Ca' Foscari
University
of Venice

Single Cycle Degree programme

in Economia e
Finanza
"Finance"

Final Thesis

# Fractional Brownian motion and option pricing under rough volatility

**Supervisor**
Ch. Prof. Marco Corazza

**Graduand**
Luca Marana
Matriculation Number 888315

**Academic Year**
2021 / 2022

# Contents

**Introduction** **5**

**1 Fractional Brownian motion and simulation methods** **7**

1.1 Some properties of the process . . . . . 7

1.2 Simulation methods . . . . . 8

1.2.1 Cholesky decomposition . . . . . 8

1.2.2 Hosking method . . . . . 10

1.2.3 Circulant embedding method (CEM) . . . . . 12

1.2.4 Hybrid scheme for truncated Brownian semi-stationary process . . . . . 14

1.3 Exact simulation methods comparison . . . . . 15

**2 Rough volatility and RSFV model** **19**

2.1 Roughness . . . . . 19

2.2 RFSV model . . . . . 20

2.3 Empirical results . . . . . 20

**3 Rough Bergomi model** **23**

3.1 Variance swap . . . . . 23

3.2 Bergomi model . . . . . 24

3.3 Rough Bergomi model . . . . . 24

**4 Simulation and calibration of rough Bergomi model** **27**

4.1 Simulation . . . . . 27

4.2 Calibration . . . . . 29

4.3 Empirical results . . . . . 31

**Conclusion** **33**

**References** **35**

**Appendix A** **39**

**Appendix B** **47**

# Introduction

Modeling volatility in financial markets has always been problematic given its non-deterministic nature. In the derivative world, where volatility plays a crucial role, different pricing models have been developed since the 20th century. The most famous and widely used is the Black-Scholes formula, proposed in their seminal work in the seventies [1].

Since then, several other models have been developed to overcome its shortcomings that question its robustness such as the assumption of constant volatility (implied). Indeed, volatility surface is well-known to presents so-called smiles due to the option moneyness and ATM skews. If volatility were a constant as described by Black and Scholes, volatility surface would be flat instead of varying across moneyness and maturities.

Hence, the need of models capable of capture consistently this feature in order to correctly price option contracts.

Among stochastic volatility models, notable mentions are Heston model [2], SABR [3], Hull and White [4] and Bergomi model [5]. They differ from each other, essentially for the specific dynamics describing the volatility process. These models are considered time-homogenous, meaning that parameters are independent from price and time. This property assumes that the overall shape of the volatility surface does almost not change in the equity market, at least in first approximation, albeit level and orientation can.

Despite these models were a notable improvement, they do not capture persistency and volatility clusters that instead are observable in time-series data. The limitation of the stochasticity term modeled as a Brownian motion process, was eliminated by Comte and Renault [6] with the introduction of fractional stochastic volatility model, where a fractional Brownian motion with long dependency drives the model. A revisitation of that has been proposed by Gatheral, Jaisson and Rosenbaum [7], who defined volatility as rough and showed that volatility has short term dependence. Bayer, Friz and Gatheral also proposed an extension of the Bergomi model for option pricing purpose under rough volatility, called the rough Bergomi model.

The work has the following structure. In the first session we describe fractional Brownian motion processes with particular focus on different traces-simulation approaches. We then, describe the so-called roughness of the process driving the volatility of many underlying, according to the work of Gatheral, Jaisson and Rosenbaum. Follows the derivation of the rough Bergomi model for option pricing. Finally, in the last session we proceed with the simulation of the model and we work on its calibration with real market data.

# 1 Fractional Brownian motion and simulation methods

A fractional Brownian motion is a generalization of the well-known Brownian motion process. It was first formally defined in [8] in 1968 by the following stochastic representation:

$$W_t^H := \frac{1}{\Gamma(H+\frac{1}{2})}\left(\int_{-\infty}^{0}[(t-s)^{H-\frac{1}{2}} - (-s)^{H-\frac{1}{2}}]\,dW_s + \int_0^t (t-s)^{H-\frac{1}{2}}\,dW_s\right) \tag{1}$$

where $\Gamma$ is the gamma function[1] and $H \in (0,1)$ denotes the Hurst parameter. $W$ is a standard Brownian motion process $W(t,w)$ where $t \in (-\infty, +-\infty)$ and $w$ is a set of all values of a random function (path) that belongs to the sample space $\Omega = C_0(\mathbb{R}_+, \mathbb{R})$.

In contrast to the ordinary Brownian motion process, its covariance structure allows for asymptotic dependence:

$$\mathbb{E}\left[W_t^H W_s^H\right] = \frac{1}{2}(t^{2H} + s^{2H} - \mid t-s \mid^{2H}). \tag{2}$$

Note that, the ordinary Brownian Motion process is easily retrievable with $H = \frac{1}{2}$, hence independence between the increments. With $H > \frac{1}{2}$ positive correlation is addressed, memory and persistence are generated. Negative correlation, thus intermittency and anti-persistency, is present instead with $H < \frac{1}{2}$.

## 1.1 Some properties of the process

A fractional Brownian motion process $W^H = W^H_{t\geq\infty}$ with $0 < H < 1$ is uniquely characterized by the following properties:

1. $W_0^H = 0$ and $\mathbb{E}\left[W_t^H\right] = 0$,
2. $W^H$ has homogeneous increments, i.e. $W^H_{t+s} - W^H_s$ has the same law of $W^H_t$ for $s, t \geq 0$,
3. $W^H$ is a Gaussian process and $\mathbb{E}\left[\left(W_t^H\right)^2\right] = t^{2H}$,
4. $W^H$ has continuous trajectories.

For the purposes of our study, it is also important that in the case of $H \neq \frac{1}{2}$, the process is neither a semi-martingale nor a Markov process. The dependence of new values from the past causes the stochastic calculus developed by Itô to be undefined. This poses a potential limitation for process simulation, especially in terms of time since there is no analytical solution.

[1] The gamma function is defined as: $\Gamma(z) = \int_0^\infty t^{z-1}e^{-t}dt$.

## 1.2 Simulation methods

In the following section, some methods are presented to simulate fractional Gaussian noise, from which, it is then possible to obtain the fractional Brownian motion process.

Process simulation is required to evaluate any model based on this type of stochastic process. Some of these algorithms, in fact, will then be employed in the last section of the document.

First, we introduce four mathematical algorithms and then implement them in Python, the code part is attached in Appendix A. The section presents the algorithms in increasing order of efficiency. In fact, for the implementation of these algorithms in the financial sector, their efficiency is fundamental: both from the point of view of the time required and from that of the memory used.

First, we introduce simulation methods considered exact: more time-consuming but generating more accurate path coordinates. For exact methods the common starting point is the covariance matrix $\Gamma$ which specifically defines the process. It follows an approximate method also used in the last section of the document.

### 1.2.1 Cholesky decomposition

Since it is a stationary process, the simulation of a path of fractional Brownian motion in a discrete time-frame $W^H = W_t^H, t \in [0, T]$ can be done via cumulative sums of simulated increments, i.e. $W_0^H = 0$ and $W_k^H = \sum_{i=0}^{k} X_i, k \geq 1$ , where $X_i$ is defined as a random variable $X_n = W_n^H - W_{n-1}^H$. Due to the properties of the process itself, it is sufficient to generate the values $W_0^H, W_1^H, ..., W_{N-1}^H$ and scale them by a factor $(\frac{T}{N})^H$ to retrieve a sequence of coordinates $W_0^H(\frac{T}{N}), W_1^H(\frac{T}{N}), ..., W_N^H(\frac{T}{N})$ of simulation values.

We introduce the covariance of a sequence of standard Gaussian random variables $X_0, X_1, ...$ belonging to a fractional Gaussian noise:

$$\gamma(k) = \mathbb{E}\left[X_1, X_{k+1}\right] = \frac{1}{2}((k+1)^{2H} + |k-1|^{2H} - 2k^{2H}), k \geq 0. \tag{3}$$

If we write the process in term of centered Gaussian vector[2] $\mathbf{X} = (X_0, X_1, ..., X_{N-1})^T$, it can represented as

$$\mathbf{X} = L\mathbf{Z} \tag{4}$$

where Z is a standard Gaussian vector $\mathbf{Z} = (Z_1, Z_2, ..., Z_N)^T$ and $L$ is the square root of the

[2]A centered normal random vector is a real random vector $\mathbf{X} = (X_0, X_1, ..., X_{N-1})^T$ with deterministic $N \times l$ matrix $\mathbf{A}$ such that $\mathbf{AZ}$, where $\mathbf{Z}$ is a standard normal random vector with $l$ components, has the same distribution as $\mathbf{X}$.

covariance matrix $\Gamma$ defined as

$$\Gamma = \Gamma_N = \begin{bmatrix} 1 & \gamma(1) & \gamma(2) & \dots & \gamma(N-2) & \gamma(N-1) \\ \gamma(1) & 1 & \gamma(1) & \dots & \gamma(N-3) & \gamma(N-2) \\ \gamma(2) & \gamma(1) & 1 & \dots & \gamma(N-4) & \gamma(N-3) \\ \gamma(3) & \gamma(2) & \gamma(1) & \dots & \gamma(N-5) & \gamma(N-4) \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \gamma(N-2) & \gamma(N-3) & \gamma(N-4) & \dots & 1 & \gamma(1) \\ \gamma(N-1) & \gamma(N-2) & \gamma(N-3) & \dots & \gamma(1) & 1 \end{bmatrix} \tag{5}$$

such that

$$LL^T = \Gamma. \tag{6}$$

Given that, applying the Cholesky decomposition gives the matrix $L$ of type:

$$L = \begin{bmatrix} l_{(0,0)} & 0 & 0 & 0 & \dots & 0 \\ l_{(1,0)} & l_{(1,1)} & 0 & 0 & \dots & 0 \\ l_{(2,0)} & l_{(2,1)} & l_{(2,2)} & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ l_{(N-1,0)} & l_{(N-1,1)} & l_{(N-1,2)} & l_{(N-1,3)} & \dots & l_{(N-1,N-1)} \end{bmatrix}. \tag{7}$$

We define the Cholesky decomposition in its element-wise form as

$$\gamma(i-j) = \sum_{k=0}^{j} l_{ik}, l_{jk}, 0 \leq j \leq i \leq N-1. \tag{8}$$

For $i = j = 0$ the equation reduces to $\gamma(0) = l_{00}^2 = 1$ and for $i = 1$ we obtain $\gamma(1) = l_{10}l_{00}$ and $l_{10}^2 + l_{11}^2 = 1$. From that, $l_{10} = \gamma(1)$ and $l_{11} = \sqrt{1-\gamma^2(1)}$. The other non-zero elements are defined as following, for $i \geq 2$:

$$\begin{gathered} l_{i0} = \gamma(i) \\ l_{ij} = \frac{1}{l_{jj}} \left( \gamma(i-j) - \sum_{k=0}^{j-1} l_{ik} l_{jk} \right), 0 < j \leq i < N-1 \\ l_{ii}^2 = \gamma(0) - \sum_{k=0}^{i-1} l_{ik}^2. \end{gathered} \tag{9}$$

Once $L$ is found, according to (4), fractional Gaussian noise is easily recovered by matrix multiplication. The fractional Brownian motion process is recovered from the cumulative sum of the increments.

The Cholesky method is relatively easy to implement but has a complexity of $\mathcal{O}(N^3)$ and requires more memory than other methods, making it rather uneconomical in terms of speed. However, the matrix $L$ can be stored once it has been calculated, reducing it to the order of $\mathcal{O}(N^2)$ for repetitive simulations. Another advantage of this algorithm is that traces can be generated on-the-fly, which means that the sample size doesn't have to be defined in advance; moreover, from a practical point of view, this guarantees some results if the algorithm stops at a random moment.

**Algorithm 1.1.** The algorithm generates a single path given granularity $N$, time $T$ and Hurst parameter $H$.

1. Compute the array generating the matrix $\Gamma$ as $\Gamma(k) = \frac{1}{2}\left((k+1)^{2H} + |k-1|^{2H} - 2(k^{2H})\right)$.
2. Implement the Cholesky composition as in (9).
3. Compute the fractional Gaussian noise through matrix multiplication between the obtained matrix $L$ and a vector of generated standard normal random variables $Z_t \sim (0, 1)$.
4. Take the cumulative sum and scale by $(\frac{T}{N})^H$ to retrieve the fractional Brownian motion process.

### 1.2.2 Hosking method

The Hosking method is an algorithm capable of simulating a general stationary Gaussian process and therefore can also be used to generate a fractional Brownian motion process. It is also known as the Durbin and Levinson method and, like the Cholesky decomposition, simulates the process as a cumulative sum of increments. The algorithm generates $X_{n+1}$ given $X_n, X_{n-1}, ..., X_0$ recursively, therefore it does not exploit any specific property of the process itself. It also has the same advantage of the Cholesky decomposition when referring to sample size.

We define an n-dimensional vector $c_n := (\gamma(1), \gamma(2), ..., \gamma(n))^T$ from the covariance matrix of the form (5) and the matrix $F_n$:

$$F_n = \begin{bmatrix} 0 & 0 & 0 & \dots & 0 & 0 & 1 \\ 0 & 0 & 0 & \dots & 0 & 1 & 0 \\ 0 & 0 & 0 & \dots & 1 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 1 & 0 & \dots & 0 & 0 & 0 \\ 1 & 0 & 0 & \dots & 0 & 0 & 0 \end{bmatrix}.$$

The matrix is capable of flipping any pre-multiplied vector of dimension $(n \times 1)$ or post-multiplied row vector of dimension $(1 \times n)$.

Then, the covariance matrix $\Gamma_{n+1}$ can be represented by block-matrix multiplication in terms of $\Gamma_n$, by:

$$\Gamma_{n+1} = \begin{bmatrix} 1 & c_n^T \\ c_n & \Gamma_n \end{bmatrix} = \begin{bmatrix} \Gamma_n & F_n c_n \\ c_n^T F_n & 1 \end{bmatrix}. \tag{10}$$

The conditional distribution of $(X_{n+1}|X_n, ..., X_0)$ is a Gaussian distribution with the following

parameters, see [9]:

$$\mu_n = \mathbb{E}(X_{n+1}|X_n, ..., X_0) = \mathbf{c}_n^T \Gamma_n^{-1} \begin{bmatrix} X_n \\ \vdots \\ X_0 \end{bmatrix}$$

$$\sigma_n^2 = Var(X_{n+1}|X_n, ..., X_0) = 1 - \mathbf{c}_n^T \Gamma_n^{-1} \mathbf{c}_n.$$

Starting from $X_0$, these parameters can be calculated recursively. The original method proposed by Hosking sees the computation of the inverse of the $\Gamma$ matrix at each step, however it would be computationally expensive. A slightly different, but more efficient, version has been proposed by Dieker [10], to which we refer for the proof.

We define $d_n := \Gamma_n^{-1} c_n$ for convenience. From (10), we can then rewrite the inverse of $\Gamma_{n+1}$ as:

$$\Gamma_{n+1}^{-1} = \frac{1}{\sigma_n^2} \begin{bmatrix} \sigma_n^2 \Gamma_n^{-1} + F_n \mathbf{d}_n \mathbf{d}_n^T F_n & -F_n \mathbf{d}_n \\ -\mathbf{d}_n^T F_n & 1 \end{bmatrix}$$

where $\sigma_{n+1}^2$ and $\mathbf{d}_{n+1}$ satisfy the recursions

$$\sigma_{n+1}^2 = \sigma_n^2 - \frac{(\gamma(n+2) - \tau_n)^2}{\sigma_n^2}$$

and

$$\mathbf{d}_{n+1} = \begin{bmatrix} \mathbf{d}_n - \phi_n F_n \mathbf{d}_n \\ \phi_n \end{bmatrix}$$

with

$$\tau_n := \mathbf{d}_n^T F_n \mathbf{c}_n = \mathbf{c}_n^T F_n \mathbf{d}_n$$

$$\phi_n = \frac{\gamma(n+2) - \tau_n}{\sigma_n^2}.$$

Note that, the algorithm needs the covariance of order $(n+2)$ to calculate the coordinates of the process at step $n$, thus $\Gamma$ should have dimension $(N + 2 \times N + 2)$.

Starting from $X_0 \sim \mathcal{N}(0, 1)$ we can easily simulate the process $X_0, X_1, X_2, ...$ recursively. The advantage of this method over Cholesky algorithm is that it has less complexity $(\mathcal{O}(N^2))$, resulting in a faster computation. However, it has the disadvantage that it does not improve the calculation speed in the case of simulation of multiple processes, since the algorithm must be executed separately each time.

The two methods presented are very similar, the triangular matrix $L$ is also implicitly calculated in the Hosking algorithm.

**Algorithm 1.2.** The algorithm generates a single path given granularity $N$, time $T$ and Hurst parameter $H$.

1. Compute the array generating the matrix $\Gamma$ as $c_n = \frac{1}{2}\left((n+1)^{2H} + |n-1|^{2H} - 2(n^{2H})\right), n \in (1, N)$.

2. Implement the Hosking algorithm setting for the initial point $X_0$, $\mu_0 = \gamma(1)X_0$, $\sigma_0^2 = 1 - \gamma^2(1)$, $\tau_0 = \gamma^2(1)$ and $\mathbf{d}_0 = (\gamma(1))$. The computation of the fractional Gaussian noise is done recursively.

3. Take the cumulative sum and scale by $(\frac{T}{N})^H$ to retrieve the fractional Brownian motion process.

### 1.2.3 Circulant embedding method (CEM)

Originally proposed by Davies and Harte in [11], this algorithm was later generalized in [12]. Analogously to the previous described methods, the circulant embedding algorithm tries to find a decomposition of the covariance matrix $\Gamma$ for some squared matrix $S$, such that

$$\Gamma = SS^T \tag{11}$$

where $S$ is not the same matrix $L$ of the Cholesky decomposition.

This method differs from the others, in applying the fast Fourier transformation (FFT) algorithm[3] to speed up the calculation. Indeed, FFT results faster than matrix multiplication, reducing the complexity to $\mathcal{O}(N \log N)$. Note that to exploit this advantage, the sample size $N$ must be a power of 2.

To be applied, the algorithm needs also a larger matrix. For this reason, we define the circulant matrix[4] $C = circ(c_0, c_1, c_2, ..., c_{M-1})$, obtained embedding the covariance matrix $\Gamma$, of size $M = 2N = 2^{g+1}$ for some $g \in \mathbb{N}$.

[3]The fast Fourier transformation, introduced by Cooley and Tukey in [13], is a more efficient algorithm to compute the discrete Fourier transformation, i.e. a matrix multiplication $\mathbf{X} = \mathbf{Mx}$ with $\mathbf{M}_{kn} = e^{-i2\pi kn/N}$. It reduces the computation complexity from $\mathcal{O}(N^2)$ of the original, to $\mathcal{O}(N \log N)$ of the fast version. The formula is expressed in its forward version by:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i2\pi kn/N}$$

where $i = \sqrt{-1}$, and in its inverse by:

$$x_n = \frac{1}{N} \sum_{n=0}^{N-1} X_k e^{i2\pi kn/N.}$$

To reduce complexity, the fast version exploits the symmetries obtained by dividing the calculation problem into smaller parts.

[4]A circulant matrix is particular matrix constructed shifting the first row vector by $i-1$ places to the right and padding the removed elements on the left side. The matrix can be fully described, thus, by the vector:

$$c_{jk} = c_{j-k}$$

where $0 \leq j, k \leq n-1$. Note that, for symmetry, it can also be displayed in column form.

Precisely, we define the circulating matrix $C$:

$$C := \begin{bmatrix} \gamma(0) & \gamma(1) & \dots & \gamma(N-1) & \gamma(N) & \gamma(N-1) & \dots & \gamma(2) & \gamma(1) \\ \gamma(1) & \gamma(0) & \dots & \gamma(N-2) & \gamma(N-1) & \gamma(N) & \dots & \gamma(3) & \gamma(2) \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \gamma(N-1) & \gamma(N-2) & \dots & \gamma(0) & \gamma(1) & \gamma(2) & \dots & \gamma(N-1) & \gamma(N) \\ \gamma(0) & \gamma(N-1) & \dots & \gamma(1) & \gamma(0) & \gamma(1) & \dots & \gamma(N-2) & \gamma(N-1) \\ \gamma(N-1) & \gamma(N) & \dots & \gamma(2) & \gamma(1) & \gamma(0) & \dots & \gamma(N-3) & \gamma(N-2) \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \gamma(1) & \gamma(2) & \dots & \gamma(N) & \gamma(N-1) & \gamma(N-2) & \dots & \gamma(1) & \gamma(0) \end{bmatrix}$$

where the matrix $\Gamma$ is recognizable in the upper-left corner.

In the case of fractional Brownian motion entries, it is ensured that there exists a matrix $S$ that multiplied by a vector $\mathbf{Z} = (Z_0, Z_1, ..., Z_{M-1})^T$ of the standard Gaussian random variables, results in a vector $\mathbf{X} = (X_0, X_1, ..., X_{M-1})^T$ of fractional Gaussian noise coordinates. See [12] for the proof and further details.

**Algorithm 1.3.** Although several algorithms based on the circulating embedding method have been developed, we implement the one proposed in [9] to which we refer for the explanation of the mathematical elements. The algorithm generates a single path given granularity $N$, time $T$ and Hurst parameter $H$.

1. Set $N = 2^g, g \in \mathbb{N}$ so thus $M = 2^{g+1}$, in order to speed up the computation time.

2. Compute the array generating the matrix $C$ defined as

$$c_0 = 1, c_k = \begin{cases} \gamma(k), k = 1, 2, ..., N-1 \\ \gamma(M-k), k = N, N+1, ..., M-1 \end{cases}, M = 2(N-1). \tag{12}$$

3. Apply the inverse FFT of the vector of standard normal random variable $\mathbf{Z}_{0,1,2,...,M}$, to obtain $\frac{1}{\sqrt{M}} Q^H \mathbf{Z}$ since:

$$\frac{1}{\sqrt{M}} Q^H \mathbf{x} = \left[ \frac{1}{M} \sum_{j=0}^{M-1} x_j exp\left(2\pi i \frac{jk}{M}\right) \right]_{k=0}^{M-1}, x = (x_0, x_1, ..., x_{M-1})^T \in \mathbb{C}.$$

4. Matrix-multiply the results by $\Lambda^{\frac{1}{2}}$, i.e. multiply it element-wise by the vector of eigenvalues $(\lambda_0^{\frac{1}{2}}, \lambda_1^{\frac{1}{2}}, ..., \lambda_{M-1}^{\frac{1}{2}})^T$ obtained in the step before.

5. Apply the FFT algorithm another time to the results. Note that, apply the FFT to $\Lambda^{\frac{1}{2}} \frac{1}{\sqrt{M}} Q^H \mathbf{Z}$ is equivalent to pre-multiply it by $\sqrt{M} Q$, since:

$$\sqrt{M} Q \mathbf{x} = \left[ \sum_{j=0}^{M-1} x_j exp\left(-2\pi i \frac{jk}{M}\right) \right]_{k=0}^{M-1}.$$

6. The fractional Gaussian noise coordinates are retrieved by $MQ\Lambda^{\frac{1}{2}}\frac{1}{\sqrt{M}}Q^H\mathbf{Z} = S\mathbf{Z}$.

7. Take the cumulative sum and scale by $(\frac{T}{N})^H$ to retrieve the fractional Brownian motion process.

### 1.2.4 Hybrid scheme for truncated Brownian semi-stationary process

There are many simulation techniques that do not rely on exact computation of fractional Brownian motion coordinates, but instead have the advantage of speed, which in some cases allows for more robust simulations. One of these is the Hybrid scheme proposed in [14]. This approximated method is useful in the simulation of the rough Bergomi model in the last section of the document.

It can be applied to the simulation of paths of a truncated Brownian motion process of type:

$$X_t = \int_0^t g(t-s)\sigma_s dW_s, t \in \mathbb{R}$$

expressed in integral form where $\sigma_s$ is a stationary predictable process representing the volatility of the process $X_t$, and $g$ is a deterministic weight function in $\mathbb{R}_+$. For completeness, we refer to [14] for the assumptions regarding the function $g$ that should be made. For the purposes of this work it is sufficient to know that in the case of the rough Bergomi model these are satisfied.

We use $g(x) = x^{H-\frac{1}{2}}$ and constant $\sigma = \sqrt{2H}$. Therefore, the discretization of the process can be represented by

$$X_t = \sum_{k=1}^{\infty} \sqrt{2H} \int_{t-\frac{k}{N}}^{t-\frac{k-1}{N}} (t-s)^{H-\frac{1}{2}} dW_s$$

where for large $k > 1$, in first order of approximation we have

$$(t-s)^{H-\frac{1}{2}} \approx \left(\frac{b_k^*}{N}\right)^{H-\frac{1}{2}}, t-s \in \left[\frac{k-1}{N}, \frac{k}{N}\right], b_k^* \in [k-1, k]$$

with

$$b_k^* = \left(\frac{k^{H+\frac{1}{2}} - (k-1)^{H+\frac{1}{2}}}{H+\frac{1}{2}}\right)^{\frac{1}{H-\frac{1}{2}}}$$

see [14]. We summarize as follows

$$X_t \approx \sqrt{2H}\left(\int_{t-\frac{1}{N}}^{t} (t-s)^{H-\frac{1}{2}} dW_s + \sum_{k=2}^{\infty}\left(\frac{b_k^*}{N}\right)^{H-\frac{1}{2}} \int_{t-\frac{k}{N}}^{t-\frac{k-1}{N}} dW_s\right). \tag{13}$$

The simulated process can be written as

$$\begin{aligned} X_0 &= 0 \\ X_1 &= \sqrt{2H} W_{1,1} \\ X_i &= \sqrt{2H}\left[W_{i-1,1} + \sum_{k=2}^{i}\left(\frac{b_k^*}{N}\right)^{H-\frac{1}{2}} W_{i-k,2}\right], i \in (2, T) \end{aligned} \tag{14}$$

where $W_{i,(1,2)}$ are two random i.i.d. vectors of a generated bivariate normal distribution with $\mu = [0,0]^T$ and covariance structure

$$\Sigma = \begin{bmatrix} \frac{1}{N} & \frac{1}{(H+\frac{1}{2})N^{(H+\frac{1}{2})}} \\ \frac{1}{(H+\frac{1}{2})N^{(H+\frac{1}{2})}} & \frac{1}{(2H)N^{2H}} \end{bmatrix}. \tag{15}$$

**Algorithm 1.4.** The algorithm generates a single path given granularity $N$, time $T$ and Hurst parameter $H$.

1. Construct the covariance matrix $\Sigma$ as indicated in (15).
2. Use it to generate a bivariate normal variable $W_t = (W_{t,1}, W_{t,2})^T$ with zero mean.
3. Implement the algorithm according to (14). Note that $\left(\frac{b_k^*}{N}\right)^{H-\frac{1}{2}} W_{i-k,2}$ can be computed as convolution[5].
4. Take the cumulative sum of the first $T * N$ components and scale by $(\frac{T}{N})^H$ to retrieve the fractional Brownian motion process.

## 1.3 Exact simulation methods comparison

The Cholesky, Hosking and CEM methods are discussed in this section. A comparison based solely on the running time of the algorithms is made. We compare only exact methods, as the comparison with approximate methods should also require a penalty on the errors due to the approximation, see the last section for such a comparison.

First, we visibly compare coordinates simulated by each algorithm.

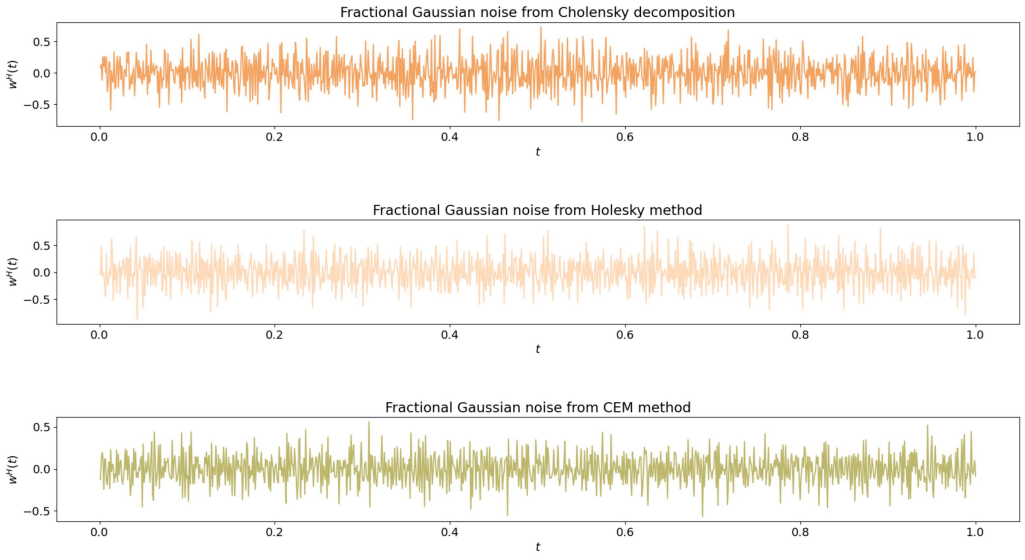



Figure 1: Fractional Gaussian noise traces generated by different algorithms[6].

[5]Discrete convolution is defined as

$$(a * v)_n = \sum_{m=-\infty}^{\infty} a_m v_{m-n}$$

where $a, v$ are complex-valued functions.

[6]All the figures and tables shown in the document are the result of a personal elaboration of data.

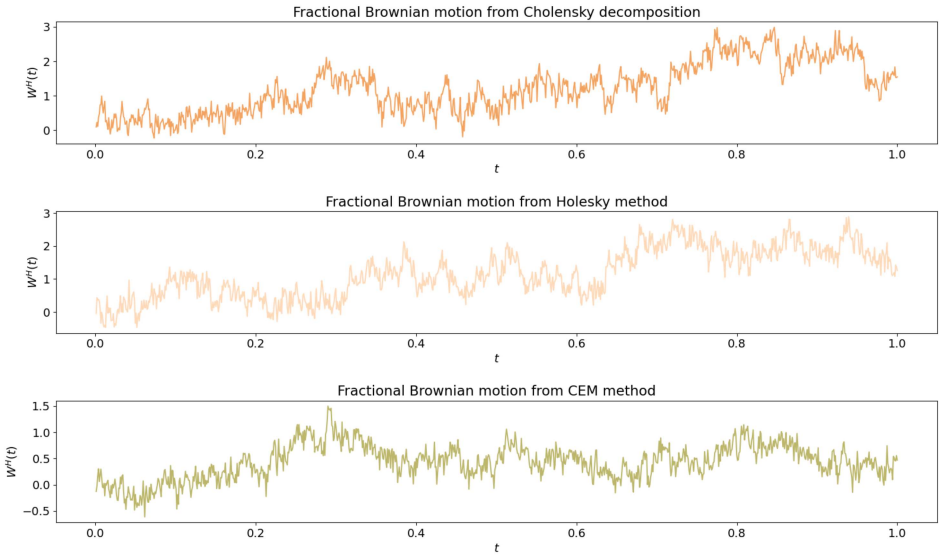



Figure 2: Fractional Brownian motion traces generated by different algorithms.

Both results refer to fractional Gaussian noise and fractional Brownian motion with $H = 0.2$, $T = 1$ and $N = 1024$.

The resulting traces are very similar to each other in terms of structure. The Hurst parameter smaller than $\frac{1}{2}$ gives the processes a short-term negative correlation, more appreciable in the figure below where 8 different processes are simulated with the CEM algorithm by varying the Hurst parameter.

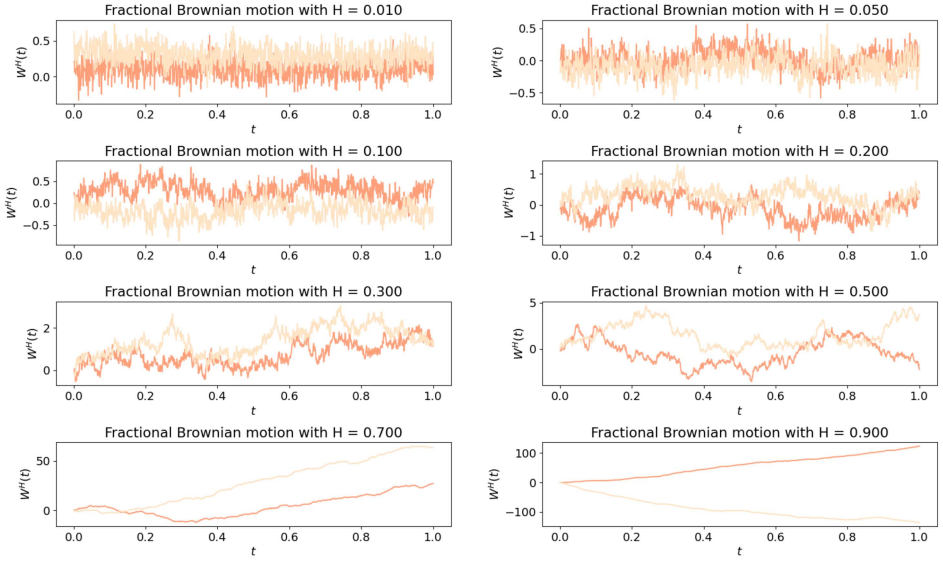



Figure 3: Fractional Gaussian noise paths generated with CEM algorithm by varying $H$.

The 3 methods are evaluated in terms of execution speed on normal hardware, in the following table[7]. For this comparison we simulate only a path with $N = 1024$ steps and $T = 1$.

| Cholesky decomposition | Hosking | CEM |
|---|---|---|
| 1.429664s | 0.022176s | 0.000969s |

Table 1: Time in seconds for one trace simulation with different algorithms.

Easy to see how the efficiency of the algorithms increases from the Cholesky decomposition to the CEM method, with a significantly shorter simulation time.

[7]The results are averaged from a sample of 10 simulations and the generation of the covariance matrix is excluded from the calculation time.

# 2 Rough volatility and RSFV model

Fractional Brownian motion processes are becoming increasingly popular in mathematical finance due to their application to a relatively new class of stochastic volatility models. They were first applied to option pricing by Comte and Renault in [6], and in recent years Gatheral, Jaisson and Rosenbaum [7] have demonstrated how the volatility process of many types of underlyings, from stocks to commodities, is driven by this type of process and not by a standard Brownian motion as in the well-known model of Black and Scholes [1].

Rough volatility is a term referring to a volatility process driven by fractional Brownian motion with Hurst parameter $< \frac{1}{2}$. According to recent studies, this framework seems to provide a more accurate description and estimation of the implied volatility surface.

In this session, we analyze why volatility is defined as rough and present a relatively simple model, namely the RFSV model where log-volatility behaves as a fractional Brownian motion. This model provides the basis for the derivation of the rough Bergomi model covered in the next section.

## 2.1 Roughness

In [7], the roughness of the volatility process is assessed defining the increments of the log-volatility in a discrete-time framework at every $q_{th}$ sample moment of a standard Gaussian random variable.

To detect any micro-structure noise in high frequency data as in this case, we have to search for irregularities that disappear in low frequencies. Specifically, we indicate:

$$m(q, \Delta) = \frac{1}{N} \sum_{k=1}^{N} |\log(\sigma_{k\Delta}) - \log(\sigma_{(k-1)\Delta})|^q \tag{16}$$

where $\Delta$ represents the mesh.

Following [15], we have that $m(q, \Delta)$ is assumed to be a stationary process, converging to a constant $b$ depending on $q$ and with the following asymptotic approximation as $\Delta \to 0$:

$$m(q, \Delta) N^{q s_q} \simeq b_q, s_q > 0, b_q > 0 \tag{17}$$

where $s_q$ is the roughness parameter. Note that, from (16) and also assuming that a law of large numbers can be applied, $m(q, \Delta)$ can be seen as an estimate of

$$\mathbb{E}\left[|\log(\sigma_\Delta) - \log(\sigma_0)|^q\right]. \tag{18}$$

Taking the minimal mesh at one business day for technical reasons, in [7] authors show how, for different $q$-moments, log-volatility has the scaling property in expectation:

$$\mathbb{E}\left[|\log(\sigma_\Delta) - \log(\sigma_0)|^q\right] = m(q, \Delta) \sim b_q \Delta^{\zeta_q} \tag{19}$$

where $\zeta_q = q s_q > 0$ represents the slope of the line associated with $q$. Particularly, it is shown that, $\zeta_q \sim Hq$ with $H \in (0, \frac{1}{2})$, as Hurst parameter of a fractional Brownian motion process, for the log-volatility of a broad class of indices and commodities as underlying.

## 2.2 RFSV model

Always in [7], it is proved that empirical distributions of log-volatility increments are reasonably approximated by a Gaussian distribution, confirming the choice of fractional Brownian motion as well-approximating process. As in other models for stochastic volatility, the volatility process is assumed to be itself to be asymptotically stationary and nowhere differentiable. The authors propose a simple volatility model using the same approach as in [6]:

$$\log(\sigma_{t+\Delta}) - \log(\sigma_t) = \nu(W^H_{t+\Delta} - W^H_t). \tag{20}$$

It is a mean-reverting volatility process in continuous time as in the Hull and White framework [4]. However, differently from that, the Wiener process is substituted by a fractional Brownian motion. And differently from Comte and Renault in [6], in [7] the case with short memory is addressed, i.e. $H \in (0, \frac{1}{2})$ instead of $(\frac{1}{2}, 1)$.

## 2.3 Empirical results

We replicate the results by implementing our code in Python using the same database of high-frequency realized variance. The log-volatility data are processed for each different $q$-moment $\in (0.5, 1, 1.5, 2, 2.5, 3)$ and $\Delta \in [1, 150]$. The graphical results for the S&P500 are shown in the figure below and are quite satisfactory confirming the approximation (19).

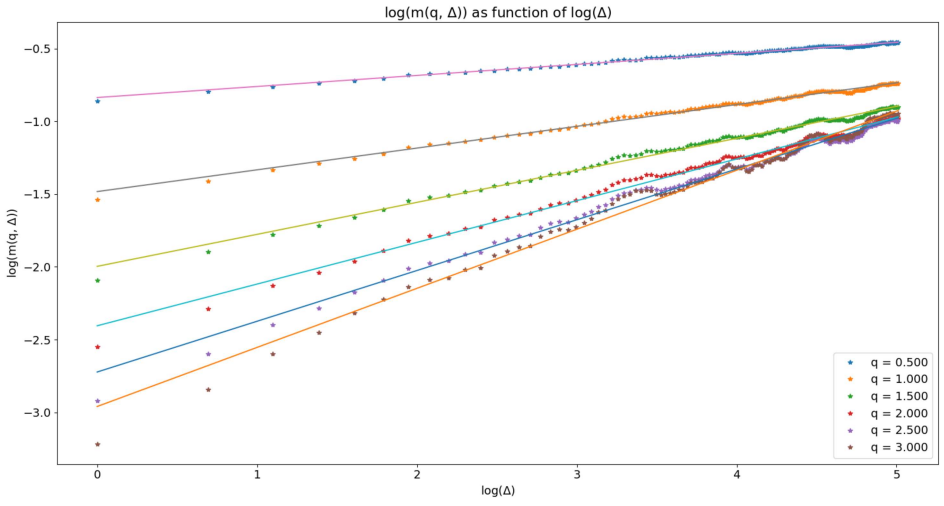



Figure 4: $\log m(q, \Delta)$ on $\log(\Delta)$ for different $q$-moments on S&P500 data.

The point-estimations of the slope of a linear approximation given by $\zeta_q$ for several indexes, are summarized in the following tables.

| S&P 500 | |
|---|---|
| $q_{th}$ | $\zeta_q$ |
| 0.5 | 0.075864 |
| 1 | 0.14944 |
| 1.5 | 0.219884 |
| 2 | 0.28648 |
| 2.5 | 0.348625 |
| 3 | 0.405837 |

| FTSE 100 | |
|---|---|
| $q_{th}$ | $\zeta_q$ |
| 0.5 | 0.073943 |
| 1 | 0.142838 |
| 1.5 | 0.205546 |
| 2 | 0.261186 |
| 2.5 | 0.309289 |
| 3 | 0.349881 |

| DAX 40 | |
|---|---|
| $q_{th}$ | $\zeta_q$ |
| 0.5 | 0.088254 |
| 1 | 0.168386 |
| 1.5 | 0.239910 |
| 2 | 0.301861 |
| 2.5 | 0.352789 |
| 3 | 0.391032 |

| NIKKEI 225 | |
|---|---|
| $q_{th}$ | $\zeta_q$ |
| 0.5 | 0.065158 |
| 1 | 0.126375 |
| 1.5 | 0.182881 |
| 2 | 0.233723 |
| 2.5 | 0.277944 |
| 3 | 0.314843 |

Table 2: Estimated $\zeta_q$ for different $q$-moments on different indexes.

We estimate $H$ exploiting the approximation $\zeta_q \sim Hq$ and thus regressing $q = (0.5, 1, 1.5, 2, 2.5, 3)$ on $\zeta_q$.

| | S&P 500 | FTSE 100 | DAX 40 | NIKKEI 225 |
|---|---|---|---|---|
| $H$ | 0.132229 | 0.110553 | 0.121660 | 0.100226 |
| $SE$ | 0.003006 | .005216 | 0.007624 | 0.004462 |
| *Upper* .95 | 0.138121 | 0.120777 | 0.136603 | 0.108972 |
| *Bottom* .95 | 0.126337 | 0.100328 | 0.106716 | 0.091481 |

Table 3: Estimated parameters $H$ on different indexes.

The point-estimations are not the same as those obtained in the original paper, however this can easily be due to longer time series with recent data not included in the authors' estimates. This suggests that $H$ may fluctuate over time as they point out.

In any case, the results confirm the roughness of the volatility process, being the Hurst parameter on the order of 0.1. This disqualifies the authors' choice in [6] of long memory in the volatility process, as well as its classic approximation by standard Brownian motion.

# 3 Rough Bergomi model

The rough Bergomi model was introduced for the first time in [16] with the motivation of pricing on realized variance and its underlying. It can be seen as another instance of the RFSV model and a generalization of the original Bergomi model.

It has the advantage of calculate the price of vanilla and exotic options more accurately than other models. Its main disadvantage is the lack of an analytical solution which causes the need of statistical techniques such as Monte-Carlo solutions.

In this section, the original version of the model is presented, followed by the rough version. However, describing these models first requires an introduction to variance swap contracts.

## 3.1 Variance swap

A variance swap contract is a derivative contract that pays out some notional amount times the realized variance of the logarithmic total returns of the underlying up to maturity $T$, less the strike called the variance swap rate:

$$\text{payoff} = (\sigma^2_{realized(0,T)} - K_{var}).$$

Particularly, let $S_t$ denote the value of the underlying at time $t \in [0,T]$ and $\sigma^2_{realized(\tau,t)}$, the realized variance of returns over the time interval $[\tau, t]$ with $\tau < t$. Hence. the realized variance is usually priced for $\tau < t_i \leq t \leq T$ as

$$\sigma^2_{realized(\tau,t)} \approx u^2 \sum_{i=\tau}^{t-1} \left( \log \frac{S_{t_{i+1}}}{S_{t_i}} \right)^2 \tag{21}$$

where $u$ represents any scaling factor, such as the annualization factor $u := 100 \times \sqrt{\frac{252}{T}}$.

Note that, (21) is a generalization useful to price the contract in any period of its life, not only at inception ($\tau = 0$). Moreover, is worth mentioned that (21) is an approximation, since $\sigma^2_{realized(\tau,t)}$ is exactly described in continuous-time framework by

$$\sigma^2_{realized(\tau,t)} := u^2 \int_\tau^t \sigma_s^2 ds. \tag{22}$$

For completeness, from (21) we have also that

$$\sigma_{realized(\tau,t)} := \sqrt{\sigma^2_{realized(\tau,t)}}$$

denoting the volatility swap with payoff $= (\sigma_{realized(0,T)} - K_{vol})$ where $K_{vol} = \sqrt{K_{var}}$.

At inception, the fair value of a swap is chosen so that the swap has zero value, meaning ATM. For the remaining life of the contract, we define $V_{var(t)}$ as the value of the variance swap at time $t > 0$ which pays out $\sigma^2_{realized(0,T)}$, and $V_{vol(t)}$ the equivalent for volatility swap. Then for both, we have that their fair value at time $t$ is their value as they were exercised in that moment,

multiplied by the factor $e^{r(T-t)}$ where $r$ represents the constant interest rate and $(T-t)$ the remaining life of the contract. So for the variance swap case:

$$V_{var(t)}^{ATM} = V_{var(t)} e^{r(T-t)}.$$

## 3.2 Bergomi model

Based on a arbitrage-free condition, the Bergomi model uses market-traded Variance swap securities as a hedging instrument [5]. From such securities it is possible to extrapolate the so-called forward variance curve of which the model describes the dynamics. We define the forward variance curve $\xi_t(T)$ observed at time $t$ for maturity T, with $T \geq t$ as

$$\xi_t(T) := \mathbb{E}[\sigma_T^2 | \mathcal{F}_t]$$

where $\mathcal{F}_t$ indicates the entire history of the process up to time $t$.

We assume that $\xi_t$ follows a log-normally distributed and drift-less dynamics and that its volatility $(\omega)$ is function of $(T-t)$. For the complete derivation of the model we refer to [5].

In the original paper, the model is presented in different versions, in fact the model can be generalized with $N$-factors. However, the authors and further research have shown that 2-factors are the best compromise between the precision in fitting the volatility surface and the parsimony principle. Therefore, we only show that version in continuous time here, referring to the original paper for the others.

$$\begin{aligned} dS_t &= r_t S_t dt + \sqrt{\xi_t(T)} S_t dW_t^S \\ d\xi_t(T) &= \omega \left( e^{-\alpha_1 (T-t)} dU_t + \theta e^{-\alpha_2 (T-t)} dW_t \right) \xi_t(T) \end{aligned} \tag{23}$$

where $S_t$ represents the log-normal dynamics of the underlying, $r_t$ the interest rate, $U_t$ and $W_t$ are correlated Brownian motion processes such that $\rho = \mathbb{E}[U_t W_t]$.

The model is ensured to be Markovian since the conditional probability distribution of future states of the process (conditional on both past and present values) depends only upon the present state.

## 3.3 Rough Bergomi model

The rough Bergomi model is a particular rough volatility model presented in [16] as another instance of RFSV models. It is consistent with the realized volatility model present in [7] and it was introduced for pricing purposes. Furthermore, again in [16], it is shown that the model outperform its classical counterparts by capturing the skew term structure observed in the market with only three parameters, instead of the seven required in the 2-factor Bergomi. The model is a non-Markovian generalization of the original. As remarked previously, this loss of Markovianity is a major drawback for the practical uses, since the phase of calibration and simulation, may be largely time-consuming.

The model is derived from the RFSV model, see [16], and we summarize it as

$$dS_u = \mu_u S_u du + \sqrt{\sigma_u^2} S_u dZ_u^{\mathbb{P}} \tag{24}$$

$$\sigma_u^2 = \sigma_t^2 exp\left(\eta \widetilde{W}_t^{\mathbb{P}}(u) + 2\nu C_H Z_t(u)\right)$$

where $\widetilde{W}_t^{\mathbb{P}}(u) := \sqrt{2H}\int_t^u (u-s)^{H-\frac{1}{2}} dW^{\mathbb{P}}(s)$ and $\rho = \mathbb{E}[Z^{\mathbb{P}} W^{\mathbb{P}}]$ is constant over time.

The model is first described under the physical measure $\mathbb{P}$. Note that, to value securities, such as derivatives, we need an artificial probability measure $\mathbb{Q}$, called a risk-neutral measure, which allows us to assign different probabilities to different states of the world including the combination which reflects the real market data. The need for a change of measures is due to the non-martingality of the true probability measure $\mathbb{P}$, as opposed to $\mathbb{Q}$, which prohibits the correct pricing of derivatives as expected discounted dividend streams.

Follows that under $\mathbb{Q} \sim \mathbb{P}$ on $[t, T]$ as in the Black and Scholes model, the rough Bergomi model can be then summarized in:

$$\begin{aligned} dS_t &= \sqrt{v_t} S_t dZ_t^{\mathbb{Q}} \\ v_t &= \xi_0(t) exp\left(\eta\sqrt{2H}\int_0^t (t-s)^{H-\frac{1}{2}} dW_s - \frac{\eta^2}{2} u^{2H}\right) \\ dZ_t^{\mathbb{Q}} &= \rho dW_t^{\mathbb{Q}} + \sqrt{1-\rho^2} dW_t^{\mathbb{Q}\perp}. \end{aligned} \tag{25}$$

# 4 Simulation and calibration of rough Bergomi model

In this section, we firstly simulate the price of a European call option and its volatility curve according to the rough Bergomi model, secondly we calibrate the model on market data.

We work on the freely-available algorithms [17] by Achraf Maalej and [18] by McCrickerd and Pakkanen.

## 4.1 Simulation

In order to effectively price option contracts, we evaluate the rough Bergomi model using a Monte-Carlo method. As mentioned at the beginning, such technique requires an efficient and fast fraction Brownian motion paths-generator algorithm. The hybrid scheme is, therefore, the recommended choice since it guarantees the fastest computation time.

**Algorithm 4.1.** The algorithm generates paths with time-horizon of $T$ in years, with $N$ steps per unit of time. Other inputs are the estimated roughness $H$, correlation $\rho$, constants $\eta$ and $\xi_0(t)$.

1. Simulate the process $V_t = \sqrt{2H}\int_0^t (t-s)^{H-\frac{1}{2}} dW_s^V$ using hybrid scheme.
2. Correlate the increments of the Brownian motion $W_t^V$ with the stock price process driven by increments of $Z_t$: $Z_{t+1} - Z_t = \rho(W_{t+1}^V - W_t^V) + \sqrt{1-\rho^2}(W_{t+1} - W_t)$ where $W_t$ is an independent Brownian motion process.
3. Compute the variance process $v_t$ using $\xi$ and $\eta$, namely $v_t = \xi_0(t)\exp\left(\eta V_t - \frac{\eta^2}{2}t^{2H}\right)$.
4. Simulate the stock price process $S_t$ using the scheme $S_{t+1} = S_t + \exp\left(\sqrt{v_t}(Z_{t+1} - Z_t) - \frac{1}{2}v_t T/N\right)$.

The option price in case of a European call option is given by $C(k,t) = \mathbb{E}\left[(S_T - k)_+\right]$. Therefore, once simulated the stock price $S_t$, it can be retrieved as

$$C(k,t) = \frac{1}{N}\sum_{i=0}^{N} \max(S_t^i - e^k, 0).$$

For the implementation we used the following estimated parameters used in literature, in addition to $\xi_0 = 0.15^2$:

| $H$ | $\rho$ | $\eta$ |
|---|---|---|
| 0.07 | -0.9 | 1.9 |

Table 4: Parameters for European call price simulation.

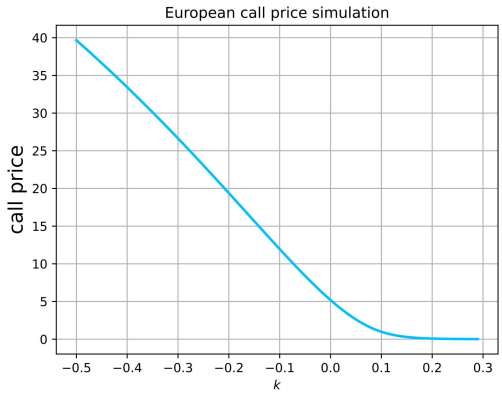



Figure 5: European call option price simulation with $H = 0.07$, $\rho = -0.9$, $\eta = 1.9$ and $k = 0$.

Calculation time for $n = 30000$ simulations is approximately 2.243330 seconds with normal hardware. The algorithm is confirmed to be efficient and potentially exploitable in the financial sector.

We report a comparison between the exact Cholesky decomposition method and the approximated hybrid scheme for $n = 100000$ simulations. The results are clear in favor of the latter with a relatively negligible loss in accuracy.

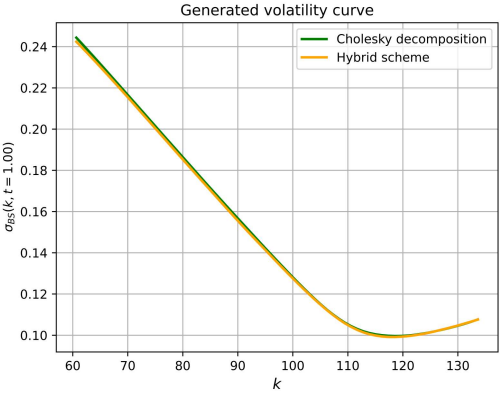



Figure 6: Volatility curve simulations comparison with $H = 0.07$, $\rho = -0.9$, $\eta = 1.9$ and $k = 0$.

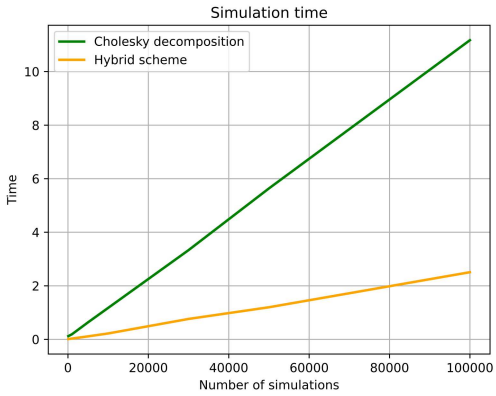



Figure 7: Simulations time comparison.

## 4.2 Calibration

The simulation of the rough Bergomi model relies on 3 parameters: $H, \eta$ and $\rho$ in addiction to the initial forward variance. Calibration of these is therefore required. A visual comparison of how each parameter affects implied volatility is shown in the figures below. In particular, it is easy to see how $H$ affects the skewness, $\eta$ the convexity of the curve, $\rho$ the explosion in terms of smiles and skew and $\xi_0$ the level of the curve.

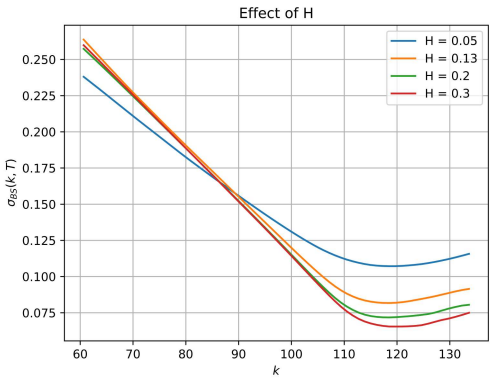



Figure 8: Volatility curve for different $H$, other parameters constant.

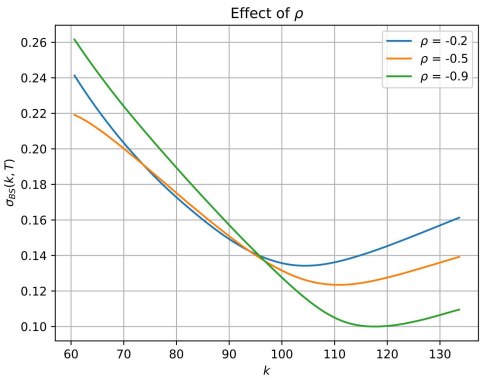



Figure 9: Volatility curve for different $\rho$, other parameters constant.

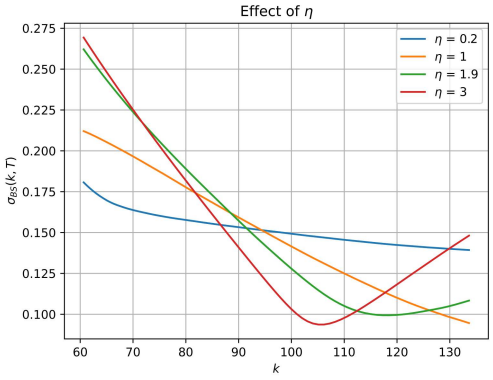



Figure 10: Volatility curve for different $\eta$, other parameters constant.

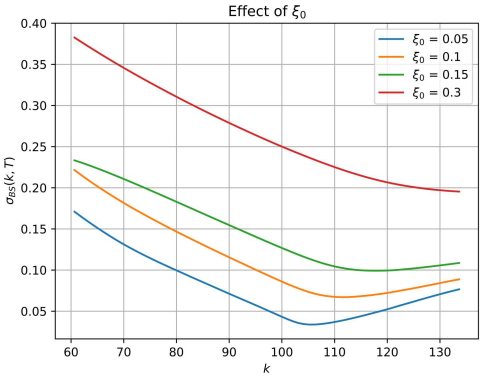



Figure 11: Volatility curve for different $\xi_0$, other parameters constant.

The calibration operation can be done from the observed volatility surface at a given day. Formally, we define $\mathcal{M} := \mathcal{M}(\theta)_{\theta\in\Theta}$ with $\Theta \in \mathbb{R}^n$ for some $n \in \mathbb{N}$. $\theta$ represent the vector of parameters of the model so that the model can be fully specified by them. We also define the fair pricing-map for a European call option as

$$\mathcal{P} : (\theta, \zeta) \times (M, T) \mapsto \mathbb{E}[S_T(\theta, \zeta) - M]^+$$

where $\zeta$ represents possible information on the corporate market and $M, T$ denote moneyness and time to maturity, respectively.

We define calibration as

$$\widehat{\theta} = \underset{\theta\in\mathcal{M}}{\arg\min}\ \delta(\widetilde{\mathcal{P}}(\theta, \zeta, M, T), P(\zeta, M, T)) \tag{26}$$

where $\delta(.,.)$ is a suitable chosen metric, $\widetilde{\mathcal{P}}$ is a numerical approximation of the theoretical pricing map $\mathcal{P}$ and $P$ the prices observed in the market.

According to the literature, instead of working directly on option prices, a better approach is to work with implied volatilities of the Black-Scholes type. Therefore, we define also the implied volatility-map as

$$\varphi : (\theta, \zeta, M, T) \mapsto \sigma_{BS}(\theta, \zeta, M, T).$$

We want optimized the model parameters generating the model surface, to fit the empirical implied volatility observed in the market of liquid European options. Thus, from (26) with $\delta(.,.)$ a least squares function, we can write

$$\widehat{\theta} = \underset{\{H,\rho,\eta\}\in\mathcal{M}}{\arg\min} \sum_{i=1}^{n} (\widetilde{\varphi}(\theta, M, T) - \sigma_{BS}(M, T, \sigma))^2. \tag{27}$$

In practice, we are trying to minimize the squared difference between the volatility surface generated by the model and the one observed in the market which is equivalent to (26).

## 4.3 Empirical results

We calibrate the model on data of traded European call options on the SPY in 02/03/2015. We isolate the implied volatility $\sigma_{BS}$ according to Black-Scholes model:

$$\begin{aligned} C_{BS} &= \mathcal{N}(d_1)S_t - \mathcal{N}(d_1)ke^{-r(T-t)} \\ d_1 &= \frac{1}{\sigma_{BS}\sqrt{T-t}}\left[\log\left(\frac{S_t}{k}\right) + \left(r + \frac{\sigma_{BS}^2}{2}\right)(T-t)\right] \\ d_2 &= d_1 - \sigma_{BS}\sqrt{T-t} \end{aligned} \tag{28}$$

where $r$ the constant interest rate and $k$ the strike price. We get three different implied volatilities by taking $S_t$ as bid price, bid price and average price (as an extrapolation).

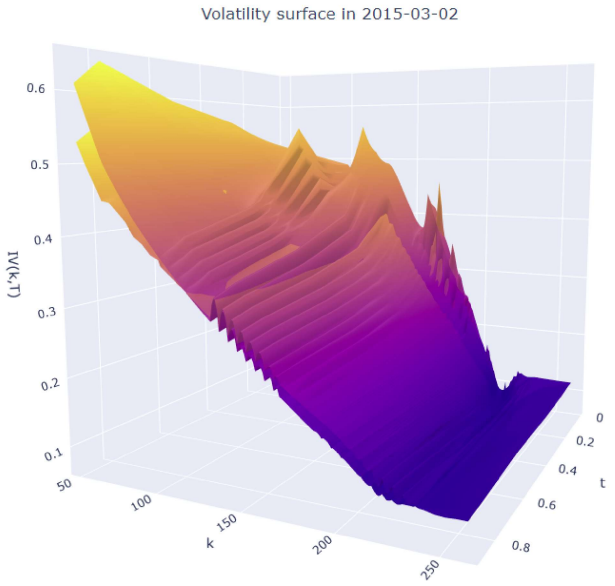



Figure 12: Mid-Implied Volatility surface of SPY in 02/03/2015.

First, we calibrate the model parameters for each expiration date rather than for the entire volatility surface. Note that the algorithm does not calibrate the forward variance parameter $\xi_0(T)$ which is calculated as a corrected approximation of the implied volatility observed for ATM options for each expiration. The results are shown in the following table.

| Expiration | $H$ | $\rho$ | $\eta$ |
|---|---|---|---|
| 0.0301 | 0.138426 | -0.816584 | 2.653066 |
| 0.0849 | 0.185319 | -0.761033 | 3.085558 |
| 0.2986 | 0.155451 | -0.840005 | 2.336378 |
| 0.8739 | 0.128683 | -0.885059 | 1.920311 |

Table 5: Calibrated parameters on the main expiration dates.

The algorithm is very time consuming, taking up to over an hour to complete the parameter calibration. The results are shown in Appendix B and are very satisfactory. In particular, the calibration of the parameters is performed only for the 4 most liquid expiration dates, due to the heaviness of the minimization problem. Of course, other optimization algorithms might be more efficient, we propose it as a further research topic.

We also note some inconsistencies with the Monte-Carlo simulations with the algorithm proposed by McCrickerd and Pakkanen. However, these seem to be related to a relatively low number of simulations, as they tend to disappear with $N > 50000$. Since this is not a serious problem, the investigation of this issue goes beyond the purposes of the document.

As a further step, we also proceed to calibrate the entire volatility surface by reformulating (27) to

$$\widehat{\theta} = \underset{\{H,\rho,\eta\}\in\mathcal{M}}{\arg\min} \sum_{i=1}^{T}\sum_{i=1}^{n}(\widetilde{\varphi}(\theta, M, T) - \sigma_{BS}(M, T, \sigma))^2. \tag{29}$$

In this case, the estimated parameters are:

| $H$ | $\rho$ | $\eta$ |
|---|---|---|
| 0.124083 | -0.775163 | 2.585495 |

Table 6: Calibrated parameters on the entire volatility surface.

Complete results are shown always in Appendix B. As might be expected, the algorithm is even heavier and the results are less accurate than the previous ones, especially for short expiration dates. Note that, this differs from the results obtained by the authors in [16] for different dates, where they do not see this effect.

However, we remark the good implied volatility approximation of the model. Easy to imagine is a comparison with the Black-Scholes model on which the rough Bergomi model offer a massive improvement. In fact, if the first were correct, it would describe a flat volatility surface.

For the comparison with classic time-homogeneus model such as Hull and White [4], SABR [3] and Heston [2], we refer to the unanimous literature on their worse fitting of the volatility surfaces ([16], [19], [7]), as it is outside the scope of this study.

That being said, those models are reasonably still widely used. In fact, as we have shown, the calibration process of rough volatility models is prohibitively expensive. We tested for a relatively small number of expiration dates, consequently for larger sets the model is unusable, also considering that the calibration process is not one-off but occurs at least daily.

# Conclusion

The aim of this work was to implement a relatively new class of stochastic volatility models for option pricing, with particular reference to the rough Bergomi. The model looks promising from the results, despite the slowness of the calibration phase which constitutes a severe drawback.

Fractional Brownian motions are processes that require computationally expensive algorithms to be simulated even though some mathematical techniques help to increase efficiency with less loss of precision.

The analysis of log-volatility time series showed that fractional Brownian motion with Hurst parameter $\in (0, 0.5)$ is a better choice to model such rough behavior than the standard Brownian motion usually used.

The option pricing model is presented as a derivation of original Bergomi model, which relies on forward variance to infer a arbitrage-free condition using available market securities, i.e. variance swaps. Its low number of parameters and its good adaptation to the volatility surface make it potentially usable in practice. However, the financial sector requires both robustness and speed of computation. In this work we have presented an algorithm based on Monte-Carlo simulations; recently other different approaches have been proposed in the literature such as mathematical approximation methods ([20], [21], [22]) and numerical approximation using Artificial Neural Networks ([23], [24]). In particular, the latter appear to be a valid research field ensuring speed in the calibration phase. However, even with machine learning methods the importance of stochastic methods is not discussed. Hence the interest in the approach we have chosen in this document.

# References


[1] Fischer Black and Myron Scholes. The pricing of options and corporate liabilities. *Journal of political economy*, 81(3):637–654, 1973.

[2] Steven L Heston. 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, 1993.

[3] Patrick S Hagan, Deep Kumar, Andrew S Lesniewski, and Diana E Woodward. Managing smile risk. *The Best of Wilmott*, 1:249–296, 2002.

[4] John Hull and Alan White. Pricing interest-rate-derivative securities. *The review of financial studies*, 3(4):573–592, 1990.

[5] Lorenzo Bergomi. Smile dynamics ii. *Available at SSRN 1493302*, 2005.

[6] Fabienne Comte and Eric Renault. Long memory in continuous-time stochastic volatility models. *Mathematical finance*, 8(4):291–323, 1998.

[7] Jim Gatheral, Thibault Jaisson, and Mathieu Rosenbaum. Volatility is rough. *Quantitative finance*, 18(6):933–949, 2018.

[8] Benoit B Mandelbrot and John W Van Ness. Fractional brownian motions, fractional noises and applications. *SIAM review*, 10(4):422–437, 1968.

[9] Oksana Banna, Yuliya Mishura, Kostiantyn Ralchenko, and Sergiy Shklyar. *Fractional Brownian motion: approximations and projections*. John Wiley & Sons, 2019.

[10] Ton Dieker. *Simulation of fractional Brownian motion*. PhD thesis, Masters Thesis, Department of Mathematical Sciences, University of Twente ..., 2004.

[11] Robert B Davies and DS Harte. Tests for hurst effect. *Biometrika*, 74(1):95–101, 1987.

[12] Claude R Dietrich and Garry N Newsam. Fast and exact simulation of stationary gaussian processes through circulant embedding of the covariance matrix. *SIAM Journal on Scientific Computing*, 18(4):1088–1107, 1997.

[13] James W Cooley and John W Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of computation*, 19(90):297–301, 1965.

[14] Mikkel Bennedsen, Asger Lunde, and Mikko S Pakkanen. Hybrid scheme for brownian semistationary processes. *Finance and Stochastics*, 21:931–965, 2017.

[15] Mathieu Rosenbaum. A new microstructure noise index. *Quantitative Finance*, 11(6):883–899, 2011.

[16] Christian Bayer, Peter Friz, and Jim Gatheral. Pricing under rough volatility. *Quantitative Finance*, 16(6):887–904, 2016.

[17] Achraf Maalej. Github repository. Available online at: `https://github.com/maalejach/Rough-volatility-model--and--calibration`.

[18] Ryan McCrickerd. Github repository. Available online at: `https://github.com/ryanmccrickerd/rough-bergomi`.

[19] Ryan McCrickerd and Mikko S Pakkanen. Turbocharging monte carlo pricing for the rough bergomi model. *Quantitative Finance*, 18(11):1877–1886, 2018.

[20] Qinwen Zhu, Grégoire Loeper, Wen Chen, and Nicolas Langrené. Markovian approximation of the rough bergomi model for monte carlo option pricing. *Mathematics*, 9(5):528, 2021.

[21] Christian Bayer, Peter K Friz, Paul Gassiat, Jorg Martin, and Benjamin Stemper. A regularity structure for rough volatility. *Mathematical Finance*, 30(3):782–832, 2020.

[22] Frederi Viens and Jianfeng Zhang. A martingale approach for fractional brownian motions and related path dependent pdes. *The Annals of Applied Probability*, 29(6):3489–3540, 2019.

[23] Blanka Horvath, Aitor Muguruza, and Mehdi Tomas. Deep learning volatility: a deep neural network perspective on pricing and calibration in (rough) volatility models. *Quantitative Finance*, 21(1):11–27, 2021.

[24] Henry Stone. Calibrating rough volatility models: a convolutional neural network approach. *Quantitative Finance*, 20(3):379–392, 2020.

[25] Francesca Biagini, Yaozhong Hu, Bernt Øksendal, and Tusheng Zhang. *Stochastic calculus for fractional Brownian motion and applications*. Springer Science & Business Media, 2008.

[26] Lorenzo Bergomi and Julien Guyon. The smile in stochastic volatility models. *Available at SSRN 1967470*, 2011.

[27] José Igor Morlanes. *Some Extensions of Fractional Ornstein-Uhlenbeck Model: Arbitrage and Other Applications*. PhD thesis, Department of Statistics, Stockholm University, 2017.

[28] Peter Carr and Roger Lee. Realized volatility and variance: Options via swaps. *Risk*, 20(5):76–83, 2007.

[29] Giulia Livieri, Saad Mouti, Andrea Pallavicini, and Mathieu Rosenbaum. Rough volatility: evidence from option prices. *IISE transactions*, 50(9):767–776, 2018.

[30] Antoine Jacquier, Claude Martini, and Aitor Muguruza. On vix futures in the rough bergomi model. *Quantitative Finance*, 18(1):45–61, 2018.

[31] Tyrone E Duncan, Yaozhong Hu, and Bozenna Pasik-Duncan. Stochastic calculus for fractional brownian motion i. theory. *SIAM Journal on Control and Optimization*, 38(2):582–612, 2000.

[32] Masaaki Kijima and Chun Ming Tam. Fractional brownian motions in financial models and their monte carlo simulation. *Theory and Applications of Monte Carlo Simulations*, pages 53–85, 2013.

[33] Jan Matas and Jan Pospíšil. On simulation of rough volterra stochastic volatility models. *arXiv preprint arXiv:2108.01999*, 2021.

[34] Sidi Mohamed Ould Aly. Forward variance dynamics: Bergomi's model revisited. *Applied Mathematical Finance*, 21(1):84–107, 2014.

[35] Chuong B Do. The multivariate gaussian distribution. *Section Notes, Lecture on Machine Learning, CS*, 229, 2008.

[36] Elizabeth Zuniga. *Volterra processes and applications in finance.* PhD thesis, Université Paris-Saclay, 2021.

# Appendix A

```
################################################################################
####    Gamma matrix                                                        ####
################################################################################

'''
Computes the Gamma matrix and its main array for a given algorithm and H
'''

import numpy as np

def covariance_f(N, H, output_gamma=False, algorithm = 'Chol'):
    if algorithm == "H":
        N +=2 #Hosking needs at least 2 correlation in advance
    elif algorithm == "CEM":
        N +=1 #Hosking needs at least 2 correlation in advance

    G = np.zeros((N,N), dtype = 'float') #gamma matrix

    c = np.zeros(N, dtype = 'float') #c vector
    c_rev = np.zeros(N, dtype = 'float') #reverse of c vector

    for k in range(0,N):
        c[k] =  1/2*((np.abs(k)+1)**(2*H)+
                np.abs(np.abs(k)-1)**(2*H)-
                2*(np.abs(k))**(2*H)) #gamma array with covariance
    if output_gamma == True:
        G[0,:] = c
        c_rev = np.flip(c[1:])
        for i in range(1,N):
            G[i,:] = np.concatenate((c_rev[N-i-1:], c[:N-i])) #gamma matrix
    return c,G
```

Listing 1: Covariance matrix algorithm.

```
################################################################################
####    Cholesky Decomposition                                                ####
################################################################################

'''
Computes a fGn path with Cholesky decomposition from the main array of a Gamma
    matrix
'''

import numpy as np
from Covmatrix_Simulations import *

def Cholesky_f(c,H,T,Nu):
    N = T*Nu
    L = np.zeros((N,N))
    L[:,0] = c #fixed computations
    L[1,1] = np.sqrt(1-(c[1]**2)) #fixed computations

    sum_squared = 0 #sum of the row elements squared for diag elements computation
    sum_mult_k = 0 #sum of elements needed for i>j elements computation

    for j in range (1,N): #columns
        for i in range (2,N): #rows
            if i > j:
                sum_mult_k = np.matmul(L[i,:j],np.transpose(L[j,:j]))
                L[i,j] = (c[i-j] - sum_mult_k)/L[j,j] #see paper

            elif i == j:
                sum_squared = np.sum(L[i,:j]**2)
                arg = c[0] - sum_squared
                if arg < 0: # to correct for negative power without using complex
    numbers
                    q = -1
                else:
                    q = 1
                L[i,j] = q*np.sqrt(arg*q) #see paper

    Z = np.random.standard_normal(N) # Standard Gaussian vector
    X = np.matmul(L,Z.T)*(T/Nu)**H #first order differtiated process
    return X
```

Listing 2: Cholesky decomposition matrix algorithm.

```
################################################################################
####    Hosking Algotrithm                                                  ####
################################################################################

'''
Computes a fGn path with Hosking algorithm from the main array of a Gamma matrix
'''

import numpy as np
from Covmatrix_Simulations import *

def Hosking_f(c,H,T,Nu):
    N = T*Nu-1
    #no need Flip matrix in python
    X = np.zeros((1)) #fBm increments storing array
    mu = np.zeros((N+1)) #mu array that memorizes all computed mus
    sigma_sq = np.zeros((N+1)) #sigma squared array that memorizes all computed
    sigmas
    tau = np.zeros((N+1)) #tau array that memorizes all computed taus
    phi = np.zeros((N))  #phi array that memorizes all computed phis
    X[0] = np.random.standard_normal(1) # 0-step
    sigma_sq[0] = 1-c[1]**2 # 0-step
    tau[0] = c[1]**2 # 0-step
    d_arr_temp = np.array(c[1]) #varing length 'd' array (no memory)
    mu[0]= d_arr_temp*X[0] # 0-step --- non-zero mean process

    for j in range(0,N):
        x = 0 #temp variable storing fBm increment
        d = np.zeros((j+1)) #temp variable storing d array

        x = mu[j] + np.random.standard_normal(1) #increment shift
        X = np.append(X, x) #storing in X array

        sigma_sq[j+1] = sigma_sq[j] - (c[j+2]-tau[j])**2/sigma_sq[j] #see paper
        phi[j] = (c[j+2]-tau[j])/sigma_sq[j] #see paper

        d = d_arr_temp - phi[j]*np.flip(d_arr_temp) #see paper
        d_arr_temp = np.append(d, phi[j]) #new d array
        tau[j+1] = np.dot(d_arr_temp, np.flip(c[1:j+3])) #see paper
        mu[j+1] = np.dot(np.flip(X), d_arr_temp) #see paper

    X *= (T/Nu)**H #first order differtiated process
    return X
```

Listing 3: Hosking algorithm.

```
################################################################################
####    CEM Algorithm                                                       ####
################################################################################

'''
Computes a fGn path with CEM algorithm from the main array of a Gamma matrix
'''

import numpy as np
from Covmatrix_Simulations import *

def CEM_f(c,H,T,Nu):
    N = T*Nu
    M = 2*N
    c_circ = np.concatenate((c, c[::-1]))
    c_circ = np.delete(c_circ, [-1])
    c_circ = np.delete(c_circ, [-N]) #array circular matrix

    eigenes = np.fft.fft(c_circ) #FFT
    eigenes = eigenes.real #get rid of immaginary part
    eigenes[eigenes<0] = 0 #since positivity is not ensure, see paper

    Z = np.random.standard_normal(M) # Standard Gaussian vector
    Z_inver = np.fft.ifft(Z) #inverse FFT
    Z_inver = Z_inver.real #get rid of immaginary part

    Y = np.sqrt(eigenes)*Z_inver

    Y_inver = np.fft.fft(Y) #inverse FFT
    X = Y_inver.real #get rid of immaginary part

    return X[:N]*(T/Nu)**H
```

Listing 4: CEM algorithm.

```
################################################################################
####    Hybrid Scheme                                                       ####
################################################################################

'''
Computes a fBm path by Hybrid scheme with convolution and the covariance matrix
    necessary
'''

import numpy as np

def cov_matrix(N,H):
    C = np.zeros((2,2))
    C[0,0] = 1/N
    C[0,1] = 1/((H+1/2)*N**(H+1/2))
    C[1,0] = C[0,1]
    C[1,1] = 1/(2*H*N**(2*H))
    return C

def Volt_sim(T,Nu,H):
    N = T*Nu
    alpha = H-1/2
    dW = np.random.multivariate_normal([0,0], cov_matrix(N,H), (N)) #binormal
    random variable
    V = np.zeros(N)
    g = np.zeros((N))
    for k in range(2,N):
        g[k] = ((k**(alpha+1)-(k-1)**(alpha+1))/(alpha+1))**(1/alpha)
        g[k] /= N
        g[k] **= alpha
    Sum = np.convolve(g,dW[:,1])
    V = np.sqrt(2*H)*(dW[:,0] + Sum[:N]) #the convolution needs to be truncated
    return V,dW[:,0]
```

Listing 5: Hybrid scheme algorithm.

```
################################################################################
####    Rough Volatility                                                    ####
################################################################################

'''
Computes m(q,mesh), $\zeta$ and checks normality of log increments for a given
    index
'''

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy.stats as stats

data = pd.read_csv(r'C:\Users\Lucam\OneDrive\Desktop\Dissertation\Codes\
    oxfordmanrealizedvolatilityindices.csv')
df = np.array(pd.DataFrame(data, columns=['Symbol', 'rk_twoscale']))

class p:
    index = '.SPX'
    max_mesh = 150
    q = np.arange(0.5, 3.5, 0.5).tolist()
    norm_vec_check = [1,2,5,50]

index_var = df[df[:,0]== p.index ,1]
index_vol = np.sqrt(index_var.astype(float))

def m(vol, q, max_mesh, norm_vec_check):
    mesh = np.arange(1,max_mesh+1)
    m = np.zeros((max_mesh,len(q)))
    norm_check = np.zeros((len(vol), 4))
    for c in q:
        for j in mesh:
            v = np.array(0)
            for i in range(j, len(vol)):
                x = 0
                arg = 0
                arg = np.log(vol[i])-np.log(vol[i-j])
                if j in norm_vec_check and c == 1:
                    norm_check[i, norm_vec_check.index(j)] = arg
                x = np.abs(arg)**c
                v = np.append(v, x)
            m[j-1,q.index(c)] = np.mean(v)
    return np.log(m), np.log(mesh), norm_check

log_m, log_mesh, norm_check = m(index_vol, p.q, p.max_mesh, p.norm_vec_check)

```

```
zeta = np.zeros((len(p.q)))
std_err = np.zeros((len(p.q)))
inter = np.zeros((len(p.q)))
for i in range(0,len(p.q)):
    zeta[i], inter[i],_, _, std_err[i] = stats.linregress(log_mesh, log_m[:,i])
print(zeta)

plt.rcParams.update({'font.size':14})
plt.plot(log_mesh, log_m, '*')
label = []
for k in range(0,len(p.q)):
    plt.plot(log_mesh, log_mesh * zeta[k] + inter[k])
    label.append('q = %1.3f' % p.q[k])
plt.title("log(m(q, $\Delta$)) as function of log($\Delta$)")
plt.ylabel(r"log(m(q, $\Delta$))")
plt.xlabel(r"log($\Delta$)")
plt.legend(label)
plt.show() #logm on logmesh

plt.plot(p.q, zeta, 'o')
H, inter2,_, _, std_err = stats.linregress(p.q, zeta)
print("Coeff =", H)
print("SE =", std_err)
print("upper .95% =", H+1.96*std_err)
print("bottom .95% =", H-1.96*std_err)
plt.plot(p.q, np.array(p.q)*H + inter2)
plt.title("$\zeta$q - H q")
plt.show() #zeta on q

_,a = plt.subplots(2, 2)
v1 = [0,0,1,1]
v2 = [0,1,0,1]
for r in range(0, 4):
    incr = norm_check[norm_check[:,r] != 0, r]
    mu = np.mean(incr)
    std = np.std(incr)
    x = np.arange(-2,2,4/1000)
    a[v1[r], v2[r]].hist(incr, bins=100, density=True)
    a[v1[r], v2[r]].plot(x, stats.norm.pdf(x, mu, std))
    a[v1[r], v2[r]].set_title("log increments of volatility $\Delta$=%i days on
    normal distribution" % p.norm_vec_check[r])
plt.show() #increments on normal distribution
```

Listing 6: Rough volatility empirical results.

# Appendix B

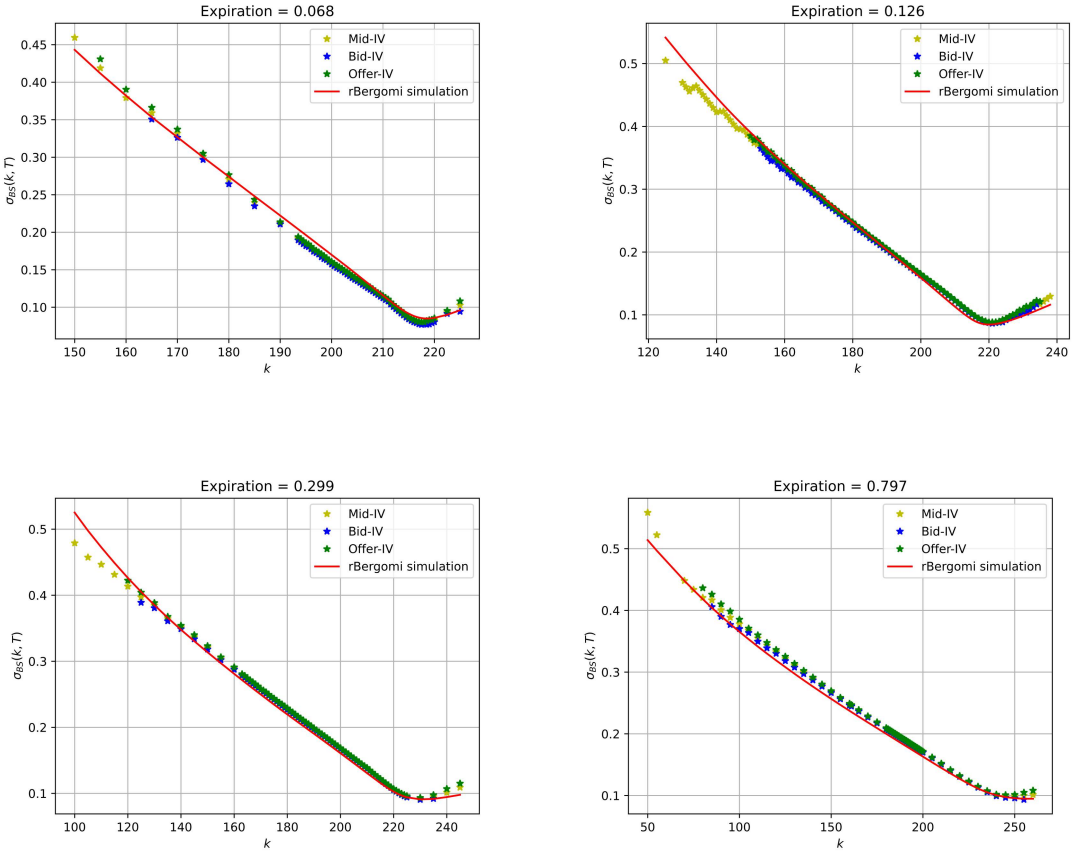
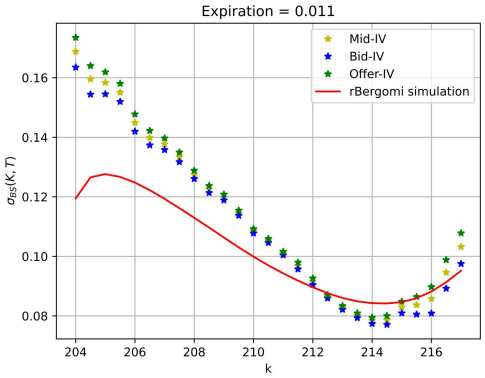
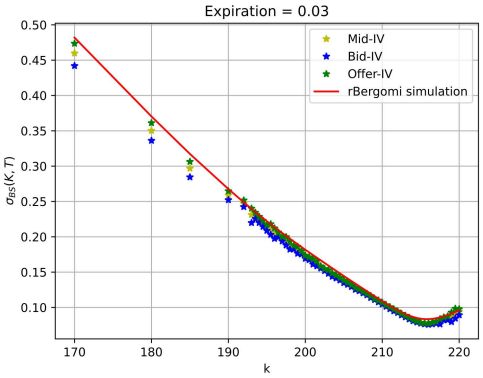
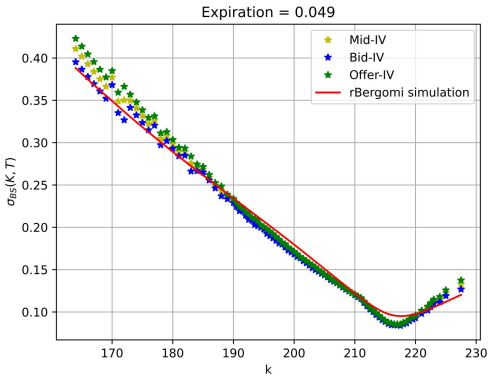
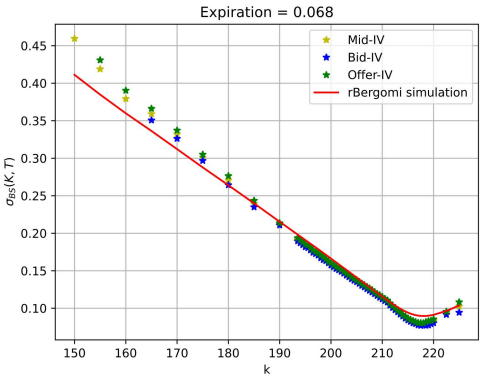


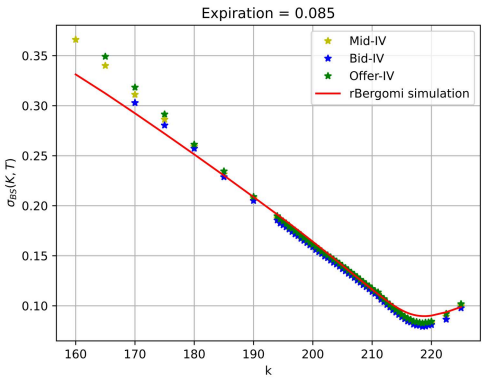

Figure B.1 : Results from rough Bergomi calibration on the main expiration dates.

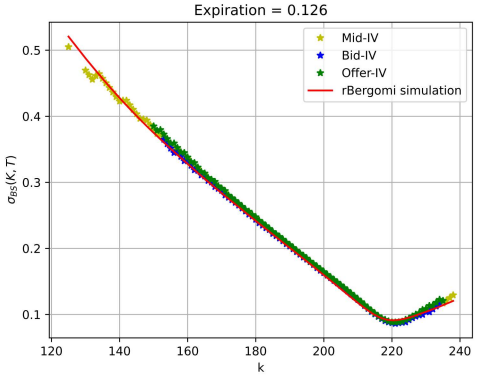

Expiration = 0.011
Mid-IV
Bid-IV
Offer-IV
rBergomi simulation
$\sigma_{BS}(K,T)$
k


Expiration = 0.03
Mid-IV
Bid-IV
Offer-IV
rBergomi simulation
$\sigma_{BS}(K,T)$
k


Expiration = 0.049
Mid-IV
Bid-IV
Offer-IV
rBergomi simulation
$\sigma_{BS}(K,T)$
k


Expiration = 0.068
Mid-IV
Bid-IV
Offer-IV
rBergomi simulation
$\sigma_{BS}(K,T)$
k


Expiration = 0.085
Mid-IV
Bid-IV
Offer-IV
rBergomi simulation
$\sigma_{BS}(K,T)$
k


Expiration = 0.126
Mid-IV
Bid-IV
Offer-IV
rBergomi simulation
$\sigma_{BS}(K,T)$
k

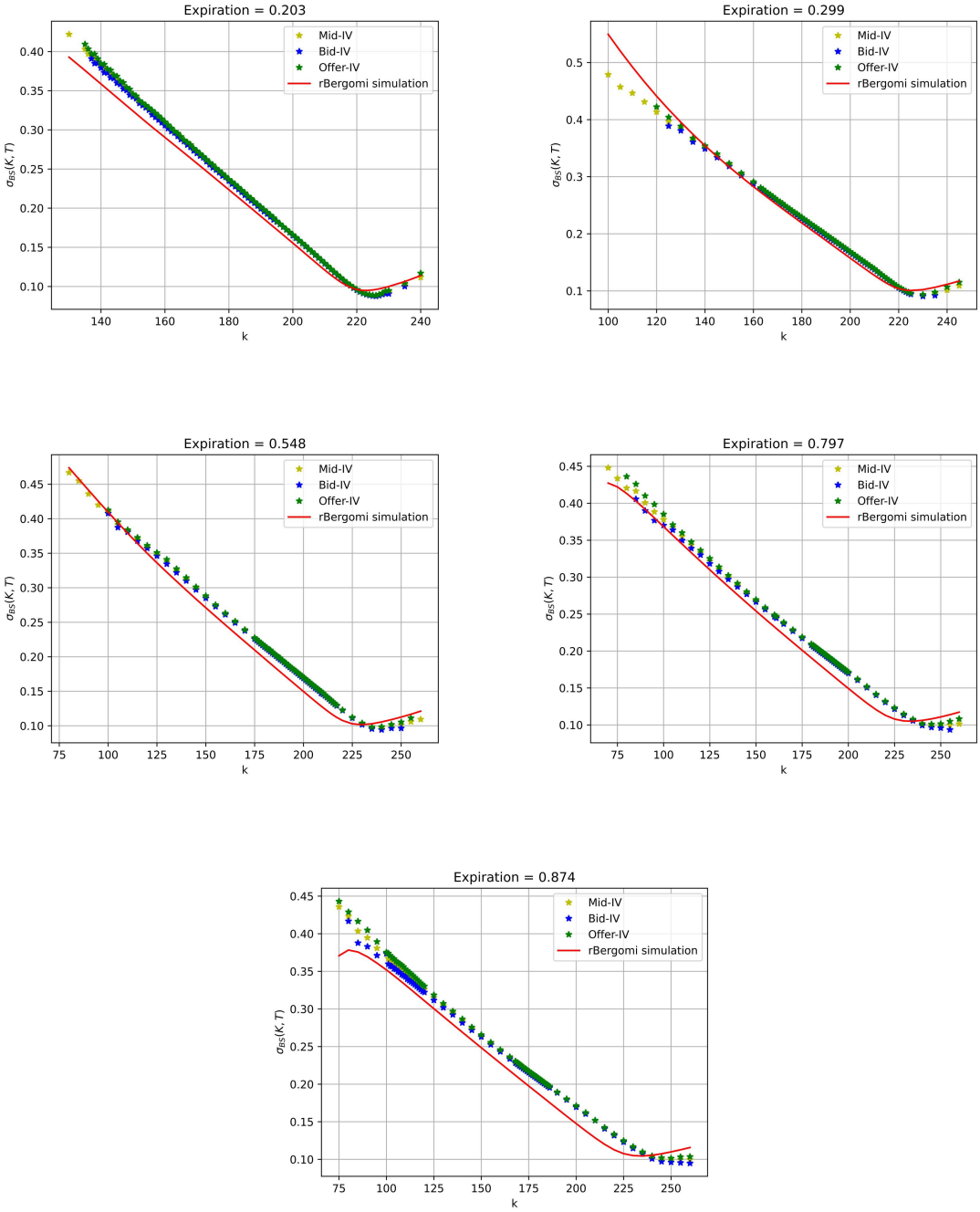



Figure B.2 : Results from rough Bergomi calibration on the entire volatility surface.