|^{2H} + |s|^{2H} - |t-s|^{2H}\right)$$

The autocovariance function, given by

$$\begin{aligned}\gamma(t-s) &= \mathrm{Cov}\left(B_H(t+1) - B_H(t), B_H(s+1) - B_H(s)\right) \\ &= \frac{1}{2}\left(|t-s-1|^{2H} - 2|t-s|^{2H} + |t-s+1|^{2H}\right)\end{aligned}$$

2.1.2 Representation

The truncated Brownian semistationary ($TBSS$) process Y is a common representation of the fractional Brownian motion.

$$Y_t = \sqrt{2\alpha+1}\int_0^t (t-s)^{\alpha}\mathrm{d}W_s$$

where $\alpha = H - \frac{1}{2}$

2.1.3 Hybrid Scheme

We define the hybrid scheme (order κ) to discretize Y_t, for any $t \geq 0$, as

$$Y_t = \int_0^t g(t-s)\sigma_s \, \mathrm{d}W_s, \quad t \geq 0$$

For the fractional Brownian motion we take:

$$\begin{gathered} g(t-s) = (t-s)^{\alpha} \\ \sigma_s = \sqrt{2\alpha+1} \\ L_g = 1 \end{gathered}$$

$$Y_t^n := \check{Y}_t^n + \hat{Y}_t^n$$

where

$$\check{Y}_t^n := \sum_{k=1}^{\min\{\lfloor nt\rfloor,\kappa\}} L_g\left(\tfrac{k}{n}\right)\sigma_{t-\frac{k}{n}}\int_{t-\frac{k}{n}}^{t-\frac{k}{n}+\frac{1}{n}}(t-s)^{\alpha}\mathrm{d}W_s = \sqrt{2\alpha+1}\sum_{k=1}^{\min\{\lfloor nt\rfloor,\kappa\}}\int_{t-\frac{k}{n}}^{t-\frac{k}{n}+\frac{1}{n}}(t-s)^{\alpha}\mathrm{d}W_s$$

$$\hat{Y}_t^n := \sum_{k=\kappa+1}^{\lfloor nt\rfloor} g\left(\tfrac{b_k}{n}\right)\sigma_{t-\frac{k}{n}}\left(W_{t-\frac{k}{n}+\frac{1}{n}} - W_{t-\frac{k}{n}}\right) = \sqrt{2\alpha+1}\sum_{k=\kappa+1}^{\lfloor nt\rfloor}(\tfrac{b_k}{n})^{\alpha}\left(W_{t-\frac{k}{n}+\frac{1}{n}} - W_{t-\frac{k}{n}}\right)$$

Following Remark 3.1 in [3], in the case of the $\mathcal{TBSS}$ process Y, the observations $Y_{\frac{i}{n}}^n, i = 0, 1, \ldots, \lfloor nT\rfloor$, given by the hybrid scheme can be computed via

$$Y^n_{\frac{i}{n}} = \sum_{k=1}^{\min\{i,\kappa\}} L_g\left(\frac{k}{n}\right)\sigma^n_{i-k}W^n_{i-k,k} + \sum_{k=\kappa+1}^{i} g\left(\frac{b^*_k}{n}\right)\sigma^n_{i-k}W^n_{i-k} = \sqrt{2\alpha+1}\sum_{k=1}^{\min\{i,\kappa\}} W^n_{i-k,k} + \sqrt{2\alpha+1}\sum_{k=\kappa+1}^{i}\left(\frac{b^*_k}{n}\right)^\alpha W^n_{i-k}$$

$$\hat{Y}^n_{\frac{i}{n}} = \sum_{k=1}^{N_n}\Gamma_k\Xi_{i-k} = (\Gamma \star \Xi)_i$$

where

$$\Gamma_k := \begin{cases} 0, & k = 1,\dots,\kappa \\ g\left(\frac{b^*_k}{n}\right) = \left(\frac{b^*_k}{n}\right)^\alpha, & k = \kappa+1, \kappa+2, \dots, \lfloor nT \rfloor \end{cases}$$

$$\Xi_k := \sigma^n_k W^n_k = \sqrt{2\alpha+1}W^n_k, \qquad k = 0,\dots,\lfloor nT \rfloor - 1$$

Proposition 2.8 in [3], the asymptotic MSE induced by the discretization, is minimized by the sequence $\mathbf{b}^*$ given by

$$b^*_k = \left(\frac{k^{\alpha+1} - (k-1)^{\alpha+1}}{\alpha+1}\right)^{1/\alpha}, \quad k \geq \kappa+1$$

2.1.4 Hurst parameter

The Hurst index H is a real number in (0, 1), it describes the roughness of the resultant motion. The value of H determines what kind of process the fBm is :

if $H = 1/2$ then the process is in fact a Brownian motion or Wiener process.

if $H > 1/2$ then the increments of the process are positively correlated.

if $H < 1/2$ then the increments of the process are negatively correlated.

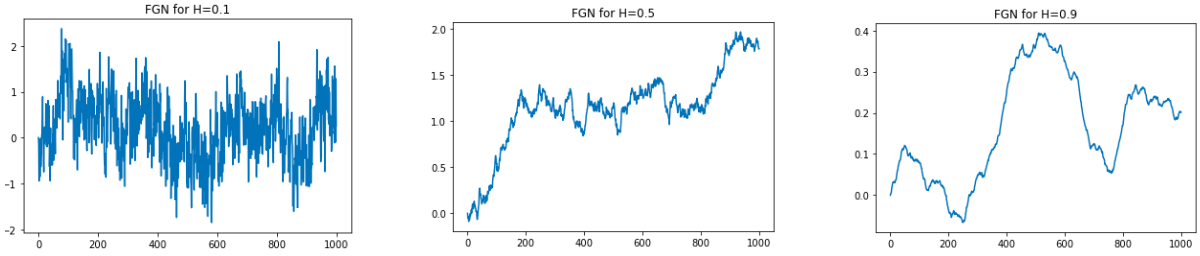

Figure 1: Fractional Gaussian Noises for different Hurst parameters

As we can see in Figure 1, for H = 0.5 we obtain a simple Brownian Motion. However, a small Hurst parameter, induce a much more irregular trajectory. Respectively, with a larger Hurst parameter we obtain a smoother path.

2.1.5 Testing our implementation of the Fractionnal Brownian motion

We will briefly test our implementation with regard to the necessary and sufficient conditions of the fractional Brownian motion, provided in [8]:

a) Normality of the process

b) Increments stationarity

$$B_H(t) - B_H(s) \sim B_H(t-s)$$

c) Self-similarity

$$B_H(at) \sim |a|^H B_H(t)$$

a) Normality of the process

First, we illustrate the distribution of the fractional Brownina motion for different value of t against its theoretical Gaussian density.

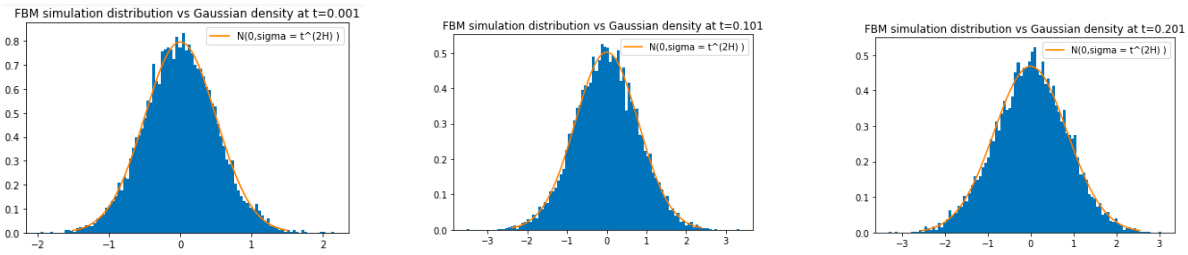

Figure 2: Fractional Brownian motion distribution for different t

We observe that the empirical distribution perfectly fits the corresponding Gaussian density for each t.

We confirm this by performing a Shapiro-Wilk test. We generated 10^4 fractional Brownian motion paths with Hurst parameter $H = 0.1$ from our class, with a discretization of $\delta t = 1/1000$. For each t_i we check that the sample of $(B_H^k(t_i))_{k=1,\dots,10^4}$ is normally distributed.

H_0 = The sample is normally distributed

Results: we found only 37 p-values<0.05. We are therefore confident that we cannot reject the null hypothesis of this test. However, we do not believe that it is sufficient to prove the normality of the process, still it gives us a good indication.

b) Increments stationarity

Mean and variance stationnarity

We illustrate the empirical mean and variance of the fractional Brownian motion increments over a sample of size 10^4 and Hurst parameter $H = 0.1$.

At first glance, the variance and the mean both look stationary, we will use a KPSS test to confirm this.

H_0 = The increment process is stationary

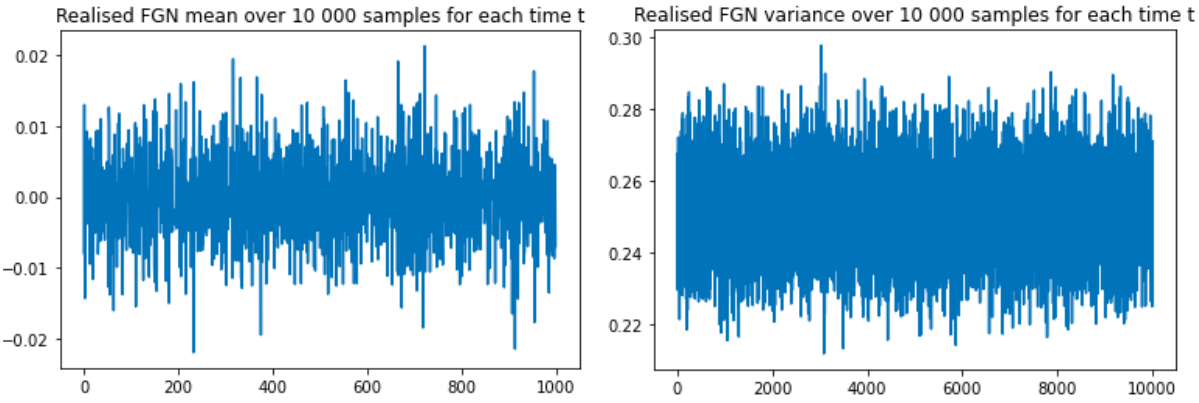

Figure 3: Fractional Gaussian noise empirical mean and variance

Results: we cannot reject the null hypothesis for both the increments variance process and increments mean process for a level of test $\alpha = 5\%$, which confirms our first impression.

c) Self-similarity

Since our process is Gaussian, it seems obvious that this property holds, i.e : $B_H(at) \sim |a|^H B_H(t)$, so we won't do further tests.

Estimating the Hurst parameter

We have for all $s, t \in R$

$$E\left(|B_H(t) - B_H(s)|^2\right) = |t-s|^{2H}$$

From the ergodic theorem, and the continuity of the fBm paths that:

$$\frac{\sum_{i=0}^{N-1}\left(B_H\left(\frac{i+1}{N}\right) - B_H\left(\frac{i}{N}\right)\right)^2}{N^{1-2H}} \underset{N\to+\infty}{\overset{p.s.}{\longrightarrow}} 1$$

Let,

$$V_N := \sum_{i=0}^{N-1}\left(B_H\left(\frac{i+1}{N}\right) - B_H\left(\frac{i}{N}\right)\right)^2$$

Then,

$$\hat{H}_N := \frac{1}{2}\left(1 - \frac{\log\left(V_N\right)}{\log(N)}\right)$$

is a strongly consistant estimator of H

$$\hat{H}_N \overset{p.s.}{\to} H$$

Results:

We have generated 100 paths with a discretization of 10000 and a theoretical $H := 0.1$. We numerically get $H_N = 0.075$, which is an another indicator of a right implementation of our fBm.

For $H := 0.15$ we get $H_N = 0.13$.

For $H := 0.3$ we get $H_N = 0.29$.

For $H := 0.7$ we get $H_N = 0.69$.

3 Rough Bergomi

3.1 Definition

Considering a constant initial forward variance curve $\xi_t^0 = \xi_0$:

$$S_t := S_0 \exp(\underbrace{\int_0^t \sqrt{v_s}\ \mathrm{d}Z_s - \frac{1}{2}\int_0^t v_s\ \mathrm{d}s}_{=:X_t}), \quad t \in [0,T]$$

$$v_t := \xi_t^0 \exp(\eta \underbrace{\sqrt{2\alpha+1}\int_0^t (t-s)^\alpha \mathrm{d}W_s^1}_{=:Y_t} - \frac{\eta^2}{2}t^{2\alpha+1}), \quad t \in [0,T]$$

$$\alpha = H - \frac{1}{2}$$

$$\mathrm{d}Z_t = \rho\ \mathrm{d}W_t^1 + \sqrt{1-\rho^2}\ \mathrm{d}W_t^2, \quad \rho \in [-1,1]$$

Euler Scheme on X_t

$$X_{t_{i+1}} = X_{t_i} - \frac{v_{t_i}}{2}\Delta t + \sqrt{v_{t_i}}\Delta Z_i$$

3.2 Examples

Here we illustrate the asset path and the corresponding variance for different Hurst parameters. We can see that high volatility results in downturns for the asset path.

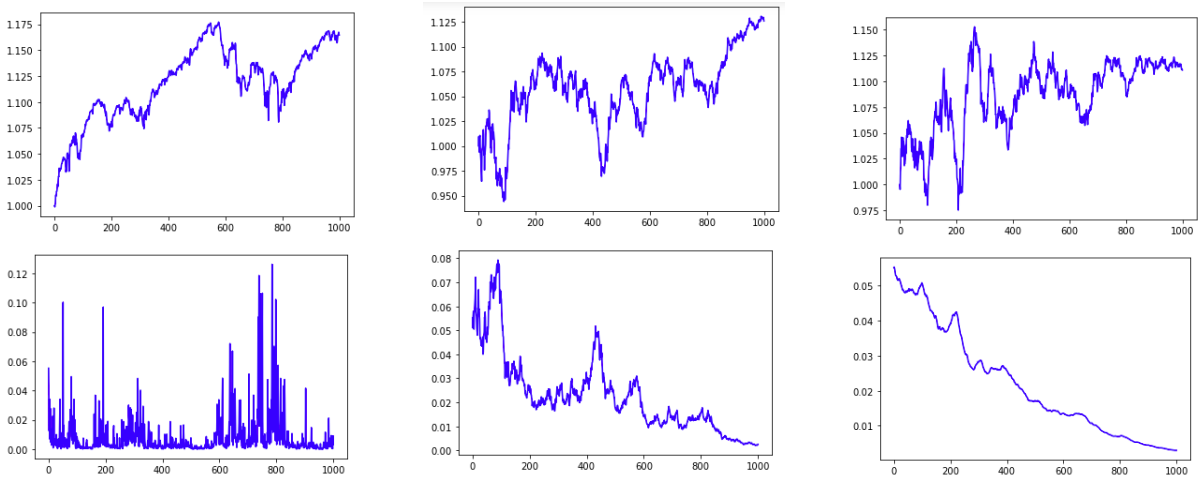

Figure 4: S_t (top) and corresponding v_t (bottom), for $H = 0.1, 0.5, 0.9$

3.3 Testing the rough Bergomi implementation

As we found no proper way to prove that the implementation of our asset path S_t is correct, we will use McCrickerd's library to verify.

In a first step, we will generate Brownian increments using our class and insert them in McCrickerd's library to generate, Y_t the fractional Brownian motion, S_t the asset path, and v_t the corresponding variance. In the meantime, we also generate $\hat{S}_t$, $\hat{v}_t$, $\hat{Y}_t$, using our class with the same Brownian increments. If our implementation is correct, we should see exactly the same paths between McCrickerd's library and our class.

Then we do the reverse, we generate Brownian increments from McCrickerd's library and insert them in our class. We plot each couple, $(\hat{S}_t, S_t)$, $(\hat{Y}_t, Y_t)$, $(\hat{v}_t, v_t)$.

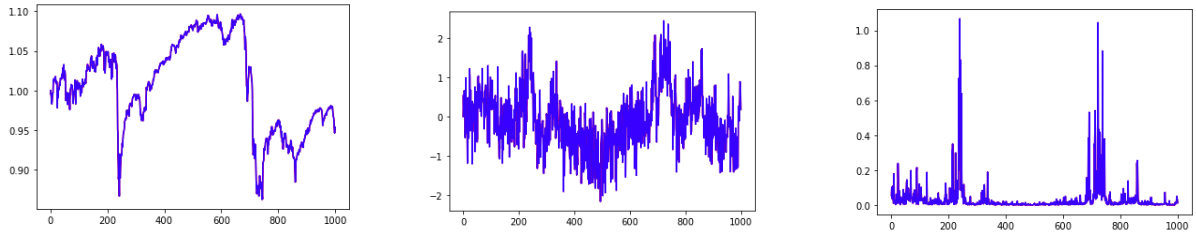

Figure 5: First step: inserting our Brownian increments in McCrickerd's library

Results: the paths are indistinguishable for each couple $(\hat{S}_t, S_t)$, $(\hat{Y}_t, Y_t)$, $(\hat{v}_t, v_t)$.

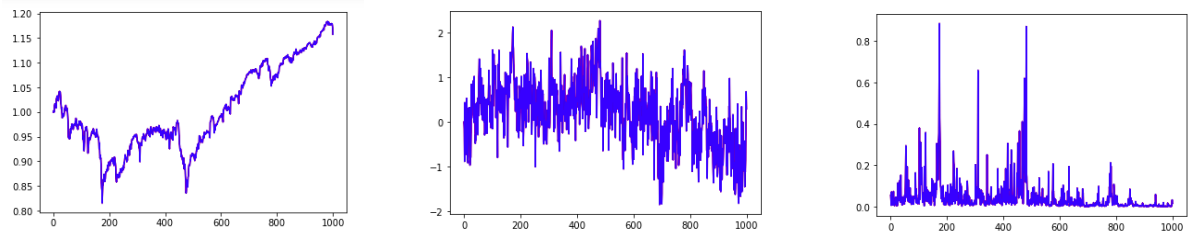

Figure 6: Reverse: inserting McCrickerd's Brownian increments in our class

Results: We find similar results. We believe our implementation is correct.

3.4 Computation time

We will generate from 1, 100,..., 10000 paths and record the time for both our class and McCrickerd's library. We can see that our implementation has the same complexity.

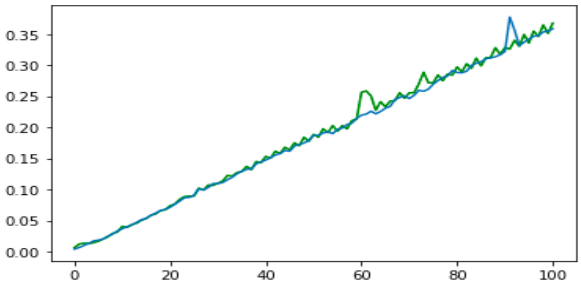

Figure 7: Computation time for our class (green) and McCrickerd's (blue)

4 Turbo charging Monte Carlo

In order to generate precise implied volatility surfaces, we need precise enough prices. Therefore, we will use the mixed estimator presented in [4].

4.1 Base estimator

$$P_n = \frac{1}{n}\sum_{i=1}^{n} max(S_T - K, 0)_i$$

4.2 Mixed estimator

Black Scholes function BS

$$BS(v; s, k) := s\mathcal{N}(d_+) - k\mathcal{N}(d_-), \quad d_\pm := (\log s - \log k)/\sqrt{v} \pm \sqrt{v}/2$$

where $\mathcal{N}(.)$ represents the Gaussian cumulative distribution function

Orthogonal separation

We consider the orthogonal separation of the rBergomi price process S_t into S_t^1 and S_t^2, where

$$S_t^1 := \mathcal{E}\left(\rho \int_0^1 \sqrt{V_u}\ \mathrm{d}W_u^1\right)_t, \quad S_t^2 := \mathcal{E}\left(\sqrt{1-\rho^2}\int_0^1 \sqrt{V_u}\ \mathrm{d}W_u^2\right)_t$$

where, $\mathcal{E}(.)$ is the Wick exponential.

Control variate

The orthogonal separation facilitates our mixed estimator, which we define using

$$X = BS\left(\left(1-\rho^2\right)\int_0^t V_u\ \mathrm{d}u; S_t^1, k\right), \quad Y = BS\left(\rho^2\left(\hat{Q}_n - \int_0^t V_u\ \mathrm{d}u\right); S_t^1, k\right)$$

to consider price estimators $\hat{P}_n(k,t)$ of the following form under the rBergomi model

$$\hat{P}_n(k,t) := \frac{1}{n}\sum_{i=1}^{n}(X_i + \hat{\alpha}_n Y_i) - \hat{\alpha}_n E[Y]$$

from which we derive the implied volatility estimators $\hat{\sigma}_{BS}^n(k,t)$, such that:

$$\hat{\sigma}_{BS}^n(k,t)^2 t = BS^{-1}\left(\hat{P}_n(k,t); 1, k\right)$$

We compute $\hat{\alpha}_n$ and $\hat{Q}_n$ post-simulation from sampled X_i, Y_i and $\left(\int_0^t V_u\ \mathrm{d}u\right)$, using

$$\hat{\alpha}_n := -\frac{\sum_{i=1}^n \left(X_i - \bar{X}_n\right)\left(Y_i - \bar{Y}_n\right)}{\sum_{i=1}^n \left(Y_i - \bar{Y}_n\right)^2}, \quad \hat{Q}_n := \sup\left\{\left(\int_0^t V_u\ \mathrm{d}u\right)_i : i = 1, \ldots, n\right\}$$

Antithetic sampling

We draw a path of W^1 over the interval $[0,t]$, and appeal to the symmetry in distribution of $S_t^{1,\pm}$, defined by

$$S_t^{1,\pm} = \mathcal{E}\left\{\pm\rho\int_0^t \sqrt{V_u^\pm}\mathrm{d}W_u^1\right\}, \quad V_t^\pm = \xi_0(t)\exp\left(-\frac{\eta^2}{2}t^{2\alpha+1}\right)(V_t^\circ)^{\pm 1}$$
$$V_t^\circ = \exp\left(\eta W_t^\alpha\right)$$

4.3 Examples

We have some known values of implied volatilities for three different strikes which were provided in McCrickerd's github [6]. We generate both base and mixed estimator with a sample size of 100.

$Strikes = 0.83, 1, 1.11$

$Maturity = 3M$

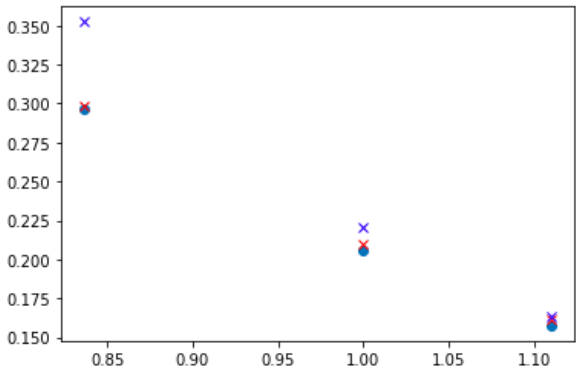

Figure 8: Mixed estimator (red), Base estimator (blue), true implied volatilities (green)

We can see that the mixed estimator is much more precise than the base estimator for the same sample size.

5 Deep learning rough volatility

5.1 Approximating the implied volatility map

Learn the map $F^*(\theta) = \left\{\sigma^{\theta}_{BS}\left(T_i, k_j\right)\right\}_{i=1,j=1}^{8,11}$ via a neural network $\widetilde{F}(\theta) := F(\theta, \hat{w})$ where

$$\begin{aligned} F^* : \Theta &\longrightarrow R^{8\times 11} \\ \theta &\mapsto F^*(\theta) \end{aligned}$$

where the input is a parameter combination θ of the rough Bergomi model and the output is a 8×11 grid on the implied volatility surface $\left\{\sigma^{\theta}_{BS}\left(T_i, k_j\right)\right\}_{i=1,j=1}^{8,11}$

Objective function

$$\hat{w} = \underset{w}{\operatorname{argmin}} \sum_{u=1}^{N_{Train}} \sum_{i=1}^{8} \sum_{j=1}^{11} \left(F\left(\theta_u, w\right)_{ij} - F^*\left(\theta_u\right)_{ij}\right)^2$$

$\hat{w}$ is the optimal network's weights.

5.2 Data generation

Training set of size 34.000 and testing set of size 6.000
Rough Bergomi sample: $(\xi_0, \nu, \rho, H) \in \mathcal{U}[0.01, 0.16] \times \mathcal{U}[0.5, 4.0] \times \mathcal{U}[-0.95, -0.1] \times \mathcal{U}[0.025, 0.5]$
Strikes: $\{0.5, 0.6, 0.7, 0.8, 0.9, 1, 1.1, 1.2, 1.3, 1.4, 1.5\}$
Maturities: $\{0.1, 0.3, 0.6, 0.9, 1.2, 1.5, 1.8, 2.0\}$

We first generate 40000 samples of rough Bergomi parameters, then for each of these samples, and for each point of the strikes/maturities grid (8×11 gridpoints), we generate a mixed estimator using a sample size of 60000 asset paths. This is a very long process, that took approximately 119 hours, however, it has to be done only once.

Examples

Here we plot some surfaces generated and their corresponding rough Bergomi parameters.

First plot (left): $\xi = 0.15$, $\nu = 1.09$, $\rho = -0.50$, $H = 0.37$

Second plot (middle): $\xi = 0.11$, $\nu = 3.26$, $\rho = -0.42$, $H = 0.27$

Third plot (right): $\xi = 0.016$, $\nu = 2.80$, $\rho = -0.20$, $H = 0.26$

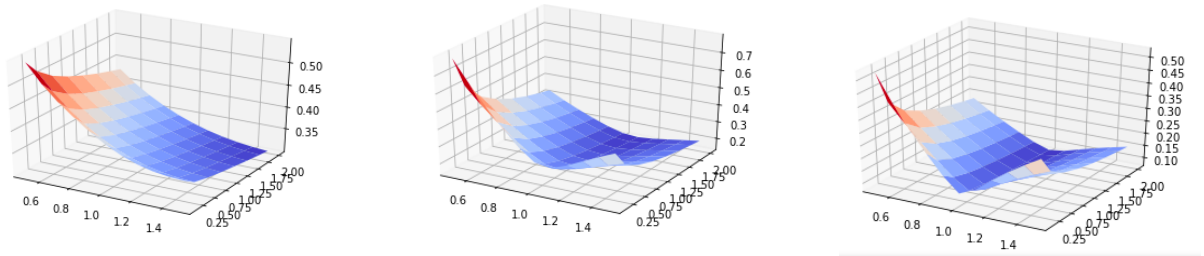

Figure 9: Implied volatility surfaces generated

5.3 Neural Network architecture

We followed the architecture proposed in [2], ie:

1. A fully connected feed forward neural network with 3 hidden layers and 30 nodes on each layers;
2. Input dimension = 4, number of model parameters
3. Output dimension = 11 strikes ×8 maturities
4. The three inner layers have 30 nodes each, which adding the corresponding biases results on a number

$$(4+1)\times 30 + 3\times(1+30)\times 30 + (30+1)\times 88 = 30\times 4 + 5548 = 5668$$

5. We choose the Elu $\sigma_{\text{Elu}} = \alpha\,(e^x - 1)$ activation function for the network, with $\alpha = 1$

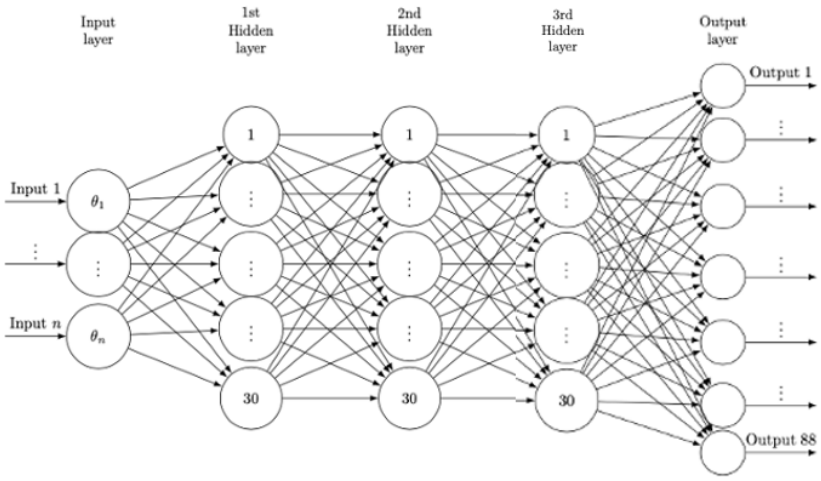

Figure 10: Neural network architecture

5.4 Layer's initialization

We follow the initialization procedure proposed in [2.1].

Suppose $w_{ij}^{(l)}$ denotes the weight of node $i, 1 \le i \le n_l$ in layer $1 \le l \le L$ being multiplied with the output of node $j, 1 \le j \le n_{l-1}$ in layer $l-1$ and n_0 denotes the number of network inputs. The weights and biases are independently drawn as follow:

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

5.5 Normalization of model parameters

As in [2], model parameters are restricted to a given domain i.e. $\theta \in [\theta_{\min}, \theta_{\max}]$. Then, we perform the following normalization transform component wise:

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

5.6 Training procedure

Adam minibatch training scheme

w denotes the set of parameters - weights and biases − of a neural network $F = F(w, x)$ Given parameters $0 \leq \beta_1, \beta_2 < 1, \epsilon, \alpha$, initial iterates $u_0 := 0, v_0 := 0, w_0 \in \Omega$, the Adam scheme has the following iterates:

$$
\begin{aligned}
g_n &:= \nabla^w \sum_{i=1}^{m} \mathcal{L}\left(F\left(w_{n-1}, X_{n,m}^{\text{batch}}\right), F^*\left(X_{n,m}^{\text{batch}}\right)\right) \\
u_{n+1} &:= \beta_1 u_n + (1-\beta_1)\, g_n \\
v_{n+1} &:= \beta_2 v_n + (1-\beta_2)\, g_n^2 \\
w_{n+1} &:= w_n - \alpha \frac{u_{n+1}}{1-\beta_1^{n+1}} \frac{1}{\sqrt{v_n / \left(1-\beta_2^{n+1}\right)} + \epsilon}
\end{aligned}
$$

Preventing early overfitting

As mentioned in [9], a solution to prevent early overfitting is to draw new minibatches uniformly from the training set between each epoch. This allows a better generalization of the network's solution. We will see its effect on the test set loss (green) and the training set loss (red) as both will follow the same decrease until the last epoch.

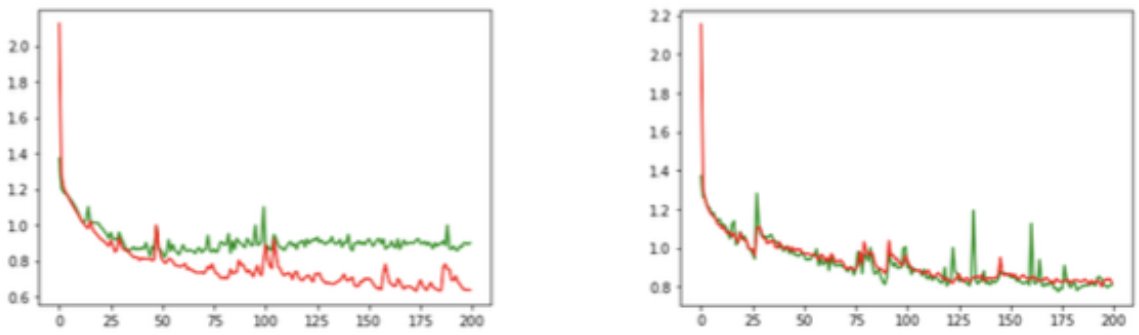

Figure 11: Loss evolution during training: Early overfitting (left), New minibatches at each epoch (right)

Early stopping variant

A variant to the early stopping training used in [2] is to keep training the network until the last epoch as the procedure is not long. However, we won't save the model unless the loss improved.

In fact, we will see that in our case the loss keeps improving until the last epoch.

Rate decay

During the training, the solution provided by the network might come close to a local or global minimum, in this situation jumps in the loss can be seen and the loss won't be able to improve further, Figure 12 is an illustration of this kind of situation. To prevent this situation, as proposed in [10], we use a linear learning rate decay during starting at epoch 0 and decreasing until the last epoch.

The learning rate at epoch k:

$$\epsilon_k = (1 - \alpha)\epsilon_0 + \alpha\epsilon_\tau$$

with $\alpha = \frac{k}{\tau}$, $\tau = 200$.

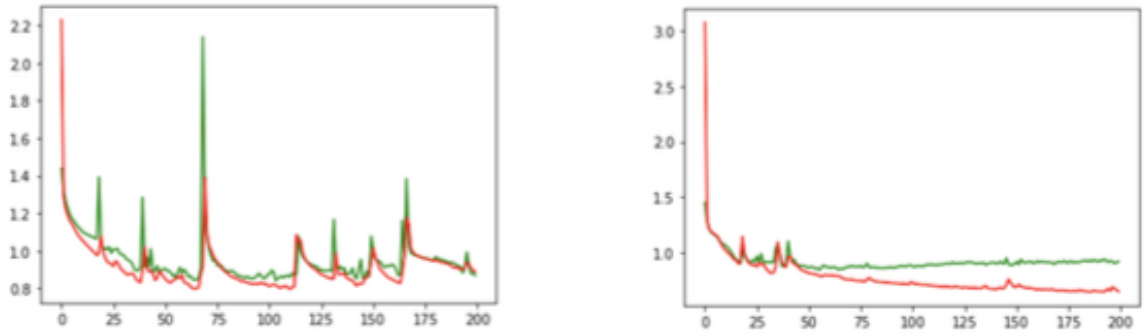

Figure 12: Loss evolution during training: without rate decay (left), with linear rate decay (right)

Training parameters

Batch size = 256

Number of epochs = 200

Starting learning rate = 5×10^{-3}

Ending learning rate = 10^{-5}

Loss function = MSE

5.7 Results

In green, we measured the loss on a test sample at each epoch, this sample was not used for any updates in the network's weights. In red is the evolution of the loss on the training set. The training procedure is fairly fast around 2 minutes.

Final loss on the test sample is 1.33×10^{-5}

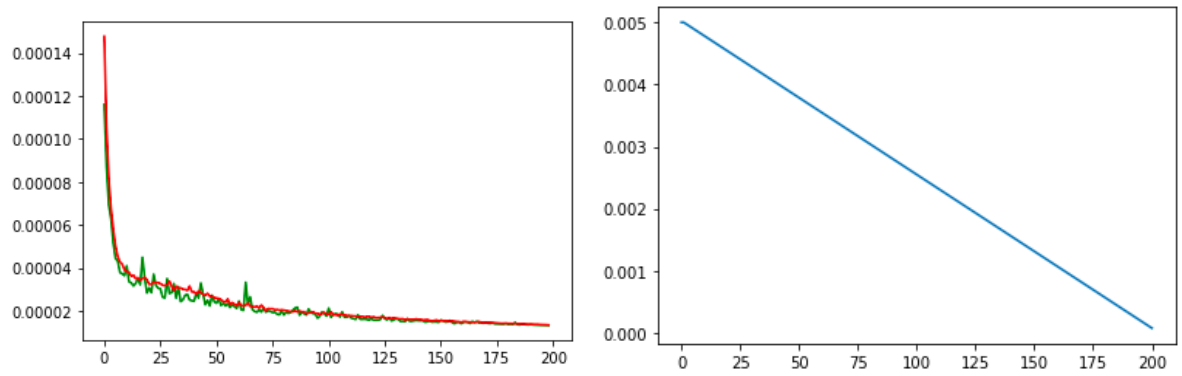

Figure 13: Loss evolution during training (left), learning rate evolution (right)

Illustration of the neural network's output

We will illustrate some true implied volatility surfaces against their neural network approximation, using test samples, ie, samples which have not been used to calibrate the network's weights.

We can see that true surfaces and surfaces generated by the network using the corresponding rough Bergomi parameters are indistinguishable by the eye.

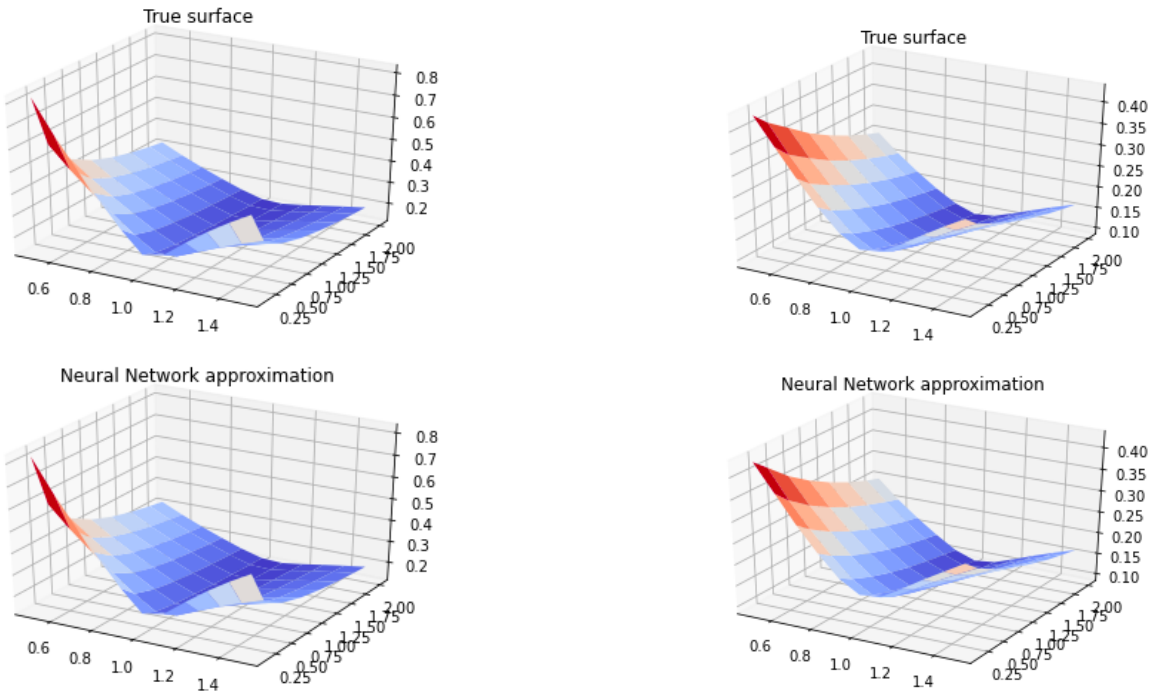

Figure 14: Neural network's output compared to true implied volatility surfaces on test samples

5.8 Accuracy of the IV map as learned by the network

Relative Error

We compare the relative errors of the neural network approximator against the true implied volatility surfaces across all test samples.

Relative error here is computed as:

$$\frac{\left|\sigma^{NN}(T,k) - \sigma(T,k)\right|}{|\sigma(T,k)|}$$

Average relative error across all strikes and maturities $= 0.45\%$

Standard deviation of the relative error all strikes and maturities $= 0.7\%$

Average maximum relative error all strikes and maturities $= 14\%$

Relative error heatmap

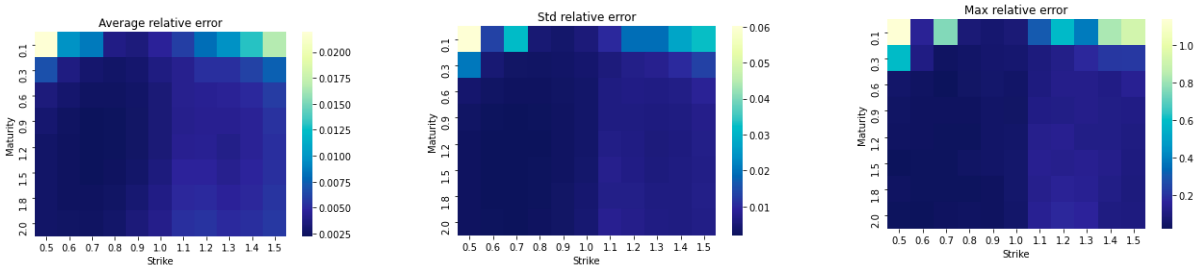

Figure 15: Average-Standard Deviation-Maximum (Left-Middle-Right) obtained with our network and our data

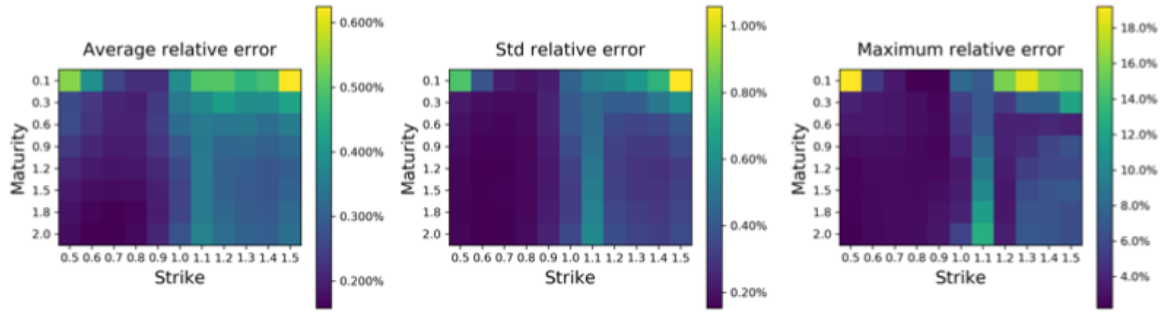

Figure 16: Average-Standard Deviation-Maximum (Left-Middle-Right) in [2]

We can see that the neural network is quite precise, with an average relative error of 0.45%. Also, looking at Figure 15 and Figure 16, we find the same patterns than the authors in [2], with the highest errors located in very short maturities and extremes strikes values. However, it seems that we have higher errors in these areas, ranging around 2%, while they find 0.6% in [2].

5.9 Relative error distribution

Quantile 90% (yellow) $= 0.9\%$

Quantile 95% (red) $= 1.3\%$

Quantile 99% (green) $= 3.4\%$

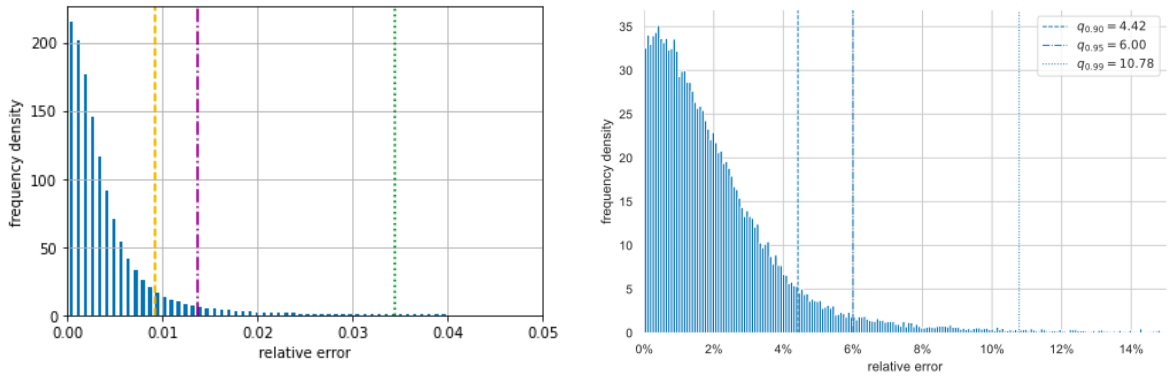

Figure 17: Relative error distribution from our network (left), compared to [2.1] (right)

Figure 17 shows that our relative error distribution is more appealing than in [2.1]. Our 99% quantile being 3.4%, while the 90% quantile in [2.1] is 4.4%.

5.10 Results using samples from [2]

The authors made their data available on Github [2.3], so we trained a new network to see if we could find more closer results.

The final loss on the test sample is 7.54×10^{-7}, which is more than 10 times lower than 1.33×10^{-5}, the loss we found with our own generated dataset.

Also, we can notice that we reproduce the exact same behavior in the relative errors, with extreme values ranging around 0.6%, same for the relative error standard deviations and the maximum relative error.

Average relative error across all strikes and maturities $= 0.2\%$

Standard deviation of the relative error all strikes and maturities $= 0.3\%$

Average maximum relative error all strikes and maturities $= 4\%$

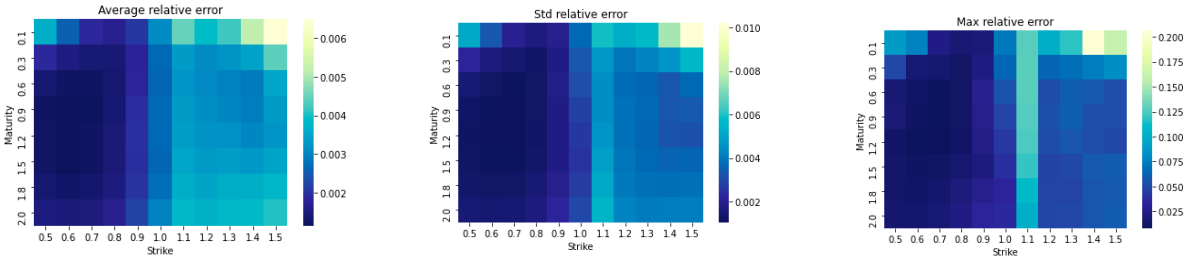

Figure 18: Average-Standard Deviation-Maximum (Left-Middle-Right) from our network using samples from [2]

6 Deep calibration of the rough Bergomi model

6.1 Approximating the inverse map

We calibrate rough Bergomi parameters, i.e., we consider the simple map

$$\Pi^{-1}\left(\Sigma_{\mathrm{BS}}^{\mathrm{rBergomi}}\right) \rightarrow \left(\hat{\xi}, \hat{\nu}, \hat{\rho}, \hat{H}\right)$$

where $\Sigma_{\mathrm{BS}}^{\mathrm{rBergomi}} \in R^{8\times 11}$ is a rBergomi implied volatility surface and $\left(\hat{\xi}, \hat{\nu}, \hat{\rho}, \hat{H}\right)$ the optimal solution to the corresponding calibration problem.

Objective function

$$\hat{w} = \underset{w}{\operatorname{argmin}} \sum_{i=1}^{N_{Train}} \left(\xi_i - \hat{\xi}_i\right)^2 + (\nu_i - \hat{\nu}_i) + (\rho_i - \hat{\rho}_i)^2 + \left(H_i - \hat{H}_i\right)^2$$

$\hat{w}$ is the optimal network's weights.

6.2 Neural network architecture

We follow the architecture proposed in [2], ie:

1. 1 convolutional layer with 16 filters and 3×3 sliding window
2. MaxPooling layer with 2×2 sliding window
3. 50 Neuron Feedforward Layer with Elu activation function
4. Output layer with linear activation function
5. Total number of parameters: 10014

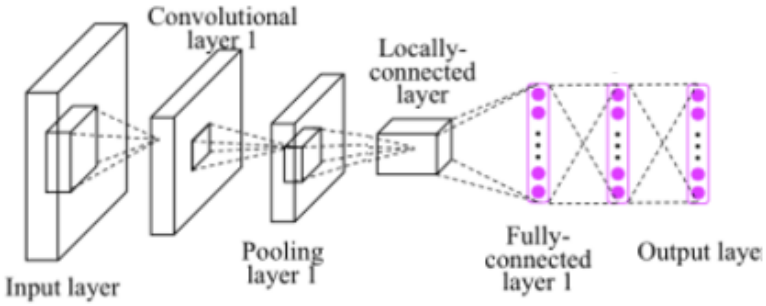

Figure 19: Neural network architecture

6.3 Normalization of implied volatilities

$\bar{iv}_{\text{train}}$ the mean of the training set and the corresponding sample standard deviation s_{train} . For each input $iv^{(i)}$, its standardized version $\hat{iv}^{(i)}$ is given by

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

where the operations are defined componentwise.

Training parameters

Batch size = 128

Number of epochs = 400

Starting learning rate = 5×10^{-3}

Ending learning rate = 5×10^{-5}

Loss function = MSE

6.4 Results

The training procedure is a little longer than our first network around 40 minutes, the final loss in the test sample is 3.82×10^{-5}

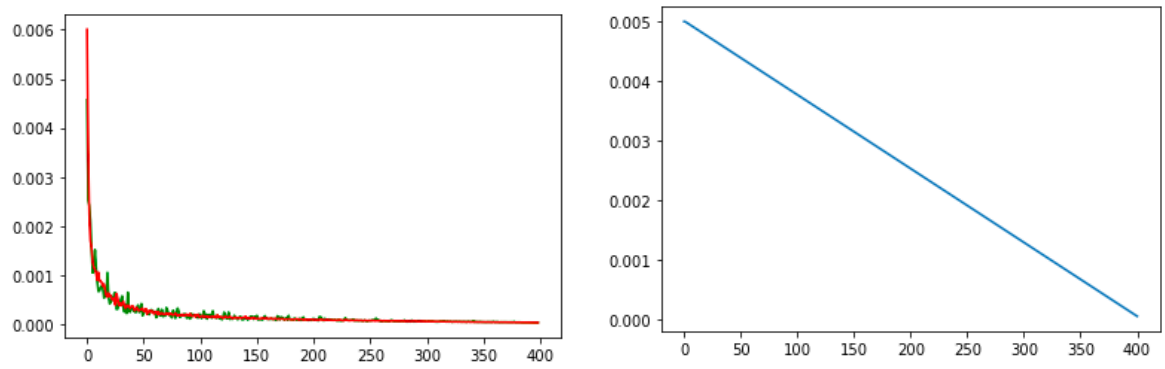

Figure 20: Loss evolution during training (left), learning rate evolution (right)

6.5 Accuracy of the inverse map as learned by the network

Relative Error

Relative error here is computed as:

$$\frac{\left|\theta^{NN} - \theta\right|}{|\theta|}$$

Average relative error across all parameters = 1.17%

Neural Network Average relative error across all parameters in [2] = 1.20%

Levengerg-Marquardt Average relative error across all parameters in [2] = 0.92%

Looking at Figure 21, we find the same patterns than in [2], with the highest errors being for small ξ, ν, H values, and ρ close to 0. This time we found a lower average relative error, however, the difference is not that significant, and we still have a greater error than the Levengerg-Marquardt calibration in [2].

Relative error per parameter

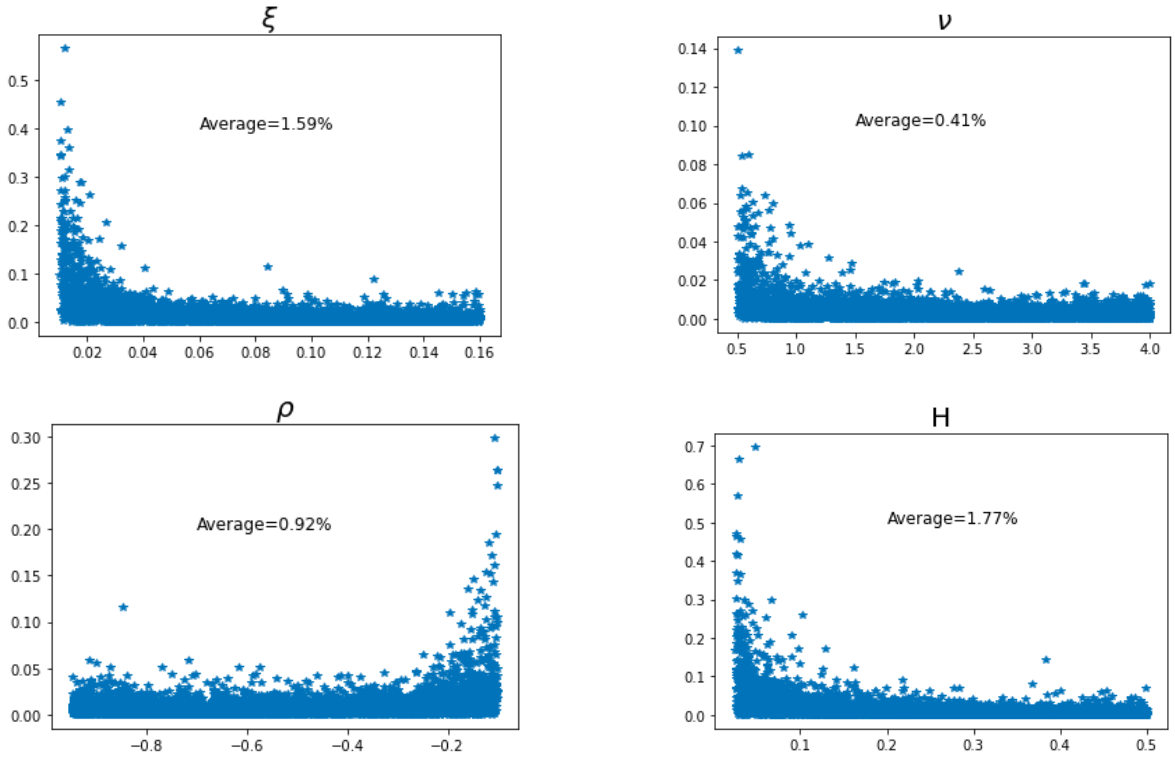

Figure 21: Relative error per parameter from our network and our dataset

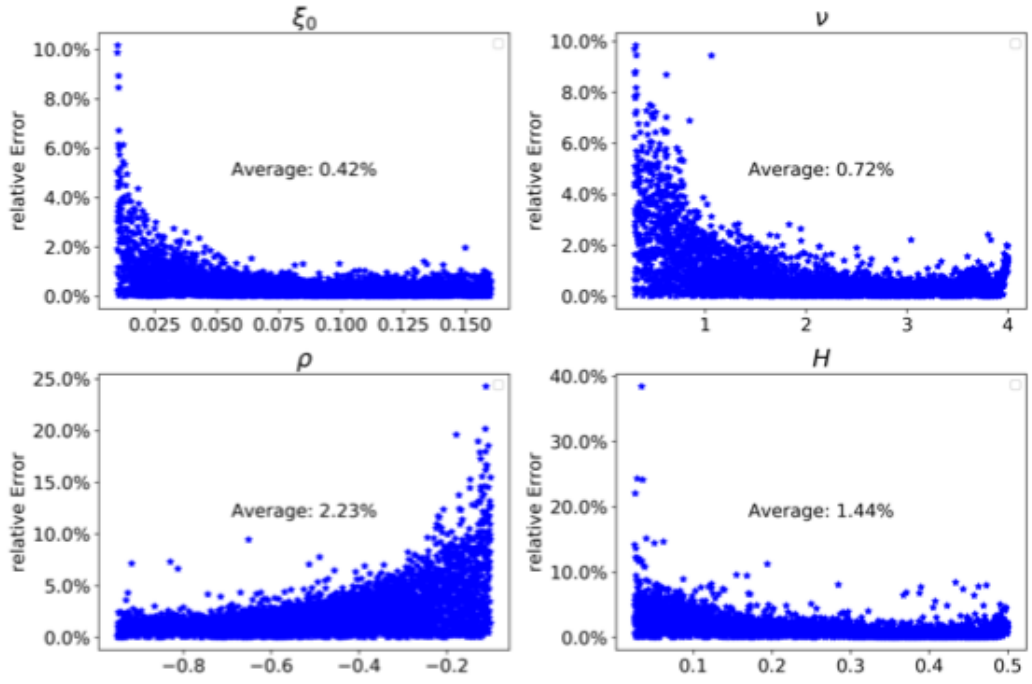

Figure 22: Neural Network relative error per parameter in [2]

6.6 Results using samples from [2]

The final loss on the test sample is 9.63×10^{-6}

Average relative error across all parameters $= 0.79\%$

This time using the authors data, we significantly decreased the average relative error across all parameters, and we even found a relative error lower than the Levengerg-Marquardt calibration in [2] (0.92%).

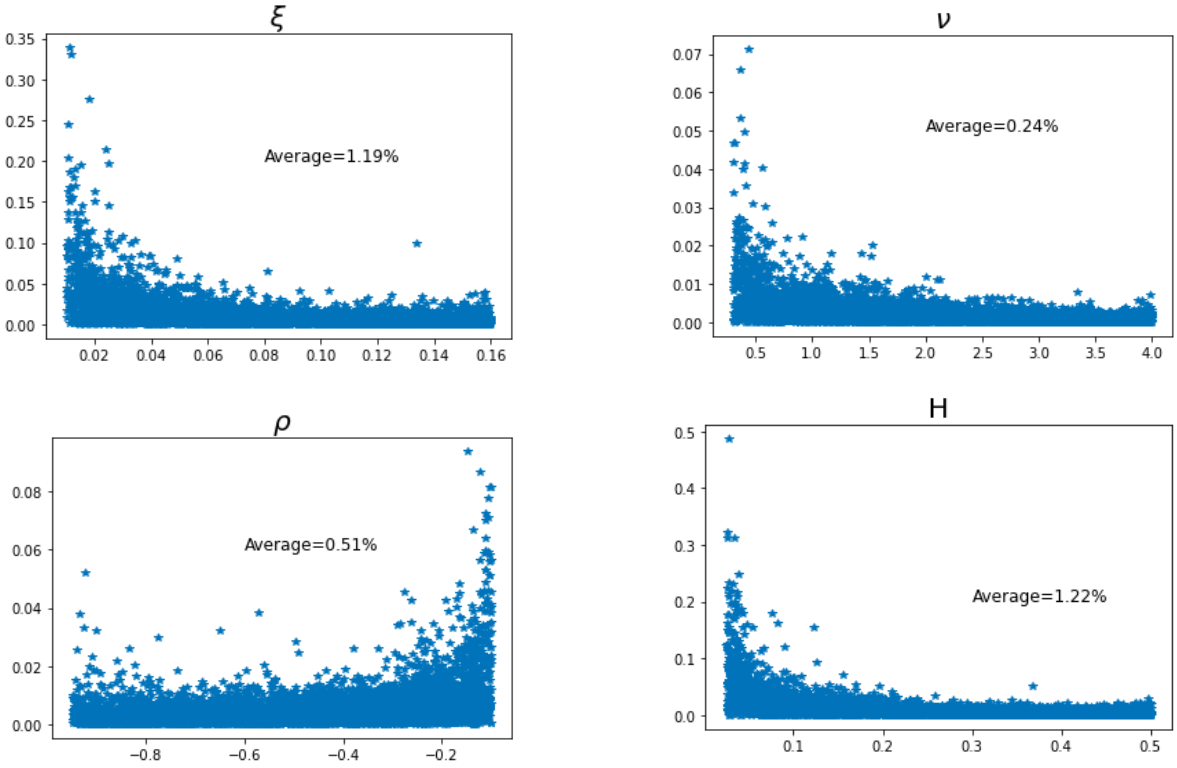

Figure 23: Relative error per parameter from our network using samples from [2]

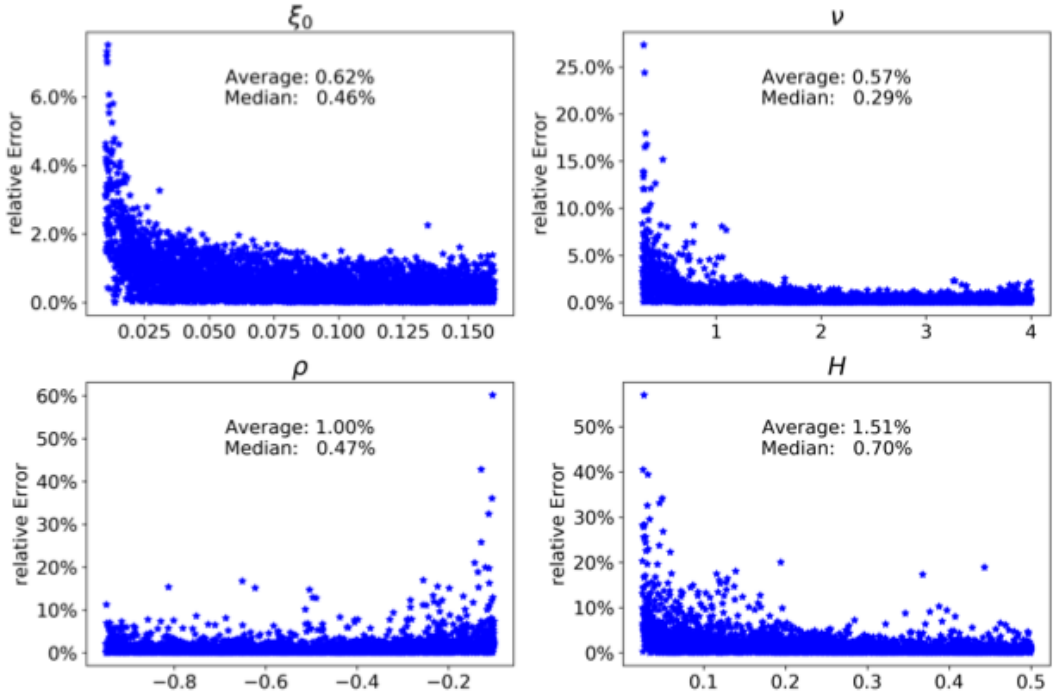

Figure 24: Levengerg-Marquardt relative error per parameter in [2]

7 CONCLUSION

In this project, we implemented the Hybrid Scheme in order to generate fast fractional Brownian motion paths, so we could work with the rough Bergomi model. Then, we looked for a precise price estimator and we found the turbo charging Monte Carlo mixed estimator to be convenient, so we could generate implied volatility surfaces that look realistic. After this, we could generate Input-Output couples of rough Bergomi parameters and their respective implied volatility surfaces, in order to train our first network.

We found, impressively low relative errors in test samples, highlighting the strength of neural networks' approximation capabilities. Still our errors were not as low as the ones we found using the authors [2] dataset. This highlights the fact that the data generation procedure is of crucial importance, as machine learning algorithms are very sensitive to the data used to train them.

Still, with our convolutional neural network used to calibrate the rough Bergomi parameters, we found better relative errors on average with both the authors' data and our own dataset. We believe that this is a consequence of our training procedure that we tried to maximize when we investigated hyperparameters. Finally, using the data from [2], we outperformed the Levengerg-Marquard with our network.

We believe that analyzing the rough Bergomi parameters impact on the implied volatility surface could be useful to understand better how to calibrate, however, we did not manage enough time to work on that point. Rather than sampling the rough Bergomi parameters, it could be interesting to reproduce the results with deterministic values for the parameters.

8 BIBLIOGRAPHY

[1] Pricing under rough volatility, Bayer, Friz and Gatheral (2016)

[2] On deep calibration of (rough) stochastic volatility models, Bayer, Horvath, Muguruza, Stemper and Tomas (2019)

[2.1] Deep calibration of rough stochastic volatility models, Bayer, Stemper (2018)

[2.2] Deep Learning Volatility, Horvath, Muguruza, and Tomas (2019)

[2.3] https://github.com/amuguruza/NN-StochVol-Calibrations/tree/master/Data, Muguruza

[3] Hybrid scheme for Brownian semistationary processes, Bennedsen, Lunde, Pakkanen (2017)

[4] Turbocharging Monte Carlo pricing for the rough Bergomi model, McCrickerd, Pakkanen (2018)

[5] Let's Be Rational, Jäckel (2015)

[6] https://github.com/ryanmccrickerd/rough$_b ergomi/, McCrickerd$

[8] Estimation de la qualité des méthodes de synthèse du mouvement brownien fractionnaire, Jennane, Harba, Jacquet (1996)

[9] Deep Hedging Under Rough Volatility, Pytoch project, Université Paris Dauphine-PSL, Aloui, Farcinade, Larbi, Delamour (2021)

[10] Deep Learning, Goodfellow, Bengio, Courville