Imperial College London

MSc Mathematics and Finance

Imperial College London

Department of Mathematics

Market Microstructure: Coursework

Author:
Antoine Balouka (02305271)

Lecturer:
Mathieu Rosenbaum

May 25, 2023

Contents

Chapter 1

Multi-variate correlated Black-Scholes Models

1.1 Some Theory

In the following first part, we focus on the famous Black-Scholes model. Under risk-neutral measure Q we modelize the evolution of an asset S_t as the solution of the stochastic differential equation :

$$dS_t = rS_t dt +_t dW_t \tag{1.1}$$

The solution is a Geometric Brownian Motion where W_t is a Brownian motion under Q and r the risk-free interest rate. Using Ito's formula to $X := ln(S_t)$ we find the closed-form :

$$S_t = S_0 exp((r - \frac{\sigma^2}{2})t + \sigma\sqrt{t}Z) \quad \mathbf{Z} \sim N(0,1) \tag{1.2}$$

Therefore we will use the foundations of the Simulation Method course to implement such simulated sample path. Further, recall that, to generate two correlated $N(0,1)$ variables with correlation coefficient ρ, we first generate two independent $N(0,1)$ variables Z_1, Z_2, then the variables X_1, X_2 are two correlated normal variables, defined by

$$\begin{cases} X_1 = Z_1 \\ X_2 = \rho Z_1 + \sqrt{1-\rho^2} Z_2 \end{cases} \tag{1.3}$$

1.2 Simulation Scheme

The simulation scheme used in the code is based on the previous Black-Scholes model for stock price dynamics. The key idea is to discretize time into small intervals and simulate the stock price at each time step.

The $BlackScholesModel$ class provide us with the three necessary attributes spot price, volatility and interest rate. The $CorrelatedBlackScholesModels$ class is responsible for simulating two correlated Black-Scholes models. It takes additional parameters for the correlation between the two models. The $generate\_correlated\_paths$ method generates correlated paths for the two models. It uses the BlackScholesModel

to initialize parameters (this structure is more complexe but can be integrated to more complex studies or more than two correlated assets).

The drift component represents the expected change in the stock price based on the interest rate and volatility. The diffusion component captures the random fluctuations in the stock price, modeled as a normally distributed random variable multiplied by the square root of the time step.

The first element of the path array is set to the initial spot price. Then, for each time step, we calculate the drift and diffusion components of the stock price dynamics. We generate two independent normally distributed random variables and combine them with appropriate weights to achieve the desired correlation. The correlation parameter is used to determine the weight between the two random variables.

Finally, the new stock price at each time step is obtained by multiplying the previous stock price by the exponential of the sum of the drift and diffusion components. This reflects the geometric Brownian motion assumption in the Black-Scholes model.

The method returns the paths for both models and the sample paths are plotted using the plot function from the matplotlib library, with the time steps on the x-axis and the stock prices on the y-axis. The resulting plot shows the simulated paths for the two correlated Black-Scholes models.

1.3 Sample Path

We use 10.000 points to simulate the paths over a year. Note that, to generate the following paths, one needs to use the *generate_correlated_paths_not_optimized* method. Note that, to use T/dt = 100.000 points, one should set $time\_space = np.arange(0, T-dt, dt)$ while for other power of 10, setting $time\_space = np.arange(0, T, dt)$ works. This anomaly comes from the way Python calculates floats.

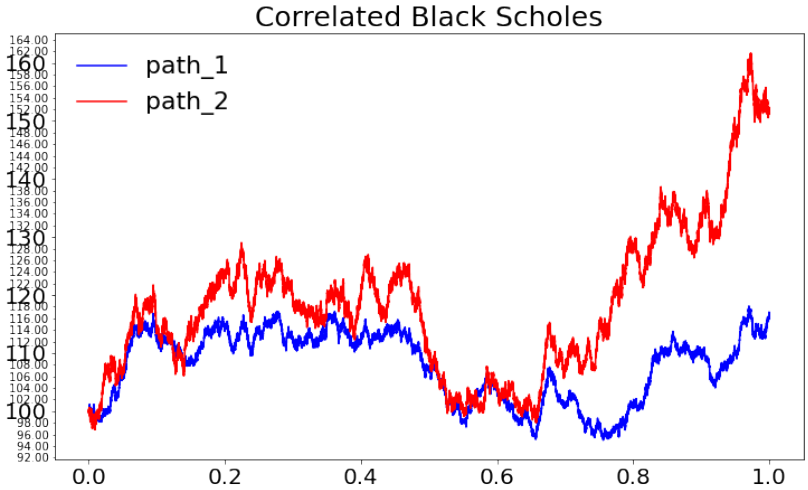


Figure 1.1: Correlated Black Scholes paths

Chapter 2

Uncertainty Zones

2.1 Some theory

Recall the notation seen in class for the Uncertainty zones model :

- Efficient price X_t
- α the tick size
- η the aversion for price change
- the Uncertainty zones $U_k = [0, \infty) \times (d_k, u_k)$ with $d_k = (k + \frac{1}{2} - \eta)\alpha$ and $u_k = (k + \frac{1}{2} + \eta)\alpha$
- We define τ_i the i-th exit time of an uncertainty zone

2.2 Simulation Scheme

The simulation scheme used in the code is based on the previous Black-Scholes model for stock price dynamics and the class we provide. The key idea is to properly defining the uncertainty zones regarding the given market tick before finding at each step in what zone the stock path lies.

We use the same $CorrelatedBlackScholesModels$ class to define the required method called $generate\_UZ\_model$. Even though we do not use the class attributes, this implementation allows for improvement and flexibility if future work is needed.

The method $generate\_UZ\_model$ takes 7 parameters including those defined above and the 2 Black-Scholes paths of stock. The parameter seed allows to reproduce the samples while using random variables.

Inside this method we first define, according to the tick sizes, the min and max prices observed, the array of all ticks and finally the uncertainty zones.

The method $generate\_UZ\_model$ is responsible for generating UZ model paths with a series of if/else conditions being the core component of the model. The complexity of the implementation lies in identifying when (index wise) the path crosses a zone resulting in a change of price. This is done with :

```
absolut_diff = np.maximum(path_X[i] - Uzones_d , 0)
absolut_diff[absolut_diff == 0] = 10000
UZM_X[i] = all_ticks[np.argmin( absolut_diff )]
```

Finally, the method returns the UZ model paths. In the main part of the code, already having the Black Scholes paths the *generate_UZ_model* method is then called to obtain the correlated paths for the two models. Finally, the sample paths are plotted using the plot function from the matplotlib library, with the time steps on the x-axis and the prices on the y-axis. The resulting plot shows the simulated UZ paths for the two correlated Black-Scholes models.

2.3 Sample Paths

We plot, for a comparison purpose, 4 graphs with $\eta = 0.8$ and $\eta = 2$ and 2 different time scales.

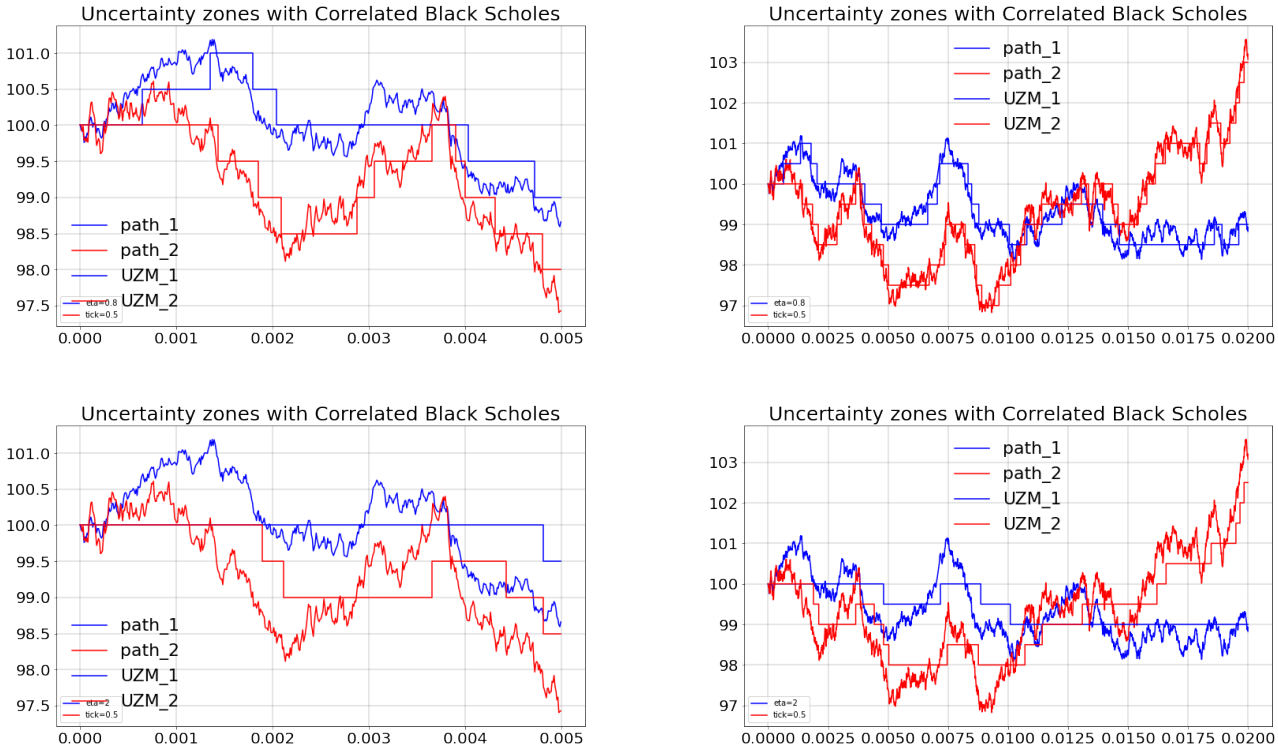


Figure 2.1: U. zones with Correlated Black Scholes

We observe, as expected, that η has a big impact on the resulting UZ model. Therefore, a further study would benefit from calibrating the most adapted η to fit the market data.

Chapter 3

Epps Effect

3.1 Simulation Scheme

The simulation scheme used in the code is based on the previous Black-Scholes model for stock price dynamics and the class we provide. The key idea is to implementing a covariation estimator and studying the impact the time horizon has.

We use the same $CorrelatedBlackScholesModels$ class to define the required method called $generate\_cov\_estimator$. It takes 3 parameters : the 2 piece_wise constant UZ model paths of stock and an array of η we want to study. It uses the function $previous\_tick\_covariation\_estimator$ to estimate the covariation based on the previous tick scheme.

On a separate study, we use the to_pwc function to "zoom in". This allows us to approach, in a simplified way, the higher frequency data to observe the epps effect. Indeed, it adds more data points to the already point-wise constant UZ paths. Note that "zooming in" with this function is equivalent to reducing the time horizon h to values between 0 and 1 which would note be accessible in an easier way but doesn't accurately add value to our calculation.

We study the epps effect over multiple η as is it the main parameter triggering a price change in the UZ model.

We provide with the values of parameters used in the code below :

```
spot_price_1 = 100
spot_price_2 = 100
volatility_1 = 0.2
volatility_2 = 0.3
interest_rate = 0  #0.05   (drift used for the first 2 chapters)
correlation = 0.7
T = 1
dt = 0.00001
seed = 123
tick   = 0.5
eta    = 0.8
```

3.2 Epps effect

We plot the graphs highlighting the Epps effect for the specific seed =123 and a null drift, ensuring the two paths to be Brownian Motions, as needed in the course's Theorem.

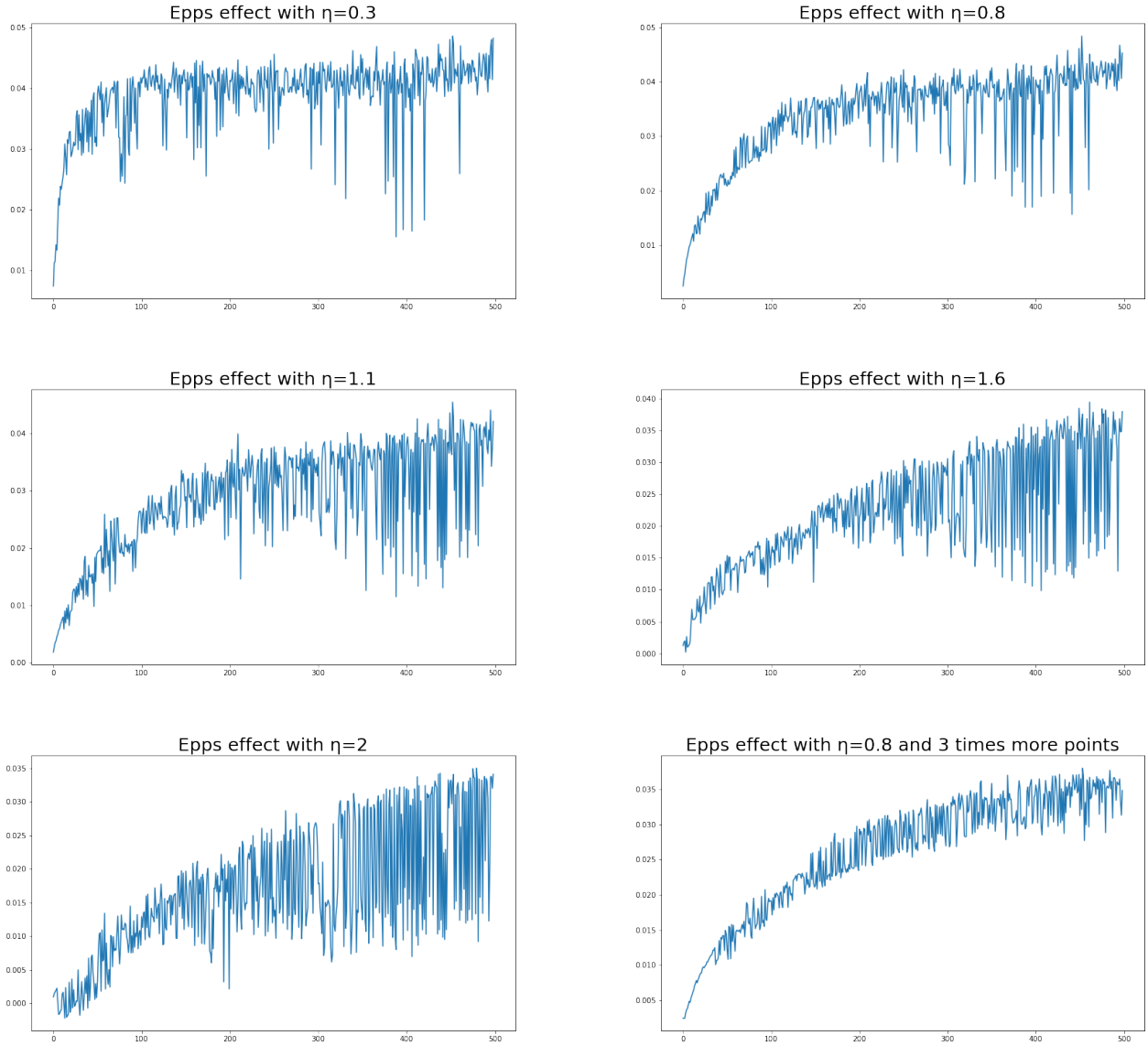
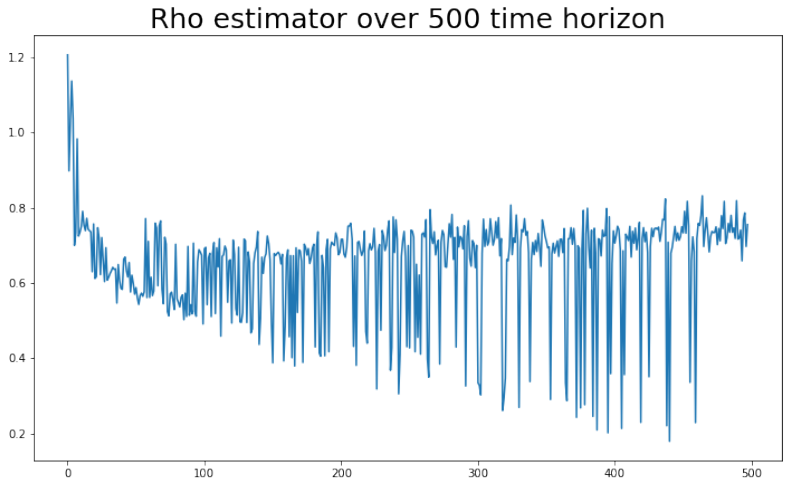


Figure 3.1: Epps effect with multiple η

We see here the Epps effect, namely the convergence to zero of the covariation estimator when the time horizon tends to 0. As h increases we observe that the covariation tends to stabilise arround a convergence positive value. However, we also see a clear instability when h increases.

Furethermore, we calculated the correlation estimator between the two Uncertainty zone model paths, converging to 0.7. In this particular example of 2 paths, the estimator properly evaluates the correlation of the paths. As expected, the higher time horizons will show better accuracy as the paths behaves more like the

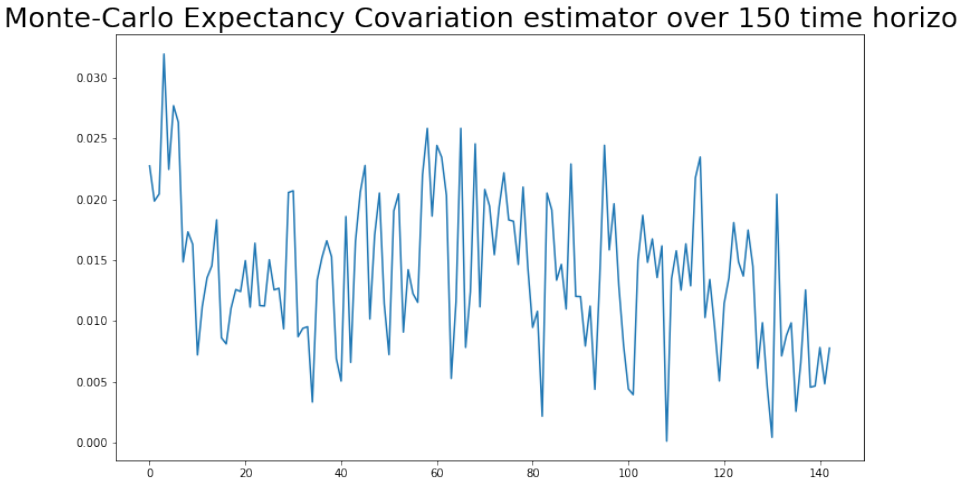
original paths. Therefore we see that the UZM generates non-linearities regarding the statistical properties of the paths.



3.2.1 Improvements

In order to improve the results we will both use Monte-Carlo Method and use 100.000 data points in the generation of the paths instead.

We implement, with 10.000 points, a Monte-Carlo to calculate the expectancy of the covariance estimator. Due to implementation inefficiency, we have a the wanted Monte-Carlo plot for the first 150 time horizon. Each data point is the result of 100 pair of generated paths. We believe with a faster implementation and more than 100 data points by step, the result would be more accurate.



Reproducing the same study for 100.000 data points instead delivers graphs closer to what we have seen in the course.

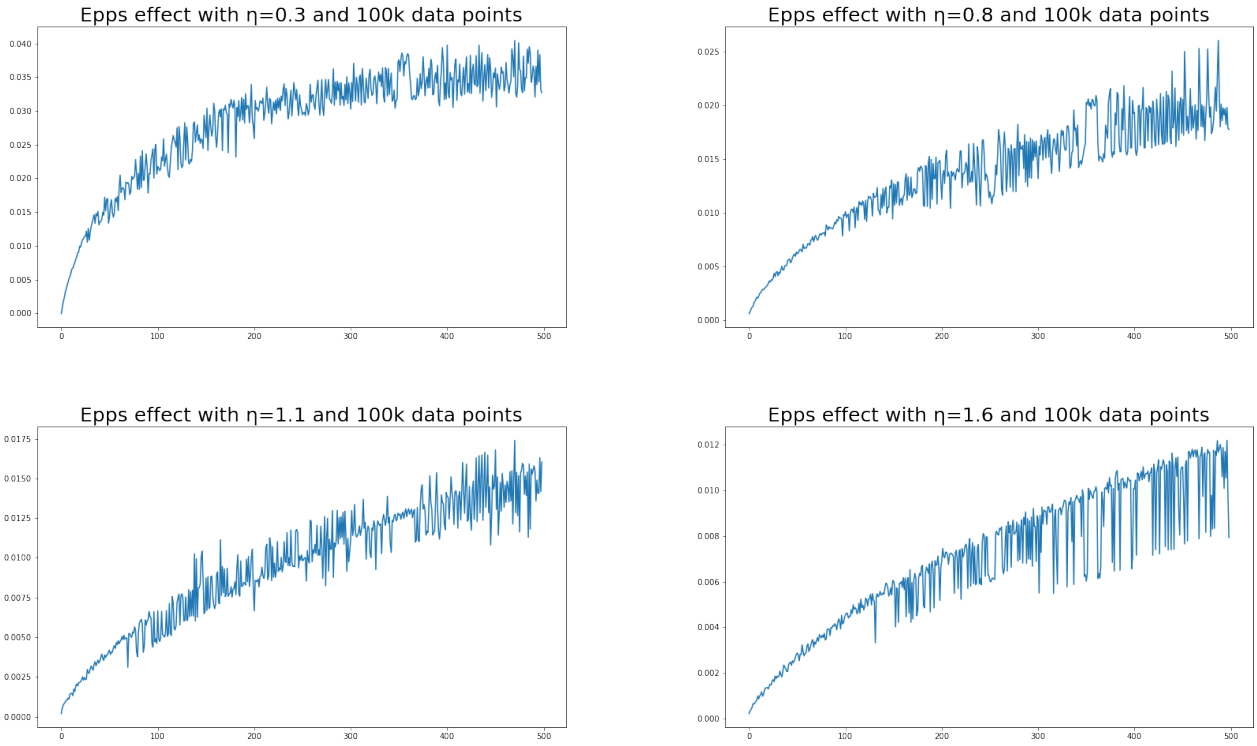


Figure 3.2: Epps effect with multiple and 100.000 data points

Chapter 4

Hayashi-Yoshida estimator

4.1 Some Theory

Recall the formula for the Hayashi-Yoshida estimator for the covariation :

$$\left\langle X^{(k)}, X^{(l)} \right\rangle_{HY} = \sum_{i=1}^{n^{(k)}} \sum_{i'=1}^{n^{(l)}} \Delta X^{(k)}_{t_i^{(k)}} \Delta X^{(l)}_{t_{i'}^{(l)}} \mathbf{1}_{\left\{ \left(t_{i-1}^{(k)}, t_i^{(k)}\right] \cap \left(t_{i'-1}^{(l)}, t_{i'}^{(l)}\right] \neq \emptyset \right\}},$$

where $\Delta X^{(j)}_{t_i^{(j)}} := X^{(j)}_{t_i^{(j)}} - X^{(j)}_{t_{i-1}^{(j)}}$ denotes the jth asset tick-to-tick log-return over the interval spanned from $\mathrm{t}_{i-1}^{(j)}$ to $t_i^{(j)}, i = 1, \cdots, n^{(j)}$.(4.1)

4.2 Simulation Scheme

The simulation scheme used in the code is based on the previous Black-Scholes model for stock price dynamics. The key idea is to first identify what are the intervals where both paths are constant before finding the intersecting intervals. Then the sum of product of log-differences should be computed in regards to the relevant non-intersecting intervals.

The function *find_intersection* provides us with a tool stating if a pair of interval is intersecting and the *HY_estimator* wraps all the necessary computations to deliver the estimate. The *MC_HY_estimator* is the Monte-Carlo implementation of the estimator for the efficient prices.

Recall the course's theorem :

Theorem 4.2.1 *In the model with uncertainty zones, the Hayashi-Yoshida estimator is a consistent estimator of the covariation provided one uses the estimated values of the efficient prices.*

We expect a difference of consistency in the estimator when using either the values of the theoretical prices or the estimated values of the efficient prices.

As explained below, we experienced computational difficulties with calculating the estimator for relevant amount of step resulting in poor estimations we will not share (but remains in our source code).

Notes on the Python code

4.3 Multi-variate correlated Black-Scholes Models

In this section, we simply used the efficiency of numpy to generate the paths with the relevant method.

4.4 Uncertainty Zones

This section is among the most important in the sense that we we enable to, after multiple attemps, to optimize the code we are using. As a consequence, it takes 2.4 seconds to compute the UZM from the paths. The complexity lies in the fact that we have multiple conditions, and the point of the next step relies on the point of the previous step. We however tried to leverage the use of other libraries and tools to build this method :

- Using DataFrames, and building the whole array output in the same computation with the relevant columns and a .apply(lambda x: with x a tuple containing all informations
- Using np.where()

Therefore, we keep the existing implementation of $generate\_UZ\_model$. Some calculations with implemented Monte-Carlo functions were too long to compute (more than 10 hours for the expectancy of the covariance estimator).

4.5 Epps

Again, it is after multiple attempts that we ended up with a pretty efficient way to calculate the previous tick covariation estimator called $previous\_tick\_covariation\_estimator\_fast$. High level, we use numpy to get rid of computationally heavy for loops. We used the same philosophy to create $rho\_estimator\_fast$.

Finally, after some research on internet, parallelising computations involving random variables and methods. One could look forward to improve our implementation of the two Monte-Carlo estimators with vectorization, allowing for more relevant results.

4.6 Hayashi-Yoshida

In those two sections, the difficulty lies in, again, developing a computationally fast process. We have tried 3 main structure while only one succeeds in delivering the

correct estimation, called $HY\_estimator$. It is using simple double loop structure. The two others, $HY\_estimator\_optimized$ 1 and 2, are based on the builing of the intersection matrix that stores the most important information. This matrix would then be used in the method np.where() that would return 0 or the formula product at pair (i,j). At row i, column j, it returns the Boolean stating if Interval I_i where path_1 is constant, is intersecting Interval I_j where path_2 is constant.

The one that had good probability of success actually doesn't provide accurrate matrix. We can see this because looking at a single row we can observe True followed by False, followed again by True. However, we know this can't happen as an interval where first path is constant can't cross interval 1, 3 but not 2 of path 2.

	0	1	2	3	4	5	6	7	8	9	...	2299	2300	2301	2302	2303	2304	2305	2306	2307	23
0	True	True	False	True	False	True	True	False	False	True	...	False	False	True	False	False	False	True	True	True	Tr
1	True	True	True	True	True	True	True	False	True	True	...	True	True	True	False	True	True	True	True	True	Tr
2	False	True	True	False	False	True	True	False	True	True	...	False	False	False	False	False	False	False	True	False	Tr
3	True	True	False	True	True	True	True	False	False	True	...	False	False	False	False	False	False	False	True	False	Tr
4	True	True	False	False	True	True	True	False	False	False	...	False	False	False	False	False	False	False	True	False	Tr
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1089	True	True	False	False	True	True	True	False	False	False	...	False	False	False	False	False	False	False	True	False	Tr
1090	False	True	False	False	True	True	True	False	False	False	...	True	True	True	False	True	True	True	True	True	Tr
1091	True	True	False	False	True	True	True	False	False	False	...	False	False	False	False	False	False	False	True	False	Tr
1092	False	True	False	False	True	True	True	False	False	False	...	True	True	True	False	True	True	True	True	True	Tr
1093	True	True	False	False	True	True	True	False	False	False	...	False	False	False	False	False	False	False	True	False	Tr