Modeling Options Prices considering Mandelbrotian Movement of Prices

Andrea Dalseno[1], Vsevolod Cherepanov[2], and Sanchit Jain[3]

[1]MScFE student at World Quant Univeristy, Malta, adalseno@gmail.com
[2]MScFE student at World Quant Univeristy, Japan, cherepanovvsb@gmail.com
[3]MScFE student at World Quant Univeristy, India, jsanchit1103@gmail.com

Abstract

This study is centered around the practical exploitation of Mandelbrotian price movements. Our objective was to implement the famous Multifractal Model of Asset return (MMAR) proposed by Benoit Mandelbrot in 1997 in order to and simulate future price behavior. Based on the simulation we developed a robust trading strategy and evaluated its efficacy in terms of cumulative profit as well as we compared the strategy with different benchmarks. One of the challenge was refining the estimation methodology for the Hurst exponent used in fractional Brownian Motion, thereby enhancing predictivity of the simulation. Subsequently, we employed machine learning approach which gave us profitable buy/sell trading signals. Thus, our final MMAR-LightGBM model generated positive cumulative profits for trading strategy based on at-the-money options of SPDR S&P 500 ETF Trust.

Key words: *Options Pricing, Mandelbrot, Multifractal Model of Asset Returns, MMAR*

1 Introduction

The efficient analysis and prediction of price actions have long been the focus of both academia and business. Stock Markets are an important being in the world today. Economists and Mathematicians have tried to study and predict the future of these instruments since ages. Stock markets are often governed by events which are of the past, anticipated events of the future , sometimes just a rumour and maybe some general sentiments. Turning it into an actual science is far for now. Bachelior (1900) was a pioneer who made assumption of independent price changes which are distributed according to a Gaussian. Though Gaussian is one of the most celebrated bride of the probability academia , financial time series has disappointed it empirically. One of the main reasons behind the same is high kurtosis i.e. fat tails. Another digression from Gaussion is temporal dependence between periods of high volatility and low volatility Mandelbrot, Fisher, and Calvet (1997).

Many models have been introduced thereafter including the ARCH/GARCH and their modifications/improvements, incorporating asymmetric behaviour of positive and negative shocks, Nelson (1991),long memory Baillie, Bollerslev, and Mikkelsen (1996) and statistically representing different time scales Drost and Werker (1996). Following this Benoit Mandelbrot introduced a Multifractal Model of Asset return (MMAR) (1997), which in words of Mandelbrot shows a warped time horizon of the clock time to something called as the trading time.

> "My model redistributes time. It compresses it in some places, stretches it out in others. The result appears very wild, very random. The two functions, of time and Brownian motion, work together in what mathematicians call a compound manner:

> Price is a function of trading time, which in turn is a function of clock time. Again, the two steps in the model combine to produce a "baby" far different from either parent." Mandelbrot and Hudson (2004)

as explained by Benoît B. Mandelbrot in his book "The (Mis)Behavior of Markets" encapsulates the idea in an ideal fashion.

The 1997 Mandelbrot paper Mandelbrot, Fisher, and Calvet (1997) is mathematically intensive. Later people have tried to simplify and elaborate the mathematical premise for example Jeon (2021). We would be relying on the original paper as well as this paper for our mathematical notation and understanding. Implementating MMAR for financial time series is an extensive task, which is described stepwise in the 2019 thesis of Sibirtsev (2019). The thesis has been also logically presented in the interesting medium article of Todd Moses (2021).

The Hurst exponent is the measure of the long term memory of a time series. Its application originated in hydrology to address the practical concern of determining the optimum dam sizing for the Nile river's volatile rain and drought conditions. The term "Hurst exponent" stems from the name of the lead researcher in these studies. Wikipedia. This was borrowed by Mandelbrot for fractal geometry and hence for MMAR. The main problem with the hurst exponent is that it is not calculated as such , but estimated using various methods. A python library for specifically estimating the Hurst exponent is present here Dmitry Mottl which uses RS analysis for estimation. The RS analysis for estimation has been described by Mansukhani (2012).

Our main intervention would be to improve the estimation of Hurst exponent and the other parameters required to build a Multifractal Model of Asset returns starting from actual data. We will see the current work on this in the next section.

2 Competitor Analysis

Wikipedia (2023) mentions the following methods to calculate Hurst Exponent: DFA, Periodogram regression, aggregated variances, local Whittle's estimator, wavelet analysis, both in the time domain and frequency domain. Another paper by Zhang and Feng (2023) surveys thirteen odd methods of estimating Hurst component and compares them on ideal time sequences. The methods are summarized in figure 1.

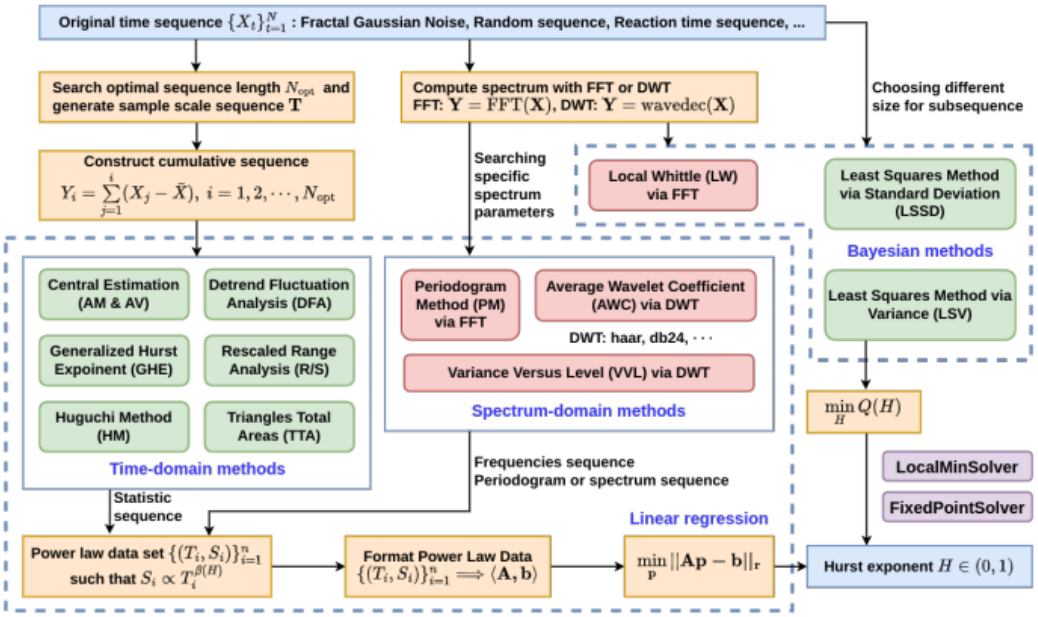


Figure 2: Typical Algorithms for Estimating Hurst Exponent

Table 2: Classification of 13 dominant estimation methods

Method / Para. Est. / Repr.	**Linear Regression Method**	**Bayesian Method**
Time-domaib	AM, AV, GHE, HM, R/S, DFA, TTA	LSSD, LSV
Spectrum-domain	PM, AWC, VVL	LW

Figure 1: Source: Zhang et al.

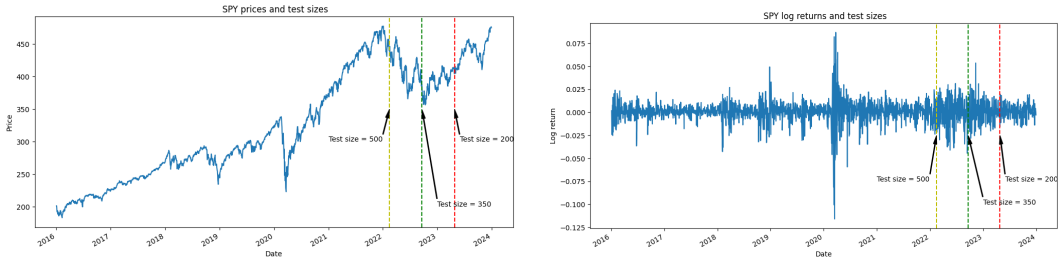
The code for their Hurst estimation could be found at github which we would be using.

Our main intervention would be to apply these methods in a Financial Series framework to see what benefits the accurate estimation of Hurst Exponent would provide in a trading strategy. Another intervention would be to incorporate the Kolmogorov-Smirnov (KS), based on the Kolmogorov-Smirnov statistic, which can be applied to any Hurst exponent estimation based on equality in distribution as an improvement mentioned in the 2022 paper Gómez-Águila, Trinidad-Segovia, and Sánchez-Granero (2022).

The multifractal concept, together with the MMAR model has been widely applied to the financial sector: for example in cryptocurrencies, Filipa and Vaz (2021), Jiang, Dev, and Maller (2020) or Saâdaoui (2023), in the stock market, Günay (2016) or Maglione (2012), in the exchange rates, Garcin (2019) or Sibirtsev (2019), and in option pricing, Nnalue et al. (2020). Particularly interesting is Maglione's work in which he describes the process for obtaining the parameters of the MMAR model starting from actual prices, using, however, R and the work of Sibirtsev who tried to do the same thing using Python, even if the resulting code was, in our opinion, quite confusing and not well structured. Moreover, on GitHub there are some Python packages dedicated to the Fractional Brownian Motion, such as fbm, fractalmarkets, or stochastic. None of them, however, provides a convenient method to build an MMAR model starting from actual data.

3 Materials and methods

We used data from Yahoo Finance for stock prices, and from OptionsDX for the option chains. We concentrated on the *SPDR S&P 500 ETF Trust*, ticker **SPY**, an Exchange Traded Fund that tries to replicate the S&P index. It's an actively traded instrument, with fair volumes, therefore liquid, and, as an index, is less subject to sudden movements that typically affect single stocks. We used the daily data for our research, with a date range from 01-01-2005 to 10-2-2024 for the ETF and the years 2017-2023 for the options. To assess the performances of our model, we used three different test sizes with completely different trends: 200, 350, and 500 days. Figure 2 shows the test sizes for the prices and the log returns.



(a) Test sizes and prices

(b) Test sizes and log returns

Figure 2: The test set goes from the vertical line to the end of the series

We had to perform data wrangling for the option chain file, first to build the dataframe from several CSV[1] files in different directories, then to select only the needed columns in the desired format. Option chain files, in fact, have an enormous number of rows: for each day we have several strike prices and several expiration dates. Just to give you an idea, one month of data contains on average around 64,000 rows. We decided to concentrate on ATM options: fat tails distributions, such as log returns, have a high kurtosis, thus ATM probabilities are higher than expected for a Normal distribution. There are, obviously, significant differences in the tails too, but we are still speaking of rare events, too sparse to feed a machine learning model. The two datasets, the prices from Yahoo Fiance and the option chain one, have minor differences in the closing prices[2], but small enough to be safely ignored. We decide to use a monthly expiration date, that is about 21 days for two reasons: a value long enough to appreciate price differences and, at the same time, short enough to quickly compute[3] and to have enough data to train the machine learning models. Using a different time horizon for the data, for example 5 or 15 minutes, allows to apply the model to shorter expiry dates, or weekly data, to longer ones.
We performed a check to see how many simulations were needed to obtain a reasonable confidence interval from the Monte Carlo simulation: the bigger the number of simulations, the narrower the confidence interval. We found out that for the 21 days horizon, 500,000 simulations were a good compromise between speed and accuracy[4].
Using the MMAR class, we simulated the price distribution for each day and price, then computed the expected Call price[5] given the interest rate, collected form the FRED site[6]. Before that, however, we tested the MMAR class on different tickers to be sure that it worked as ex-

[1] Comma separate values

[2] We compared both the Close price and the Adjusted Close, and found out that the correct value to use is the Close price.

[3] We used Monte Carlo simulation and the number of steps (days) increase exponentially the computing time.

[4] At present the code does not use GPU, but uses *Numba* to speed up the calculations.

[5] We concentrated only on Call options, but the methodology can be applied to Puts too.

[6] The Federal Reserve of St. Louis

pected[7] and the outcome of the model was reasonable[8].
With the results from the simulations we built a dataframe containing as many statistical data as possible (see A.1 for the structure of the file). The description of the methodology used to build the file, and the pseudocode are in A.2. We got a total of 1,508 observations. Not a huge number for Machine Learning models, and definitively not enough to use Deep Learning, especially considering that using a test size of 500 leaves on 1,008 observations for the train set, in its turn split into train and validation set. As a matter of fact, we struggled to find a suitable model for our predictions[9]. The best model was LightGBM, Ke et al. (2017), which is also extremely fast to train. We used *Cross Validation*[10] and *Optuna*, Akiba et al. (2019), to find the best *hyperparameters*. Despite our efforts, however, the model has a high tendency to overfit[11]. Since we wanted to reduce the *False Positive Rate* as much as possible[12], we used the *predict_proba* method and computed, based on the train data, a threshold for the given maximum acceptable *False Positive Rate*, and then used the computed value for the final predictions[13]. We performed the model tuning for each test size obtaining different *hyperparameters* and different feature importance, see figure 3.

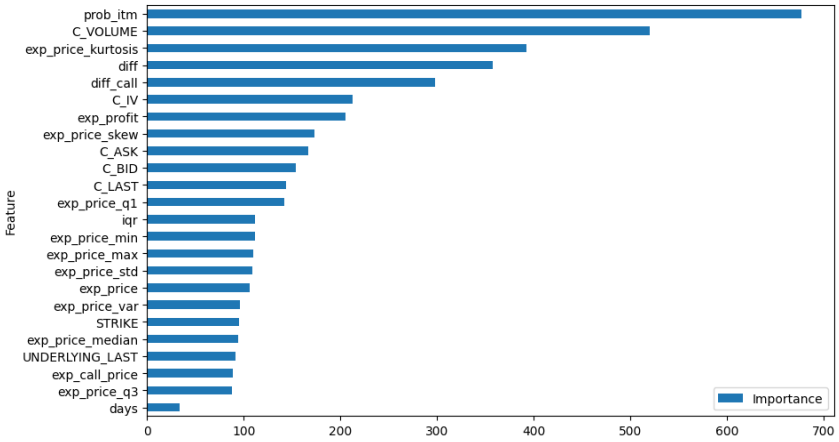


Figure 3: Example of feature importance for the 200 point test size

3.1 The MMAR class

Following the implementation used by Maglione (2012), we created the MMAR class to compute the required parameters for a Monte Carlo simulation following the *Multifractal Model of Asset Returns* starting from actual data. The original work used R as a programming language. We freely translated it into Python, using ChatGPT, OpenAI (2024), to suggest a Python replacement for the R functions. In addition to the "canonical" implementation to compute θ[14]

[7] Refer to the *test_mmar.ipynb* notebook in the GitHub repository.

[8] Refer, for example, to *check_mc_simulation.ipynb* notebook.

[9] We also tried Logistic Regression, Random Forest and CatBoost, See Prokhorenkova et al. (2017). The examples can be found in the experiments branch on GitHub,

[10] From a minimum of 5 folds to as much as 12

[11] We not only took the mean of each folding round, but also added a penalty term based on the standard deviation. See the *objective_lightgbm* function in the *utility_functions* module.

[12] That is predicting 1, Buy, when the true value is 0.

[13] Usually the predicted class is 1 if the probability is bigger than 0.5, 0 otherwise; using the threshold we can change the default behavior increasing or decreasing the value. For example, we may decide to predict 1 only if the probability is bigger than 0.7. In this way, we reduce the number of false positives, and likely the number of true ones. That is, we consider the cost of a false positive higher than the opportunity cost of missing a true positive. As shown in figure 5, the cost of a wrong prediction, the negative profit, is usually much higher than the positive profit for a right prediction.

[14] See section 4.

based on Random Normal, we used the method suggested by Batten, Kinateder, and Wagner (2014) based on Volume data. According to our tests, the latter provides a better way to simulate the price distribution, closer to reality, and more consistent since it does not rely on a random number different at each simulation[15].

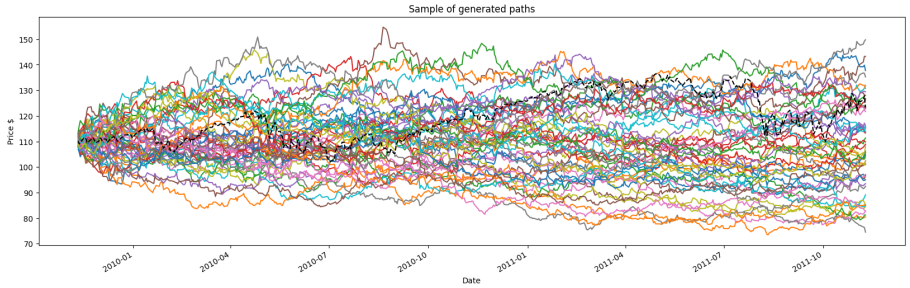


Figure 4: Example of simulated path using MMAR class and volume to compute θ

3.2 The machine learning model

We built a target dataframe in a simple way: if the final price, estimated using the MMAR class, is bigger than the *Strike price* plus the *Call price*, we should Buy (class = 1), otherwise not (class = 0), see figure 5.

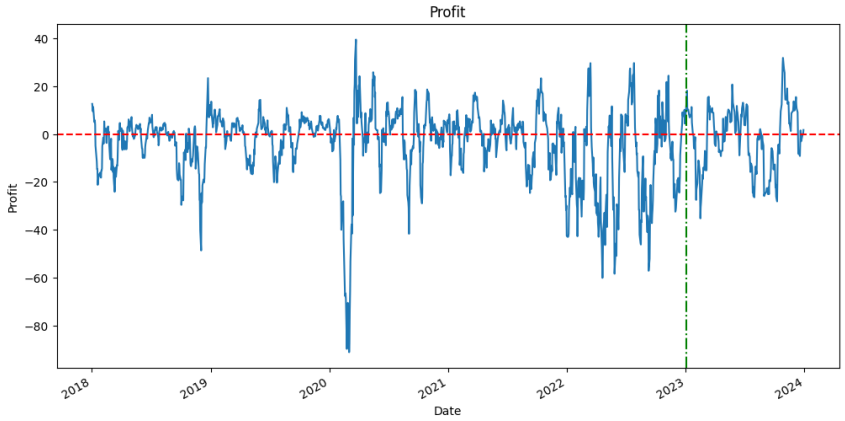


Figure 5: Estimated profit

The result was a fairly balanced dataset, see figure 6.

[15]Refer to *compare_theta.ipynb* notebook.

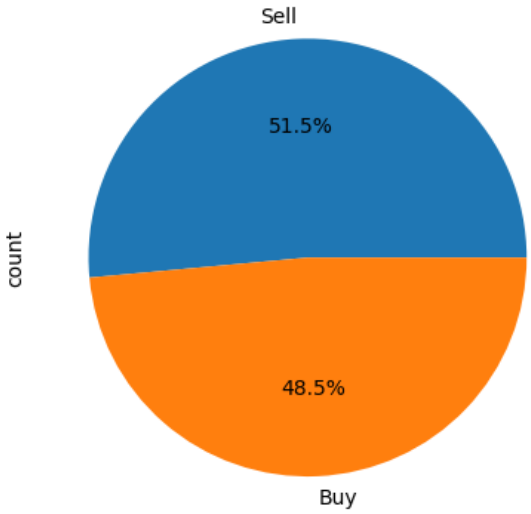


Figure 6: Class distribution for the selected target

3.3 Boosting the LightGBM strategy with Reinforcement Learning

In order to improve performance of the strategy based on LightGBM, we employed Upper Confidence Bound algorithm (UCB1) from k-armed bandit algorithms family. The decision making strategy involved the dilemma if it's better to buy underlying asset (S&P500 ETF) or S&P500 option in case it was suggested to buy by the LightGBM model. The bandit was updated after each option maturation. We calculate UCB using the formula:

$$\text{UCB1}(i) = \bar{x}_i + C\sqrt{\frac{\ln N}{n_i}} \tag{1}$$

UCB1(i) - Upper Confidence Bound for arm i;

$\bar{x}_i$ - Average reward obtained from arm i;

N - Total number of plays or rounds so far;

n_i - Number of times arm i has been played.

C - confidence interval (we used $C = 3$)

The algorithm updates the estimated value of an arm using running average. In total our k-armed bandit had two arms: buy S&P500 ETF or buy S&P500 option. Fees and spread were not considered in the model.

4 The MMAR model

While the *fractal* model addresses the problem of fat tails, the variance cluster remains uncovered by that model. For this reason Mandelbrot, Fisher, and Calvet (1997) developed the Multifractal Model of Asset Returns (MMAR). The central idea of the model is that the trading

time does not follow the clock time, but it shows moment of intense activity followed by moment of moderate activity. The main formula is:

$$X(t) \equiv B[\theta(t)] \tag{2}$$

where "the index t denotes clock time, and $\theta(t)$ is called trading time or the time deformation process" Mandelbrot, Fisher, and Calvet (1997).

4.1 The steps to build the model

First of all, in the model we work with the log returns in the form:

$$X(t) = \ln P(t+1) - \ln P(t) \quad [P(t); 0 < t < T] \tag{3}$$

and we build the stochastic process as:

$$X(t) \equiv B_H[\theta(t)] \tag{4}$$

where $B_H(t)$ is a classical fractional Brownian motion, while $\theta(t)$ is stochastic trading time Mandelbrot, Fisher, and Calvet (1997).

To simulate a MMAR process we need to estimate[16] four parameters:

- H: the Hurst exponent;
- α: the scaling (Hölder) exponent;
- μ: the mean of the series;
- σ: the standard deviation of the series.

The first step, we need to define some values for q, the central moments. Then, taking advantage of the definition:

$$\mathbb{E}\left[|X(t)|^q\right] = c(q)t^{\tau(q)+1} \tag{5}$$

we can compute $\tau(q)$, the so called *scaling function*, using the definition of the *partition function*:

$$\mathbb{E}\left[S_q(T, \Delta_t)\right] = c(q)\Delta_t^{\tau(q)+1} \tag{6}$$

The partition function $S(q,h)$ of a time series process $X(t)$ with $\leq 1 \leq t \leq T$ is obtained by *dividing*[17] the series into $N = \lfloor \frac{T}{h} \rfloor$ _non overlapping sub-intervals of length $h \in [1, T]$ using the formula below:

$$S(q,h) = \left[\sum_{i=0}^{N-1} |X_{i\times h+h} - X_{i\times h}| \right]^q, \quad h \in [1, T] \tag{7}$$

[16] the parameters cannot be computed, but only estimated, for example See Goddard and Onali (2014).
[17] partitioning

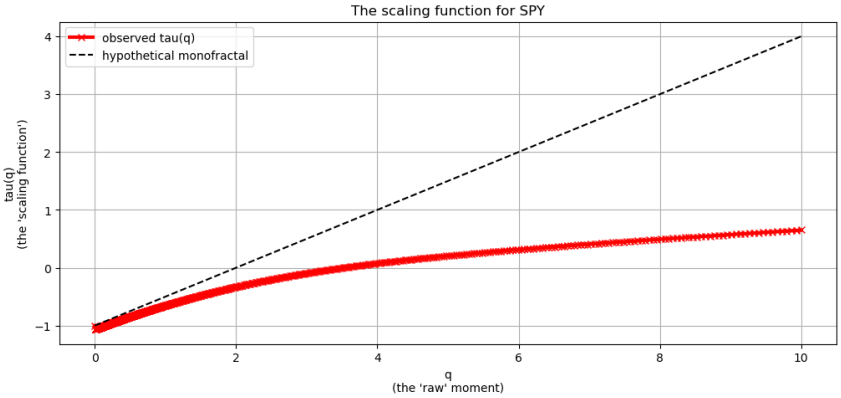


Figure 7: An example of scaling function for the SPY

After having computed the values for the *partition function*, we can infer $\tau(q)$ using a linear regression[18].

Having estimated q and $\tau(q)$, we can estimate α, the *scaling exponent* and then, via Legendre transformation, $\hat{f}(\alpha)$, the *multifractal spectrum*. Now we have all the elements to compute $\hat{\alpha}_0 = \hat{f}^{-1}(max(\hat{f}(\alpha))$, the Hölder exponent such as $\hat{f}(\alpha_0) = 1$. Since

$$H = \frac{1}{\alpha_0} \tag{8}$$

we can estimate the Hurst exponent, and, lastly[19],

$$\mu = \frac{\widehat{\alpha_0}}{\hat{H}} \text{ and } \hat{\sigma}^2 = \frac{2(\mu - 1)}{\ln b}$$

where b is a positive integer, usually set to 2[20].

To complete our simulation we need to estimate a last parameter, θ, the trading time function. In the original paper a random function is used to estimate θ, but in doing so at each run we will get a different value that may lead to extremely different outcomes (distributions), making our process unreliable. We may fix a specific seed, but in this way we would limit the stochastic nature of the process, without any clue about which might be the optimal seed. Or we may leverage the central limit theorem and compute the mean of a huge number of simulations, but it would make to computation heavy[21]. An alternative has come from a paper by Batten, Kinateder, and Wagner (2014) where the Volume[22] is used to estimate θ, with the following formula:

$$\overline{V_t} = \frac{V_t}{\sum_{t=1}^{T} V_t} \tag{9}$$

We decided to concentrate to the ATM options, since the MMAR model returns a fat tailed series, with high Kurtosis (see figure 8), so the ATM may suffer of a poor estimation for any model based on a Normal assumption, such as the Black & Scholes model, figure 8[23].

[18] where $\tau(q)$ is the slope of the regression line

[19] In some cases μ is defined as λ.

[20] The combination of b and k allows us to estimate up to b^k data points.

[21] If we want to compute one million simulation, each one with one million simulation, we will need to compute one trillion paths.

[22] The normalized volume. The paper used the Forex exchange rate data, but the reasoning seems to fit well for stocks, too.

[23] The figure code is available in Initial_test_MMAR_class.ipynb in our GitHub repository.

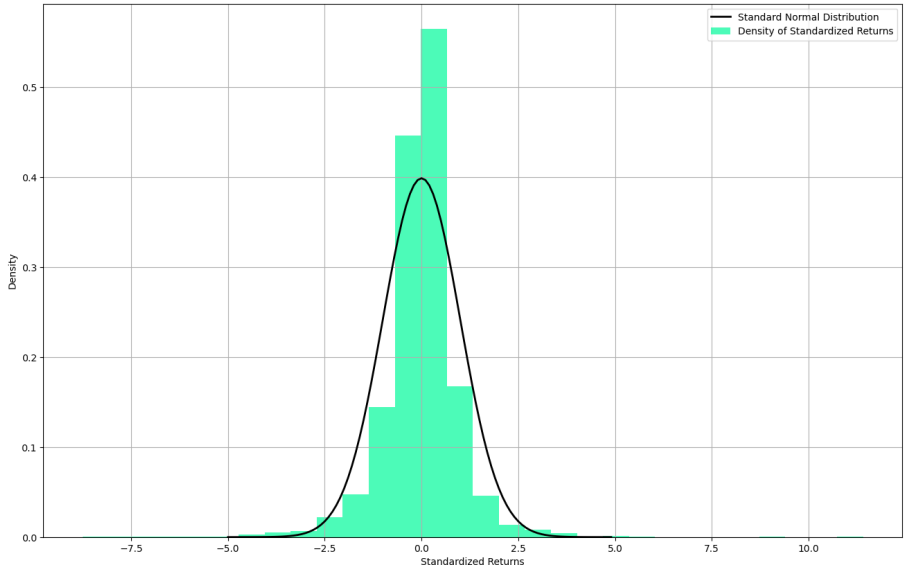


Figure 8: The standardized log returns distribution vs a Normal one

4.2 Some caveats about the model

4.2.1 The importance of the time scales

The choice of the time scale values plays a pivotal role in the computation of the scaling function. Unfortunately, there is not a defined rule to choose those values; a general rule of thumb is to select a wide range of values in order to capture both short-term and long-term fluctuations. In the original paper Maglione (2012) used an arbitrary range of values, [2, 3, 4, 5, 10, 15, 20], which was totally unfit in our case, see figure 9. Extending the range with some additional, bigger, values ([2, 3, 4, 5, 10, 15, 20, 50, 100, 300]) gave more reasonable results. Since we are dealing with daily data, a range like [1, 5, 10, 15, 21, 63, 126, 252], i.e. daily, weekly, bi-weekly, three-week, month, quarter, half-year, and year, seems a good choice. We opted for a more *time agnostic* method: the divisors of the time series. For this reason, the length of the original series is trimmed, during class initialization, to a value with a sufficient number of divisors[24].

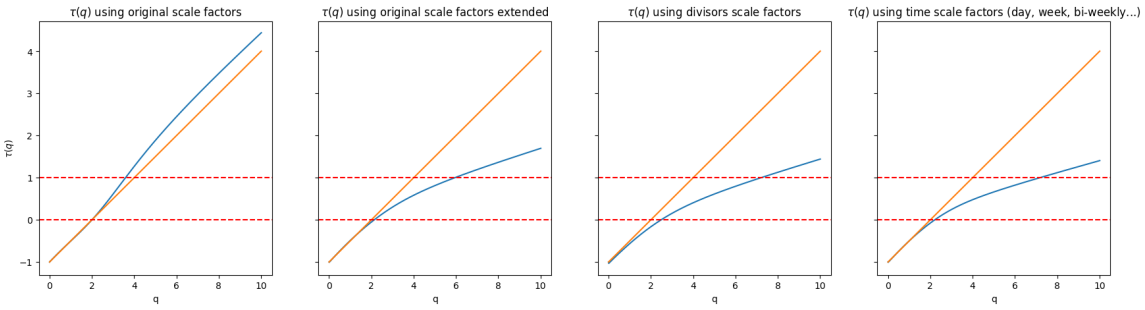


Figure 9: Observed $\tau(q)$ for different partition ranges

4.2.2 The simulated data

We performed a series of tests with the simulated data and compared them (the standardized log returns, actually) with the original data using the *Kolmogorov-Smirnov* test for goodness

[24] In our case the original time series had 3,273 observations. The divisors [1, 3, 1091, 3273] were not enough, and the series was trimmed to 3,272.

of fit for different simulation lengths. For all selected values, the match is almost perfect, see figure 10.

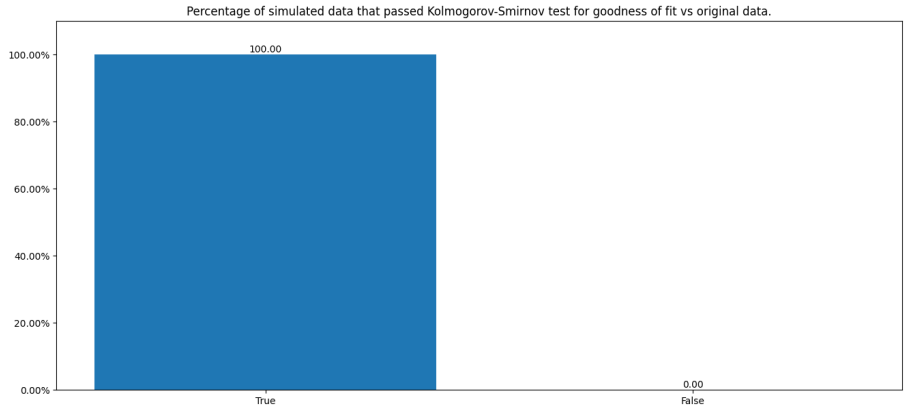


Figure 10: Percentage of simulated data that passed Kolmogorov-Smirnov test for goodness of fit vs original data.

However, especially for shorter intervals, the standardized log returns follow an almost Normal distribution. This is no longer true for bigger values, such as 2,048 steps (simulated days), see figure 11

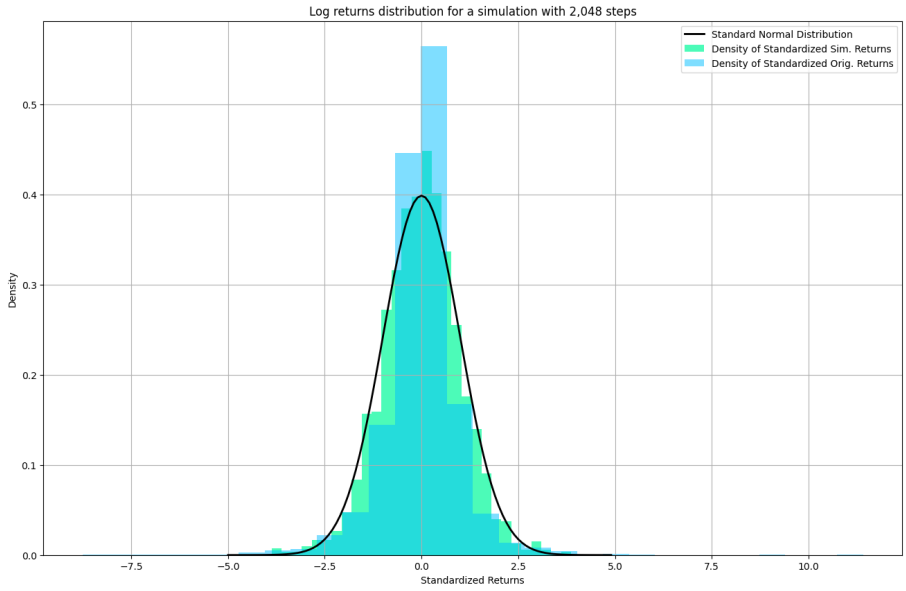


Figure 11: Log returns distribution for a simulation with 2,048 steps.

With one million simulations of 252 steps(days), the simulated log returns have wide range of *(Fisher's) Kurtosis* and *Skewness*, as shown in table 1. In table 2, instead, we can see the values of Skewness and Kurtosis for the original log returns compute in chunks of the same length (252 days). On average, the original returns show a higher *Kurtosis* and left *Skewness*, while the simulated data show lower average *Kurtosis* and slightly positive *Skewness*, together with more extreme fluctuations for both[25].

[25]see notebook check_mc_simulation_year.ipynb in our GitHub repository.

	Kurtosis	Skewness
mean	0.281136	0.000704
std	0.461464	0.190210
min	-0.830049	-1.222896
25%	-0.027780	-0.121822
50%	0.203358	0.000657
75%	0.496865	0.124739
max	8.098781	1.127628

Table 1: Simulated.

	Kurtosis	Skewness
mean	2.107914	-0.265645
std	1.265798	0.267305
min	0.788667	-0.544498
25%	1.328798	-0.457191
50%	1.891251	-0.372867
75%	2.393861	-0.053320
max	5.637375	0.287554

Table 2: Original

This is mainly due to the fact that θ, that is the trading time function, is computed using the Volume[26], but the simulated data use only a small fraction of it, namely the latest n data[27] and is, therefore, partial. Using the maximum admissible[28] number of steps, i.e. 2,048, the values for the simulated data are closer, see table 3.

	Kurtosis	Skewness
mean	0.920611	-0.002696
std	0.339303	0.096850
min	0.113021	-0.497040
25%	0.692311	-0.066946
50%	0.867238	-0.003204
75%	1.089176	0.059949
max	4.135019	0.419596

Table 3: Kurtosis and Skewness for the simulated data (10,000 simulations with 2,048 steps).

In light of the above, the ideal strategy would be to remove the limitation, in case of using the Volume to calculate θ, and to simulate a number of steps as close as possible to the number of observations of the original series.

4.2.3 Call price formula

The usual way to get a Call price from a Monte Carlo simulation is using the following formula:

$$P_{Call} = \frac{1}{n}\sum_{i=1}^{n} \text{maximum}(S_T - K, 0)$$

we preferred to write it as

$$P_{Call} = \mathbb{E}(\text{Payoff } | S_T > K) \cdot P(S_T > K)$$

It's just a different way to express the same quantity. In fact,

$$\mathbb{E}(\text{Payoff}) = \frac{1}{m}\sum_{i=1}^{m} S_T - K, \text{ and } P(S_T > K) = \frac{m}{n}$$

[26] Actually, for consistency with the original method, a fraction of the Volume series using the biggest power of 2 smaller than or equal to the length of the time series; in our case, 2,048: $2^{11} = 2048 < 3272$.

[27] In this case 252.

[28] According to the self imposed limitation described above.

4.2.4 Fractional Gaussian Noise vs Gaussian Noise

For computational reasons and reproducibility, we opted to use a standard Gaussian Noise instead of a Fractional Gaussian Noise (fgn). There are several implementations to estimate an *fgn* process[29], see for example Dieker (2004), but we struggled to find a suitable way to obtain consistent outcomes together with acceptable computing times. For short time horizons, like the one used in our simulations (21 days), the difference becomes tiny, and the benefit of obtaining replicable results far outweighs, in our view, the disadvantages, see figure 12.

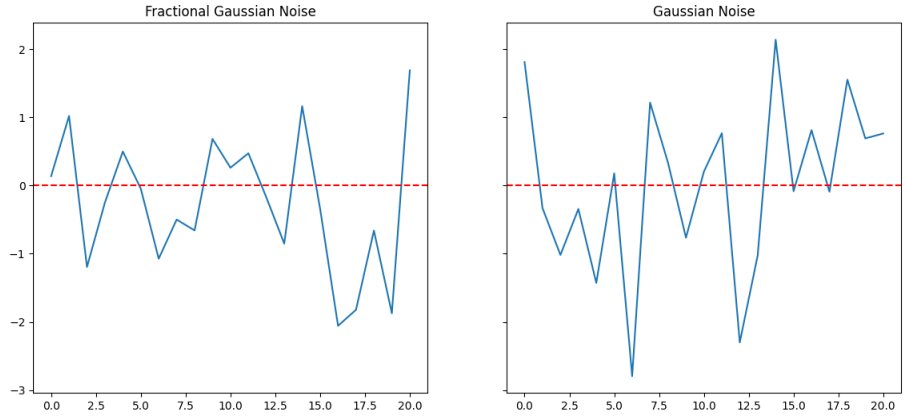


Figure 12: Comparison between FGN and GN for 21 steps and Hurst exponent = 0.45.

5 Results and conclusion

5.1 Backtesting

We performed a back-testing on the unseen data to verify if the addition of the data obtained from the MMAR simulation was able to increase the performance. We compared a baseline strategy, that is buying the ATM Call each day with the control, that is the predictions obtained using LightGBM and just the option chain data, and with the predictions obtained using LightGBM with our data. Overall our model performed quite well, in particular during downswings, such as in the 350 point test size, where we clearly outperformed both alternative strategies avoiding the losses, see figure13.

[29]The Fractional Gaussian Noise is defined as $B_H(t) - B_H(t - \Delta t)$, that is the first derivative of a Fractional Brownian Motion $B_H(t)$, which is not existent See Sottinen (2003)

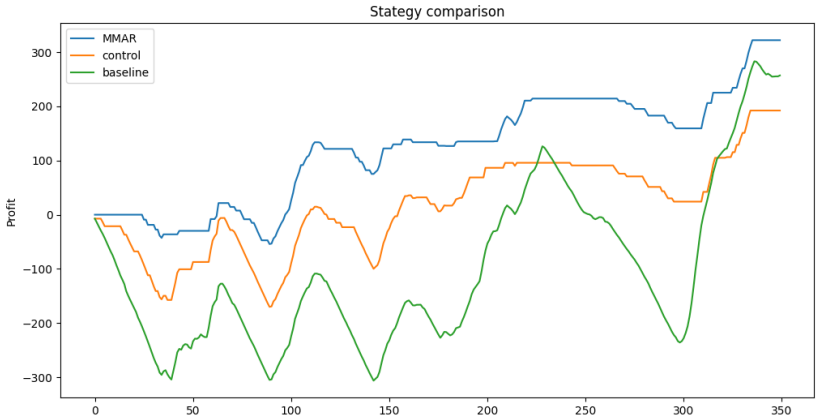


Figure 13: Backtesting of different strategies, test size 350

We got similar results, outperforming both alternative strategies, but without avoiding losses (just reducing them) for the 500 day test size, see figure 14.

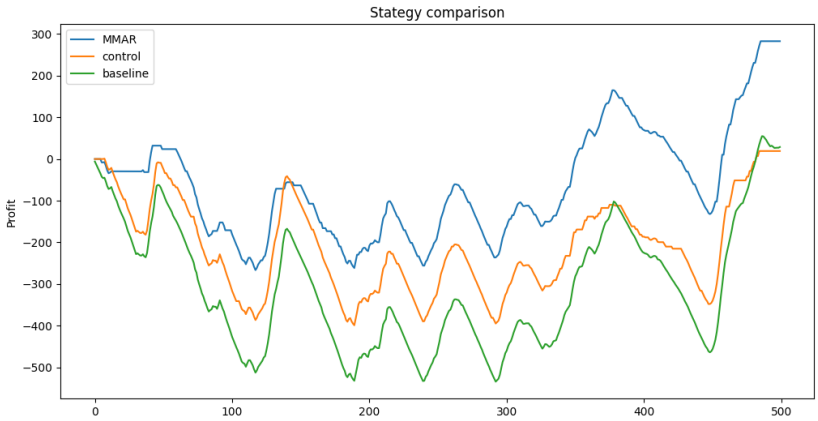


Figure 14: Backtesting of different strategies, test size 500

In the case of the 200 day test size, mainly characterized by an upward trend, the baseline strategy offered an higher final cumulative result even if it showed big swings. Our model outperformed the control strategy, and showed less variance in the cumulative results, see figure 15.

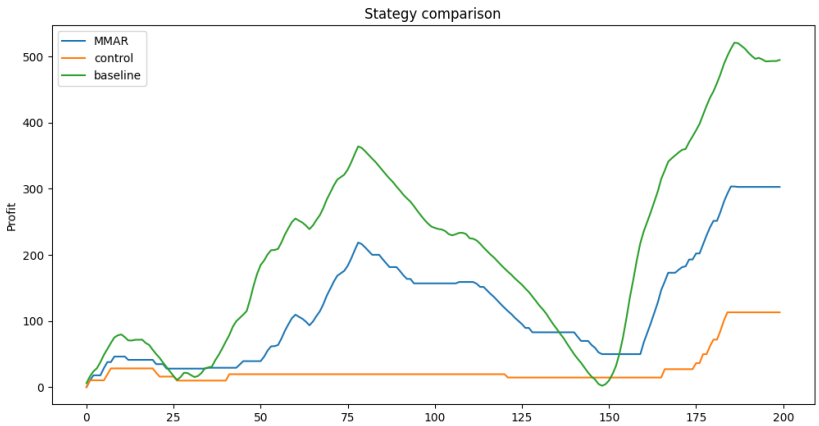


Figure 15: Backtesting of different strategies, test size 200

Future works may investigate if adding more data can increase the model reliability and reduce overfitting, or if different and more sophisticated strategies can provide better performances, maybe involving Put options. As far as we know, this is the first full working example to build simulated data starting from actual time-series in Python. The simulated data, with their fat tails, may be extremely useful for risk management and VaR models.

5.2 Performance of UCB1 algorithm

The best performance was reached with the LightGBM model with 200 sample size (see the figure below). Before rewards became negative the algorithm were choosing the options. Thus, overall performance was increased. In the figure below Underlying - S&P500 ETF; the Strategy - LightGBM model prediction; Bandit - UCB1.

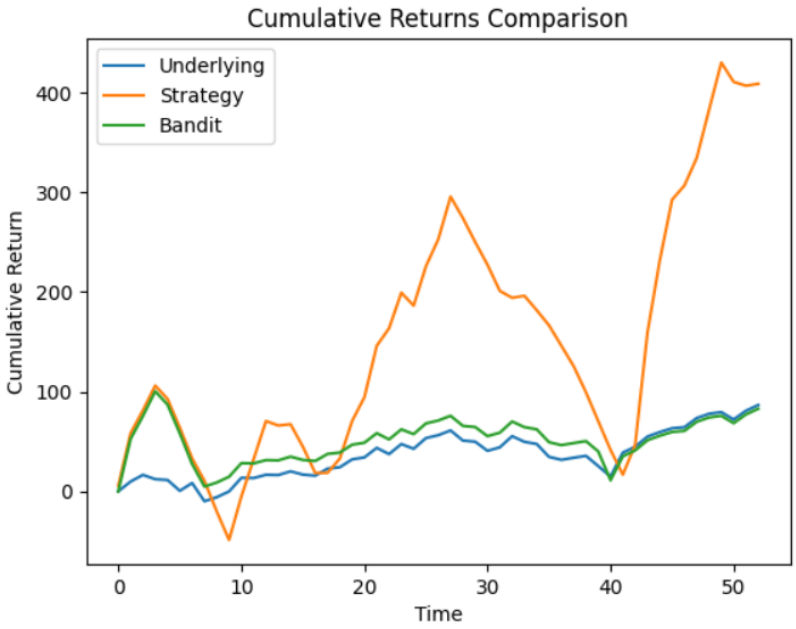


Figure 16: Cumulative returns from UCB1 algorithm

5.3 Conclusions

We think that the MMAR class is something new and convenient in the Python ecosystem. The simulated data proved to be effective in improving option trading strategies with better overall performances and reduced swings. More work is needed to refine the class, improving computational speed, possibly using GPUs when available, and memory usage. As described in the caveats, we should find a consistent and effective way to replace the Gaussian Noise with a Fractional Gaussian Noise. We should also find a suitable way to implement (shrink) the *Volume model* for θ in the case of simulations with a small number of steps, which is, at present, far from optimal.

A Appendix

A.1 Built DataFrame

Field	Description	Type
Date	Date	date
last_quote	latest underlying price	float64
strike	strike price	float64
exp_price	expected price according to simulation	float64
exp_call	expected Call price	float64
exp_price_min	expected Call price minimum	float64
exp_price_max	expected Call price maximum	float64
exp_price_std	expected Call price Standard Deviation	float64
exp_price_median	expected Call price median	float64
exp_price_q1	expected Call price Q1 (first interquartile)	float64
exp_price_q3	expected Call price Q3 (third interquartile)	float64
exp_price_kurtosis	Kurtosis of simulated data	float64
exp_price_skew	Skenewss of simulated data	float64
days	Days to expiration	int64
r	Intersest Rate	float64
prob_itm	Probability that the simulation closes ITM	float64

Table 4: MMAR file structure

A.2 Pseudocode for DataFrame

We must identify which option data to use: for every day, we have many possible expiration dates and strike prices. We created two columns:

- diff: the absolute difference between the strike price and the underlying price;
- diff_date: number of days to expiration.

The first step is to find a suitable expiration date. The ideal target, 21 days, is not always available, so we must perform the search for different values by alternatively adding 1 or subtracting 1. Then we query the table for the desired diff_date, and if the query is not empty, we get the value with the smallest diff, that is the value closest to the strike price, that is an ATM option.

Algorithm 1 Get data for option

```
Require: option_data, dates, day_diff
Ensure: filled_option_data
  filled_option_data = copy_structure(option_data) ;
  for date in diff_date do
      last_price = last_quote;
      query_df = query(option_data where QUOTE_DATE==date);
      while True do
          desired_diff = get_desired_diff(day_diff);
          work_df = query(query_df where diff_date == desired_diff);
          if work_df is not empty then
              row = get_best_row();                    ▷ The row with the lowest diff
              update(filled_option_data,row);
              break;
          end if
      end while
  end for
```

Bibliography

Akiba, Takuya, et al. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2019.

Bachelier, Louis. "Théorie de la Spéculation." *Annales Scientifiques de L'Ecole Normale Supérieure*, vol. 17, 1900, Reprinted in P. H. Cootner (ed), 1964, The Random Character of Stock Market Prices, Cambridge, Mass. MIT Press, pp. 21–88.

Baillie, Richard T., Tim Bollerslev, and Hans Ole Mikkelsen. "Fractionally integrated generalized autoregressive conditional heteroskedasticity." *Journal of Econometrics*, vol. 74, no. 1, 1996, pp. 3–30. https://doi.org/https://doi.org/10.1016/S0304-4076(95)01749-6, https://www.sciencedirect.com/science/article/pii/S0304407695017496.

Batten, Jonathan A., Harald Kinateder, and Niklas Wagner. "Multifractality and value-at-risk forecasting of exchange rates." *Physica A: Statistical Mechanics and its Applications*, vol. 401, May 2014, pp. 71–81. https://doi.org/10.1016/j.physa.2014.01.024.

Dieker, Ton. Simulation of fractional Brownian motion. 2004. http://www.cwi.nl/~ton.

Drost, Feike C., and Bas Werker. "Closing the GARCH gap: Continuous time GARCH modeling." *Journal of Econometrics*, vol. 74, no. 1, 1996, pp. 31–57. https://EconPapers.repec.org/RePEc:eee:econom:v:74:y:1996:i:1:p:31-57.

Filipa, Cristiana, and Teixeira Vaz. Multifractality in bitcoin. 2021.

Garcin, Matthieu. Fractal analysis of the multifractality of foreign exchange rates. 2019.

Goddard, John, and Enrico Onali. Self-affinity In Financial Asset Returns. 2014.

Gómez-Águila, A., J. E. Trinidad-Segovia, and M. A. Sánchez-Granero. "Improvement in Hurst exponent estimation and its application to financial markets." *Financial Innovation*, vol. 8, no. 1, Sept. 2022. https://doi.org/https://doi.org/10.1186/s40854-022-00394-x.

Günay, Samet. "Performance of the multifractal model of asset returns (Mmar): Evidence from emerging stock markets." *International Journal of Financial Studies*, vol. 4, 2 June 2016. https://doi.org/10.3390/ijfs4020011.

Jeon, Hye Woong. Overview of the Multifractal Model of Asset Returns. 2021.

Jiang, Chuxuan, Priya Dev, and Ross A. Maller. "A Hypothesis Test Method for Detecting Multifractal Scaling, Applied to Bitcoin Prices." *Journal of Risk and Financial Management*, vol. 13, 5 May 2020. https://doi.org/10.3390/jrfm13050104.

Ke, Guolin, et al. "Lightgbm: A highly efficient gradient boosting decision tree." *Advances in neural information processing systems*, vol. 30, 2017, pp. 3146–54.

Maglione, Federico. Fractal and multifractal models for price changes An attempt to manage the Black Swan. 2012. https://www.academia.edu/89617926/Fractal%5C%5Fand%5C%5FMultifractal%5C%5Fmodels%5C%5Ffor%5C%5Fprice%5C%5Fchanges.

Mandelbrot, Benoit, Adlai Fisher, and Laurent Calvet. A Multifractal Model of Asset Returns. 1997. http://www.econ.yale.edu/%5Csim%7B%7Dfisher/papers.html.

Mandelbrot, Benoit, and Richard L Hudson. Overall Applicability Innovation Style The Misbehavior of Markets A Fractal View of Risk, Ruin, and Reward. 2004. www.getAbstract.com.

Mansukhani, Subir. The Hurst Exponent: Predictability of Time Series. 2012. https://pubsonline.informs.org/do/10.1287/LYTX.2012.04.05/full/.

Moses, Todd. Mandelbrot's Multifractal Model of Asset Returns Explained. 2021. https://medium.datadriveninvestor.com/mandelbrots-multifractal-model-of-asset-returns-explained-bde7b5121be0.

Mottl, Dmitry. Hurst , Python Package Index - PyPI. 2019. https://pypi.org/project/hurst/. Accessed 28 Mar. 2021.

Nelson, Daniel B. "Conditional Heteroskedasticity in Asset Returns: A New Approach." *Econometrica*, vol. 59, no. 2, 1991, pp. 347–70. http://www.jstor.org/stable/2938260. Accessed 11 Feb. 2024.

Nnalue, Ugochukwu, et al. Modeling Options Prices Considering Mandelbrotian Movement of Prices. 2020. https://ssrn.com/abstract=3542764.

OpenAI. ChatGPT. *chat.openai.com*, Feb. 2024. https://chat.openai.com/.

Prokhorenkova, Liudmila Ostroumova, et al. "CatBoost: unbiased boosting with categorical features." *arXiv (Cornell University)*, June 2017. https://doi.org/https://doi.org/10.48550/arxiv.1706.09516.

Saâdaoui, Foued. "Structured Multifractal Scaling of the Principal Cryptocurrencies: Examination using a Self-Explainable Machine Learning." Apr. 2023. http://arxiv.org/abs/2304.08440.

Sibirtsev, Rostislav. How to Avoid Bankruptcy?: Monte Carlo Simulation of Three Financial Markets, using the Multifractal Model of Asset Returns. 2019. https://www.uppsatser.se/uppsats/e13a6dfd51/.

Sottinen, Tommi. "Fractional Brownian motion in finance and queueing." Apr. 2003.

Wikipedia. Hurst Exponent. 2023, Accessed: Dec 18, 2023. https://en.wikipedia.org/wiki/Hurst%5C%5Fexponent.

Zhang, Hong-yan, and Zhi-qiang Feng. A Survey of Methods for Estimating Hurst Exponent of Time Sequence. Sept. 2023. *arXiv*, arxiv.org/abs/2310.19051.