Heston model simulation
Python implementation and a statistical analysis

D'Ignazi Jacopo

jacopo.dignazi@edu.unito.it

Abstract

In this study I discuss an application of Heston model for stock prices' simulation, and provide some interpretation of his behaviour based on statistical analysis. The model was implemented from scratch on Python and the most relevant results from the script are summarized in this essay

In section 1 I briefly describe the theory and motivation behind the model, while in section 2 it is described how to use it for simulating stock prices. In section 3 I define some metric and conduct some preliminar analisys on the generated data, and in section 4 these metrics are used to test the model under different parametrization.

1. The Heston model of volatility

The most basic model that can describe the price evolution in a stock market is the geometric brownian motion. In this model, it is defined that

$$dS_t = \mu S_t + \sqrt{v} S_t dW_t \tag{1}$$

where v is the volatility and dW a Wiener process. The limit of this kind of model is that it assumes the volatility to be a constant, while this might not be the case of a financial phenomenon.

Heston model was introduced in 1993 [1] to indeed relax this assumption, defining the volatility as a stochastic process itself. It defines that

$$\begin{cases} dS_t = \mu S_t + \sqrt{v_t} S_t dW_t^s \\ dv_t = k(\theta - v_t)dt + \sigma\sqrt{v_t} dW_t^v \end{cases} \tag{2}$$

where dW^v and dW^s are two Wiener processes, corralated by a factor of ρ. In this model, the volatility v_t follows an Orstein Uhlembeck process, and σ is the volatility of that process.

Given initial values S_0, V_0 and a parametrization Ω, the evolution of the pair S_t, v_t is uniquely determined by the evolution of the underlying Wiener processes Z_t^s, Z_t^v. In addition, the Feller condition can ensure that S_t, v_t are always non negative [2].

The advantages of this approach, compared to a geometric brownian motion, is that it lets you model the volatility as its own phenomenon. This means that the model will be able to reproduce events of high and

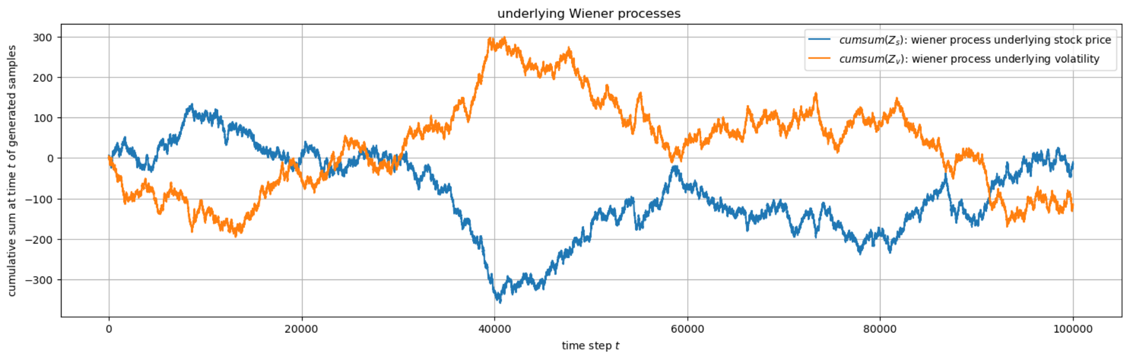


Figure 1: An example of two correlated Wiener processes, where $\rho = -0.3$. Here is represented the cumulative sum of each process, defined as $cumsum(Z_t) = \sum_{\tilde{t}=1}^{\tilde{t}=t} Z_{\tilde{t}}$

low volatility, which are typical of financial world. On the contrary, its limit is that the volatility will in any case be a random process, so the occurrences of these events are left to randomness.

In this essay I will explore the consequences of this choice and its limitations, by generating synthetic prices with the method defined in the next section and measure its behaviour with the criteria I will define in section 3

2. Model simulation

To use the equation 2 in a simulation, the time's space first needs to be discretized. This operation can be performed in different ways, and it represents an approximation of the initial equations. These "discretization scheme" have been object of study [3] since the particular choice of scheme can yield different results. Following the prescriptions of [3] in my program I impemented three of these schemes: "basic Euler", "log Euler", and "Milstein" scheme.

For each scheme it is defined that

$$\begin{cases} Z_t^v &= Z_1 \\ Z_t^s &= \rho Z_t^v + \sqrt{1-\rho^2} Z_2 \end{cases} \tag{3}$$

Where Z_1, Z_2 are two independent sample from a standard normal distribution $Z_{1,2} \sim N(0,1)$. These samples will be independent and identically distributed at any time t, so $Z_t^s \equiv Z_s$ and $Z_t^v \equiv Z_v$. This ensures that the two Wiener processes underlying prices and volatility are correlated. An example of these processes is shown in fig. 1.

The three schemes will now be defined defined as follow:

Euler scheme.

$$\begin{cases} v_{t+1} &= v_t + k(\theta - v_t^+)\Delta_t + \sigma\sqrt{v_t^+} Z_v \sqrt{\Delta_t} \\ S_{t+1} &= S_t + rS_t\Delta_t + S_t\sqrt{v_t^+} Z_s \sqrt{\Delta_t} \end{cases} \tag{4}$$

Log-Euler scheme.

$$\begin{cases} v_{t+1} &= v_t + k(\theta - v_t^+)\Delta t + \sigma\sqrt{v_t^+}Z_v\sqrt{\Delta t} \\ S_{t+1} &= S_t e^{(r-\frac{1}{2}v_t^+)\Delta t + \sqrt{v_t^+}Z_s\sqrt{\Delta t}} \end{cases} \tag{5}$$

Milstein scheme.

$$\begin{cases} v_{t+1} &= v_t + k(\theta - v_t^+)\Delta t + \sigma\sqrt{v_t^+}Z_v\sqrt{\Delta t} + \frac{1}{4}\sigma^2(Z_v^2-1)\Delta t \\ S_{t+1} &= S_t + rS_t\Delta t + S_t\sqrt{v_t^+}Z_t^s\sqrt{\Delta t} + \frac{1}{2}S_t v_t^+(Z_s^2-1)\Delta t \end{cases} \tag{6}$$

Parameters. In each discretization scheme:

- r is the risk-neutral rate of return
- θ is the long term expected value of volatility
- k is the rate at which the volatility revert to its expected value
- σ is the volatility of the volatility
- ρ is the correlation coefficient between Z^s and Z^v
- Δ is the size of the time interval
- S_0 is the initial price
- V_0 is the initial volatility

As a "default" parametrization I chose $\Omega^d : \{\rho = -0.6; r = 0.05; \theta = 0.1; k = 2.0; \sigma = 0.57\ \Delta_t = 10^{-5}; S_0 = 100., V_0 = \theta\}$. This choice is based on parameters fitted on real data [4], therefore can be considered "a realistic parametrization" [1]. In the next section I will use this parametrization for generating prices evolution, while in section 4 the testing will be performed changing each one of the parameters in Ω^d at a time.

The simulation consists in generating simulated prices and volatility by first generating a the Wiener processes 3, then iterate the scheme of choice starting from initial conditions S_0, V_0; the algorithm can run for N_{iter} time steps, and can generate N_{paths} different realization of the simulation. The generated sequences $\{S_t^n, v_t^n\}_{t=1,\dots,N_{iter}}^{i=1,\dots,N_{paths}}$ are commonly defined "Heston paths".

Python implementation. The equations 3 and 2 can be easily reproduced in a script, and the random normal samples are generated through Numpy library.

The technical limitation I encountered was a tradeoff between speed of the generation process and memory it took: the fastest way to generate a big sample is to generate it altogether, meaning producing a Numpy array of sizes $N_{paths} * N_{iter} * 2 * 8$ bytes [2]. This could easily overload memory of a common PC when the

[1] I chose $r = 0.05$ instead of that in the paper [4], in order to visualize the drift in a smaller time scale

[2] The numeric type of used was the common numpy $np.float64$, where each value takes 8 bytes of memory

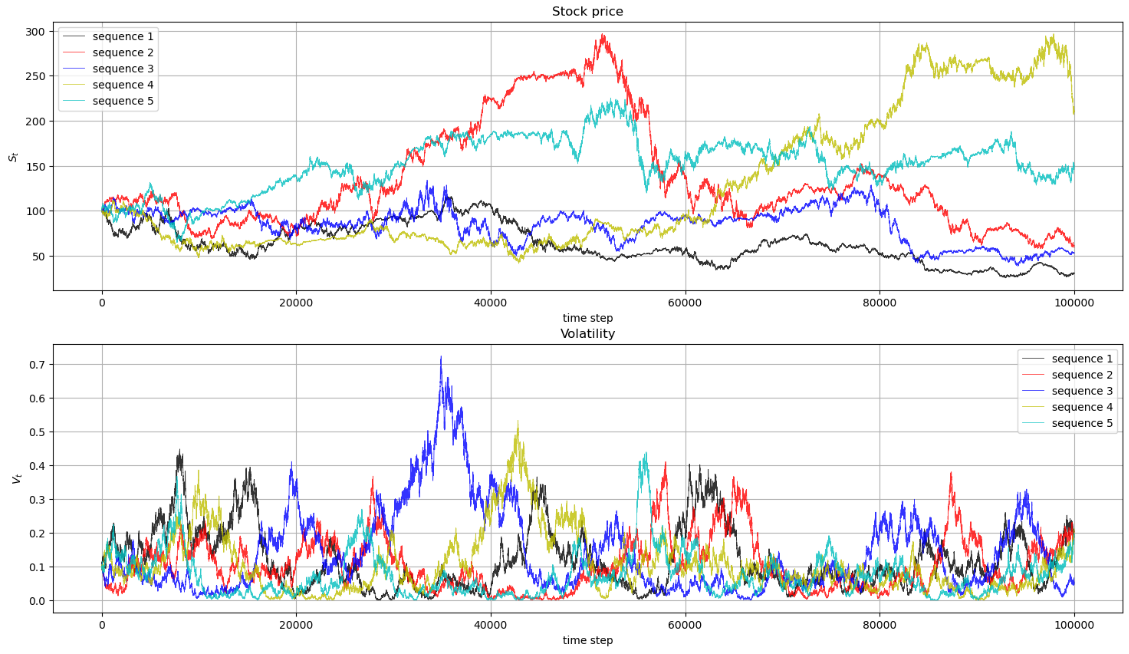


Figure 2: Five instances of Heston path generated with parametrization Ω^d

array size is in the order of 10^9 bytes ($\sim Gb$), so for $N_{paths} * N_{iter} \sim 10^8$. The solution is batching the tensor in smaller sized ones and working with one batch at a time but it is slower due to Python interface with Numpy, since iterating over a sampling process is slower when managed by Python than by Numpy itself.

For this reason I limited the data I studied to have a maximum size of $500Mb$, meaning that I had to choose roughly $N_{paths} * N_{iter} \sim 10^7$; in this case, when generating the data altogether, the generation process takes less than a second.

I also implemented a version of the generation algorithm that output one path at a time, but I only used it for visualization purposes due to the much bigger time it requires.

In order to optimize either the generating algorithm or the metrics computation, I implemented in the code some timer functionality to tell how much time the program spent in each function. By modularization of each sub-process and searching for the best computation strategy, I was able to limit the time cost of the whole study to a really manageable amount on a common desktop PC.

In an Intel Core i5-7200U (3.1 GHz) with 4 Gb of RAM, the whole script ran in around only 5 minutes.

Python packages and versions. For reproducibility the code is accompained by an Environment.yml file, with all the versions of each specific package. In particular, I used Python 3.8.5 and Numpy 1.19

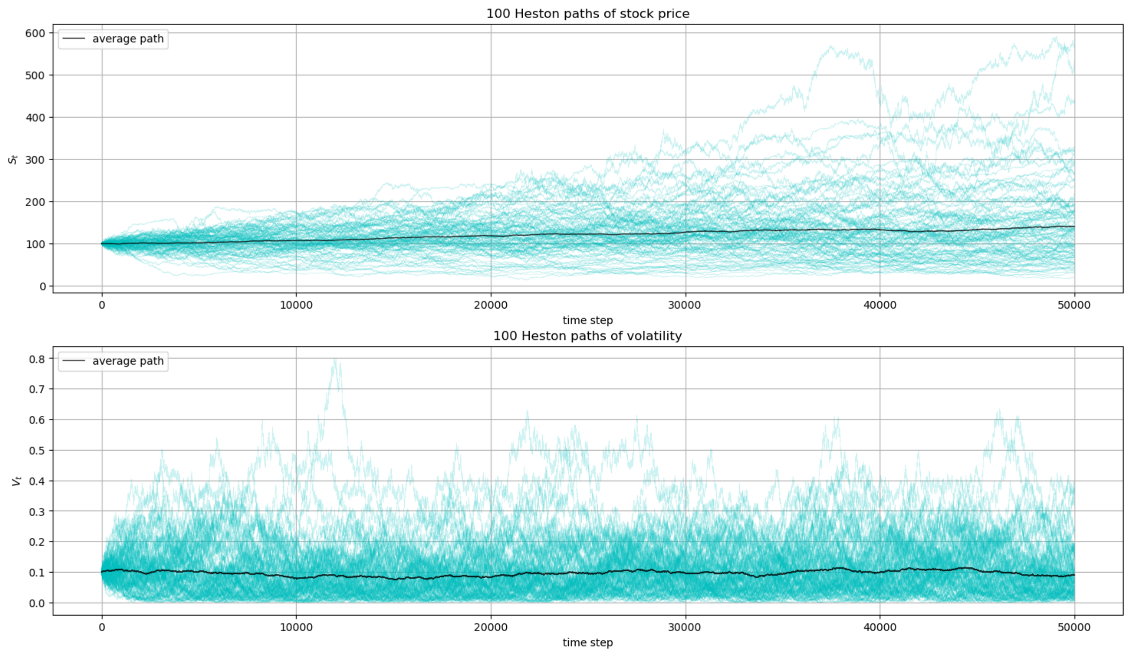


Figure 3: 100 instances of Heston path generated with parametrization Ω^d, and their average path

3. Metrics and observations

Heston paths. Figure 2 shows 5 realizations of the same simulation. The first interesting thing to notice is that the same parametrization can yield very different paths and these paths tend to spread out more and more as time goes by. I will refer to this propriety as "susceptibility to randomness" and I will discuss it in the next section. The second thing to notice is that the paths can have very suddent and dramatic changes, as it is to be expected in a brownian-like process.

Defining the average path:

$$\begin{cases} < v_t > & = \frac{\sum_{n=1}^{n=N_{paths}} v_t^n}{N_{paths}} \\ < S_t > & = \frac{\sum_{n=1}^{n=N_{paths}} S_t^n}{N_{paths}} \end{cases} \tag{7}$$

Figure 3 shows how the average of a bigger amount of paths mitigate these effects and the generated paths shows some "average regularity". The relation between "average regularity" and "susceptibility to randomness" will be the focus of my statistical analysis

Scheme comparison. Under the same parametrization Ω^d a sequence $\{Z^s, Z^v\}$ is generatd and then used to calculate prices and volatility for the three different schemes. The comparison is made by taking the difference of the schemes output at each timestep. The procedure has been repeated for 1000 paths of lenght 5000 and the histograms of all the difference values are shown in Figure 4.

With respect to the order of S_0, V_0 these differences stand in a scale of $\sim [0, 10^{-3}]$ and averagely in the

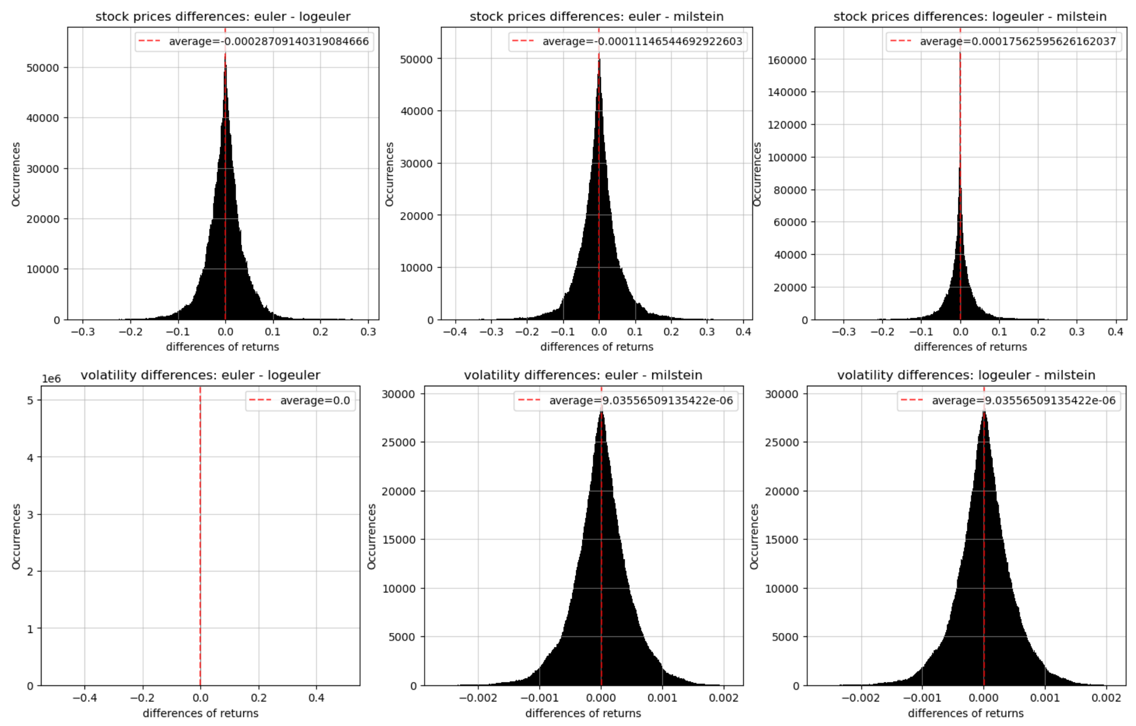


Figure 4: Differences in all generated values. Each entry of these instogram is a pointwise difference a path generated with same underlying Wiener sequences, evaluated for different schemes at time t

order $\sim 10^{-5}$, so they can be considered be not relevant to statistical observations on the model's general behaviour[3].

Scheme	Time	Avg. diff. Euler (prices)	Avg. diff. Euler (volatility)
Euler	$\sim 0.23sec$	-	-
Log-Euler	$\sim 0.28sec$	$\sim 10^{-3}$	-
Milstein	$\sim 0.38sec$	$\sim 10^{-3}$	$\sim 10^{-5}$

The main difference I observed was in computational costs: as it can be seen from equations in section 2, Log-Euler scheme and Milstein scheme need to perform more operations than basic Euler scheme. This results in a slight increase of time for the former, and around 3/2 increase for the latter.

For this reason and since neither significant nor sistematic difference was observed in the generated paths, I decided to focus my attention on the Euler scheme alone.

[3]The second and third columns are the order of average distance from the Euler schemes, when looking at price paths and volatility path respectively. The $'-'$ entries are zero by definition, and the values are not yet scaled with respect to S_0, V_0

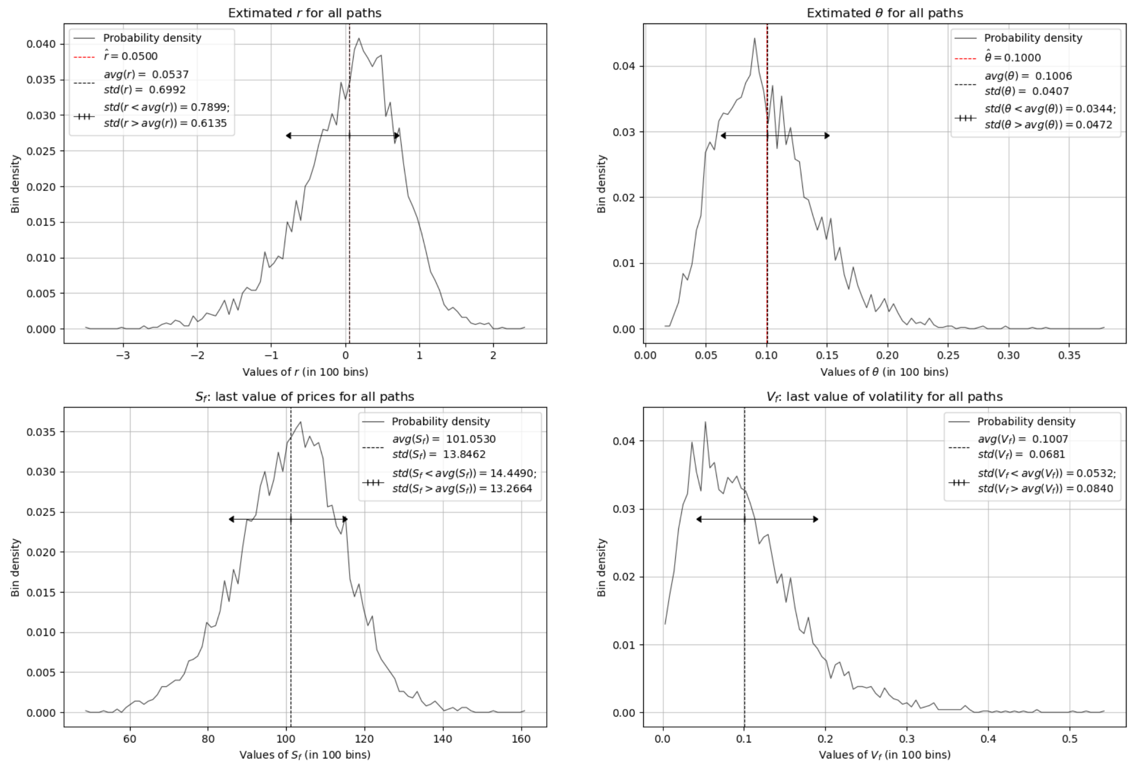


Figure 5: Distribution of pathwise metrics. The values on y axis represent the probability of a value falling in one of the 100 intervals used to partition the x axis

pathwise metrics. Taking the average of Euler scheme in 2, and considering that the average of a Wiener processes is always 0, with some algebra we get that

$$\begin{cases} r &= <\frac{S_{t+1}-S_t}{S_t}> \\ \theta &= \quad < v_t > \end{cases} \tag{8}$$

meaning that r can be extimated as the average of returns, and θ can be extimated as the average value of volatility. This means that given a path generated with $\hat{r}, \hat{\theta}$, we can use these equations to extimate the quantity r_{ext}, θ_{ext} that "have most probably generated that path". The equality between the two whould be hold exactly only for $N_{iter} \equiv t \to \infty$, while the extimated quantity for a certain simulation N_{iter}, N_{paths} represent the model consistency in generating paths.

From figure 5 we can see that the average $< r_{ext} >$ is indeed close to $\hat{r}$ but its standard deviation is realatively high. This is also true for θ_{ext} and for the values of price and volatility at last iteration, representing the spreading of the paths.

The values of average and standard deviation of these metrics, are a quantitative measure of the model average regularity and susceptibility to randomness. In the next section, these metrics will be used to evaluate simulation performances on different parametrizations

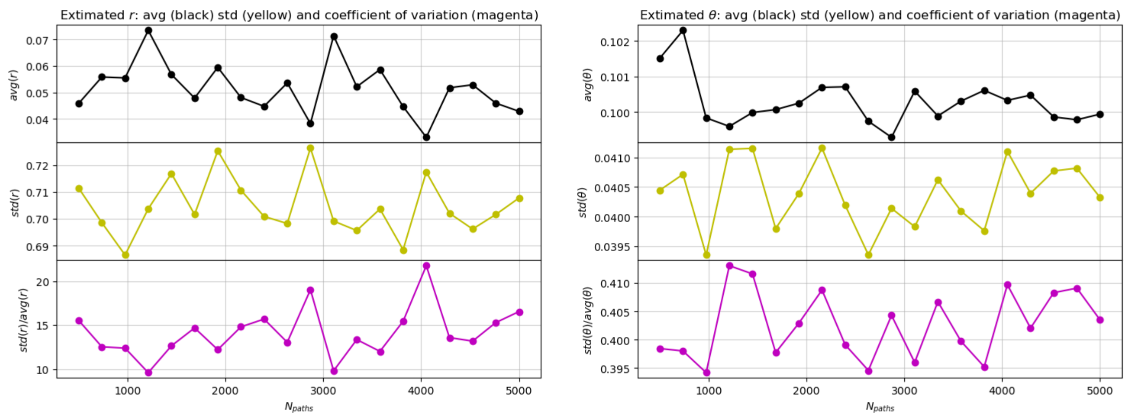
4. Model testing

With synthetic data consisting in N_{paths}, the metrics above consist in distributions of N_{paths} values -one for each path; given a certain set of parametrizations $\{\Omega\}$ the test will consist in mapping these parametrizations to the average and standard deviation of those distributions. It is useful to also look at coefficient of variation[4], since it represent the balance between model average regularity and its susceptibility to randomness.

The choice of the number of paths and iterations has also been tested with this strategy, while in all the other cases the choice was $N_{iter} = 2000, N_{paths} = 5000$. While changing one of the parameters, the others will be the ones in Ω^d. I will use notation $\hat{x}$ to indicate that a parameter "x" has been used to *generate* the data, while x_{ext} means that the quantity is *inferred* from the generated path itself.

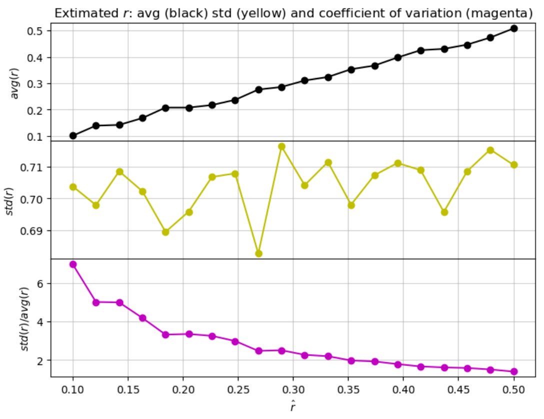
In this section I will only display the most relevant graphs, while the whole table of results can be retrieved in the script and/od easily reproduced.

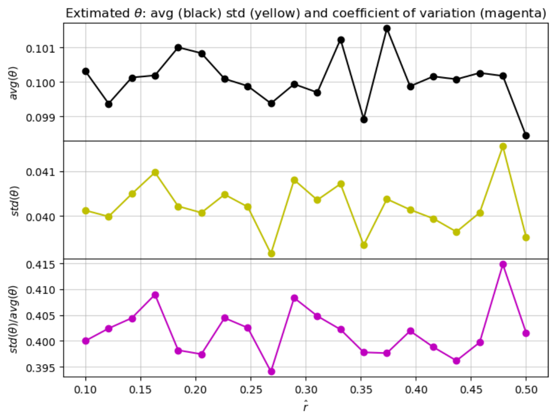
Changing N_{paths}. The first thing to notice is that for a statistically significant sample, the metrics defined above do not change with the size of the sample. At most, they do not show any obvious trend with respect to N_{paths}. This implies that the observed statistical propriety are an intrsic propriety of the model itself -and the specific parametrization- and not of the size of the sample



Changing r. The parameter $\hat{r}$ sets the model stochastic drift, meaning that the higer its value the more a path will tend to grow in time. Figure in the next page shows how $< r_{ext} >$ is always really close to $\hat{r}$, meaning that the model generates path that are actually averagely regular. As observed before, the standard deviation is high for this distribution, and it apparently slowly grows around the value $std(r_{ext}) \sim 0.7$. This means that while regular in average, the model is higly noisy and expecially for small values of $\hat{r}$ a single generated path's behaviour can be much different from the average

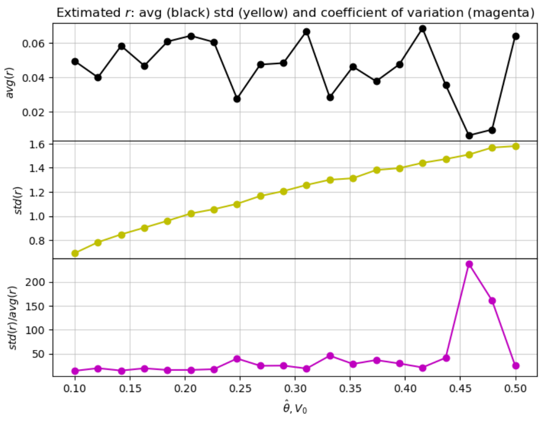
[4]Given a distribution P, the coefficient of vriation of P is defined as the ratio $CV(P) = \frac{std(P)}{avg(P)}$

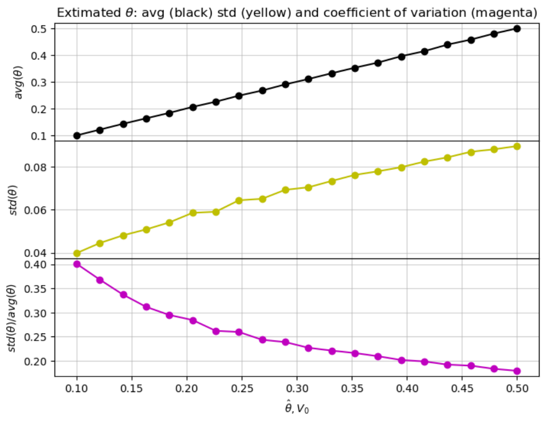




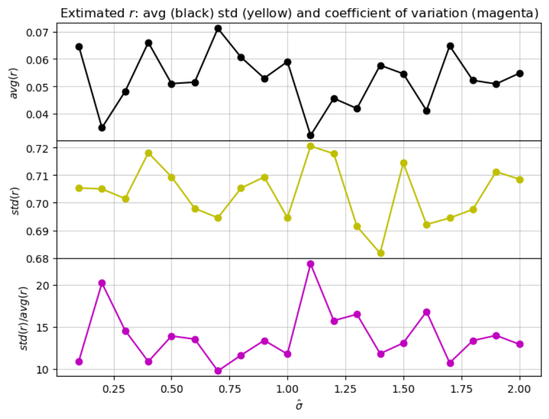
Changing θ. Changing $\hat{\theta}$ means changing the average volatility of the model, meaning that the higer it is the more the model will be allowed to fluctuate. The test on this parameter has been performed changing $V_0 = \theta$ accordingly, in order to avoid biases due to the time the volatility could take to reach its average value.

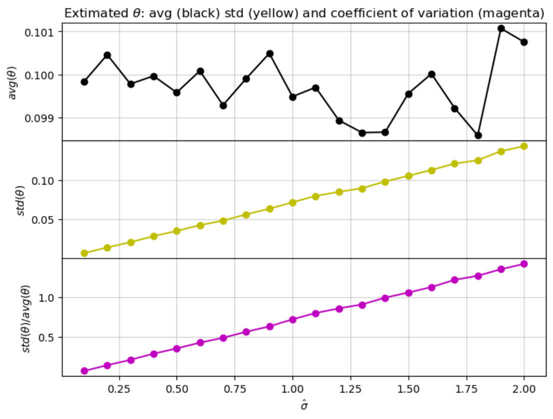
Changing those parameters increases the standard deviation of both r_{ext} and θ_{ext}; this is not surprising, since the model susceptibility to randomness is actually a function of the intrinsic volatility of the model itself. Nonetheless, the paths still mantain their average regularity, even if possibly less consistently for higer value of average volatility

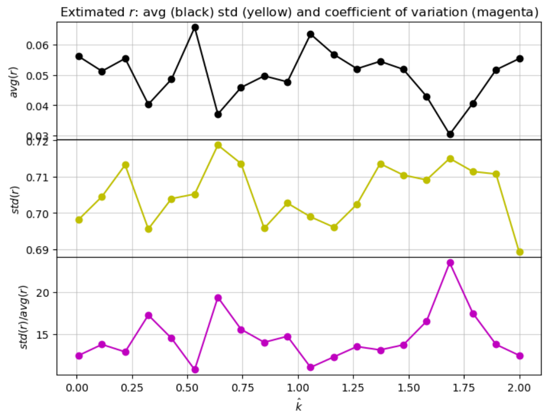


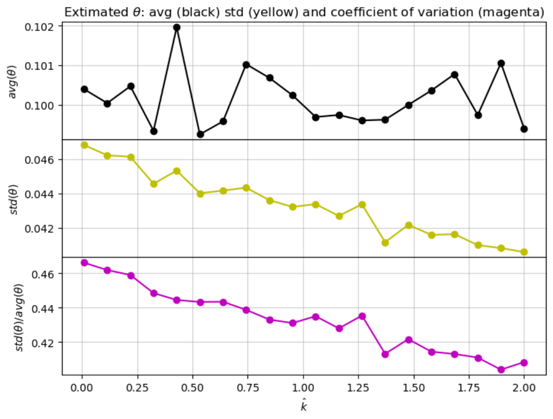


Changing σ and k. The parameter $\hat{\sigma}$ and $\hat{k}$ are responsible of the volatility behaviour: the higer $\hat{\sigma}$ the higer the variance of volatility, while higer $\hat{k}$ means a bigger "step size" of the volatility paths. These parameters do not have significant effects on the average r_{ext} and its standard deviation but can increase or reduce the susceptibility to randomness of the volatility. The relation between these parameters and the volatility is well studied, since the volatility on its own is an instance of the well known Ornstein Uhlembeck process

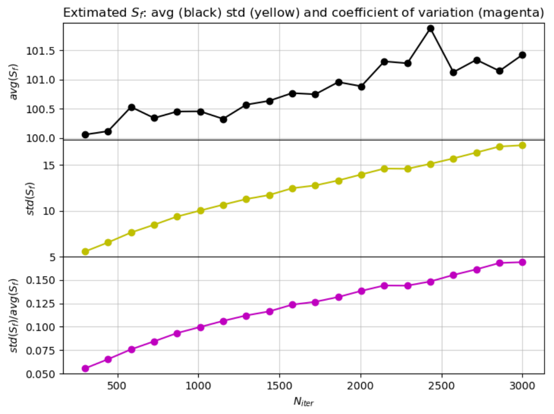


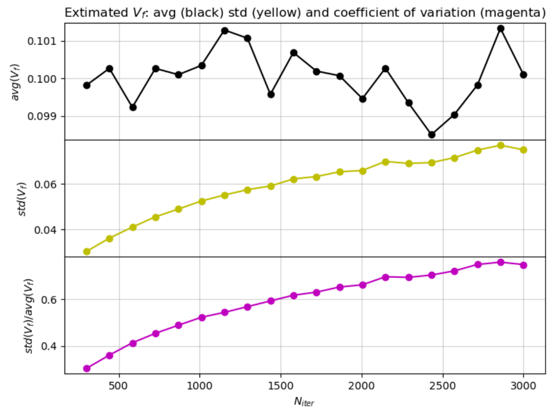






Changing N_{iter}. Changing the number of iterations means changing the lenght of the simulation; or from a different perspective, looking what is happening in the simulation at different times. The most relevant thing to notice is how this changes the behaviour of price paths at time $t = N_{iter}$: the average linearly increases with the time, but so does the standar deviation of that value. The increase of average prices is consistent with a drift-like stochastic process, where the drift is indeed given by the risk-neutral rate of return $\hat{r}$. The increases in standard deviation and coefficient of variation is instead consistent with a diffusion-like stochastic process, meaning that given enough time the paths will spread out on all feasible values $[0, \infty[$.





This tells us that while Heston model is a generalization of a geometric brownian motion, the two still share some relevant peculiarity. The reason of its susceptibility to randomness and average regularity brings back to the propriety of a brownian motion itself: the fact that the volatility is a randon process adds complexity to the paths behaviour, and makes it better suited to locally reproduce the realistic behaviour good's prices. Nonetheless, the model remains statistically similar to a random walk

Conclusions

The topic of simulating stock prices with a stochastic model is still an open problem, and how to use models like Heston's in real case scenario is still an open debate. Two important factor to consider, which have not been discussed in this essay, are the possibility of using this model for prediction of derivatives and of fitting the parameters on real data.

Nonetheless, the observations above tells much of its known limitations: the occurrences of low and high volatility are still left to randomness. Since the volatility of a process is to be considered a representation of its underlying microstructure, this means that the Heston model still represent a rough approximation of economic actor's behaviour.

This leads to, in the first place, the Heston model's inability to represent the causality of any local change in collective behaviour; and secondly, the model's noisiness and the unpredictability of the paths. While the second one is an intrinsic characteristic of an SDE-based model, state of the art model are indeed born in attempt to solve the non locality limitation. [a] Furthermore, a good practice is fitting the model's parameter locally [3][4].

Despite these limitations, this model manages to give an interesting qualitative insight to prices evolution with respect to their volatility. Furthermore, in absence of drastic local changes, a statistical analysis over Heston paths can still give informations about the probability of future events, together with the risks associated with an actor's possible choice. From an historical point of view, the Heston model is indeed an important and fascinating step in economic science's capability of effectively describe human behaviour

[a] In more recent models, all of the parameters of Heston model can be time-dependent. For further references see "local volatility model"

References

[1] Heston, S. (1993). *A Closed-Form Solution for Options with Stochastic Volatility with Applications to Bond and Currency Options.* Review of Financial Studies, 6, 327-343. https://doi.org/10.1093/rfs/6.2.327

[2] Feller, W. (1951). *Two Singular Diffusion Problems.* Annals of Mathematics, 54(1), 173–182. https://doi.org/10.2307/1969318

[3] Mrázek, M. & Pospíšil, J. (2017). *Calibration and simulation of Heston model.* Open Mathematics, 15(1), 679-704. https://doi.org/10.1515/math-2017-0058

[4] Crisóstomo, R (2014). *An Analysis of the Heston Stochastic Volatility Model: Implementation and Calibration Using Matlab.* CNMV Working Paper No 58. http://doi.org/10.2139/ssrn.2527818