King's College London
Department of Mathematics
Submission Cover Sheet for Coursework

The following cover sheet must be completed and submitted with any dissertation, project, coursework essay or report submitted as part of formal assessment for degree in the Mathematics Department.

You are not required to write your name on your work

Candidate number (this is found on your student record account)	AF24124
Module Code and Title	7CCMFM50 MSc Financial Mathematics Project
Title of Project/Coursework	Estimating rough volatility

Declaration

By submitting this assignment I agree to the following statements:
I have read and understand the King's College London Academic Honesty and Integrity Statement that I signed upon entry to this programme of study.
I declare that the content of this submission is my own work.
I understand that plagiarism is a serious examination offence, an allegation of which can result in action being taken under the College's Misconduct regulations.

Your work may be used as an example of good practice for future students to refer to. If chosen, your work will be made available either via KEATS or by paper copy. Your work will remain anonymous; neither the specific mark nor any individual feedback will be shared. Participation is entirely optional and will not affect your mark. If you consent to your submission being used in this way please add an X in the box to the right.	

Estimating rough volatility

by

AF24124

Department of Mathematics
King's College London
The Strand, London WC2R 2LS
United Kingdom

Email: -
Tel: -

28 August 2025

Report submitted in partial fulfilment of the requirements for the degree of MSc in Financial Mathematics in the University of London

Abstract

Rough volatility is a novel concept that has gained popularity in the past decade due to its empirical advantages over traditional financial models; however, recent literature critiques the very estimation techniques that led to its discovery, and, thereafter, the usage of rough volatility models has now declined in favour of more parsimonious models that can reproduce those similar empirical characteristics without the need of rough volatility. As such, in this project, we implement parametric and non-parametric estimators of the Hurst exponent and stress-test them under model misspecification (departures from fractional Brownian Motion) and with real data to ascertain their consistency and reliability, which is paramount to estimating true rough volatility. From simulations, in general no one estimator is dominates; q-th order Power Variation is most accurate for clean fBM but develops a bias floor under time-change (RFSV) and a small positive bias under fOU; Wavelet estimators remain well calibrated on bias across misspecification and H at the cost of higher variance and larger sample sizes; likelihood and the formative Log–Log regression estimators degrade as persistence increases. On SPX realised variance (1-min RV aggregated to daily), estimates remain in the rough range ($H < 0.5$), but vary across years and methods, despite being well adjusted for when targeting $H = 0.1$, the proposed volatility roughness level in the literature, which may instead suggest material scale/model effects. These patterns may be explained by recent critiques that have observed roughness can reflect estimation and sampling artefacts, particularly those of estimated realised volatility that acts as a surrogate for (unobservable) instantaneous spot volatility rather than true roughness itself.

Acknowledgements

I am grateful to my supervisor for their constructive feedback and guidance throughout the development of this work.

Contents

Introduction

In the early 1980s, the foundations for volatility modelling began to develop, from simpler ARCH/GARCH models that leveraged discrete time-series data to more complex stochastic models, such as Heston [20], Hull and White [21], and SABR [18], which utilised continuous time. However, these models struggle to reproduce stylised market features [11], like the power-law decay of autocorrelations or ATM volatility skews ([16]). Given these inconsistencies, Gatheral et al.'s [16] paper on "Volatility is Rough", provided a new outlook on modelling volatility, empirically demonstrating with Log–Log regression that volatility is rough, consistently obtaining a Hurst exponent of $H \approx 0.10$ across multiple asset classes as well as offering the rough stochastic fractional volatility model that was competitive against traditional autoregressive models such as AR(5), AR(10) and HAR(3). With this, Bayer et al. [3] developed pricing models, such as the rough Bergomi model, that fitted market volatility ATM skews with power-law $\psi(\tau) \sim \tau^{H-1/2}$, as Fukasawa [15] theorised that rough models would achieve. However, calibration of VIX/VVIX proved more challenging, with relatively flat/inconsistent smiles in some implementations. Later, Quadratic Rough Heston [17] achieved joint calibration of SPX and VIX smiles. However, literature that had previously favoured fractional models with long-range dependence where $H > 0.5$ (e.g., Comte and Renault's FSV model [10]) was now at odds with this new rough volatility paradigm that supports the opposite $H < 0.5$. Hence, to investigate this matter further, the key results of this project are about the estimator performance due to their foundational significance in developing the rough volatility paradigm, in this we replicate Gatheral et al.'s [16] Log–Log Regression methodology and compare them with other estimators such as maximum likelihood [8], Wavelet [1] and semi-parametric version of q-th order Power Variation (developed in section 3.1.4) and assess their strengths and weaknesses as well as whether the claim of $H \approx 0.10$ was misinterpreted due to some estimation error; indeed, the estimator they had utilised arguably had better alternatives depending on context; q-th order Power Variation leads

for clean fBM, but under RFSV it exhibits a persistent downward offset and under fOU a small upward bias; Wavelet remains closest on average under misspecification across H, but is variance-heavy; likelihood and the formative Log–Log regression are increasingly biased as H grows, however, based on these results there was another inconsistency without any resolution: the well-defined estimators would not agree upon a singular Hurst exponent for SPX daily realised Variance data, ranging from $H \in [0.05, 0.30]$ and being inconsistent to themselves at different time scales, the results were inconclusive despite them fitting well within the 'rough' ($H < 0.5$) regime. These findings may echo those by Cont and Das [12], who, with the advent of the roughness exponent as derived by Han and Schied [19], recently explored a critical inconsistency where, regardless of the true roughness of the instantaneous volatility of the model or data, the realised volatility would consistently be rough $R < 0.25$. Cont and Das [12] attributed it to estimation errors: from estimator errors and finite sample errors, to the errors in deriving estimations for unobservable instantaneous spot volatility itself. Therefore, other, more parsimonious and traditional models, such as diffusion-based stochastic volatility models (like OU-OU by Rogers [23]), which can explain market features just as well as rough volatility models, retain their convention in the world of financial modelling against rough volatility modelling.

The given problem set is solved in the Numerical Computations in Section 2; although it is mostly stand-alone, it is also highly relevant, as the results often conform to the Literature Review in Section 1, where the history of rough volatility is further explored in more detail, and some of the methodologies are replicated or adapted for our key results that are developed for the Original Contribution in Section 3 for estimator analysis.

Chapter 1

Literature Review

1.1 Foundations of Volatility Modelling

This section examines the foundational volatility models that preceded rough volatility, analysing their theoretical contributions and empirical successes, as well as the critical limitations that may have directly motivated the development of the rough volatility paradigm in Section 1.2.

1.1.1 ARCH/GARCH Models

The first formal econometric model to capture volatility clustering, the tendency for large (absolute) returns to be followed by large moves, was Engle's [14] autoregressive conditional heteroscedastic (ARCH) model, which imposes an autoregressive structure on the conditional variance, in contrast to earlier AR models that targeted only the conditional mean:

$$\sigma_t^2 = \alpha_0 + \sum_{i=1}^{q} \alpha_i \epsilon_{t-i}^2$$

Previously, traditional models assumed constant variance over time. Engle's [14] key innovation was to let the variance depend on past squared shocks, directly capturing volatility clustering and conditional heteroscedasticity. He demonstrated the model's practical value by applying it to U.K. inflation data, showing that estimated variance rose substantially during the turbulent 1970s, a period marked by high and volatile inflation, oil shocks, and macroeconomic instability.

This paved the way for other models such as generalised ARCH (GARCH), where Bollerslev [5] extended Engle's ARCH model [14] by introducing an autoregressive variance term to its structure:

$$\sigma_t^2 = \alpha_0 + \sum_{i=1}^{q} \alpha_i \epsilon_{t-i}^2 + \sum_{j=1}^{p} \beta_j \sigma_{t-j}^2$$

The addition of past volatility dependence helps model volatility persistence, which refers to the tendency for high or low volatility to persist for an extended period. As a result, Bollerslev's model [5] is also more parsimonious, as it requires fewer parameters due to its efficiency. Both ARCH/GARCH models are still widely used due to their simplicity and tractability.

1.1.2 Stochastic Volatility Models

In the continuous-time framework, the well-known Black-Scholes model [4] was a key stepping stone in option pricing theory.

The model assumes the underlying asset price S_t follows geometric Brownian motion with constant volatility σ and risk-free rate r:

$$dS_t = rS_t dt + \sigma S_t dW_t$$

Its closed-form solution for European options revolutionised derivatives markets by providing an analytical pricing formula. However, the model's assumption of constant volatility contradicts empirical observations of time-varying volatility and the volatility smile, which is the systematic variation in implied volatility across strikes. This limitation impedes accurate market calibration, particularly for options far from ATM or with short maturities.

Heston [20] adapts the model by amending volatility to be a correlated, mean-reverting stochastic process, rather than a simple constant:

$$dS_t = \mu S_t dt + \sqrt{V_t} S_t dW_t^S$$

$$dV_t = \kappa(\theta - V_t)dt + \sigma\sqrt{V_t} dW_t^V, \qquad dW_t^S dW_t^V = \rho dt$$

As such, Heston's model [20] captures volatility clustering, mean reversion, and fat tails. The introduction of the correlation parameter, unlike ARCH [14]/GARCH [5], which lacks asymmetry due to ϵ_{t-1}^2, can produce skewed implied volatility smiles and capture stylised leverage effects [11], where volatility tends to increase following negative returns, and to decrease or remain lower following positive returns.

1.1.3 Key Limitations & Motivation

Despite their contributions, these early models share critical limitations with respect to stylised market features [11], for example, their volatility autocorrelation decays geometrically for GARCH [5] and exponentially for Heston [20], contradicting the stylised power-law decay with $C(k) \sim k^{-\gamma}$ as observed in market data. Additionally, the Heston model also fails to replicate the market's ATM skew term structure as $\tau \to 0$, Heston's skew approaches a constant, whereas empirical skews diverge as $\tau^{H-1/2}$, as shown by Gatheral et al. [16]. These limitations motivate the rough-volatility paradigm: if volatility is neither constant nor well-captured by Markovian volatility models (e.g., Heston [20] /GARCH [5]), a natural next step might be to model it as a rough process; indeed, the project explores its empirical successes that led to its coming to fame and challenges that otherwise undermine the whole hypothesis.

1.2 Emergence of Rough Volatility

With prior models being unable to reproduce some of the stylised market features [11], the new paradigm of rough volatility was investigated and found to have some justification, leading to an alternative branch of rough models that were competitive and more consistent with stylised market features than conventional models.

1.2.1 The Hurst Exponent: A Measure of Roughness

The Hurst exponent (H) is a fundamental parameter characterising the roughness or smoothness of a stochastic process. Originating from Mandelbrot and Van Ness's [22] formalisation of fractional Brownian motion (fBm), it quantifies the path regularity and correlation structure of time series:

- $H = 0.5$: Uncorrelated increments (standard Brownian motion)
- $H > 0.5$: Positively correlated increments (long memory, smoother paths)
- $H < 0.5$: Negatively correlated increments (anti-persistence, rough paths)

Mathematically, for fBm $B^H(t)$, the covariance structure exhibits H-scaling:

$$\mathbb{E}[B^H(t)B^H(s)] = \frac{1}{2}\left(|t|^{2H} + |s|^{2H} - |t-s|^{2H}\right), \quad 0 < H < 1.$$

In volatility modelling, H determines the local regularity: lower H values ($H < 0.5$) indicate jagged, rapidly mean-reverting volatility paths characteristic of rough volatility. See figure 2.1 for visualisation.

1.2.2 Empirical Discovery:"Volatility is Rough"

Gatheral, Jaisson, and Rosenbaum's breakthrough paper, *"Volatility is Rough"*, provided the first systematic evidence that log-volatility behaves like a rough process (H $\approx$ 0.1), diverging from traditional models with smoother paths (H $\approx$ 0.5).

In more detail, from the high-frequency realised volatility data, they constructed a time series of log-volatility. For each scale Δ (time-interval) and moment order 2, they measured:

$$m(2, \Delta) = \mathbb{E}[|\Delta \log \sigma_t|^2]$$

Where $\Delta \log \sigma_t = \log \sigma_{t+\Delta} - \log \sigma_t$.

Then, from the fBM property of self-similarity ($B^H_{\Delta t} \sim \Delta^H B^H_t$), it is expected that:

$$m(2, \Delta) \propto \Delta^{2H}$$

Naturally, to obtain H, estimation via Log–Log Regression leads to a $\log m(2, \Delta)$ vs. $\log \Delta$ plot, forming a straight line with slope $\approx 2H$, hence, $H \approx$ slope/2.

They applied this methodology across various q, Δ, and asset classes, consistently finding $H \approx 0.1$. This key result was a motivator to explore this rough volatility paradigm further, as if it were the case that this roughness was a market feature, it then stands to reason that if a model integrates this framework, it could potentially be more congruent with other market features and, as a result, perform better than non-rough models. Consequently, Gatheral et al. [16] proposed the Rough Fractional Stochastic Volatility (RFSV) model, where log-volatility evolves as an fBM with Hurst exponent $H < 0.5$:

$$\log \sigma_{t+\Delta} - \log \sigma_t = \nu(W^H_{t+\Delta} - W^H_t), \quad \sigma_t = \sigma_0 \exp(\nu W^H_t + \beta)$$

where ν is a positive constant that scales the vol-of-vol, σ_0 is the baseline volatility, and the drift correction $\beta = -\frac{1}{2}\nu^2 t^{2H}$ ensures $\mathbb{E}[\sigma_t] = \sigma_0$.

However, this anti-persistent memory of rough volatility went against stylised views on long memory, particularly the Fractional Stochastic Volatility (FSV) framework of Comte and Renault [10], which required $H > 0.5$ for long-memory persistence. Gatheral et al. [16] argued that the reason this was consistent with the rough model was that statistical tools at the time conflated roughness with persistence; in fact, they demonstrated that statistical tests spuriously detected long memory in their own rough model.

As demonstrated by Fukasawa [15], such a volatility model produces an ATM volatility skew of the form $\psi(\tau) \sim \tau^{H-1/2}$, coincidently, Gatheral et al., [16] found the S&P ATM skews to take a similar power-law form $\psi(\tau) \sim \tau^{-0.4}$. With a rough Hurst exponent $H = 0.1$, both forms become equal; not only does this validate their findings for $H = 0.1$, but it also improves upon other models, such as Heston [20], Hull and White [21], and SABR [18], which do not conform to the power-law form. In their paper, they also compared their RFSV model with autoregressive models, such as AR(5), AR(10), and HAR(3), for forecasting purposes. The results showed that RFSV forecast consistently outperforms the AR and HAR forecasts, especially at longer horizons, while also being more parsimonious since it only requires H to forecast the log-variance, as Gatheral et al. [16] suggest, this does not necessarily mean their model is superior, rather, being at least competitive to other predictors whilst being simpler.

With this, rough models also retain other stylised market features [11]:

- Volatility Clustering: rough paths exhibit strong short-scale dependence and clustering, without displaying true long memory.
- Heavy Tails: log-normal volatility $\sigma_t = e^{X_t}$ conditions for positive returns and fat-tails.
- Leverage Effect: rough volatility models can utilise the correlative structure $dW_t^S dW_t^V = \rho dt$.

1.2.3 Pricing under Rough Volatility: success and failure

Building upon Gatheral et al.'s [16] findings, Bayer et al. [3] developed an arbitrage-free pricing model under rough volatility using their rough Bergomi model. They chose to model ATM implied volatilities at two specific days: a day with a flatter term structure, with (guessed) parameters $H = 0.07$, $\eta = 1.9$ and $\rho = -0.9$ and a day with short expirations, with (also guessed) parameters $H = 0.05$, $\eta = 2.3$ and $\rho = -0.9$.

As illustrated by Bayer et al. [3], despite the guessed parameters, the modelled ATM skews were indeed consistent with the market and Fukasawa's [15] theoretical $\psi(\tau) \sim \tau^{H-1/2}$. This directly gave credence to the idea that volatility is rough, especially and offers utility. Our RFSV model in Section 2.5 was consistent with their findings, as shown in Figure 2.4.

While rBergomi achieves exceptional SPX fits, its inconsistencies in VIX reveal deeper challenges. It was conjectured that continuous sample-path models could not jointly calibrate the SPX and VIX smiles; however, the model was unable to counter this conjecture, producing flat VIX smiles, which contradicted the expected upward-sloping market smiles. Additionally, its VVIX term structure is also inconsistent with the market. They suggest that the inconsistency might be "ascribed to a misspecified model, wrong market prices, or indeed both of these". They left this problem for future research to uncover. [3].

Despite this, subsequent rough-volatility models were developed, such as the rough Heston model [13], which was later built upon to form the Quadratic Rough Heston model [17], a model that addressed the conjecture against continuous sample-path models. By incorporating an affine structure and a price-feedback (Zumbach) effect, the Quadratic Rough Heston succeeded where rBergomi failed, providing a counterexample to the conjecture by achieving joint calibration of SPX and VIX smiles. Although this was a great advancement, achieving convincing fits to the broader VVIX surface within the rough volatility paradigm remains a more complex and less explored challenge. Subtly, one may question why less parsimonious models were needed to achieve empirical market features or why the problem of VVIX smiles remains unaddressed, especially if volatility was truly rough. These discrepancies may reflect limitations of the rough-volatility hypothesis or sensitivity to model overfitting.

1.3 Questioning the Rough Volatility Paradigm

A fact remains that the formative research conducted by Gatheral et al. [16] that led to the rough volatility paradigm relied on the estimations to be accurate; critically, realised variances themselves were used as a proxy for instantaneous spot volatility - the very data they used to determine that volatility is rough, is unobservable and an estimate itself, all their work hinges on this estimation to be sufficiently accurate. Consequently, Cont and Das [12] shed light on this, through their research they provide compelling evidence to revaluate the rough volatility paradigm as an artefact of estimation error, underpinning this are Han and Schied [19], who propose that roughness exponent (R) is separate from the Hurst exponent, allowing for a more formal read on true 'roughness'.

1.3.1 What Exactly is Roughness?

Han and Schied [19] distinguish the roughness exponent R from the Hurst exponent H. While the Hurst exponent reflects long-range dependence and autocorrelation structure, the roughness exponent measures the local 'jaggedness' of a continuous path and is independent of such statistical properties.

They do this through p-th variation limits along dyadic partitions:

$$\langle x \rangle_n^{(p)} := \sum_{k=0}^{2^n-1} \left| x\left((k+1)2^{-n}\right) - x\left(k2^{-n}\right) \right|^p$$

then x admits the roughness exponential if:

$$\lim_{n\to\infty} \langle x \rangle_n^{(p)} = \begin{cases} 0 & \text{if } p > \frac{1}{R}, \\ \infty & \text{if } p < \frac{1}{R}. \end{cases}$$

For example, standard Brownian motion has $R = 1/2$, and more generally, fractional Brownian motion satisfies $R = H$. Han and Schied [19] argue in favour of p-th variation on grounds of tractability, convergence, and the absence of any alternative measure.

The principal advantage of the exponent R is its model-free nature: it quantifies local irregularity without reference to underlying statistical assumptions. In particular, observing a Hurst estimate $H = 0.1$ does not imply that the path itself is as rough as $R = 0.1$. This distinction casts doubt on whether

Gatheral et al [16] truly measured roughness in the first place via their methods. Cont and Das [12] later leveraged this insight, constructing a roughness index based on p-th variation to test the rough volatility hypothesis empirically.

1.3.2 Roughness: Estimation Artefact

Considering that literature that had argued for fractional models with long-range dependence where $H > 0.5$ (Comte and Renault [10]), Cont and Das [12] investigated this conflict with rough models that had recently proposed the opposite with $H < 0.5$, in their research they argued that volatility may not be rough after all, but rather just an artefact caused by estimation error.

They begin this by highlighting the distinction that realised volatility is not directly observed; rather, as with the SPX Daily Realised Variance data, it is simply an estimation based on observed prices:

$$RV_n = \sum_{i=1}^{n} (\log S_{t_i} - \log S_{t_{i-1}})^2$$

They also mention the estimation error that may be prevalent due to discrete sampling, which can introduce downward biases, as well as pointing out how Gatheral et al.'s [16] Log–Log Regression method amplifies small errors. Together, they argue that this accumulation of errors creates enough noise that it is conflated with roughness by the very estimators that previously had empirically demonstrated roughness.

To test this, they employ a robust p-th variation estimator, which is parameter-free and avoids assumptions on noise distribution (unlike MLE or Method of Moments). Following Han and Schied [19], a similar roughness index is then derived:

$$\hat{H}^{\pi}_{L,K}(X) = \frac{1}{p^{\pi}_{L,K}(X)}$$

where $p^{\pi}_{L,K}(X)$ solves $W(L, K, \pi, p, T, X) = T$ for normalised p-th variations. This directly links scaling behaviour to Hölder regularity, which can be understood as a measure of a path's local roughness.

With this, they first ran stochastic-based models, such as where volatility was modelled via simple Brownian motion and then the OU-SV model. They

compared realised volatility with instantaneous volatility, the idea being that the instantaneous volatility represented the true underlying volatility structure and the realised variance served as an estimated analogue. For both models, the results were inconsistent, with realised volatility consistently being far rougher. Likewise, even replicating this with fOU they found similar results, the realised volatility was consistently far below with $H_{RV_n} \in [0.05, 0.30]$, yet $H_{true} \in [0.1, 0.8]$, demonstrating that roughness ($H < 0.5$) would almost always be perceived by the estimator regardless of the true roughness/smoothness, this is already a critical concern that suggests that realised volatility alone is not a sufficient proxy for actual volatility. [12].

They then tested their p-th variation estimator on AAPL and the S&P500 stock data using realised volatility. They obtained consistent results with Gatheral et al. [16], yielding $R \in [0.05, 0.25]$, which falls well within the rough regime. However, because their estimator measures a more formal roughness than Log–Log regression, the consistency only highlights that the estimators utilised were not reporting inaccurate findings, or that if they were, using just realised volatility is insufficient to discern them from one another, and once again does not make for an apt approximation for spot volatility for the underlying asset.

This critical discovery casts legitimate doubt on the rough volatility hypothesis - either if it exists, then current empirical evidence cannot distinguish it from noise-induced bias, or volatility truly was not rough in the first place, thus resorting back to traditional models where $H > 0.5$, and indeed, through 'Occam's Razor' they argue the usage of "diffusion-based stochastic volatility models which seem compatible with the empirical evidence but are far more tractable" - Cont and Das [12].

1.3.3 Alternative Theories and Models

This view for retaining conventional non-rough volatility models is strongly supported by Rogers [23], who demonstrates that the apparent roughness can be reproduced by much more parsimonious, Markovian multi-scale models. In particular, Rogers writes down and investigates an "OU-OU" specification:

$$dY_t = \sigma_Y \, dW'_t - \beta Y_t \, dt, \qquad dX_t = \sigma_X \, dW_t + \lambda(Y_t - X_t) \, dt,$$

in which a fast mean-reverting volatility factor X_t and slower Y_t interplay.

Rogers [23] demonstrates that this bivariate diffusion exhibits the same $\log m(q, \Delta)$ scaling observed in the rough-vol literature for the daily/medium lags of interest, while remaining Markovian, Gaussian, and straightforward to calibrate and filter. He also emphasises that high-frequency estimation error and microstructure effects can bias roughness estimators, so reproducing the empirical signature at the time-scales relevant for option pricing (days–months) with a tractable multi-factor diffusion argues in favour of strongly mean-reverting, fast–slow Markovian specifications rather than non-Markovian rough processes. These alternatives, therefore, avoid the computational and interpretational burdens of fractional models while providing comparable empirical fit. [23].

Another alternative line of thought is that volatility might not be just "rough" but actually multifractal. Instead of being described by a single Hurst exponent, multifractal models assume a whole range of scaling behaviours across different time scales. Empirical work, such as Brandi and Di Matteo's [6] investigation of multi-scaling and roughness, shows that multifractal features can coexist with, or even explain, the apparent roughness in financial markets. Wu, Muzy, and Bacry [24] introduced the "log S-fBM" model, which brings together both rough and multifractal features. Mathematically, the log S-fBM model defines log-volatility as:

$$\log \sigma_t = \mu + X_t,$$

where X_t is a stationary Gaussian process with Hurst parameter $H \in (0, 1/2)$. For $0 < H < 1/2$ the model behaves like a rough-volatility process; as $H \to 0$ it approaches the log-normal multifractal random measure.

In this framework, index volatilities exhibited $H \approx 0.1$, but individual stocks can appear even rougher, approaching the multifractal extreme. An additional benefit is that, unlike the variance of log-volatility (often used in rough volatility studies but shown to be unreliable), the log S-fBM framework highlights the "intermittency coefficient" λ^2, which is a product of H and ν^2, as a more stable and empirically robust quantity. Wu et al. [24] find that this coefficient takes nearly universal values across indices and across individual stocks, suggesting it may provide a more reliable empirical signature of volatility dynamics than estimations of realised volatilities.

Altogether, this suggests that volatility may lie on a spectrum between rough and multifractal regimes. While alternatives, such as Markovian multi-scale or multifractal volatility, provide value, this project will concentrate on fBM rough volatility models, as these remain the central focus of finance research.

Chapter 2

Numerical Computations

2.1 Deriving Estimator $\hat{H}$ for H

The approach used to obtain $\hat{H}$ here is the Method of Moments (MoM), which involves applying fundamental properties of fBM to the expected value, deriving a simplified formula, and then taking logs. To determine the nature of the estimator, Jensen's Inequality can be conveniently used.

Theoretically expected value of the q-th empirical absolute moment:

$$\frac{1}{n}\mathbb{E}(\Sigma_{i=1}^{n}|B_{i\Delta}^{H} - B_{(i-1)\Delta}^{H}|^{q}) \tag{2.1}$$

on $[0, T]$ for $q > 0$, where $\Delta = T/n$.

Recall the self-similarity property:

$$(B^H(at))_{t\geq 0} \stackrel{d}{=} a^H(B^H(t))_{t\geq 0},$$

Applying to the increment of fBM gives:

$$B_{i\Delta}^{H} - B_{(i-1)\Delta}^{H} \stackrel{d}{=} \Delta^H(B_t^H - B_{t-1}^H)$$

Applying the stationary increment property of fBM:

$$B_t^H - B_s^H \sim \mathcal{N}(0, (t-s)^{2H}) \implies \Delta^H(B_t^H - B_{t-1}^H) \stackrel{d}{=} \Delta^H Z_i \tag{2.2}$$

where $Z \sim \mathcal{N}(0, 1)$.

Substituting eq 2.2 back to eq **??** and simplifying:

$$\frac{1}{n}\,\mathbb{E}(\sum_{i=1}^{n}|B^H_{i\Delta}-B^H_{(i-1)\Delta}|^q)=\frac{1}{n}\,\mathbb{E}(\Delta^{qH}|Z_i|^q)\implies\Delta^{qH}K_q$$

Where $K_q=\mathbb{E}(|Z|^q)$

Let $\mathbb{E}[\hat{V_q}]=\Delta^{qH}K_q$, take logs and solve for H:

$$\hat{H}=\frac{1}{q}\frac{\log(\hat{V_q})-\log(K_q)}{\log(T)-\log(n)} \tag{2.3}$$

Now, to determine the nature of this estimator, $\hat{H}$, it is important to note that the logarithm is a concave function; this detail allows the application of Jensen's Inequality, applying this to $\log(\hat{V_q})$ from 2.3:

$$\mathbb{E}(\log(\hat{V_q}))\leq\log(\mathbb{E}[\hat{V_q}])=\log(\Delta^{qH}K_q) \tag{2.4}$$

Making H the subject, signs flip after dividing eq 2.4 through $\log\Delta$:

$$\begin{gathered}\mathbb{E}[\log\hat{V_q}]\leq\log\mathbb{E}[\hat{V_q}]=\log(\Delta^{qH}K_q)=qH\log\Delta+\log K_q,\\ \mathbb{E}[\hat{H}]=\frac{1}{q}\frac{\mathbb{E}[\log\hat{V_q}]-\log K_q}{\log\Delta}\;\geq\;\frac{1}{q}\frac{qH\log\Delta}{\log\Delta}=H\end{gathered} \tag{2.5}$$

Hence, the estimator, $\hat{H}$, for finite n systematically overestimates or is positively biased, implying that on average the estimated values for $\hat{H}$ will tend to be higher than the true value.

However, after adding and subtracting $qH\log(\Delta)$ in the numerator of eq 2.5:

$$\mathbb{E}[\hat{H}]=\frac{1}{q}\frac{(\mathbb{E}[\log(\hat{V_q})]-\log(\Delta^{qH}K_q))+qH\log\Delta}{\log(\Delta)}=H+\frac{1}{q}\frac{\mathbb{E}[\log(\hat{V_q})]-\log(\Delta^{qH}K_q)}{\log(\Delta)}$$

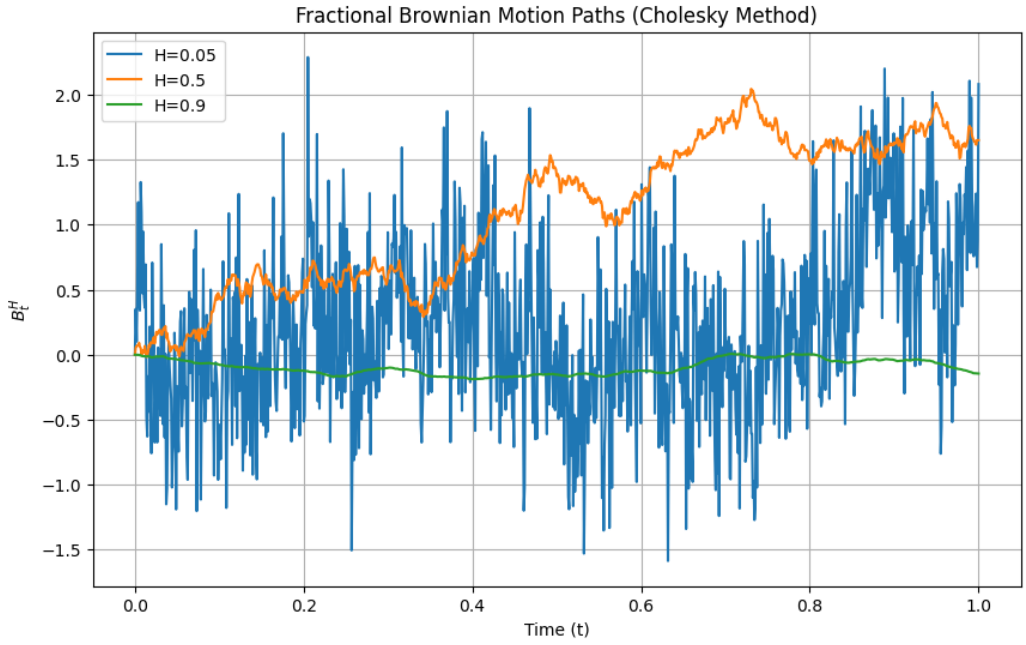
As $n\to\infty$, it implies $|\log(\Delta)|\to\inf$, meaning the estimator is asymptotically unbiased; this is because the bias term vanishes for larger n.

2.2 Simulation of fBM with Hurst coefficients

The Hurst exponent determines the roughness or how jagged the fBM path will be; it is expected that a low value $H<0.5$ will be rougher, whereas a

high value $H > 0.5$ will be smoother and $H = 0.5$ reflects standard BM.

Figure 2.1: fBM path evolution with different Hurst Coefficients



In Figure 2.1, the fBM paths were computed via Cholesky decomposition, which uses the covariance matrix aligned with $\gamma(k) = \frac{1}{2}(t^{2H} + s^{2H} - |t-s|^{2H})$.

It is worth noting that Cholesky requires strict positive definiteness to do $LL^T = \Sigma$, so if it has a path with 0, its first row/column is zero, making it only positive semidefinite ($x^T \Sigma x > 0$) and becomes invertible, so the Cholesky decomposition fails. Hence, as a workaround for this, since it cannot have 0 as its starting path, it shall always ignore the first term and insert it back afterwards for consistency.

From visual inspection, it confirms the expectation: for $H \in \{0.05, 0.50, 0.90\}$, comparatively, the path for $H = 0.90$ was smooth, for $H = 0.05$ it was very jagged, and for $H = 0.50$ its roughness was in-between the other two, as it is simply standard Brownian motion. Moreover, the smoothness of $H = 0.90$ suggests persistence, as it appears to remain close to its origin throughout with minimal divergent steps, which is characteristic of a long-memory process. In contrast, for $H = 0.05$, it displays anti-persistence due to its more

erratic and unpredictable 'rough' behaviour, characteristic of a no-memory process.

2.3 Computing Maximum Likelihood Estimator (MLE) for ν and H for a sample fBM path

In this subsection, the computation for the MLEs of ν and H using standard maximum-likelihood techniques for Gaussian models (also guided by Chang [8]); the idea is to utilise the log-joint density from the joint Gaussian log-likelihood to form a one-parameter negative log-likelihood function which can then be optimised to obtain $\hat{H}$ and then can recover $\hat{\nu}$. To improve numerical stability, we evaluate the likelihood via Cholesky factorisation using scaling. The sample path, 'fBMpath2.txt', is a fBM sample path of $[0, 1]$ at $n = 1024$ equidistant time points on $[0, 1]$ multiplied by an unknown constant ν.

Since B^H is a zero-mean Gaussian process with the covariance matrix of $\nu^2\Sigma(H)$, where $\Sigma(H)_{ij} = \frac{1}{2}(t_i^{2H} + t_j^{2H} - |t_i - t_j|^{2H})$, then it shall have a log-joint density:

$$\ell(X|H, \nu) = -\frac{1}{2}[n \log(2\pi) + \log\det(\nu^2\Sigma(H)) + X^T(\nu^2\Sigma(H))^{-1}X] \quad (2.6)$$

Now fix H and then maximise the log-joint density over ν. From this, obtain an estimator for ν:

$$\frac{\partial \ell}{\partial \nu} = 0 \implies \hat{\nu}^2 = \frac{1}{n}X^T\Sigma(H)^{-1}X$$

Plug back $\hat{\nu}^2$ in eq.2.6 make one-parameter profile negative log-likelihood:

$$\mathcal{L}(H) = \frac{n}{2}\log(\frac{1}{n}X^T\Sigma(H)^{-1}X) + \frac{1}{2}\log\det\Sigma(H)$$

Then the optimisation problem can be set up as follows:

$$\hat{H} = \arg\min_{H\in(0,1)} \mathcal{L}(H)$$

To compute numerically, we utilised a scaling factor $s = 1000$ to prevent the covariance from shrinking to zero:

$$X = sX_0, \quad \Sigma(H) = s^2\Sigma_0(H)$$

Thus, let L be the Cholesky factor: $LL^T = s^2\Sigma_0(H)$ and solve $Ly = sX_0$ to get $y = L^{-1}(sX_0)$.

Evaluate:

$$Q(H) = ||y||^2 = X^T\Sigma(H)^{-1}X, \quad \log\det(\Sigma(H)) = 2\Sigma_{i=1}^{n}\log L_{ii}$$

Compute

$$\mathcal{L}(H) = \frac{n}{2}\log(Q/n) + \Sigma_{i=1}^{n}\log L_{ii}$$

In order to recover ν, use $\hat{H}$ and solve $\hat{\nu} = \sqrt{\frac{1}{n}X^T\Sigma(\hat{H})^{-1}X} = \sqrt{\frac{Q(\hat{H})}{n}}$.

Using the method as outlined, these were the results for $\hat{H}$ and $\hat{\nu}$ obtained for 'fBMpath2.txt':

Table 2.1: Maximum Likelihood Estimates of H and ν for 'fBMpath2.txt'

Parameter	**MLE Estimate**
Hurst parameter H	0.0357
Variance ν	1.1597

2.4 Simulating RFSV model & Estimating H using V_t

In this subsection, we will be simulating a sample path of an RFSV model with data that contains sample paths of fBM of a given H and standard BM at $n = 16384$ equidistant time points on $[0, 1]$, generated using the same i.i.d. Normals, and an additional independent Brownian motion W. After simulation, we obtain estimates for the Hurst parameter, H, using spot variance, V_t, at different subwindows by using the previously discussed MLE methodology in section 2.3.

2.4.1 Simulating RFSV model

The RFSV model:

$$dS_t = S_t\sqrt{V_t}(\rho dB_t + \bar{\rho} dW_t)$$
$$V_t = V_0 e^{vB_t^H}$$

$\rho B + \bar{\rho} W$ and B are two correlated BMs with $\mathbb{E}((\rho B_t + \bar{\rho} W_t)B_t = \rho t$, where $\bar{\rho} = \sqrt{1-\rho^2}$ with (fixed) remaining model parameters $S_0 = 1$, $\nu = 1$, $\rho = -0.65$ and $V_0 = 0.1$.

To simulate this, apply Euler-Maruyama Discretisation to dS_t:

$$S_{t_{t+1}} = S_{t_i}(1 + \sqrt{V_{t_i}}(\rho \Delta B_i + \bar{\rho} \Delta W_i))$$

where $\Delta B_i = B_{t_{i+1}} - B_{t_i} \sim \mathcal{N}(0, \Delta t)$ and $\Delta W_i = W_{t_{i+1}} - W_{t_i} \sim \mathcal{N}(0, \Delta t)$.

Below are two simulation plots, the first using a fBM sample with a Hurst parameter of $H = 0.05$, and the second using $H = 0.10$.

Figure 2.2: RFSV Simulation of V_t and S_t with a sample fBM where $H = 0.05$; B_t^H - 'BHpath.05.txt', B_t - 'Bpath.05.txt', W_t - ' Wpath.05.txt'.

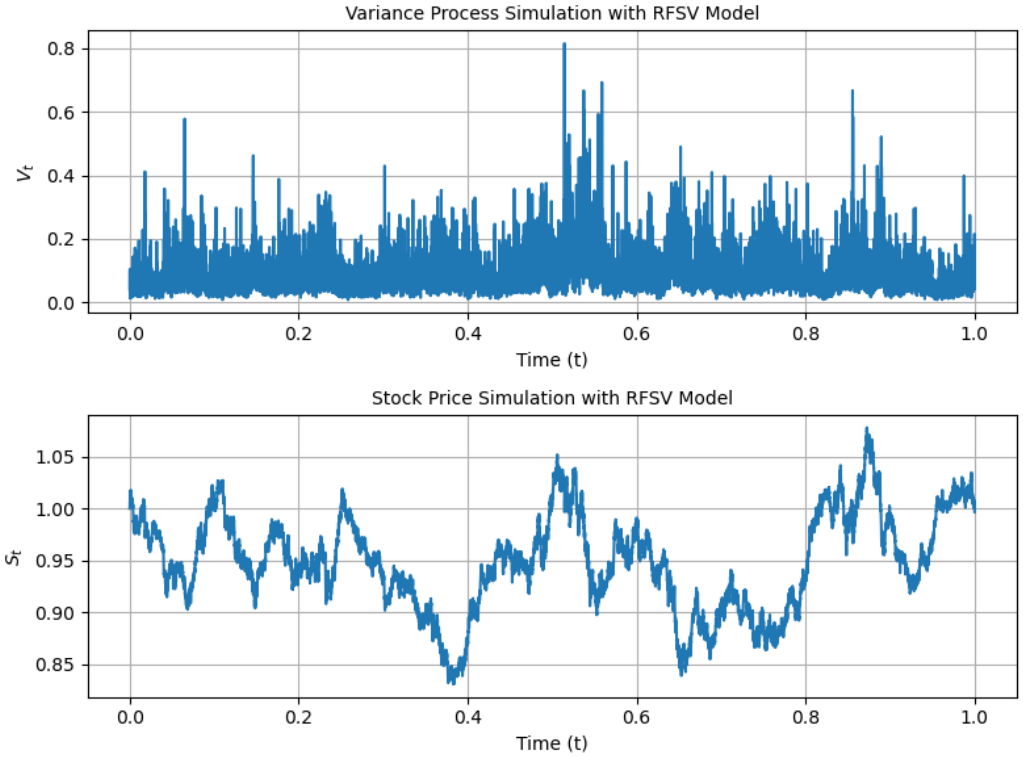
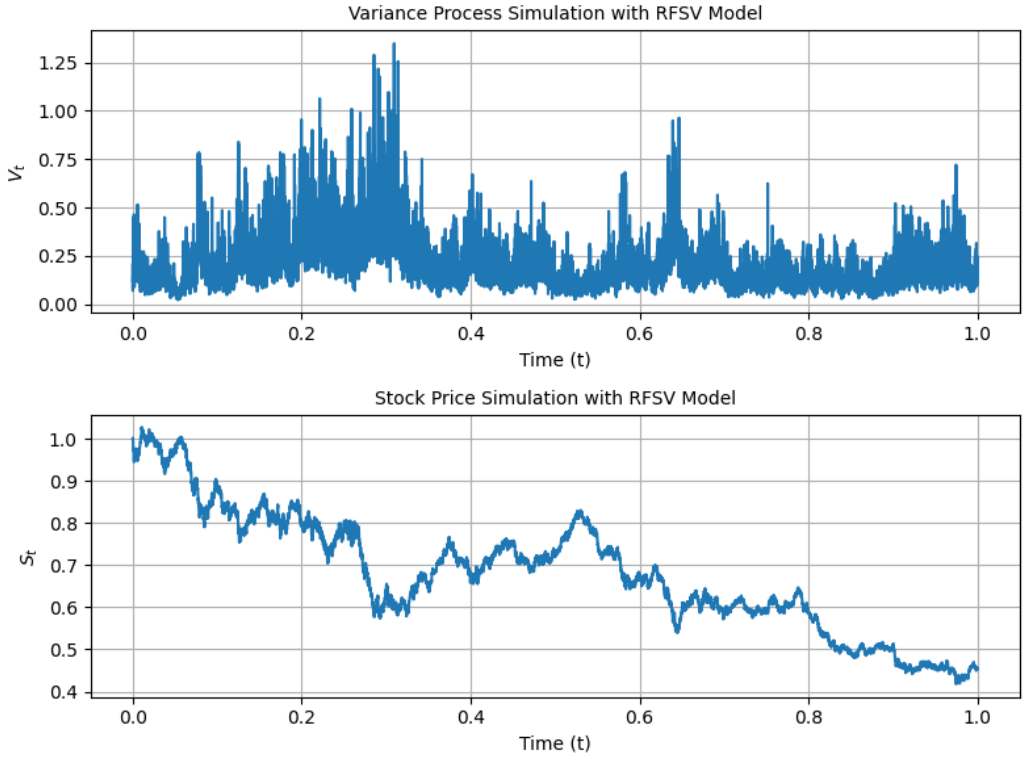


Figure 2.3: RFSV Simulation of V_t and S_t with a sample fBM where $H = 0.10$; B_t^H - 'BHpath.10.txt', B_t - 'Bpath.10.txt', W_t - ' Wpath.10.txt'.



2.4.2 Estimating H using V_t

Now to obtain an estimate for the spot variance, V_t, we will refer back to the dynamics of the RFSV and get an estimate for it through applying Itô's lemma and the second moment:

$$dS_t = S_t\sqrt{V_t}dM_t$$

where $M_t = \rho B_t + \bar{\rho}W_t$ is a Brownian motion. This holds because:

1. B_t and W_t are independent Brownian motions by construction
2. M_t is a continuous martingale with quadratic variation:

$$\langle M\rangle_t = \rho^2\langle B\rangle_t + \bar{\rho}^2\langle W\rangle_t = (\rho^2 + 1 - \rho^2)t = t$$

3. By Lévy's characterization, any continuous martingale with $\langle M \rangle_t = t$ is a Brownian motion

Then apply Itô's lemma to $f(S_t)$=logS_t to obtain:

$$\begin{aligned} d(\log S_t) &= f'(S_t)\, dS_t + \tfrac{1}{2} f''(S_t)\, d\langle S \rangle_t \\ &= \frac{1}{S_t}\, dS_t - \frac{1}{2S_t^2}\, d\langle S \rangle_t \\ &= \sqrt{V_t}\, dM_t - \tfrac{1}{2} V_t\, dt. \end{aligned}$$

where $d\langle S \rangle_t = S_t^2 V_t dt$

The increment over Δt suggests:

$$\Delta_i = \log S_{t_{i+1}} - \log S_{t_i} \approx \sqrt{V_{t_i}} \Delta M_i - \frac{1}{2} V_{t_i} \Delta t$$

with $\Delta M_i \sim \mathcal{N}(0, \Delta t)$

Applying the second moment:

$$\mathbb{E}[\Delta_i^2 | \mathcal{F}_{t_i}] = \mathbb{E}[(\sqrt{V_{t_i}} \Delta M_i)^2] + O(\Delta t^2) \approx V_{t_i} \Delta t$$

Hence, summing squared increments: $\Sigma(\Delta_{jm+k})^2 \approx V_{t_{jm}} m \Delta t$. Therefore, the formula used for the spot variance shall be:

$$V_{t_{jm}} \approx \frac{\Sigma(\Delta_{jm+k})^2}{m \Delta t}$$

where m will be the subwindow number.

Since the RFSV model assumes $\log V_t = \log V_0 + \nu B_t^H$, the centred log-spot variance inherits the covariance structure of the fBM $\log V_t - \mathbb{E}[\log V_t] = \nu B_t^H$, thus allowing to use the same methodology outlined in Section 2.3 to compute the MLE of H.

Below are the results obtained from the RFSV model simulations with $m \in \{16, 32, 64\}$.

Table 2.2: H estimation via MLE for true H values and block sizes m.

True H	**Estimated** $\hat{H}$		
	$m = 16$	$m = 32$	$m = 64$
0.05	0.0358	0.0581	0.1364
0.10	0.0637	0.1086	0.1736

From the results, $m = 32$ yields the most optimal solution with the lowest errors for both H's; it implies that $m = 16$ under-smooths in the sense that it magnifies noise, resulting in lower H, whereas $m = 64$ over-smooths in the sense that it underplays noise by averaging it out. Moreover, this also highlights a calibration problem when choosing block sizes for estimators, as finding the optimal block size can be challenging, especially considering that an optimal block size for one estimator/model may not be optimal for different estimators/models.

Additionally, for $H = 0.5$ (standard Brownian motion), we did the same procedure as before, but this time computed $\hat{H}$ 100 times with a 95% confidence interval using our own B and W paths at $n = 65536$ with $m = 512$.

Table 2.3: Estimation results for standard Brownian motion ($H = 0.5$) with $m = 512$, $n = 65536$.

Statistic	**Value**
True H	0.5000
Mean estimated $\hat{H}$	0.3622
Standard deviation	0.0557
95% Confidence Interval	[0.3510, 0.3733]
Bias ($\hat{H} - H$)	-0.1378

These results demonstrate that the estimator systematically underestimates the Hurst parameter with a negative bias. Moreover, although the 95% confidence interval is precise due to the small standard deviation, it does not contain the true value of $H = 0.5$. The likely culprit for this is the choice of block size $m = 512$, which causes over-smoothing that leads to inaccurate results.

2.5 Monte Carlo Simulation of Volatility Smile

In this subsection, we will use Monte Carlo to simulate a volatility smile with $H = 0.1$, $\nu = 1$, $V_0 = 0.04$,$\rho = -0.65$, with log-moneyness $x = \frac{K}{S_0}$ ranging from -0.5 to 0.5.

To achieve this, we employed the Riemann-Liouville (similar ideas to Bayer et al.'s Voltera Kernel [3]) representation of fractional Brownian motion, which is more efficient than the Cholesky method, and then discretised it. Through Itô isometry:

$$X_t = \int_0^t (t-s)^{H-\frac{1}{2}}\, dW_s \implies \mathrm{Var}(X_t) = \int_0^t (t-s)^{2H-1} ds = \frac{t^{2H}}{2H}. \quad (2.7)$$

To obtain the correct variance $\mathrm{Var}(B_t^H) = t^{2H}$, we multiply 2.7 by $\sqrt{2H}$:

$$B_t^H = \sqrt{2H} \int_0^t (t-s)^{H-\frac{1}{2}}\, dW_s.$$

On a uniform grid $t_i = i\Delta t$, this can be approximated by

$$B_{t_i}^H \approx \sqrt{2H} \sum_{k=0}^{i-1} \big((i-k)\Delta t\big)^{H-\frac{1}{2}}\, \Delta W_k,$$

which corresponds to the simple matrix form

$$B^H \;\approx\; L\,\Delta W,$$

where the convolution matrix is given by

$$L[i,k] = \big((i-k)\Delta t\big)^{H-\frac{1}{2}}, \quad 0 \le k < i.$$

Next apply Itô correction as $\mathbb{E}[V_t] = V_0 e^{\frac{1}{2}\nu^2 t^{2H}}$ grows exponentially, hence the process will be:

$$V_t = V_0 \exp(\nu B_t^H - \frac{1}{2}\nu^2 t^{2H}) \quad (2.8)$$

where $-\frac{1}{2}\nu^2 t^{2H}$ is the correction allowing for a stable expectation, $\mathbb{E}[V_t] = V_0$.

We implement the simulation using a log-transform for numerical stability and apply the Itô correction from eq. 2.8 to ensure $\mathbb{E}[V_t] = V_0$. For each batch, we generate four antithetic paths using antithetic variables $\pm\Delta B_i$, $\pm\Delta W_i$ for each S_t path:

1. Generate fractional Brownian motions:

$$B_t^{H,(1)} = L \cdot dB$$
$$B_t^{H,(2)} = L \cdot (-dB)$$

2. Apply Itô correction to volatility:

$$V_t^{(1)} = V_0 \exp\left(\nu B_t^{H,(1)} - \frac{1}{2}\nu^2 t^{2H}\right)$$
$$V_t^{(2)} = V_0 \exp\left(\nu B_t^{H,(2)} - \frac{1}{2}\nu^2 t^{2H}\right)$$

3. Simulate log-prices using antithetic Brownian increments:

$$d\log S_t^{(1)} = -\frac{1}{2}V_t^{(1)}dt + \sqrt{V_t^{(1)}}(\rho dB_t + \bar{\rho}dW_t)$$
$$d\log S_t^{(2)} = -\frac{1}{2}V_t^{(2)}dt + \sqrt{V_t^{(2)}}(-\rho dB_t + \bar{\rho}dW_t)$$
$$d\log S_t^{(3)} = -\frac{1}{2}V_t^{(1)}dt + \sqrt{V_t^{(1)}}(\rho dB_t - \bar{\rho}dW_t)$$
$$d\log S_t^{(4)} = -\frac{1}{2}V_t^{(2)}dt + \sqrt{V_t^{(2)}}(-\rho dB_t - \bar{\rho}dW_t)$$

Discretising the log-price SDE gives:

$$\log S_{t_{i+1}}^{(1)} = \log S_{t_i}^{(1)} - \frac{1}{2}V_{t_i}^{(1)}\Delta t + \sqrt{V_{t_i}^{(1)}}(\rho\Delta B_i + \bar{\rho}\Delta W_i)$$
$$\log S_{t_{i+1}}^{(2)} = \log S_{t_i}^{(2)} - \frac{1}{2}V_{t_i}^{(2)}\Delta t + \sqrt{V_{t_i}^{(2)}}(-\rho\Delta B_i + \bar{\rho}\Delta W_i)$$
$$\log S_{t_{i+1}}^{(3)} = \log S_{t_i}^{(3)} - \frac{1}{2}V_{t_i}^{(1)}\Delta t + \sqrt{V_{t_i}^{(1)}}(\rho\Delta B_i - \bar{\rho}\Delta W_i)$$
$$\log S_{t_{i+1}}^{(4)} = \log S_{t_i}^{(4)} - \frac{1}{2}V_{t_i}^{(2)}\Delta t + \sqrt{V_{t_i}^{(2)}}(-\rho\Delta B_i - \bar{\rho}\Delta W_i)$$

where $\Delta B_i \sim \mathcal{N}(0, \Delta t)$, $\Delta W_i \sim \mathcal{N}(0, \Delta t)$, and final stock prices are obtained via exponentiation: $S_T = \exp(\log S_T)$.

Applying the Law of Large Numbers and Monte Carlo integration by averaging over all generated paths (4 paths per batch, M batches), the expected payoff is:

$$\mathbb{E}[X] \approx \lim_{n\to\infty} \frac{1}{n}\sum_{i=1}^{n} X_i \quad \text{(Law of Large Numbers)}$$

$$\implies C \approx e^{-rT} \cdot \frac{1}{N}\sum_{i=1}^{N} \max(S_T^{(i)} - K, 0)$$

With $N = 4M$ (total paths), and r=0:

$$\implies C(K) \approx \frac{1}{4M}\sum_{j=1}^{M}\sum_{k=1}^{4} \max\left(S_T^{(j,k)} - K, 0\right)$$

Next, to obtain implied volatility, the computed Monte Carlo prices will be treated as synthetic option prices, and they will be equated with the Black-Scholes formula to solve for the unknown volatility:

$$C_{BS}(\sigma) = C_{MC}$$

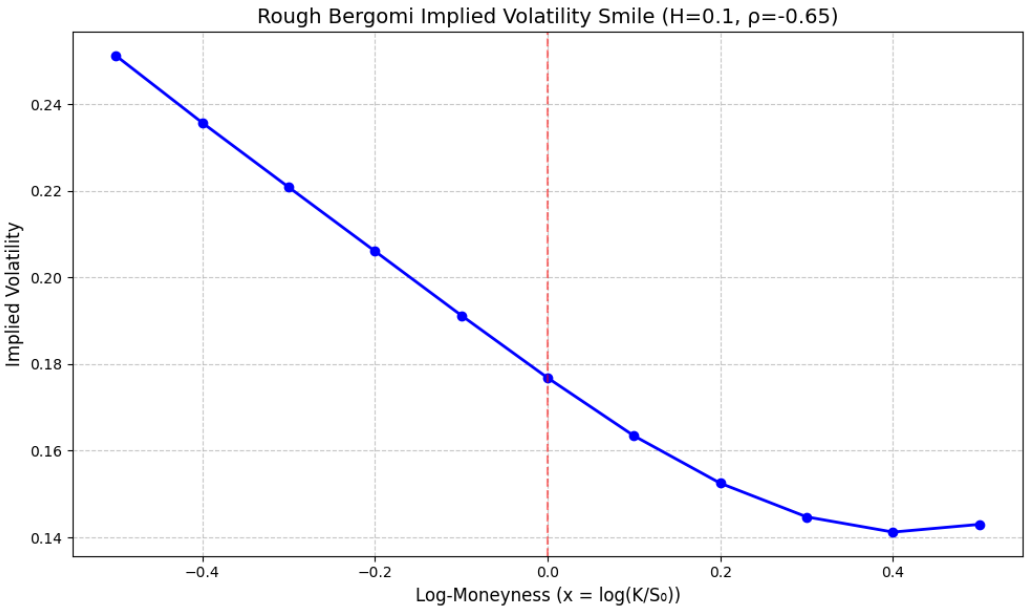
used Brent's root-finding algorithm to solve [7].

Below are the results: tabulation and plot of the volatility smile with $M = 100,000$ simulations and $N = 252$ time steps.

Table 2.4: Implied Volatility Smile Plot: RFSV ($H = 0.1$, $\rho = -0.65$, $V_0 = 0.04$)

Log-Moneyness (x)	**Strike (K)**	**Call Price**	**Implied Volatility**
-0.5	0.6065	0.3954	0.2511
-0.4	0.6703	0.3334	0.2357
-0.3	0.7408	0.2669	0.2208
-0.2	0.8187	0.1976	0.2060
-0.1	0.9048	0.1299	0.1912
0.0	1.0000	0.0707	0.1768
0.1	1.1052	0.0287	0.1634
0.2	1.2214	0.0077	0.1525
0.3	1.3499	0.0013	0.1447
0.4	1.4918	0.0002	0.1412
0.5	1.6487	0.0000	0.1430

Figure 2.4: Plot of implied volatility smile for RFSV ($H = 0.1$, $\rho = -0.65$, $V_0 = 0.04$)



Analysis of results: A strong leverage effect is evident, as indicated by the significant negative skew, where OTM puts have greater implied volatilities than OTM calls, with a correlation coefficient of $\rho = -0.65$. A steep decline is observed on the negative log-moneyness side, which is consistent with a rough $H = 0.10$. The smile's general structure exhibits left tail (puts) concavity, which is attributed to market crash fear, while the right tail (calls) displays convexity, resulting from the leverage effect suppressing upside volatility.

2.6 Estimating H and ν using real-world data

In this subsection, we will use SPX 3-year 1min daily realised variance, which has been computed from the sum of squares of 1min SPX log returns from 2nd Feb 2020 to 24th Feb 2023, to estimate H and ν, however, we will also estimate H and ν for the final year and analyse the results.

The methodology to be utilised will be as outlined in Section 2.4.2, where V_t can be used to estimate H and thereby also ν.

Table 2.5: MLE Estimates of H and ν from Daily realised Variance Data

Parameter	**Full Sample**	**Final Year Only**
Hurst parameter H	0.2098	0.1366
Vol-of-Vol ν	2.2742	1.3171

Examining the raw data, the market volatility became rougher as the Hurst parameter decreased in the final year; however, despite the increase in roughness, ν decreased slightly, indicating that the magnitude of these volatility fluctuations decreased. Moreover, the fact that $H < 0.5$ in both cases lends some credence to the rough volatility hypothesis.

Chapter 3

Original Contribution

3.1 Analysis of Estimators

The empirical validation of rough volatility hinges critically on accurate estimation of the Hurst exponent H. Estimators applied in this project are diverse and span a range of methodological assumptions. First, Log–Log Regression, formative to the rough-volatility literature (as implemented by Gatheral [16]). Second, the MLE is included as a fully parametric benchmark, relying explicitly on fGN, and is often used as a parametric benchmark; under correct specification and standard regularity conditions, the MLE is asymptotically efficient, attaining the Cramér–Rao lower bound. Third, we incorporate a Wavelet-based estimator, chosen for its non-parametric nature and robustness to noise and multi-scale features. Lastly, the q-th order Power Variation, grounded in the method of moments, offers a semi-parametric alternative ($q = 2$) that estimates roughness via scaling laws in absolute increments. Hence, to evaluate the performance of roughness estimators, we use four benchmark models. The baseline is fBM, chosen for its simplicity and clean scaling properties as a canonical fractional process. Alongside it, we include the RFSV model, inspired by Gatheral et al. [16], which extends fBM by introducing stochastic log-volatility dynamics that are more representative of financial markets. These two models serve as controlled environments where fractional Gaussianity (fGN) and ideal scaling laws are expected to hold - settings in which parametric and semi-parametric estimators should, in theory, perform well. To further test these estimators, we introduce two additional models: the 'Contaminated fBM' (C-fBM) and the fractional Ornstein–Uhlenbeck (fOU) process. These are designed to break

idealised assumptions and reflect market-like conditions. Lastly, after evaluating the strengths and weaknesses of each parameter, we will test them using market data to determine if any trends emerge.

3.1.1 Log–Log Regression: The Formative Approach

The Log–Log Regression method, as introduced by Gatheral et al. [16], is an intuitive approach for estimating the Hurst exponent H. The method relies on the scaling property of fractional Brownian motion increments: for any lag k,

$$\mathbb{E}[|X_{t+k} - X_t|^q] \propto k^{qH}.$$

Taking logarithms yields the linear relation

$$\log \mathbb{E}[|X_{t+k} - X_t|^q] = qH \log k + C,$$

where C is a constant. In this work, we fix $q = 2$, corresponding to variance scaling, consistent with Gatheral et al. [16]. The empirical analogue is

$$M_2(k) = \frac{1}{n-k} \sum_{i=1}^{n-k} |X_{i+k} - X_i|^2,$$

and the estimator is obtained by regressing $\log M_2(k)$ against $\log k$. The slope $\hat{\beta}$ of this regression satisfies $\hat{\beta} \approx 2H$, so the Hurst exponent is estimated as

$$\hat{H}_{\text{LLR}} = \frac{\hat{\beta}}{2}.$$

In practice, lags k are chosen on a dyadic grid ($k = 2^j$), ensuring good scale separation and numerical stability. This estimator is non-parametric, as it does not rely on Gaussianity or any underlying distribution; therefore, it remains valid under heavy-tailed or skewed increments. This is why Gatheral et al. highlighted its appeal for financial volatility data.

3.1.2 Maximum Likelihood Estimation: The 'Gold Standard'

As implemented in Section 2.3, Maximum Likelihood Estimation (MLE) offers a theoretically rigorous framework for Hurst exponent estimation. Unlike

the non-parametric log–log approach, this method is parametric, fully exploiting the Gaussian structure of fractional Brownian motion (fBM) through exact likelihood computation; moreover, its convergence rate is $O(n^{-1/2})$, superior to Log–Log Regression's $O(n^{-1/3})$ rate. One of the significant theoretical advantages of MLE is that when the Gaussian assumption holds, it achieves the Cramér–Rao lower bound, making it the most statistically efficient estimator possible.

The MLE is defined as the maximiser $\hat{H}_{MLE} = \arg\max_H \ell(H; \mathbf{X})$. Under standard regularity conditions, the MLE satisfies:

$$\sqrt{n}(\hat{H}_{MLE} - H_0) \xrightarrow{d} \mathcal{N}\big(0, \mathcal{I}^{-1}(H_0)\big)\,,$$

where the Fisher information is

$$\mathcal{I}(H) = \tfrac{1}{2}\,\mathrm{tr}\left(\Sigma(H)^{-1}\frac{\partial\Sigma(H)}{\partial H}\,\Sigma(H)^{-1}\frac{\partial\Sigma(H)}{\partial H}\right).$$

Although we do not implement this for simplicity, a key computational insight from Chang [8] recognises that when the likelihood is written for the increments of fBM, the covariance is Toeplitz, enabling $O(n^2)$ time via the Levinson–Durbin algorithm, compared with the standard $O(n^3)$ Cholesky approach. This efficiency makes MLE more feasible for moderately large samples.

Despite its theoretical appeal, MLE faces several practical limitations:

- The Gaussianity assumption may not hold, as real volatility increments often exhibit heavy tails and skewness.
- Financial volatility and other empirical data often display multi-scale behaviour beyond the pure fBM framework [6].
- Even with Toeplitz structure, computation remains more demanding than wavelet-based or MoM estimators, especially for very large samples.
- Performance can deteriorate under model misspecification or in the presence of non-stationarities.

3.1.3 Wavelet Estimation: Noise-Robust Multi-scale Analysis

Wavelet-based methods [1] provide a principled, multi-scale approach for estimating roughness and long-memory parameters in time series. Unlike classical estimators such as log–log regression [16] or maximum likelihood [8], the Wavelet transform decomposes the signal across dyadic scales, offering both time and frequency localisation. This enables the separation of persistent low-frequency dynamics from high-frequency microstructure noise.

Formally, the Wavelet transform represents a process $\{X_t\}$ as

$$X_t = \sum_{j=-\infty}^{J} \sum_{k\in\mathbb{Z}} d_X(j,k)\, \psi_{j,k}(t),$$

where $\psi_{j,k}(t) = 2^{-j/2}\psi(2^{-j}t - k)$ are scaled and shifted copies of a mother wavelet ψ, and $d_X(j,k)$ are the detail coefficients capturing local variations at scale j. For fractional processes, these coefficients satisfy the scaling law

$$\mathbb{E}[|d_X(j,\cdot)|^2] \propto 2^{j(2H+1)},$$

which implies a linear relation between scale and the logarithm of empirical energies:

$$\log_2 S_2(j) = \alpha + \beta j + \varepsilon_j, \qquad S_2(j) = \frac{1}{n_j}\sum_k |d_X(j,k)|^2.$$

We then estimate the Hurst parameter H by performing a weighted least squares regression of $\log_2 S_2(j)$ on the scale index j. The scale index is defined as

$$j = -\log_2(n_j),$$

and the Hurst exponent is estimated from the slope $\hat{\beta}$ as

$$\hat{H}_{\text{wav}} = \tfrac{1}{2}(\hat{\beta} - 1).$$

This regression-based procedure is non-parametric, relying solely on the scaling property of the wavelet coefficients rather than assuming a specific distributional form for the underlying process.

Weights $w_j^2 \propto n_j$ are used so that scales with more coefficients receive a higher weight. An appropriate range of scales $[j_1, j_2]$ is typically selected to

mitigate boundary artefacts and reduce the influence of high-frequency noise. In our implementation, we employ the pyramid algorithm, which yields $O(n)$ computational complexity. This algorithm utilises the Daubechies-3 basis (db3) mother wavelet (as used by Arby et al. [1]), whose three vanishing moments strike a balance between filtering low-frequency trends and preserving time localisation.

This estimator is favoured for its robustness and interpretability:

- Fine-scale noise is isolated, while coarser scales capture long-memory structure.
- Haar and other compactly supported wavelets naturally attenuate jumps and discontinuities.
- The fast wavelet transform ensures scalability to large datasets.
- No Gaussianity or specific distributional assumptions are required.

On the other hand, performance can be sensitive to the choice of mother wavelet and to low sample sizes, as well as the subwindow, which limits the effective number of scales available; therefore, calibration is paramount for this estimator.

3.1.4 q-th Order Power Variation: a parametric alternative

The PV estimator directly adapts from the Method of Moments framework discussed in Section 2.1 and becomes equivalent to Log–Log regression if it is multi-scaled. In that setting, the Hurst exponent is identified via the scaling of moments of increments. Specifically, for a fractional Brownian motion B^H, we have:

$$\mathbb{E}\left[\left|B^H_{i\Delta} - B^H_{(i-1)\Delta}\right|^q\right] = \Delta^{qH} \cdot \mathbb{E}[|Z|^q] = \Delta^{qH} K_q,$$

where $Z \sim \mathcal{N}(0,1)$ and

$$K_q = 2^{q/2} \cdot \frac{\Gamma\left(\frac{q+1}{2}\right)}{\sqrt{\pi}}.$$

In our implementation, we fix $q = 2$. In this case,

$$K_2 = \mathbb{E}[|Z|^2] = \mathbb{E}[Z^2] = 1,$$

So the Gaussian normalising constant disappears, and the estimator simplifies to:

Given discrete observations $X_0, \ldots, X_n$ over a horizon T with mesh size $\Delta = T/n$, We compute the quadratic power variation:

$$V_2 = \frac{1}{n} \sum_{i=1}^{n} \left| X_i - X_{i-1} \right|^2.$$

The estimator of the Hurst exponent is then

$$\hat{H}_{\mathrm{PV}} = \frac{1}{2} \cdot \frac{\log V_2}{\log \Delta}.$$

Keeping the scale single, this corresponds to the semi-parametric form of the estimator: the constant K_q is absorbed into the intercept, so no distributional assumption beyond the scaling property is required. While this form sacrifices some efficiency relative to the fully parametric version, it is more robust to deviations from Gaussianity, such as jumps, noise, or heavy tails, and remains sensitive to the self-similar scaling that characterises fBM.

3.2 Models

The models utilised shall be fBM (derived in section 2.2) for its simplicity as a fractional model, providing a controlled environment where fGN and scaling principles are exactly obeyed, as well as the slightly more complex RFSV model (derived in section 2.4) with $\nu = 1.5,\ \rho = -0.65,\ V_0 = 0.04,\ S_0 = 1$, which introduces realistic rough stochastic volatility dynamics. These models serve as complementary baselines: fBM to test estimator behaviour under idealised fractional assumptions, and RFSV to assess robustness in a more market-like, heterogeneous volatility setting. It is expected that parametric or semi-parametric estimators that rely on these properties will perform worse compared to non-parametric estimators. Alongside fBM and RFSV, we have decided to add C-fBM, as well as fOU, for their diametric properties. This enables the simulation of external factors, such as noise, jumps, and mean reversion, allowing for stress testing and more realistic market conditions.

3.2.1 Contaminated fBM

The C-fBM model is constructed to test the robustness of estimators under extreme market distortions. It extends standard fBM by introducing two exogenous effects commonly encountered in high-frequency financial data: microstructure noise and discrete Poisson jumps. Although there are more realistic jump dynamics, such as Hawkes-based self-exciting jumps [2], discrete Poisson jumps were chosen for their tractability and are sufficient to strain the estimators. Formally, the observed process is given by:

$$X_i^{\text{cont}} = X_i^0 + \varepsilon_i + \sum_{k=1}^{N} J_k \cdot \mathbf{1}_{\{i=\tau_k\}}, \quad i = 0, ..., n.$$

Where $X_i^0 = B_H(t_i)$ is a discretely sampled fractional Brownian motion (fBM) with Hurst index $H \in (0, 1)$, defined on a uniform grid $t_i = \frac{iT}{n}$.

The noise term ε_i is independent Gaussian noise scaled to the standard deviation of the underlying process increments:

$$\varepsilon_i \sim \mathcal{N}\left(0,\, (\eta \cdot \text{vol})^2\right), \quad \text{vol} = \text{SD}(X_i^0 - X_{i-1}^0), \quad \varepsilon_0 = 0 \quad \eta = 0.3.$$

This mimics microstructure noise, such as bid-ask bounce or other high-frequency distortions in financial markets.

Jumps occur at distinct time grid times; the number of events over n steps:

$$N \sim \text{Poisson}(\lambda_{\text{step}} n), \quad \lambda_{\text{step}} = 0.02$$

with jump times chosen uniformly at random without replacement, and jump sizes:

$$J_k = \pm\sigma \cdot \text{vol} \cdot \exp(\mathcal{N}(0, 1)), \quad \sigma = 0.5$$

The resulting process X^{cont} after rescaling by T^H, is neither self-similar nor Gaussian. Its scaling law is corrupted at fine scales, making it a valuable stress test for estimators that rely on pure fBM assumptions. In particular, it helps to distinguish between estimators that are robust to high-frequency noise and jumps, and those that implicitly assume a clean scaling structure. We decided not to add jumps or noise to other models, as it may hamper their distinctive characteristics, which we are testing with the estimators.

3.2.2 Fractional Ornstein–Uhlenbeck

The fOU process [9] introduces mean-reversion into a rough volatility setting by replacing standard Brownian motion in the classical OU model with fBM. This results in a process that retains long memory while incorporating a central tendency, which is more reflective of empirical volatility dynamics observed in financial markets, given appropriate parameters.

The continuous-time dynamics are governed by the stochastic differential equation:

$$dX_t = \theta(\mu - X_t)\,dt + \sigma\,dB^H(t), \quad X_0 = \mu,$$

Where $\theta > 0$ is the rate of mean reversion, μ is the long-run mean, $\sigma > 0$ is the volatility coefficient, and $B^H(t)$ is a fBM with $H \in (0, 1)$.

The process is approximated using an Euler–Maruyama scheme over an equidistant grid $t_i = \frac{iT}{n}$:

$$X_{i+1} = X_i + \theta(\mu - X_i)\,\Delta t + \sigma\,\Delta B_i^H, \quad \Delta t = \tfrac{T}{n}, \quad \Delta B_i^H = B_H(t_{i+1}) - B_H(t_i).$$

With $\theta = 2.5,\ \mu = 0,\ \sigma = 1$.

Unlike standard fBM, which exhibits non-stationary increments and no mean reversion, the fOU process introduces both temporal correlation and a pull-back towards a central level, while still preserving roughness for $H < 0.5$. This model is beneficial for testing estimators under non-stationary scaling distortions, where the presence of drift and reversion may bias techniques that rely strictly on self-similarity or stationary increments. Although we fix $\mu = 0$ and $\sigma = 1$ in this case, different choices of μ and σ would only shift or rescale the path. Scale-invariant estimators (e.g. MLE, Wavelet) remain unaffected, whereas scale-dependent methods (e.g. Power Variation, Log–Log regression) may perform suboptimally without standardisation.

3.3 Methodology

The estimators will be applied across all models, using path partitions of size 32 for a balance in efficiency and accuracy. This was chosen from analysis in Section 2.4.2 as well as general heuristics and testing. The goal is to evaluate their statistical behaviour, convergence properties, and practical relevance. To that end, the following analyses will be conducted:

1. **Estimator Benchmarking:** For the range of models, Hurst parameters, and fixed time scale, we will assess each estimator's bias via Monte Carlo, construct confidence intervals, perform normality and examine distribution via box-plots and histograms. This serves as a baseline comparison of estimator performance under controlled conditions.

2. **Convergence Analysis:** Fixing the Hurst exponent and time-scale, we will progressively increase the number of step sizes n with M simulation runs to observe whether the estimators demonstrate consistent convergence properties. This will be repeated across the different model types to gauge robustness.

3. **Practical Evaluation:** Finally, we will apply all estimators to real-world financial data, specifically SPX realised variance, to assess how they perform outside of synthetic settings, offering insight into their potential viability in applied contexts.

3.4 Results

3.4.1 Estimator Benchmarking

Estimator benchmarks will be achieved via Monte Carlo simulations. Default simulations $M = 50$, with a base $n = 264$ that are scaled by time-scale $T = 5$ (so total $n = 1320$) and with $H \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$.

Since Monte Carlo uses randomness, our estimates of H vary across runs. To form a simple confidence interval for H, take the M Monte Carlo estimates $H_1, \ldots, H_M$ and define

$$m = \frac{1}{M}\sum_{i=1}^{M} H_i, \qquad \hat{\sigma}_H = \sqrt{\frac{1}{M-1}\sum_{i=1}^{M}(H_i - m)^2}.$$

Assuming an approximately Normal shape, use:

$$Z = \frac{m - H}{\hat{\sigma}_H} \sim \mathcal{N}(0,1) \quad \Longrightarrow \quad H \in \left[m - z_{\alpha/2}\,\hat{\sigma}_H,\ m + z_{\alpha/2}\,\hat{\sigma}_H\right].$$

For a 95% interval, $\alpha = 0.05$ so $z_{\alpha/2} = \Phi^{-1}(0.975) \approx 1.96$

Another important property of estimators is whether their sampling distribution is close to Normal in large samples. The Jarque–Bera (JB) test is

commonly used to test this, based on skewness and kurtosis:

$$JB = \frac{M}{6}\left(S^2 + \frac{(K-3)^2}{4}\right),$$

where S is the sample skewness and K the sample kurtosis of the Monte Carlo estimates.

H_0 : The sample comes from a Normal distribution ($S \approx 0,\ K \approx 3$),

H_1 : The sample does not come from a Normal distribution.

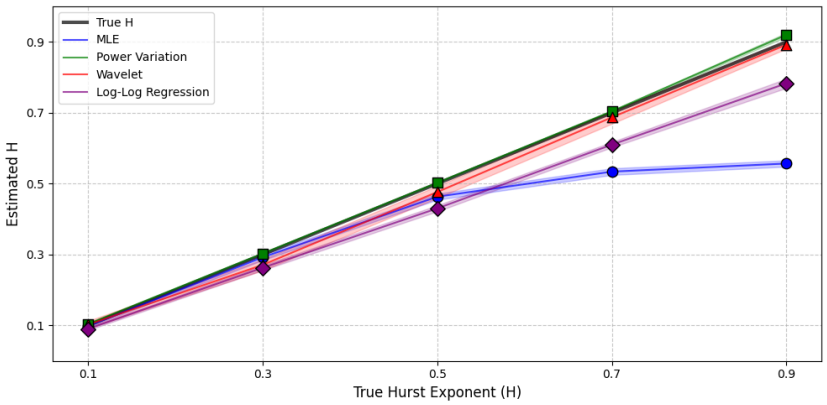
If $p > 0.05$, we fail to reject H_0, indicating no statistical evidence against Normality at this sample size; if $p < 0.05$, the null is rejected, suggesting the distribution shows skew/heavy tails and Normal-based intervals may be unreliable. In that case, prefer percentile (empirical) CIs and report median/IQR alongside mean/std. Stars (*) flag JB rejection levels: * 5%, ** 1%, *** 0.1%.

Clean fBM

Table 3.1: Performance of estimators for Clean fBM across Hurst exponents. Values show 'bias' and '(std)' with '95% [CI] for $\hat{H}$' with JB rejection flags.

H	MLE	PV	Wavelet	Log–Log
0.1	-0.0018 (0.0128) [0.0947, 0.1018]	0.0043 (0.0042) [0.1031, 0.1055]	0.0023 (0.0377) [0.0918, 0.1127]	-0.0097 (0.0132) [0.0866, 0.0940]
0.3	-0.0077 (0.0298) [0.2840, 0.3005]	0.0029* (0.0038) [0.3019, 0.3040]	-0.0291 (0.0561) [0.2553, 0.2864]	-0.0373* (0.0249) [0.2559, 0.2696]
0.5	-0.0367 (0.0307) [0.4548, 0.4718]	0.0027 (0.0033) [0.5018, 0.5036]	-0.0236 (0.0614) [0.4593, 0.4934]	-0.0698 (0.0313) [0.4216, 0.4389]
0.7	-0.1660 (0.0340) [0.5245, 0.5434]	0.0042 (0.0043) [0.7030, 0.7054]	-0.0129 (0.0650) [0.6690, 0.7051]	-0.0900 (0.0273) [0.6024, 0.6175]
0.9	-0.3432 (0.0320) [0.5480, 0.5657]	0.0200*** (0.0208) [0.9142, 0.9258]	-0.0080 (0.0400) [0.8809, 0.9031]	-0.1169 (0.0470) [0.7700, 0.7961]

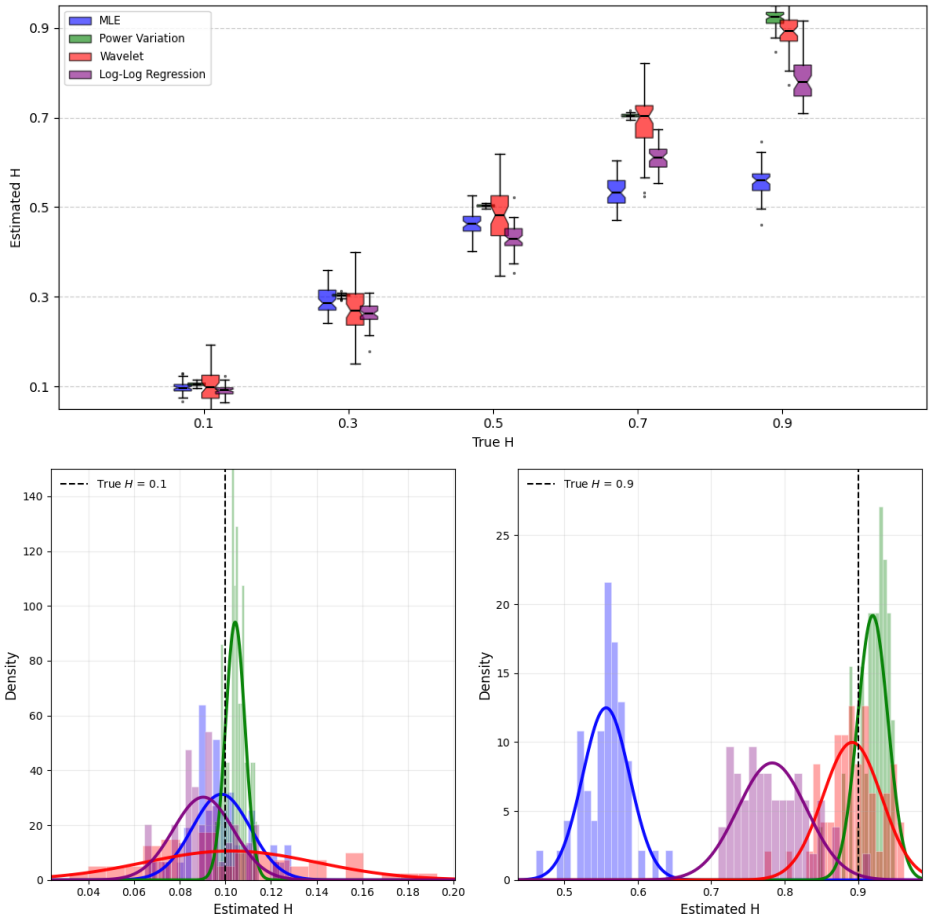
Figure 3.1: Estimators estimated versus True Hurst Exponent - Clean fBM



PV exhibits a tiny positive bias, as predicted in Section 2.1, with the tightest dispersions/CI's among all estimators. MLE and Log–Log bias become increasingly negative as H grows. Wavelet has the largest variance overall but remains near unbiased at high H. Patterns are consistent with increasing covariance ill-conditioning for MLE, making inversion numerically unstable; log–log slope suffers from finite-scale regression effects, while PV's $q = 2$

scaling trades a tiny bias for very low variance and Wavelet trades bias for variance at coarse scales.

Figure 3.2: Top: Box plots for estimators estimated versus true Hurst exponent. Bottom: Histograms for $H = 0.1$ and $H = 0.9$ - Clean fBM.



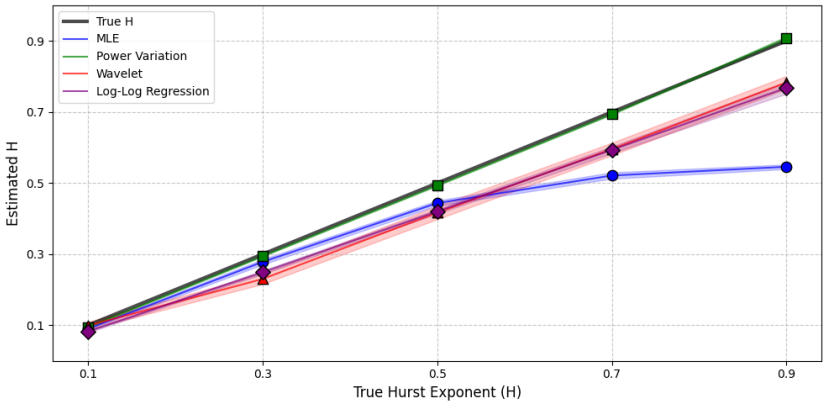
PV shows the tightest box/whiskers throughout; Wavelet exhibits the largest IQR and range; MLE and Log–Log boxes shift downward as H increases. The histograms at the extremes clarify the distributional shape behind the JB flags. At H=0.1, PV concentrates sharply around the truth, MLE and Log–Log are slightly left of the truth (small negative bias) with mild skew, and Wavelet is centred close to 0.1 but with heavy tails. At H=0.9, PV remains very concentrated but is marginally right-shifted (explaining the JB rejection via slight skew); Wavelet is broadly centred near the truth with long tails; Log–Log underestimates, and MLE collapses far left, reflecting the difficulty of MLE and log–log estimations as long-range dependence strengthens.

Contaminated fBM

Table 3.2: Performance of estimators for C-fBM across Hurst exponents. Values show 'bias' and '(std)' with '95% [CI] for $\hat{H}$' with JB rejection flags.

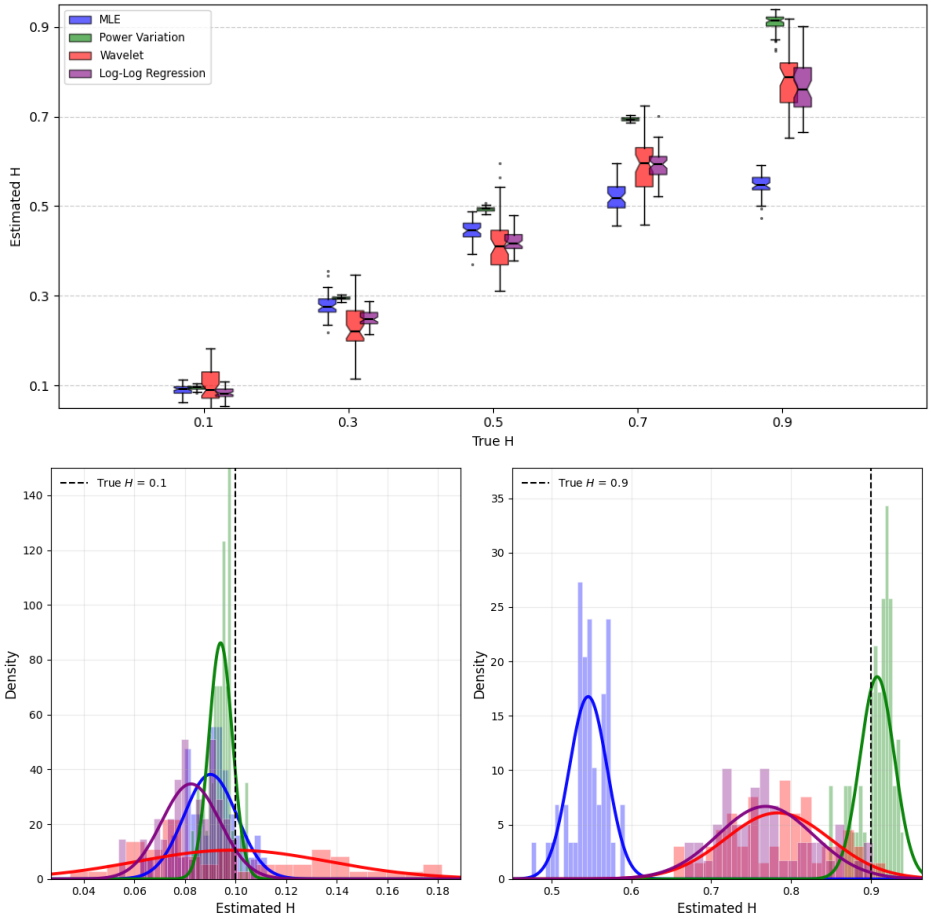
H	MLE	PV	Wavelet	Log–Log
0.1	-0.0099 (0.0105) [0.0872, 0.0930]	-0.0059 (0.0046) [0.0928, 0.0954]	-0.0011 (0.0380) [0.0884, 0.1094]	-0.0177 (0.0115) [0.0791, 0.0855]
0.3	-0.0218 (0.0271) [0.2707, 0.2857]	-0.0058 (0.0042) [0.2930, 0.2953]	-0.0698 (0.0519) [0.2158, 0.2446]	-0.0509 (0.0176) [0.2442, 0.2540]
0.5	-0.0562 (0.0265) [0.4364, 0.4511]	-0.0067 (0.0051) [0.4919, 0.4947]	-0.0830 (0.0674) [0.3983, 0.4356]	-0.0797 (0.0235) [0.4138, 0.4268]
0.7	-0.1787 (0.0321) [0.5125, 0.5302]	-0.0063 (0.0044) [0.6925, 0.6949]	-0.1041 (0.0638) [0.5783, 0.6136]	-0.1064 (0.0346) [0.5840, 0.6032]
0.9	-0.3540 (0.0238) [0.5394, 0.5526]	0.0078** (0.0215) [0.9018, 0.9137]	-0.1167 (0.0657) [0.7651, 0.8015]	-0.1319 (0.0598) [0.7515, 0.7846]

Figure 3.3: Estimators estimated versus True Hurst Exponent - C-fBM



Contamination pushes MLE and Log–Log further downward relative to the clean case, inflating Wavelet variability; PV stays tight and close to unbiased (slightly negative for $H \leq 0.7$, turning mildly positive at $H = 0.9$). JB rejections concentrate at the edges, particularly for the more model–sensitive methods (MLE/Log–Log), with PV only showing significance at very high H. Interestingly, our implementation of Wavelet was not proven to be noise-robust in this context.

Figure 3.4: Top: Box plots for estimators estimated versus true Hurst exponent. Bottom: Histograms for $H = 0.1$ and $H = 0.9$ - C-fBM.



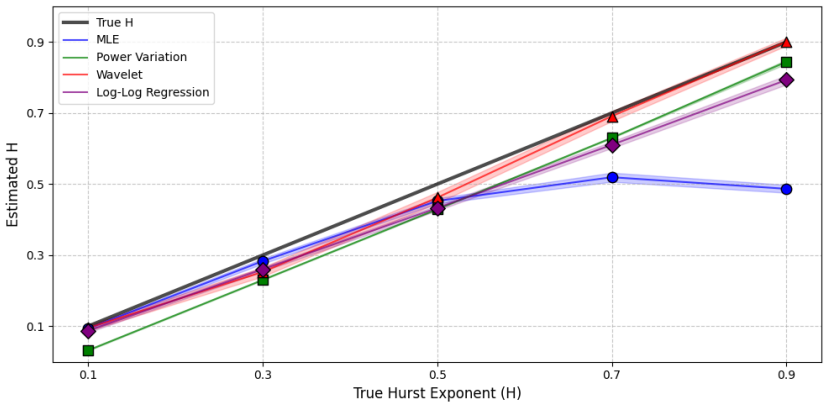
The box plots show broader boxes/whiskers than in clean fBM, with the MLE and Log–Log medians drifting downward as H increases; Wavelet remains the most dispersed, while PV retains the tightest spread. At $H = 0.1$, PV is sharply concentrated near the true H; Log–Log sits slightly left with mild skew, likewise for MLE that fails JB; Wavelet is centred lower with heavier tails. At $H = 0.9$, PV is very concentrated but a touch right-shifted (small positive skew; JB rejection), Wavelet is broad and left of the true H, Log–Log underestimates with wide tails, and MLE is far left.

RFSV

Table 3.3: Performance of estimators for RFSV across Hurst exponents. Values show 'bias' and '(std)' with '95% [CI] for $\hat{H}$' with JB rejection flags.

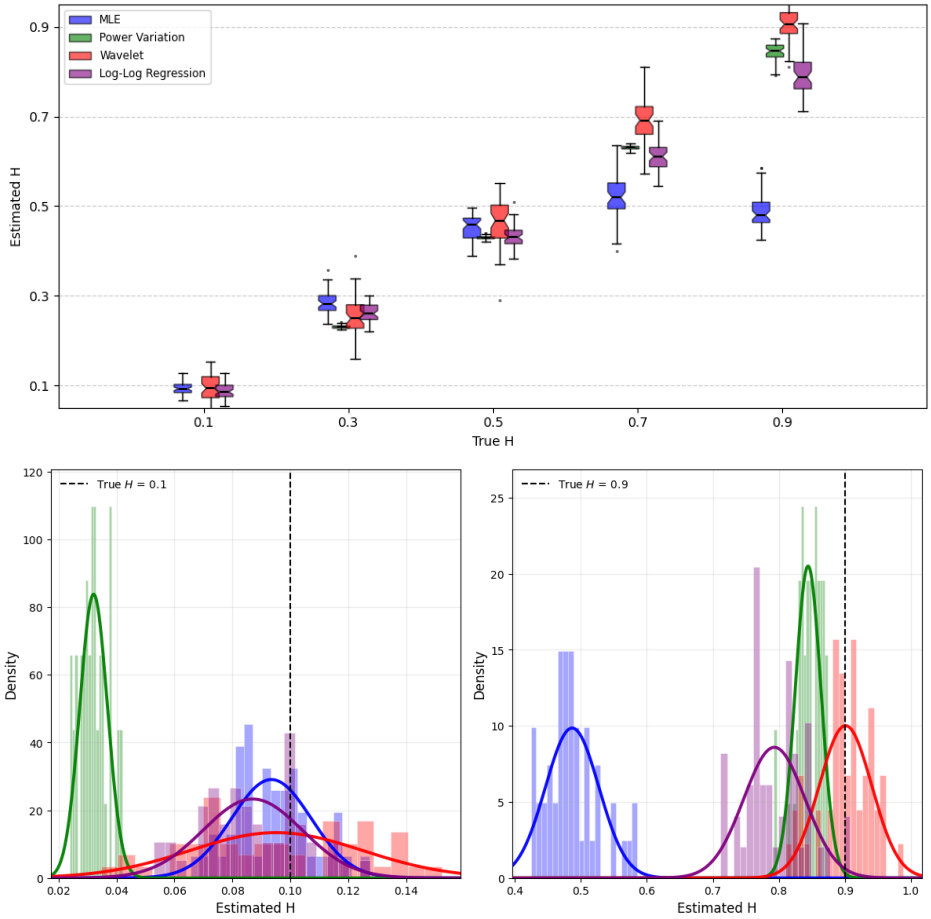
H	MLE	PV	Wavelet	Log–Log
0.1	-0.0065 (0.0137) [0.0897, 0.0973]	-0.0679 (0.0048) [0.0308, 0.0334]	-0.0048 (0.0298) [0.0869, 0.1034]	-0.0130 (0.0171) [0.0823, 0.0918]
0.3	-0.0168 (0.0262) [0.2760, 0.2905]	-0.0697 (0.0036) [0.2293, 0.2313]	-0.0472 (0.0464) [0.2399, 0.2657]	-0.0396 (0.0211) [0.2545, 0.2662]
0.5	-0.0473 (0.0273) [0.4451, 0.4602]	-0.0706 (0.0039) [0.4283, 0.4305]	-0.0381 (0.0560) [0.4463, 0.4774]	-0.0672 (0.0259) [0.4257, 0.4400]
0.7	-0.1806 (0.0461) [0.5066, 0.5322]	-0.0697** (0.0044) [0.6290, 0.6315]	-0.0088 (0.0509) [0.6770, 0.7053]	-0.0894 (0.0331) [0.6014, 0.6197]
0.9	-0.4132 (0.0405) [0.4756, 0.4980]	-0.0557 (0.0195) [0.8389, 0.8497]	0.0009 (0.0398) [0.8899, 0.9120]	-0.1068 (0.0464) [0.7803, 0.8061]

Figure 3.5: Estimators estimated versus True Hurst Exponent - RFSV



PV is systematically low by roughly a constant amount (about -0.07) across H but remains extremely tight; MLE and Log–Log deteriorate as dependence strengthens and become strongly negative at high H (MLE starts reversing near $H = 0.9$). The wavelet is often closest on average, with a moderate spread (near zero bias at the ends and modest negatives in the middle).

Figure 3.6: Top: Box plots for estimators estimated versus true Hurst exponent. Bottom: Histograms for $H = 0.1$ and $H = 0.9$ - RFSV.



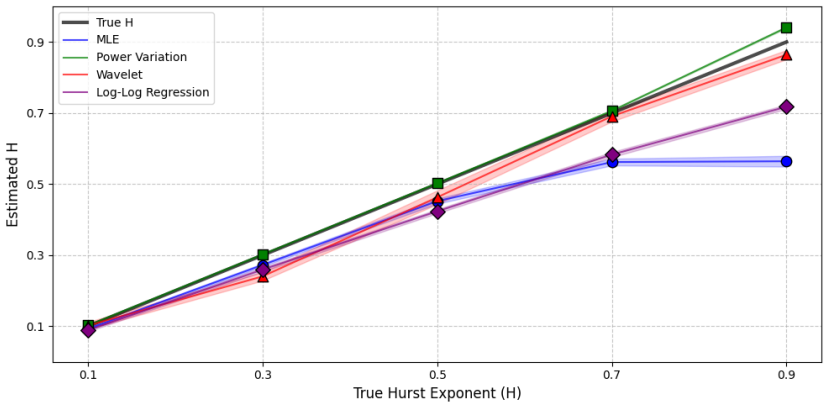
Box plots show PV consistently tight but shifted below the truth, Wavelet wider yet centred, and MLE/Log–Log drifting down with H. In the histograms, at $H = 0.1$ PV is sharply left of the dashed line (time-change bias), and Wavelet is near-centred with heavier tails. At $H = 0.9$, PV remains tight but left, Wavelet stays close to the truth with longer tails, Log–Log is broad and low, and MLE collapses far left, hallmarks of misspecification under strong long-memory.

fOU Processes

Table 3.4: Performance of estimators for fOU across Hurst exponents. Values show 'bias' and '(std)' with '95% [CI] for $\hat{H}$' with JB rejection flags.

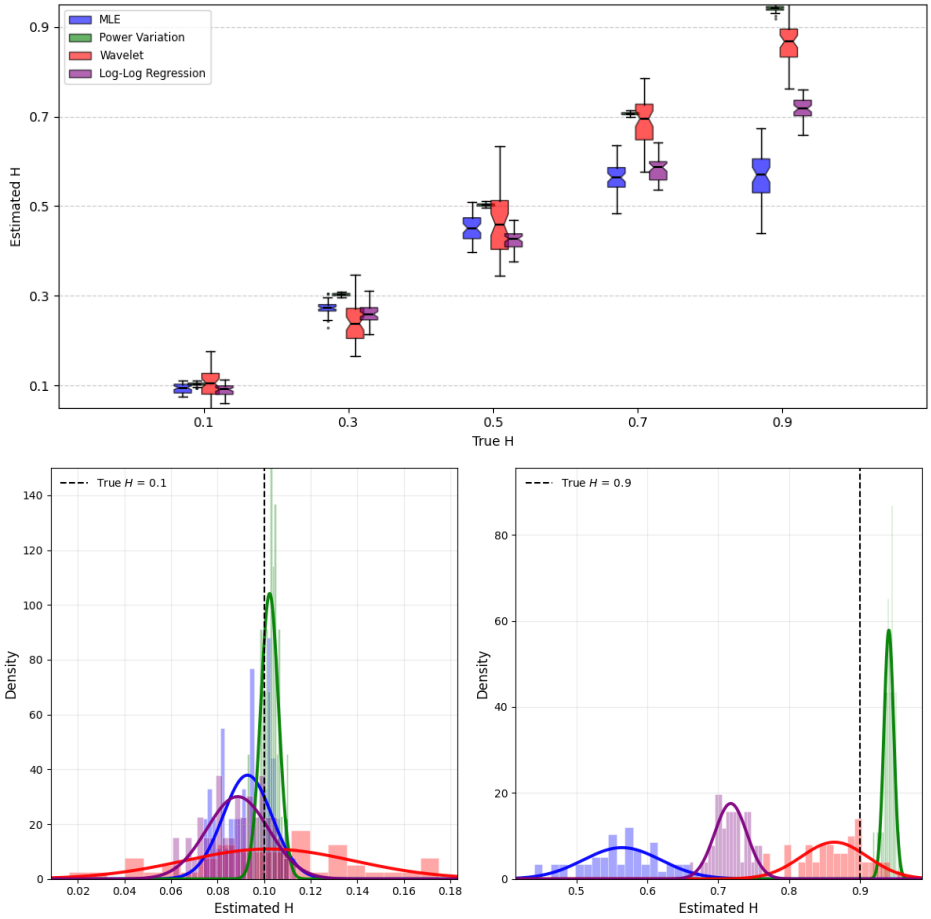
H	MLE	PV	Wavelet	Log–Log
0.1	-0.0071 (0.0105) [0.0900, 0.0958]	0.0024 (0.0038) [0.1013, 0.1035]	0.0020 (0.0365) [0.0919, 0.1121]	-0.0113 (0.0133) [0.0850, 0.0923]
0.3	-0.0275 (0.0157) [0.2682, 0.2768]	0.0022 (0.0035) [0.3013, 0.3032]	-0.0596 (0.0436) [0.2283, 0.2525]	-0.0405 (0.0210) [0.2537, 0.2653]
0.5	-0.0478 (0.0270) [0.4447, 0.4597]	0.0026 (0.0034) [0.5016, 0.5035]	-0.0371 (0.0700) [0.4434, 0.4823]	-0.0765 (0.0206) [0.4178, 0.4292]
0.7	-0.1379 (0.0343) [0.5526, 0.5716]	0.0063 (0.0041) [0.7052, 0.7074]	-0.0091 (0.0534) [0.6761, 0.7056]	-0.1169 (0.0264) [0.5758, 0.5904]
0.9	-0.3357 (0.0584) [0.5491, 0.5795]	0.0409* (0.0069) [0.9390, 0.9428]	-0.0359 (0.0466) [0.8511, 0.8770]	-0.1822 (0.0227) [0.7115, 0.7241]

Figure 3.7: Estimators estimated versus True Hurst Exponent - fOU.



Because fOU is not self-similar misspecification: PV stays very tight but drifts upward with H (small at low H, more noticeable by $H = 0.9$); MLE and Log–Log drift steadily downward as H increases; Wavelet tends to be comparatively well–behaved under mean reversion, often closer than covariance/slope methods at mid/high H, albeit with wider spread than PV.

Figure 3.8: Top: Box plots for estimated versus true Hurst exponent. Bottom: Histograms for $H = 0.1$ and $H = 0.9$ - fOU.



Similar to before, the box plots show PV with the smallest IQR yet consistently overestimating; MLE and Log–Log shift left as H grows; Wavelet stays near-centred with longer tails. The histograms at $H = 0.1$ and $H = 0.9$ are roughly symmetric, but PV's narrow modes sit just to the right of the dashed line, and once again deviate from Gaussian for $H = 0.9$ and MLE/Log–Log lie to the left; Wavelet sits near the line with habitual heavier tails.

3.4.2 Convergence Testing

Setup: fixed Hurst exponent $H = 0.10$ (following [16]), $T = 1$, $M = 50$ Monte Carlo replications, and subwindow sizes $n \in \{128, 256, 512, 1024, 2048\}$. For each n and estimator, we compute

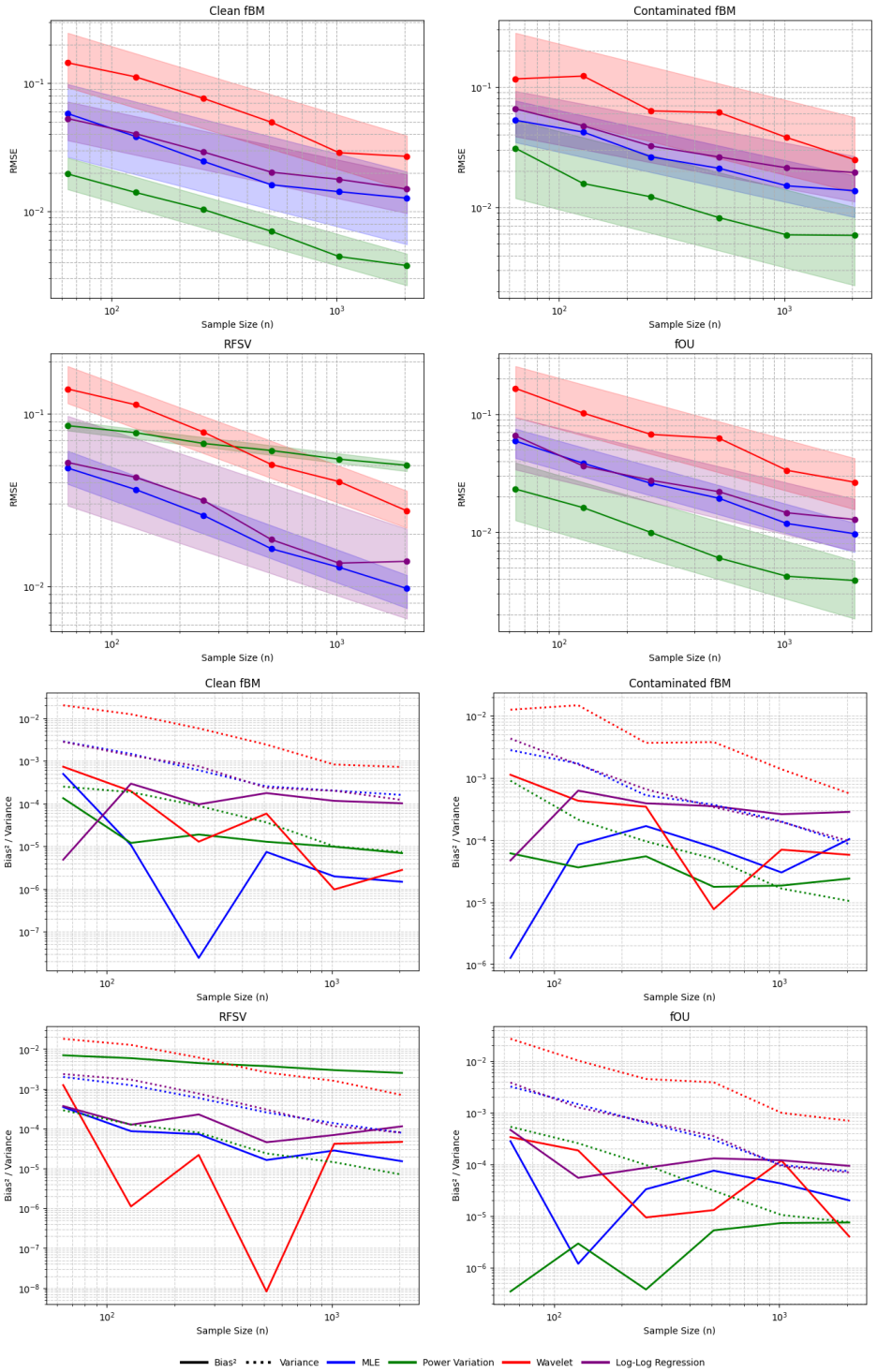
$$\mathrm{MSE}(n) = \frac{1}{M}\sum_{i=1}^{M}\left(\widehat{H}_i - H\right)^2 = \underbrace{\left(\overline{H} - H\right)^2}_{\text{bias}^2} + \underbrace{\mathrm{Var}(\widehat{H})}_{\text{variance}}.$$

Thus, the bias–variance panels plot the two additive components whose sum gives the MSE curves. However, in practice, we use RMSE instead of MSE because it has the same scaled units, making it more interpretable. The convergence rate α is estimated from the slope of a log–log regression, $\mathrm{RMSE}(n) \approx c\, n^{-\alpha} + \varepsilon_n$; $\alpha \approx 1/2$ indicates the canonical $1/n$ Monte Carlo decay.

Table 3.5: Convergence at $H = 0.1$, $T = 1$. First row in each cell: RMSE at $n = 2048$ with JB rejection flags; second row: $\alpha\,(\pm\mathrm{se})$.

Estimator	Clean fBM	C-fBM	RFSV	fOU
MLE	0.0127	0.0138	0.0097	0.0097
	0.45 ± 0.06	0.41 ± 0.03	0.48 ± 0.02	0.53 ± 0.02
PV	0.0038	0.0059	0.0502	0.0039
	0.50 ± 0.02	0.48 ± 0.06	0.16 ± 0.01	0.55 ± 0.05
Wavelet	0.0269	0.0249	0.0273*	0.0265
	0.53 ± 0.04	0.46 ± 0.06	0.48 ± 0.02	0.52 ± 0.04
Log–Log	0.0149	0.0195	0.0139	0.0128
	0.38 ± 0.03	0.36 ± 0.04	0.44 ± 0.05	0.46 ± 0.04

Figure 3.9: Convergence across models/estimators ($H = 0.1$, $T = 1$). Top 2x2: RMSE vs n (log–log). Bottom 2x2: bias^2 and variance vs n (log–log).



Clean fBM PV has the lowest RMSE for all n with $\alpha \approx 0.5$. Wavelet also has $\alpha \approx 0.5$ but sits higher because of variance, while its bias^2 is competitive. MLE decays slightly more slowly, and Log–Log is the slowest. Both bias^2 and variance decline with n for all methods, supporting consistency.

Contaminated fBM PV, MLE and Wavelet keep $\alpha \approx 0.5$; Log–Log decays more slowly. PV remains most accurate across n; Wavelet's main issue is variance at small/mid n. For all estimators, bias^2 and variance fall with n, which supports consistency in practice (with larger constants than in clean fBM).

RFSV Wavelet Log–Log, and MLE have near–optimal rates. However, PV exhibits a very slow decay ($\alpha \approx 0.16$) because a persistent bias term from $\nu = 1.5$ dominates under the time change, resulting in their RMSE plateauing. PV is not consistent for the fBM H under RFSV (bias floor), whereas the others continue to improve with n.

fOU Despite the mean reversion of $\theta = 2.5$, PV, all estimators are close to the $1/n$ decay. PV leads on RMSE; Wavelet is well calibrated on bias but has higher variance. PV, MLE, and Wavelet show both bias^2 and variance decreasing with n, supporting consistency.

Bias–Variance Interestingly, the bias of all the estimators appears to decrease slowly relative to the variance as n increases, implying that these estimators are noise-dominated over finite n. Moreover, there was only one JB rejection, indicating that the empirical distribution of the estimators is already close to Gaussian, in line with the asymptotic theory.

Caveat on H All convergence slopes α, RMSE levels, and bias/variance decompositions here are computed at $H = 0.1$. The qualitative takeaways generally carry over, but performance changes with H: (i) MLE and Log–Log exhibit increasing downward bias as H grows; (ii) PV remains low-variance but shows a small positive bias in clean fBM/fOU and a persistent negative offset under RFSV; (iii) Wavelet stays bias-competitive while its variance increases with H.

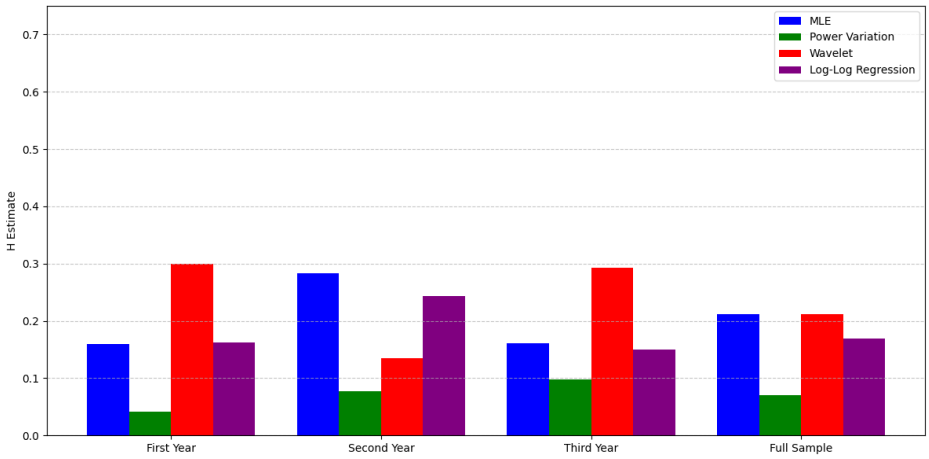
3.4.3 Practical Evaluation

The data here will be the same data that was used in the section 2.6; estimated daily SPX volatility from 2nd Feb 2020 to 24th Feb 2023:

Table 3.6: Estimator Hurst exponent estimates across time-periods

Period	MLE	Power Variation	Wavelet	Log–Log Regression
Year 1	0.1601	0.0410	0.2992	0.1618
Year 2	0.2831	0.0773	0.1353	0.2438
Year 3	0.1605	0.0981	0.2931	0.1495
Full Sample	0.2118	0.0704	0.2112	0.1696

Figure 3.10: 3-year Rough Volatility Estimate on Realised (SPX) Volatilities on Estimators



These results reveal more complexities. PV is most consistent $H \approx 0.1$ as reported by Gatheral et al. [16] and is the most stable across years (~ 0.04 - 0.10). MLE and Log–Log are more period-sensitive: both peak in Year 2 (0.28 and 0.24), but sit near 0.16 in Years 1/3. Wavelet swings the most (high in Years 1/3, low in Year 2), in line with its larger variance. Based on prior results, especially when $H < 0.5$, one would expect a tighter agreement across years and estimators. These inconsistencies might instead point to a model/scale mismatch rather than simply estimator failure, which may be explainable in light of the artefact concerns by Cont and Das [12].

3.4.4 Summary of Results

Overall, the simulations and the SPX application suggest a straightforward conclusion: no single estimator is uniformly best; performance depends on the model assumptions, H, and the sample size.

Benchmark overview

- Clean fBM: PV is near-unbiased with the smallest dispersion; MLE and log–log drift downward as H increases; Wavelet is well centred on average but has the largest spread.
- Contaminated fBM: the same directions persist. PV stays tight with small bias; MLE/log–log shift further down; Wavelet degrades here.
- RFSV: PV shows an approximately constant negative bias, so it remains tight but below the truth; Wavelet is near-centred with moderate variance; MLE/log–log drift downward as persistence strengthens.
- fOU: PV is tight with a mild positive bias that grows with H; MLE/log–log underestimates because mean reversion breaks pure fBM scaling; Wavelet is closest on average with wider spread/tails.

Convergence (α) and consistency at $H = 0.1$

- Clean and mildly contaminated fBM: PV and Wavelet have $\alpha \approx 0.5$ and both bias^2 and variance fall with n; MLE is slightly slower; log–log is slowest. This supports practical consistency for all four.
- RFSV: PV has slow decay, so it is not consistent for the model; Wavelet Log–Log, and MLE continue to improve optimally.
- fOU: all estimators close to $1/n$ decay. Bias^2 and variance continue to shrink with n.
- Bias-Variance: estimators are noise-dominated over finite n.

Real data (SPX RV, 2020–2023) Estimators, although broadly agree within the rough regime, are inconsistent among each other and at different time scales, when they are expected to be more similar than different, especially if the true roughness of volatility is $H = 0.1$, leading to suspect other failures rather than simple estimator failure.

Chapter 4

Conclusion

The comprehensive analysis conducted in this project highlights the methodological trade-offs and fragility that accompany Hurst-exponent estimation, the cornerstone of the rough-volatility paradigm. This work speaks directly to the field's evolution, from the foundational empirical claim by Gatheral et al. [16] to the more recent scepticism by Cont and Das [12]. PV performed best in clean fBM (near-unbiased with very small dispersion), but its robustness breaks under misspecification: a persistent negative offset under RFSV (time-change) and a small positive bias that grows with H in fOU; contamination also perturbs its scaling constant. The wavelet is the most bias-calibrated across misspecified settings (RFSV, fOU) for all H and maintains a favourable convergence rate, but it comes at the cost of higher variance, thus requiring a larger n. Likelihood and the formative Log-Log estimators are not recommended for long-range dependent processes, as they degrade as persistence increases (downward drift for large H). The simulations suggest that no single estimator is universally reliable; performance is model-dependent, which helps reconcile findings by Gatheral et al. [16] with the concerns raised by Cont and Das [12]: clean, short RV can yield low H, whereas discretisation and noise can make RV-based H spuriously low.

However, most critically, the application of all estimators to real-world SPX realised variance data yielded material dispersion and contradictory estimates of the Hurst exponent ($\hat{H} \in [0.05, 0.30]$), which were highly sensitive to the chosen time window. This empirical result may serve as a potent confirmation of the scepticism raised by Cont and Das [12]: the pervasive finding of "roughness" ($H \approx 0.1$) is likely an artefact of estimation error. largely driven by the inherent limitations of estimating the unobservable instantaneous volatility

from realised variance. The initial allure of the rough volatility paradigm was its elegant explanation of the power-law decay of the volatility skew, but it is severely undermined by the inability of its estimation tools to reliably identify true roughness.

Consequently, this project concludes that while mathematically elegant and initially compelling, the rough volatility paradigm rests on an empirically unstable foundation and requires further investigation to clarify the inconclusiveness of rough volatility. The empirical success of more parsimonious, traditional models (e.g., the multi-factor diffusion models advocated by Rogers [23], which can reproduce the same empirical signatures without fractional dynamics) as well as alternative frameworks (e.g., the multifractal log S-fBM of Wu et al. [24], which seeks to unify rough and multifractal behaviours), suggests that the pursuit of roughness, rough volatility may be premature as a stand-alone paradigm. In line with the principle of Occam's razor, and until the challenge of estimation can be decisively overcome, simpler conventional models retain their practical advantage. Hence, until more evidence is presented, the supposed 'roughness' of volatility appears less a discovery of a deep market feature and perhaps more a cautionary tale on the perils of model overfitting and the profound gap between theoretical innovation and empirical validation.

Bibliography

[1] Abry, P., Flandrin, P., Taqqu, M. S., & Veitch, D. (2000). *Wavelets for the analysis, estimation and synthesis of scaling data*. In K. Park & W. Willinger (Eds.), *Self-Similar Network Traffic and Performance Evaluation* (pp. 39–88). Wiley.

[2] Bacry, E., Mastromatteo, I., & Muzy, J.-F. (2015). *Hawkes processes in finance*. *Market Microstructure and Liquidity*, **1**(1), 1550005.

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

[4] Black, F., & Scholes, M. (1973). *The pricing of options and corporate liabilities*. Journal of Political Economy, 81(3), 637–654.

[5] Bollerslev, T. (1986). *Generalized autoregressive conditional heteroskedasticity*. Journal of Econometrics, 31(3), 307–327.

[6] Brandi, G., & Di Matteo, T. (2022). *Multiscaling and rough volatility: An empirical investigation*. International Review of Financial Analysis, 7(2), 174–196.

[7] Brent, R. P. (1973). *An algorithm with guaranteed convergence for finding a zero of a function*. In *Algorithms for Minimisation without Derivatives* (chap. 4). Englewood Cliffs, NJ: Prentice-Hall.

[8] Chang, Y.-C. (2014). *Efficiently implementing the maximum likelihood estimator for Hurst exponent*. Mathematical Problems in Engineering, vol. 2014.

[9] Cheridito, P., Kawaguchi, H., & Maejima, M. (2003). *Fractional Ornstein–Uhlenbeck processes*. Electronic Journal of Probability, 8, Paper No. 3, 1–14.

[10] Comte, F., & Renault, E. (1998). *Long memory in continuous-time stochastic volatility models*. Mathematical Finance, 8(4), 291–323.

[11] Cont, R. (2001). *Empirical properties of asset returns: Stylized facts and statistical issues*. Quantitative Finance, 1(2), 223–236.

[12] Cont, R., & Das, P. (2024). *Rough volatility: fact or artefact?* Sankhya B, 86, 191–223.

[13] El Euch, O., & Rosenbaum, M. (2019). The characteristic function of rough Heston models. Mathematical Finance, 29(1), 3-38.

[14] Engle, R. F. (1982). *Autoregressive conditional heteroskedasticity with estimates of the variance of United Kingdom inflation*. Econometrica, 50(4), 987–1007.

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

[16] Gatheral, J., Jaisson, T., & Rosenbaum, M. (2018). *Volatility is rough*. Quantitative Finance, 18(6), 933–949.

[17] Gatheral, J., Jusselin, P., & Rosenbaum, M. (2020). The quadratic rough Heston model and the joint S&P 500/VIX smile calibration problem.

[18] Hagan, P. S., Kumar, D., Lesniewski, A. S., & Woodward, D. E. (2002). *Managing smile risk*. Wilmott Magazine, 84–108.

[19] Han, X., & Schied, A. (2024). *The roughness exponent and its model-free estimation*. Annals of Applied Probability, 35(2), 1049–1082.

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

[21] Hull, J., & White, A. (1987). *The pricing of options on assets with stochastic volatilities*. The Journal of Finance, 42(2), 281–300.

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

[23] Rogers, L. C. G. (2019). *Things we think we know*. Statistical Laboratory, University of Cambridge.

[24] Wu, P., Muzy, J.-F., & Bacry, E. (2022). *From rough to multifractal volatility: The log S-fBM model*. Physica A: Statistical Mechanics and its Applications, 604, 127682.

Appendix A

Python code

FM50FinalCode

August 28, 2025

[0]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.lines import Line2D
from matplotlib.patches import Patch

import pywt
from scipy.special import gamma
from scipy import linalg as la
from scipy import optimize as opt
from scipy import stats

import statsmodels.api as sm

##########################################################################
 ↪PARAMETER/DATA LOADING
 ↪#############################################################

"""Random Seed"""
seed = 42


"""Data Loading"""
X = np.loadtxt("fBMPath2.txt")
BH05 = np.loadtxt("BHpath.05.txt")
B05 = np.loadtxt("Bpath.05.txt")
W05 = np.loadtxt("Wpath.05.txt")
BH10 = np.loadtxt("BHpath.10.txt")
B10 = np.loadtxt("Bpath.10.txt")
W10 = np.loadtxt("Wpath.10.txt")
realized_var3 = np.loadtxt('SPX3yrDailyRealizedVariance1minBins.txt')


"""Simulating Standard fBM Paths """
scaling_factor = 1000
fbm_n = 1024
fbm_H_values = [0.05, 0.5, 0.9]
```

```
T = 1


"""Simulating RSFV """
H = 0.1
S0 = 1
V0_1 = 0.1
nu = 1
rho = -0.65


"""MC MLE StdBM Bias"""
m_stdBM = 512
n_sim_stdBM = 100
n_steps_stdBM = 65536
H_stdBM = 0.5
p = 95
m = [16, 32, 64]


"""RSFV ATM Vol Smile"""
V0_2 = 0.04
smile_N = 252
smile_M = 100000


"""Estimator, Models, Colors"""
models = ['Clean fBM', 'Contaminated fBM', 'RFSV', 'fOU']
estimators = ['MLE', 'Power Variation', 'Wavelet', 'Log-Log Regression']
colors = ['blue', 'green', 'red', 'purple']


"""Models C-fBM, RSFV, fOU Parameters"""
noise_level=0.2
jump_intensity=0.05
jump_size=0.25

nu_3 = 1.5

theta=2.5
mu=0
sigma=1.0


"""MC Bias + SPX"""
H_vals = (0.1, 0.3, 0.5, 0.7, 0.9)
hist_Vals=(0.1, 0.9)
```

```
T_vals = [5]

monte_N = 256
monte_M = 50


"""Estimator Convergence"""
conv_H = 0.1
conv_T = 1

conv_N = [64, 128, 256, 512, 1024, 2048]
conv_M = 50


"""Windows"""
monte_window = 32
spx_window = 32
conv_window = 32


"""Faster Sims"""
#smile_N = 128
#smile_M = 10000

#monte_N = 128
#monte_M = 25

#conv_N = [64, 128, 256, 512]
#conv_M = 25

############################################################################ Main␣
 ↪Functions ############################################################################

############################################################################ ␣
 ↪Universal Functions  #

def compute_covariance_matrix(H, t):
    t1, t2 = np.meshgrid(t, t)

    cov = 0.5 * (t1**(2*H) + t2**(2*H) - np.abs(t1 - t2)**(2*H))

    return cov

def generate_fbm(H, n, T):
    t = np.arange(1, n+1)/n

    cov = compute_covariance_matrix(H, t)
```

```
    L = la.cholesky(cov, lower=True)
    Z = np.random.normal(0, 1, size=n)
    clean_path = np.zeros(n+1)
    clean_path[1:] = L @ Z

    timeline = np.linspace(0, T, n+1)
    clean_scale = clean_path* (T**H)

    return timeline, clean_scale, clean_path

def confidence_interval(true_value, n, arr, p=p):
    std = np.std(arr, ddof=1)
    mean = np.mean(arr)

    se = std / np.sqrt(n)
    z = stats.norm.ppf(1 - (100 - p) / 200.0)

    ci_low = mean - z * se
    ci_high = mean + z * se

    bias = mean - true_value

    return mean, ci_low, ci_high, bias, std

########################################################################  Other␣
 ↪Path Generators  #

def generate_c_fbm(H, n, T):
    _,_,clean_path = generate_fbm(H, n, T)

    vol = np.std(np.diff(clean_path))
    microstructure_noise = noise_level * vol * np.random.normal(0, 1, size=n+1)
    microstructure_noise[0] = 0

    total_jumps = np.random.poisson(jump_intensity * n)
    jump_times = np.random.choice(np.arange(1, n+1), size=total_jumps,␣
 ↪replace=False)
    jump_sizes = jump_size * vol * np.exp(np.random.normal(0, 1,␣
 ↪size=total_jumps))
    signed_jumps = jump_sizes * np.sign(np.random.normal(0, 1,␣
 ↪size=total_jumps))

    contaminated_path = clean_path + microstructure_noise
    contaminated_path[jump_times] += signed_jumps

    contaminated_scale = contaminated_path * (T**H)
```

```
    return contaminated_scale

def generate_rfsv(H, n, T, S0=S0, V0=V0_2, nu=nu_3, rho=rho):
    t, BH, _ = generate_fbm(H, n, T)
    dt = T/n

    B = np.zeros(n+1)
    W = np.zeros(n+1)
    dB = np.random.normal(0, np.sqrt(dt), n)
    dW = np.random.normal(0, np.sqrt(dt), n)
    B[1:] = np.cumsum(dB)
    W[1:] = np.cumsum(dW)

    rho_bar = np.sqrt(1 - rho**2)
    S, V = simulate_rfsv_path(BH, B, W, S0, V0, nu, rho)

    logV = np.log(V)
    logV = logV - np.mean(logV)

    return logV

def generate_fou(H, n, T, theta=theta, mu=mu, sigma=sigma):
    t, fbm, _ = generate_fbm(H, n, T)
    dt = T/n

    dfbm = np.diff(fbm)
    X = np.zeros(n+1)
    X[0] = mu

    for i in range(n):
        X[i+1] = X[i] + theta*(mu - X[i])*dt + sigma*dfbm[i]

    return X

########################################################################  RSFV␣
 ↪Functions  #

def simulate_rfsv_path(BH, B, W, S0=S0, V0=V0_1, nu=nu, rho=rho):
    rho_bar = np.sqrt(1 - rho**2)
    n = len(B) - 1
    V = V0 * np.exp(nu * BH)
    S = np.zeros(n + 1)

    S[0] = S0

    dB = np.diff(B)
    dW = np.diff(W)
```

```
    for i in range(n):
        sqrtV = np.sqrt(V[i])
        S[i+1] = S[i] * (1 + sqrtV * (rho * dB[i] + rho_bar * dW[i]))

    return S, V

def simulate_paths(n_sim=n_sim_stdBM, n_steps=n_steps_stdBM,␣
 ↪block_size=m_stdBM, S0=S0, V0=V0_1, nu=nu, rho=rho):
    rho_bar = np.sqrt(1 - rho**2)
    dt = 1.0 / n_steps
    estimates = []

    for i in range(n_sim):
        dB = np.sqrt(dt) * np.random.randn(n_steps)
        dW = np.sqrt(dt) * np.random.randn(n_steps)

        B_path = np.concatenate([[0], np.cumsum(dB)])
        W_path = np.concatenate([[0], np.cumsum(dW)])
        BH_path = B_path.copy()

        S, _ = simulate_rfsv_path(BH_path, B_path, W_path, S0=S0, V0=V0, nu=nu,␣
 ↪rho=rho)

        log_rets = np.diff(np.log(S))

        spot_vars = compute_realized_variances(log_rets, block_size)
        log_vars = np.log(spot_vars)
        log_vars_centered = log_vars - np.mean(log_vars)

        H_est, _ = mle_estimator(log_vars_centered)
        estimates.append(H_est)
        estimates_array = np.array(estimates)

    return estimates_array

def compute_realized_variances(log_returns, block_size):
    n = len(log_returns)
    dt = T / n

    num_blocks = n // block_size
    spot_vars = np.zeros(num_blocks)

    for j in range(num_blocks):
        start_idx = j * block_size
        end_idx = start_idx + block_size
        block = log_returns[start_idx:end_idx]
```

```
        spot_vars[j] = np.sum(block**2) / (block_size * dt)

    return spot_vars

def estimate_params(BH, B, W, m=m, S0=S0, V0=V0_1, nu=nu, rho=rho):
    rho_bar = np.sqrt(1 - rho**2)
    S, V = simulate_rfsv_path(BH, B, W, S0, V0, nu, rho)

    log_returns = np.diff(np.log(S))
    dt = 1.0 / (len(B) - 1)

    results = {}
    for block_size in m:
        spot_vars = compute_realized_variances(log_returns, block_size)

        log_vars = np.log(spot_vars)
        log_vars_centered = log_vars - np.mean(log_vars)

        H_est, nu_est = mle_estimator(log_vars_centered)
        results[block_size] = (H_est, nu_est)

    return results

######################################################################  Smile  #

def simulate_rough_bergomi_paths(H=H, N=smile_N, V0=V0_2, nu=nu, rho=rho, T=T):
    dt = T / N
    sqrt_dt = np.sqrt(dt)
    t_grid = np.linspace(0, T, N+1)
    correction = 0.5 * nu**2 * (t_grid**(2*H))

    lags = dt * np.arange(1, N+1)
    c_H = np.sqrt(2*H)
    kernel_full = c_H * (lags ** (H - 0.5))

    L = np.zeros((N+1, N))
    for i in range(1, N+1):
        L[i, :i] = kernel_full[:i][::-1]

    return L, correction, dt, sqrt_dt

def compute_call_prices(L, correction, dt, sqrt_dt, K, M=smile_M, N=smile_N,␣
 ↪S0=S0, V0=V0_2,  nu=nu, rho=rho):
    rho_bar = np.sqrt(1 - rho**2)
    call_price_sum = np.zeros(len(K))

    for m in range(M):
```

```
        dW = np.random.normal(0, sqrt_dt, N)
        dB = np.random.normal(0, sqrt_dt, N)

        BH_orig = L @ dB
        BH_ant = L @ (-dB)

        V_orig = V0 * np.exp(nu * BH_orig - correction)
        V_ant = V0 * np.exp(nu * BH_ant - correction)

        logS = np.log(S0) * np.ones(4)

        for i in range(N):
            sqrtV_orig = np.sqrt(V_orig[i])
            sqrtV_ant = np.sqrt(V_ant[i])

            dZ1 = rho * dB[i] + rho_bar * dW[i]
            dZ2 = rho * (-dB[i]) + rho_bar * dW[i]
            dZ3 = rho * dB[i] + rho_bar * (-dW[i])
            dZ4 = rho * (-dB[i]) + rho_bar * (-dW[i])

            logS[0] += -0.5 * V_orig[i] * dt + sqrtV_orig * dZ1
            logS[1] += -0.5 * V_ant[i] * dt + sqrtV_ant * dZ2
            logS[2] += -0.5 * V_orig[i] * dt + sqrtV_orig * dZ3
            logS[3] += -0.5 * V_ant[i] * dt + sqrtV_ant * dZ4

        S_T = np.exp(logS)

        payoffs = np.maximum(S_T - K[:, np.newaxis], 0)
        avg_payoffs = np.mean(payoffs, axis=1)
        call_price_sum += avg_payoffs

    avg_call = call_price_sum / M

    return avg_call

def bs_price_call(S, K, T, r, sigma):
    d1 = (np.log(S/K) + (r + 0.5*sigma**2)*T) / (sigma*np.sqrt(T))
    d2 = d1 - sigma*np.sqrt(T)

    call = S * stats.norm.cdf(d1) - K * np.exp(-r*T) * stats.norm.cdf(d2)

    return call

def implied_vol_call(price, S, K, T, r, lower=1e-6, upper=10):
    f = lambda sig: bs_price_call(S, K, T, r, sig) - price

    return opt.brentq(f, lower, upper)
```

```
def compute_implied_vols(call_prices, K, S0=S0, T=T, r=0):
    implied_vols = []

    for i, k in enumerate(K):
        mc_price = call_prices[i]
        iv = implied_vol_call(mc_price, S0, k, T, r)
        implied_vols.append(iv)

    implied_vol_array = np.array(implied_vols)

    return implied_vol_array

########################################################################  Monte␣
 ↪Carlo Estimators  #

def monte_carlo_bias(H, T, M, n, monte_window, models=models,␣
 ↪estimators=estimators):

    estimates = {model: {est: [] for est in estimators} for model in models}

    for _ in range(M):
        _, clean_fbm, _ = generate_fbm(H, n, T)
        contaminated_fbm  = generate_c_fbm(H, n, T)
        BH_rfsv = generate_rfsv(H, n, T)
        fou = generate_fou(H, n, T)

        paths = [clean_fbm, contaminated_fbm, BH_rfsv, fou]

        for model, path in zip(models, paths):

            H_mle = chunk_array_mle(path, monte_window)
            estimates[model]['MLE'].append(H_mle)

            H_pv = chunk_array_power(path, T, monte_window)
            estimates[model]['Power Variation'].append(H_pv)

            H_wave = chunk_array_wavelet(path, monte_window)
            estimates[model]['Wavelet'].append(H_wave)

            H_loglog = chunk_array_loglog(path, monte_window)
            estimates[model]['Log-Log Regression'].append(H_loglog)

    results = {model: {} for model in models}

    for model in models:
        for est in estimators:
```

```
            arr = np.array(estimates[model][est])
            arr = arr[~np.isnan(arr)]

            mean_hat, ci_low, ci_high, bias_val, std_val  = confidence_interval(H, len(arr), arr, p=95)
            _, jb_pvalue = stats.jarque_bera(arr)


            results[model][est] = {
                'mean_hat': mean_hat,
                'bias': bias_val,
                'std': std_val,
                'conf_interval': (ci_low, ci_high),
                'jb_pvalue': jb_pvalue
            }

    return results, estimates

def collect_monte_carlo_data(H_vals=H_vals, T_vals=T_vals, monte_N=monte_N, monte_M=monte_M, monte_window=monte_window, models=models, estimators=estimators):
    all_results = {}
    all_raw = {}

    for T in T_vals:
        print(f"\n\n=== T = {T} years ===")
        n = int(monte_N * T)

        for H in H_vals:
            print(f"\nH = {H:.1f}")

            results, raw_estimates = monte_carlo_bias(H, T, monte_M, n, monte_window)
            all_results[(H, T)] = results
            all_raw[(H, T)] = raw_estimates

    return all_results, all_raw

########################################################################## Estimators  #

def prof_neg_loglik(H, X, t):
    n = len(X)
    S0 = compute_covariance_matrix(H, t)

    X_scaled = X * scaling_factor
    S0_scaled = S0 * (scaling_factor ** 2)
```

```
    L = la.cholesky(S0_scaled, lower=True)
    y = la.solve_triangular(L, X_scaled, lower=True)

    quad = np.dot(y, y)
    logdet = 2 * np.sum(np.log(np.diag(L)))
    nu = (n / 2) * np.log(quad / n) + 0.5 * logdet

    return nu

def mle_estimator(X, year_correction = 1):
    n = len(X)
    t = np.arange(1, n + 1) / (n/year_correction)
    res = opt.minimize_scalar(
        prof_neg_loglik,
        args=(X, t),
        bounds=(0.0001, 0.9999),
        method='bounded'
    )
    H_hat = res.x

    S0_hat = compute_covariance_matrix(H_hat, t) * (scaling_factor ** 2)
    Lh = la.cholesky(S0_hat, lower=True)
    y_h = la.solve_triangular(Lh, X * scaling_factor, lower=True)
    nu_hat = np.sqrt(np.dot(y_h, y_h) / n)

    return H_hat, nu_hat

def log_log_estimator(X, q=2):
    n = len(X)
    k_values = [2**j for j in range(int(np.log2(n)))]

    m_q = np.array([np.mean(np.abs(X[k:] - X[:-k])**q) for k in k_values])
    log_m = np.log(m_q)
    log_deltas = np.log(np.array(k_values))
    slope = stats.linregress(log_deltas, log_m)[0]
    H_est = slope / q

    return H_est

def power_variation_estimator(X, T, q=2):
    inc = np.diff(np.asarray(X))
    n = len(inc)
    dt = T / n

    K_q = 2**(q/2) * gamma((q+1)/2) / np.sqrt(np.pi)
    V_q = np.mean(np.abs(inc)**q)
```

```
    H = (np.log(V_q) - np.log(K_q)) / (q * np.log(dt))
    return float(np.clip(H, 1e-4, 0.9999))


def wavelet_estimator(X, wavelet='db3'):
    X = np.asarray(X, dtype=float)

    W = pywt.Wavelet(wavelet)
    max_level = pywt.dwt_max_level(len(X), W.dec_len)

    coeffs = pywt.wavedec(X, W, level=max_level)
    details = coeffs[1:]

    n_j = np.array([len(d) for d in details], dtype=float)
    S2_j = np.array([np.mean(d**2) for d in details], dtype=float)

    j = -np.log2(n_j)
    y = np.log2(S2_j)
    w_j = np.sqrt(n_j)

    beta_hat, alpha_hat = np.polyfit(j, y, deg=1, w=w_j)
    H_hat = 0.5 * (beta_hat - 1.0)

    return float(np.clip(H_hat, 1e-4, 0.9999))



########################################################################  Monte␣
 ↪Carlo Blocking Functions  #

def blocks_with_tail(a, l):
    a = np.asarray(a, dtype=float)
    if a.size < l:
        return np.empty((0, l))
    k = a.size // l
    starts = list(range(0, max(k-1, 0) * l, l))
    starts.append(a.size - l)
    return np.stack([a[s:s+l] for s in starts], axis=0)

def chunk_array_wavelet(X, block_size):
    blocks = blocks_with_tail(X, block_size)
    if blocks.size == 0:
        return np.nan

    H_values = []
    for i in range(blocks.shape[0]):
```

```
        b = blocks[i]
        H_block = wavelet_estimator(b)
        H_values.append(H_block)

    H_values = np.array(H_values)
    return np.mean(H_values)

def chunk_array_mle(X, block_size):
    blocks = blocks_with_tail(X, block_size)
    if blocks.size == 0:
        return np.nan

    H_values = []
    for i in range(blocks.shape[0]):
        b = blocks[i]
        H_block, _ = mle_estimator(b)
        H_values.append(H_block)

    H_values = np.array(H_values)
    return np.mean(H_values)

def chunk_array_power(X, T, block_size):
    blocks = blocks_with_tail(X, block_size)
    if blocks.size == 0:
        return np.nan

    dt_global = T / (len(X) - 1)
    T_block = dt_global * (block_size - 1)

    H_values = []
    for i in range(blocks.shape[0]):
        b = blocks[i]
        H_block = power_variation_estimator(b, T_block)
        H_values.append(H_block)

    H_values = np.array(H_values)
    return np.mean(H_values)

def chunk_array_loglog(X, block_size):
    blocks = blocks_with_tail(X, block_size)
    if blocks.size == 0:
        return np.nan

    H_values = []
    for i in range(blocks.shape[0]):
        b = blocks[i]
        H_block = log_log_estimator(b)
```

```
        H_values.append(H_block)

    H_values = np.array(H_values)
    return np.mean(H_values)

########################################################################
 ↪Plotting Functions  #

def plot_fbm_paths(H_values, t_full, T):
    n = len(t_full) - 1
    plt.figure(figsize=(10, 6))

    for H in H_values:
        _,_,fBm_full = generate_fbm(H, n, T)
        plt.plot(t_full, fBm_full, label=f'H={H}')

    plt.title("Fractional Brownian Motion Paths (Cholesky Method)")
    plt.xlabel("Time (t)")
    plt.ylabel("$B_t^H$")
    plt.legend()

    plt.grid(True)
    plt.show()

def plot_RFSV_path(BH, B, W, S0=S0, V0=V0_1, nu=nu, rho=rho):
    rho_bar = np.sqrt(1 - rho**2)
    n = len(BH) - 1
    dt = 1.0 / n
    tgrid = np.linspace(0, 1, n + 1)

    V = V0 * np.exp(nu * BH)
    dB = np.diff(B)
    dW = np.diff(W)
    S = np.empty(n + 1)
    S[0] = S0
    for i in range(n):
        S[i+1] = S[i] + S[i] * np.sqrt(V[i]) * (rho * dB[i] + rho_bar * dW[i])

    plt.figure(figsize=(8,6))
    plt.subplot(2,1,1)
    plt.plot(tgrid, V)
    plt.title('Variance Process Simulation with RFSV Model', fontsize = 10)
    plt.xlabel('Time (t)')
    plt.ylabel('$V_t$')
    plt.grid(True)

    plt.subplot(2,1,2)
```

```
    plt.plot(tgrid, S)
    plt.title('Stock Price Simulation with RFSV Model', fontsize = 10)
    plt.xlabel('Time (t)')
    plt.ylabel('$S_t$')
    plt.grid(True)

    plt.tight_layout()
    plt.show()

def plot_smile(x, implied_vols, H=H, nu=nu, V0=V0_2, rho=rho):
    plt.figure(figsize=(10, 6))
    plt.plot(x, implied_vols, 'bo-', linewidth=2, markersize=6)
    plt.title(f'Rough Bergomi Implied Volatility Smile (H={H},  ={rho})',␣
 ↪fontsize=14)
    plt.xlabel('Log-Moneyness (x = log(K/S ))', fontsize=12)
    plt.ylabel('Implied Volatility', fontsize=12)
    plt.grid(True, linestyle='--', alpha=0.7)
    plt.axvline(x=0, color='r', linestyle='--', alpha=0.5)

    plt.tight_layout()
    plt.show()

def plot_spx_hurst(results):
    plt.figure(figsize=(12, 6))

    segments = [r['Segment'] for r in results]
    x = np.arange(len(segments))
    width = 0.8 / len(estimators)

    for i, e in enumerate(estimators):
        vals = [r[e] for r in results]
        plt.bar(x + width * (i - (len(estimators) - 1) / 2), vals, width,␣
 ↪label=e, color=colors[i])

    plt.xticks(x, segments)
    plt.ylabel('H Estimate')
    plt.legend()
    plt.grid(axis='y', linestyle='--', alpha=0.7)
    plt.ylim(0, 0.75)
    plt.tight_layout()
    plt.show()

def plot_line_H(results_dict, H_vals=H_vals, colors=colors,␣
 ↪estimators=estimators):
    models = list(results_dict[(H_vals[0], T_vals[0])].keys())
    markers = ['o', 's', '^', 'D']
```

```
    for model in models:
        print(f"Model: {model}")
        plt.figure(figsize=(10, 5))
        plt.xlabel('True Hurst Exponent (H)', fontsize=12)
        plt.ylabel('Estimated H', fontsize=12)
        plt.grid(True, linestyle='--', alpha=0.7)

        plt.plot(H_vals, H_vals, 'k-', linewidth=3, alpha=0.7, label='True H')

        plt.ylim(0.0, 1.0)
        plt.yticks(np.arange(0.1, 1.0, 0.2))
        plt.xticks(H_vals)

        for est_idx, (estimator, color, marker) in enumerate(zip(estimators,␣
↪colors, markers)):
            mean_hats = []
            ci_lows = []
            ci_highs = []

            for H in H_vals:
                res = results_dict[(H, T_vals[0])][model][estimator]
                mean_hats.append(res['mean_hat'])
                ci_low, ci_high = res['conf_interval']
                ci_lows.append(ci_low)
                ci_highs.append(ci_high)

            plt.plot(H_vals, mean_hats, '-', color=color, alpha=0.7,␣
↪label=estimator)
            plt.fill_between(H_vals, ci_lows, ci_highs, color=color, alpha=0.2)
            plt.scatter(H_vals, mean_hats, marker=marker, s=80, color=color,␣
↪edgecolor='black', zorder=10)

        plt.legend(loc='best')
        plt.tight_layout()
        plt.show()

def plot_boxplots(all_raw, H_vals, T, colors=colors):
    nH = len(H_vals)
    nE = len(estimators)
    group_gap = 1.0
    width = 0.6

    offsets = np.linspace(-1.2, 1.2, nE) * (width / 1.0)

    for model in models:
        print(f"Model: {model}")
        fig, ax = plt.subplots(1, 1, figsize=(10,5))
```

```
        data_list = []
        positions = []
        for h_idx, H in enumerate(H_vals):
            group_center = h_idx * (nE + group_gap)
            for e_idx, est in enumerate(estimators):
                pos = group_center + offsets[e_idx]
                positions.append(pos)

                raw = all_raw.get((H, T), {}).get(model, {}).get(est, [])
                arr = np.array(raw, dtype=float)
                arr = arr[~np.isnan(arr)]
                if len(arr) == 0:
                    data_list.append(np.array([np.nan]))
                else:
                    data_list.append(arr)

        bp = ax.boxplot(
            data_list,
            positions=positions,
            widths=width,
            notch=True,
            patch_artist=True,
            showmeans=False,
            meanprops=dict(marker='D', markeredgecolor='k',␣
↪markerfacecolor='white', markersize=4),
            medianprops=dict(color='black', linewidth=1.5),
            flierprops=dict(marker='.', markersize=2.5, alpha=0.6),
            manage_ticks=False
        )

        for k, patch in enumerate(bp['boxes']):
            color = colors[k % nE]
            patch.set_facecolor(color)
            patch.set_alpha(0.65)

        if 'fliers' in bp:
            for fl in bp['fliers']:
                try:
                    fl.set_marker('.')
                    fl.set_markersize(2.5)
                    fl.set_alpha(0.6)
                except Exception:
                    pass

        group_centers = [h_idx * (nE + group_gap) for h_idx in range(nH)]
        ax.set_xticks(group_centers)
```

```
        ax.set_xticklabels([f"{H:.1f}" for H in H_vals])
        ax.set_xlim(- (nE + group_gap), group_centers[-1] + (nE + group_gap))

        ax.set_yticks(H_vals)
        ax.set_ylim(min(H_vals) - 0.05, max(H_vals) + 0.05)

        ax.set_xlabel("True H")
        ax.set_ylabel("Estimated H")
        ax.grid(axis='y', linestyle='--', alpha=0.6)

        proxies = [Patch(facecolor=colors[k], edgecolor='k', alpha=0.6) for k␣
 ↪in range(nE)]
        ax.legend(proxies, estimators, loc='upper left', fontsize='small')

        plt.tight_layout()
        plt.show()

def plot_histograms(all_raw, H_vals=hist_Vals, T_val=T_vals[0], colors=colors,␣
 ↪estimators=estimators, models=models, bins=20):
    target_H = H_vals[:2]

    for model in models:
        print(f"Model: {model}")
        fig, axes = plt.subplots(1, 2, figsize=(12, 6))

        for ax, H in zip(axes, target_H):
            model_data = all_raw.get((H, T_val), {}).get(model, {})

            all_chunks = []
            for est in estimators:
                arr = np.asarray(model_data.get(est, []), dtype=float)
                arr = arr[~np.isnan(arr)]
                if arr.size > 0:
                    all_chunks.append(arr)

            if not all_chunks:
                ax.set_visible(False)
                continue

            all_vals = np.concatenate(all_chunks)
            xmin, xmax = np.min(all_vals), np.max(all_vals)
            if xmin == xmax:
                xmin -= 0.05
                xmax += 0.05

            pad = 0.05 * (xmax - xmin)
            x_grid = np.linspace(xmin - pad, xmax + pad, 1000)
```

```
            local_ymax = 0.0
            for est, color in zip(estimators, colors):
                clean = np.asarray(model_data.get(est, []), dtype=float)
                clean = clean[~np.isnan(clean)]
                if clean.size == 0:
                    continue

                n, _, _ = ax.hist(clean, bins=bins, density=True, alpha=0.35,
                                  color=color, edgecolor="white", linewidth=0.8)
                local_ymax = max(local_ymax, float(np.max(n)))

                mu, sd = np.mean(clean), np.std(clean, ddof=1)
                if sd > 0:
                    y = stats.norm.pdf(x_grid, mu, sd)
                    ax.plot(x_grid, y, color=color, linewidth=2.8, alpha=0.95)
                    local_ymax = max(local_ymax, float(np.max(y)))

            ax.axvline(H, color="black", linestyle="--", linewidth=1.5,␣
 ↪label=f"True $H$ = {H:g}")
            ax.legend(loc="upper left", frameon=False, fontsize=10,␣
 ↪handlelength=2.5)

            ax.set_xlim(xmin - pad, xmax + pad)
            ax.set_ylim(0, min(local_ymax * 1.10, 150))

            ax.set_xlabel("Estimated H", fontsize=12)
            ax.set_ylabel("Density", fontsize=12)
            ax.grid(True, alpha=0.25)

        plt.tight_layout()
        plt.show()

def plot_rmse_convergence(results_rmse, conv_H, conv_T, p=p, models=models,␣
 ↪estimators=estimators, colors=colors):
    z = stats.norm.ppf(1 - (100 - p) / 200.0)
    fig, axs = plt.subplots(2, 2, figsize=(16, 12))
    model_positions = {m: divmod(i, 2) for i, m in enumerate(models)}

    for model, (i, j) in model_positions.items():
        ax = axs[i, j]
        for est, color in zip(estimators, colors):
            n_vals = np.array(sorted(results_rmse[model][est].keys()))
            rmse_vals = np.array([results_rmse[model][est][n] for n in n_vals])

            ln_n = np.log(n_vals)
            ln_rmse = np.log(rmse_vals)
```

```
            X = sm.add_constant(ln_n)
            fit = sm.OLS(ln_rmse, X).fit()
            slope, se_slope = fit.params[1], fit.bse[1]

            ax.loglog(n_vals, rmse_vals, 'o-', color=color)

            ci_upper = np.exp(fit.params[0] + z*fit.bse[0]) * n_vals**slope
            ci_lower = np.exp(fit.params[0] - z*fit.bse[0]) * n_vals**slope
            ax.fill_between(n_vals, ci_lower, ci_upper, color=color, alpha=0.2)

        ax.set_title(f"{model}")
        ax.set_xlabel("Sample Size (n)")
        ax.set_ylabel("RMSE")
        ax.grid(True, which="both", ls="--")

    plt.show()

def plot_bias_variance_decomposition(results_bias2, results_var, conv_H,␣
 ↪conv_T, models=models, estimators=estimators, colors=colors):

    fig, axs = plt.subplots(2, 2, figsize=(16, 12))
    model_positions = {m: divmod(i, 2) for i, m in enumerate(models)}
    color_map = dict(zip(estimators, colors))

    for model, (i, j) in model_positions.items():
        ax = axs[i, j]

        for est in estimators:
            c = color_map[est]
            n_vals = np.array(sorted(results_bias2[model][est].keys()))
            vals   = np.array([results_bias2[model][est][n] for n in n_vals])
            ax.loglog(n_vals, vals, linestyle='-', color=c, linewidth=2)

        for est in estimators:
            c = color_map[est]
            n_vals = np.array(sorted(results_var[model][est].keys()))
            vals   = np.array([results_var[model][est][n] for n in n_vals])
            ax.loglog(n_vals, vals, linestyle=':', color=c, linewidth=2)

        ax.set_title(f"{model}")
        ax.set_xlabel("Sample Size (n)")
        ax.set_ylabel("Bias² / Variance")
        ax.grid(True, which="both", ls="--", alpha=0.6)

    style_handles = [
        Line2D([0], [0], linestyle='-', color='black', lw=3, label='Bias²'),
        Line2D([0], [0], linestyle=':', color='black', lw=3, label='Variance'),
```

```
    ]
    est_handles = [
        Line2D([0], [0], linestyle='-', color=color_map[e], lw=3, label=e)
        for e in estimators
    ]
    handles = style_handles + est_handles
    labels  = [h.get_label() for h in handles]

    fig.legend(handles, labels,loc='lower␣
 ↪center',ncol=len(handles),frameon=False,bbox_to_anchor=(0.5, 0.
 ↪02),handlelength=2.8,columnspacing=1.0)
    plt.show()

########################################################################  tables ␣
 ↪#

def spx_results(data, window_size, trading_days_per_year=252):
    n = len(data)
    seg_len = n // 3
    segments = {
        'First Year': data[:seg_len],
        'Second Year': data[seg_len:2*seg_len],
        'Third Year': data[2*seg_len:],
        'Full Sample': data
    }

    results = []
    for name, seg in segments.items():
        centered = np.log(seg) - np.mean(np.log(seg))
        years = len(seg) / trading_days_per_year

        results.append({
            'Segment': name,
            'Days': len(seg),
            'Years': years,
            'MLE': chunk_array_mle(centered, window_size),
            'Power Variation': chunk_array_power(centered, years, window_size),
            'Wavelet': chunk_array_wavelet(centered, window_size),
            'Log-Log Regression': chunk_array_loglog(centered, window_size)
        })

    col1, colw = 20, 20
    header = "Segment".ljust(col1)
    for e in estimators:
        header += f"{e}".center(colw)
    print(header)
    print("-" * len(header))
```

```
    for r in results:
        row = str(r["Segment"]).ljust(col1)
        for e in estimators:
            row += f"{r[e]:.4f}".center(colw)
        print(row)

    return results

def print_all_MC_tables(all_results, H_vals=H_vals, T_vals=T_vals,␣
 ↪models=models, estimators=estimators):
    for T in T_vals:
        print(f"\n\n=== T = {T} years ===")
        for H in H_vals:
            print(f"\nH = {H:.1f}")
            results = all_results[(H, T)]

            header = "Estimator".ljust(20)
            for model in models:
                header += f"{model}".center(30)
            print(header)
            print("-" * len(header))

            for est in estimators:
                row1 = est.ljust(20)
                row2 = "bias".ljust(20)
                row3 = "std".ljust(20)
                row4 = "CI".ljust(20)

                for model in models:
                    res = results[model][est]
                    jb_star = format_jb_star(res['jb_pvalue'])

                    row1 += f"{res['mean_hat']:.4f}{jb_star}".center(30)
                    row2 += f"{res['bias']:.4f}".center(30)
                    row3 += f"{res['std']:.4f}".center(30)

                    ci_low, ci_high = res['conf_interval']
                    row4 += f"[{ci_low:.4f}, {ci_high:.4f}]".center(30)

                print(row1)
                print(row2)
                print(row3)
                print(row4)
                if est != estimators[-1]:
                    print("-" * len(header))
            print("-" * len(header))
```

```
def create_convergence_table(results_rmse, results_jb, alpha_results, n_max,␣
 ↪models=models, estimators=estimators):
    print(f"\n\n=== Convergence Results Table (n = {n_max}) ===")

    header = "Estimator".ljust(20)
    for model in models:
        header += f"{model}".center(30)
    print(header)
    print("-" * len(header))

    for est in estimators:
        row1 = est.ljust(20)
        row2 = "rate (±se)".ljust(20)

        for model in models:
            rmse_val  = results_rmse[model][est][n_max]
            jb_pvalue = results_jb[model][est][n_max]
            jb_star   = format_jb_star(jb_pvalue)
            rmse_str  = f"{rmse_val:.4f}{jb_star}"

            alpha_val, alpha_se = alpha_results[model][est]
            alpha_str = f"{alpha_val:.2f}±{alpha_se:.2f}"

            row1 += rmse_str.center(30)
            row2 += alpha_str.center(30)

        print(row1)
        print(row2)
        if est != estimators[-1]:
            print("-" * len(header))

    print("-" * len(header))
    return alpha_results

########################################################################## ␣
 ↪Formating  #

def format_jb_star(p):

    if np.isnan(p):
        return ""
    if p < 0.001:
        return "***"
    if p < 0.01:
        return "**"
    if p < 0.05:
        return "*"
```

```
    return ""

############################################################################  Outputs␣
 ↪ ############################################################################


############################################################################ ␣
 ↪Simulating fBM, Estimating H, Nu w/MLE  #
np.random.seed(seed)

plot_fbm_paths(fbm_H_values, np.linspace(0, T, fbm_n+1), T)
H_hat, nu_hat = mle_estimator(X)

print(f"MLE H = {H_hat:.10f}")
print(f"MLE   = {nu_hat:.10f}")

############################################################################ ␣
 ↪Simulating RSFV, Estimating H, Nu w/MLE  #

res05 = estimate_params(BH05, B05, W05)
res10 = estimate_params(BH10, B10, W10)
estimates = simulate_paths()

mean, ci_low, ci_high, bias, std = confidence_interval(H_stdBM, n_sim_stdBM,␣
 ↪estimates, p)

print("\nH=0.05 results:")
for m_val, vals in res05.items():
        print(f"m={m_val}: H_hat={vals[0]:.4f}, nu_hat={vals[1]:.4f},␣
 ↪relErr={np.abs(vals[0]-0.05)/0.05:.4f}")

print("\nH=0.10 results:")
for m_val, vals in res10.items():
        print(f"m={m_val}: H_hat={vals[0]:.4f}, nu_hat={vals[1]:.4f},␣
 ↪relErr={np.abs(vals[0]-0.1)/0.1:.4f}")

print("\nH=", H_stdBM, "(std BM) results:")
print(f"Mean H_hat: {mean:.4f}")
print(f"Std Dev H_hat: {std:.4f}")
print(f"{p}% CI: [{ci_low:.4f}, {ci_high:.4f}]")
print(f"Bias: {bias:.4f}")

print("\nBH05 Simulation")
plot_RFSV_path(BH05, B05, W05)

print("\nBH10 Simulation")
```

```
plot_RFSV_path(BH10, B10, W10)

########################################################################  RFSV␣
 ↪Implied Volatility Skew  #

x = np.linspace(-0.5, 0.5, 11)
K = S0 * np.exp(x)
L, correction, dt, sqrt_dt = simulate_rough_bergomi_paths()
call_prices = compute_call_prices(L, correction, dt, sqrt_dt, K)
implied_vols = compute_implied_vols(call_prices, K)

results = pd.DataFrame({
    'Log-Moneyness (x)': x,
    'Strike Price (K)': K,
    'Call Price': call_prices,
    'Implied Volatility': implied_vols
})
print("\nImplied Volatility Smile Results:")
print(results.round(4))

plot_smile(x, implied_vols)

######################################################################## ␣
 ↪Estimating H on spx w/MLE  #

realized_var1 = realized_var3[-251:]

log_realized_var3 = np.log(realized_var3)
centered_log_var3 = log_realized_var3 - np.mean(log_realized_var3)

log_realized_var1 = np.log(realized_var1)
centered_log_var1 = log_realized_var1 - np.mean(log_realized_var1)

H_MLE3spx, nu_MLE3spx  = mle_estimator(centered_log_var3, 3)
H_MLE1spx, nu_MLE1spx  = mle_estimator(centered_log_var1)

print("\nMLE Estimates w/SPX volatility data")
print("3 Years :  " f"H = {H_MLE3spx:.4f},  nu = {nu_MLE3spx:.4f} ")
print("Final Yr:  "f"H = {H_MLE1spx:.4f},  nu = {nu_MLE1spx:.4f} ")

######################################################################## ␣
 ↪Estimating H on SPX w/Estimators w/Subwindoes  #

print("\nSPX Hurst Exponent: Yearly Segments vs Full Sample")
spx_results = spx_results(realized_var3, spx_window)
plot_spx_hurst(spx_results)
```

```
######################################################################  MC␣
 ↪Estimators Bias  #
np.random.seed(seed)

all_results, all_raw = collect_monte_carlo_data()
print_all_MC_tables(all_results)

plot_line_H(all_results)
plot_boxplots(all_raw, H_vals, T_vals[0])
plot_histograms(all_raw)

###################################################################### ␣
 ↪Estimator Convergence  #

results_rmse = {model: {est: {} for est in estimators} for model in models}
results_bias2 = {model: {est: {} for est in estimators} for model in models}
results_var   = {model: {est: {} for est in estimators} for model in models}
results_jb    = {model: {est: {} for est in estimators} for model in models}

for n in conv_N:
    print(f"\nRunning n={n} (T={conv_T}, H={conv_H})")
    results, _ = monte_carlo_bias(conv_H, conv_T, conv_M, n, conv_window)

    for model in models:
        for est in estimators:
            res = results[model][est]
            b2  = res['bias']**2
            var = res['std']**2
            mse = b2 + var
            rmse = np.sqrt(mse)

            results_rmse[model][est][n] = rmse
            results_bias2[model][est][n] = b2
            results_var[model][est][n]   = var
            results_jb[model][est][n]    = res['jb_pvalue']

n_max = max(conv_N)
alpha_results = {model: {est: (0.0, 0.0) for est in estimators} for model in␣
 ↪models}

for model in models:
    for est in estimators:
        n_vals   = np.array(sorted(results_rmse[model][est].keys()))
        rmse_vals = np.array([results_rmse[model][est][n] for n in n_vals])

        ln_n    = np.log(n_vals)
```

```
        ln_rmse = np.log(rmse_vals)
        X = sm.add_constant(ln_n)
        fit = sm.OLS(ln_rmse, X).fit()

        beta = -fit.params[1]
        beta_se = fit.bse[1]
        alpha_results[model][est] = (beta, beta_se)

create_convergence_table(results_rmse, results_jb, alpha_results, n_max)
plot_rmse_convergence(results_rmse, conv_H, conv_T)
plot_bias_variance_decomposition(results_bias2, results_var, conv_H, conv_T)


#end
```