How to simulate a stock market with less than 10 lines of Python code

Let's simulate a financial market using geometric brownian motion

Gianluca Malato Apr 23 · 5 min read ★

Photo by Maxim Hopman on Unsplash

Stochastic processes theory is wonderful and full of theoretical opportunities for those who are interested in quantitative trading. Sometimes, in order to test a trading strategy, simulating a stock market could be useful. Let's see how theory comes into help and how to convert it into practice using Python.

Note from Towards Data Science's editors: *While we allow independent authors to publish articles in accordance with our rules and guidelines, we do not endorse each author's contribution. You should not rely on an author's works without seeking professional advice. See our Reader Terms for details.*

Market returns

More than the price, when it comes to modeling a stock market, the most important object we can model is the return. If we consider a particular day *n* and its previous day *n-1* the return is calculated as:

$$r_n = \frac{p_n - p_{n-1}}{p_{n-1}}$$

Starting from this definition, we can calculate the price time series using the following formula:

$i=1$

So, knowing the start price *p0*, we can calculate future prices using the sequence of returns.

The problem is that the returns are stochastic objects, so each realization of the returns will give us a different time series for the prices. The stochastic behavior of the price is modeled in this way.

So, if we want to simulate price time series, we need to make some assumptions about the returns and then apply the formula above. Geometric Brownian Motion is one of these assumptions.

What is Geometric Brownian Motion?

GBM is a particular model of the stock market in with the returns are uncorrelated and normally distributed.

We can mathematically translate this sentence as:

$$r_i \sim \mathcal{N}(\mu, \sigma)$$

that is, the returns are normally distributed with mean μ and standard deviation σ. Keep in mind that these parameters are time-independent, so the process is called "stationary".

μ is the mean value of the returns. If it's positive, we have a bullish trend. If it's negative, we have a bearish trend. The higher the absolute value of this parameter, the stronger the trend.

σ is the volatility of the returns. The higher this value if compared with μ, the more erratic the price.

Going back to portfolio theory, the Sharpe ratio with risk-free return equal to 0 is μ/σ.

$$dr_t = \mu r_t dt + \sigma r_t dW_t$$

where *W* is the Wiener process.

This model is often used in financial mathematics because it's the simplest stock market model you can build. For example, it's the theoretical base of Nobel-award Black-Sholes options theory. However, this model has been proven not to be completely correct, because stock market returns aren't normally distributed nor are stationary, but it's a good point to start from.

The SDE can be analytically solved under, for example, Ito's interpretation, but in reality we never have a continuous time, since the transactions are discrete and finite. So, if we want to simulate a GBM, we can simply discretize the time keeping the normality of the returns. It is mathematically equivalent to numerically solving the SDE using Euler-Maruyama method.

The idea is, then, very simple: generate *n* normally distributed random variables and calculate future prices starting from a start price.

Let's see how to do it in Python in less than 10 lines of code.

Python simulation

First, let's import some useful libraries:

```
import numpy as np
import matplotlib.pyplot as plt
```

Now, we have to define μ, σ and the start price. We can use, for example, these values:

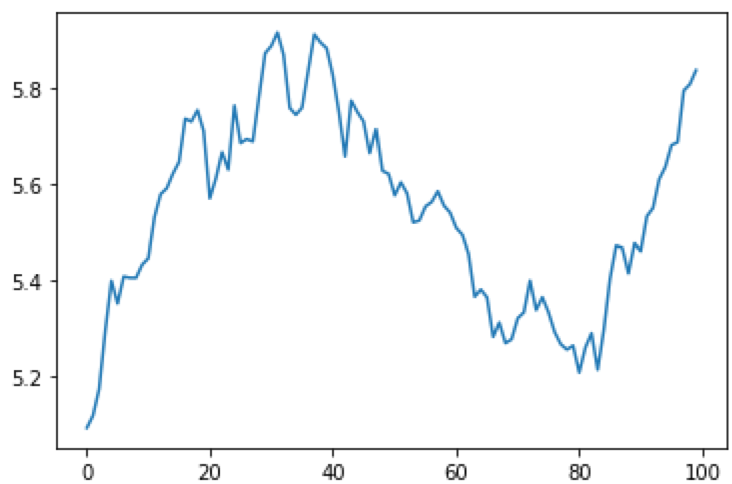
```
mu = 0.001
sigma = 0.01
start_price = 5
```

number generator in order to make reproducible results. Then we generate, for example, 100 values for the returns and finally we build the price time series starting from the start price.

```
np.random.seed(0)
returns = np.random.normal(loc=mu, scale=sigma, size=100)
price = start_price*(1+returns).cumprod()
```

Finally, we can plot these results:

```
plt.plot(price)
```



Simulation of a stock market. Image by the author

That’s it. 9 lines total. It really looks like a stock price, doesn’t it?

We could, for example, increase the value of μ and see what happens.

For example, with μ=0.004, we have:

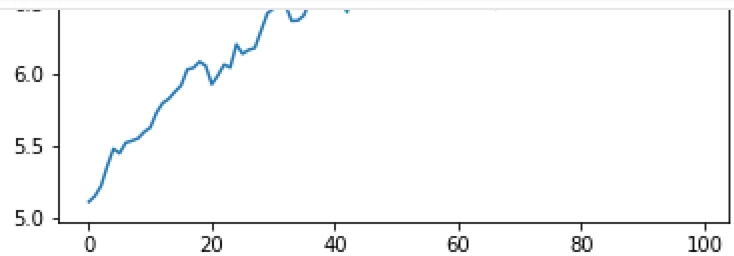


Image by the author

As we can see, a higher value of μ leads us to a stronger bullish trend.

Conclusions

Simulating a stock market in Python using Geometric Brownian Motion is very simple, but when we do this exercise we need to keep in mind that the stock market is not always normally distributed nor it is stationary. We could, for example, apply a time dependence on μ and σ or use a different probability distribution for the returns.

Gianluca Malato is an Italian Data Scientist and a fiction author. He is the founder of YourDataTeacher.com, an online school of data science, machine learning and data analysis.