Geometric Brownian Motion, Black-Scholes and the Financial Market

Joseph Iovine
Physics Department, Cornell University

(Dated: March 15th, 2021)

This report outlines the first step of mathematical modeling in the financial markets through geometric Brownian Motion. The building block for the Black-Scholes Model used as the first mathematical model in the options market. This was done through a script in python along with data provided by yahoo finance.

I. Introduction

In order to make financial investment decisions, simulated price paths of financial assets are often used to make future predictions of its value. These simulated paths are modelled by a stochastic process known as geometric Brownian motion. Using estimates of an assets drift and volatility, any trader can simulate thousands of paths to aid their investments [1].

How did a theory popularized by Einstein in 1905 modelling the randomness of a molecules path break its way into the financial field? Although not given the name Brownian motion, five years before Einstein, French mathematician Louis Bachelier introduced the idea that asset prices in the short term were completely random. He argued that if asset prices were not completely random, investors would exploit the pattern and eliminate it. Scrutinized for this idea, it was buried for over fifty years until resurfaced in the hands of American economist Paul A. Samuelson. Samuelson further developed Bachelier's ideas in his paper *Proof That Properly Anticipated Prices Fluctuate Randomly*, a cornerstone of financial literature. Geometric Brownian motion (Equation 1) was now the dominant model for simulating asset prices, composed of the randomness of an assets price and its expected rate of return [2].

$$S_t = S_0 exp(\left(\mu - \frac{\sigma^2}{2}\right) t + \sigma W_t)$$

Eq. 1, GBM formula

In 1973, economists Fischer Black and Myron Scholes had an idea to introduce the principles of geometric Brownian motion to the options market. The model they developed, known as the Black-Scholes Model (Equation 2), took into account the value of a stock option and its risk-free return rate to estimate the price of an option in the future [2].

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} = rV - rS\frac{\partial V}{\partial S} \quad \text{(a)}$$

$$C = S_t N(d_1) - Ke^{-rt} N(d_2) \quad \text{(b)}$$

Eq. 2, Black-Scholes in PDE form (a), solved form for a call option (b)

Enter Python. Python is a versatile programming language with strong numerical analysis power. Utilizing this along with well-developed modules for grabbing asset data, simulating the path of hundreds of stocks using years of historical data can be done by any investor from their own laptop. In this report I outline how to use python and the theories of geometric Brownian motion and Black-Scholes to develop simulations for any stock and price any call option.

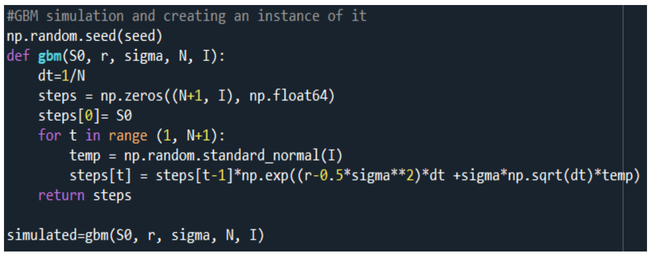
II. Getting Started

Before building the models, I had to familiarize myself with coding in Python and how to extract historical stock data. The overall goal was to plot simulated paths of a stock using geometric Brownian motion, and then use the Black-Scholes Model to price the value of an option and if it's worth it to buy.

To aid in this project there were three modules that I installed to my IDE: MatPlotLib, Numpy, and Yahoo-Finance. These modules helped in generating plots, using mathematical functions and extracting stock data. After familiarizing myself with the methods used in these packages and relearning Python, it was time to start building the models.

III. GBM Model

The geometric Brownian motion function (Figure 1) required five parameters: initial stock price **(S0)**, the expected rate of return or drift coefficient **(r)**, the diffusion coefficient **(sigma)**, time frame **(N)**, and number of iterations **(I)**.



```
#GBM simulation and creating an instance of it
np.random.seed(seed)
def gbm(S0, r, sigma, N, I):
    dt=1/N
    steps = np.zeros((N+1, I), np.float64)
    steps[0]= S0
    for t in range (1, N+1):
        temp = np.random.standard_normal(I)
        steps[t] = steps[t-1]*np.exp((r-0.5*sigma**2)*dt +sigma*np.sqrt(dt)*temp)
    return steps

simulated=gbm(S0, r, sigma, N, I)
```

Fig. 1, GBM function and instance of it

Starting with an array of just the initial price, the parameters would get inputted into the GBM formula contained in the for-loop and the value returned is then appended to the array. This process continues until the entire array is filled. Utilizing the Yahoo-Finance package, I was able to grab data for a stock and then calculate the drift and diffusion coefficients over a specific time period by using Numpy. Now with accurate parameters that reflect the real stock, I can request any number of simulations to be ran and generate a plot to compare them. I found the most pleasing results to be several simulations as there was too much noise with larger numbers. Below is an example for Microsoft using 2020 for data and ran six times (Figure 2).

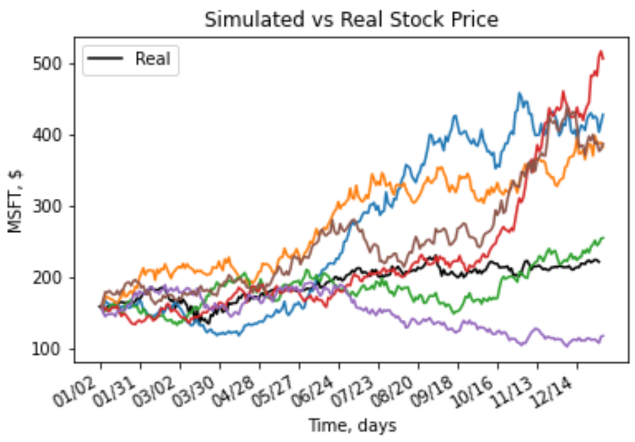


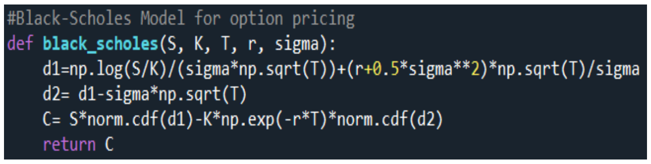
Fig. 2, Microsoft over course of 2020

Although very useful for modelling stock behavior, there are two shortcomings when using GBM: volatility is not constant in the real world, but is assumed to be in this calculation, and unpredictable events can cause a drastic change in a stock's price.

Due to the random nature of GBM, it makes it difficult to eliminate risk in the short term. This is where Black-Scholes enters the room. Their big idea was to hold a long position in the stock and pivot to a short position in the option in order to create a hedged position.

IV. Black-Scholes Model (BSM)

The function for the Black-Scholes equation (Figure 3) is short and sweet. It consists of a stock's price **(S)**, the option's strike price **(K)**, time until the option expires **(T)**, the drift coefficient **(r)**, and the diffusion coefficient **(sigma)**.

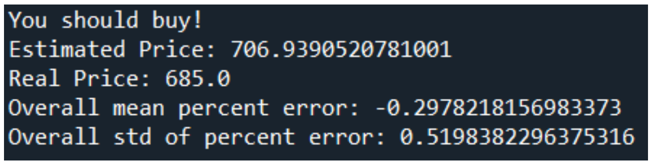


```
#Black-Scholes Model for option pricing
def black_scholes(S, K, T, r, sigma):
    d1=np.log(S/K)/(sigma*np.sqrt(T))+(r+0.5*sigma**2)*np.sqrt(T)/sigma
    d2= d1-sigma*np.sqrt(T)
    C= S*norm.cdf(d1)-K*np.exp(-r*T)*norm.cdf(d2)
    return C
```

Fig. 3, Black-Scholes Model

The more difficult portion of this model was parsing through the options data and picking specific contracts to analyze. Like the GBM model, I took an empty array and embedded the above function in a for-loop that ran through all of the contracts that I had stored in a data frame that I grabbed from Yahoo-Finance. This outputted an array of estimated prices for each contract on a given day.

I took two approaches to analyzing the data: as a collective for all contracts with a specific expiration date, and as a standalone contract. The first was done by calculating the mean percent error for the real price vs the estimated price given from the BSM. Meaning if the mean percent error was positive or negative the stock was generally over or underpriced. For a standalone contract, I just used a direct comparison. If the BSM gave a price higher than the real price it meant to go buy it. I show a sample output using a Tesla contract that expires March 19th (Figure 4).

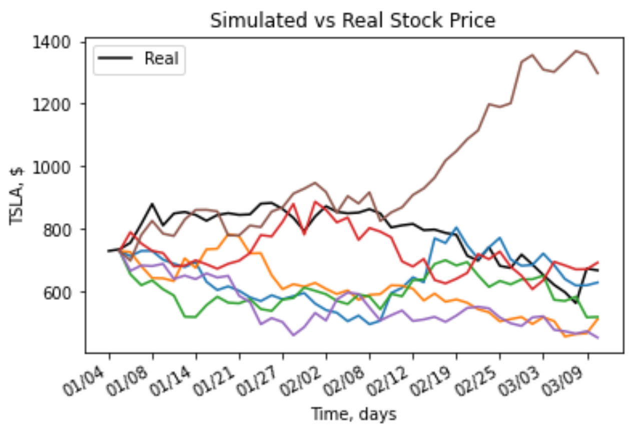


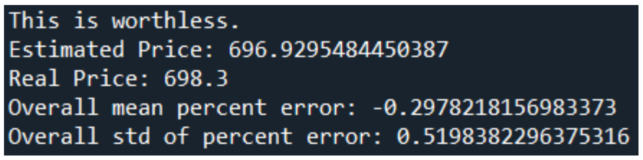
```
You should buy!
Estimated Price: 706.9390520781001
Real Price: 685.0
Overall mean percent error: -0.2978218156983373
Overall std of percent error: 0.5198382296375316
```

Fig. 4, Sample output for a Tesla option

V. Final Product

The final step was to combine the two models so that I can input any stock ticker with a corresponding contract name to receive data on its behavior so far this year and if the contract should be bought. This involved rewriting my whole code so that the two functions shared the same parameters when necessary and automatically grabbed the proper columns for the inputted stock data frame (which I should have done originally, but you live and you learn). Now using a not so enticing Tesla contract, I can get all this data (Figure 5) shown below simply by entering any option's contract name.





```
This is worthless.
Estimated Price: 696.9295484450387
Real Price: 698.3
Overall mean percent error: -0.2978218156983373
Overall std of percent error: 0.5198382296375316
```

Fig. 5, Final sample output for a Tesla contract

VI. Conclusion

My original goal was to study geometric Brownian motion and its relationship to the financial market based on an interest I had in quantitative finance and a small excerpt in a book I read discussing it. Through this, my goal was expanded to building a Black-Scholes Model in python, something that I did not know existed until I started researching the topic. The beginning was intimidating as I have not coded using Python in over three years and was diving into a topic I had very little knowledge of. In the end I am satisfied with what I have completed and have completed my overall goal.

I plan to expand on this in the future as a personal project and take a more rigorous approach since these two models are just approximations and blindly following them offers no real insight. These concepts are incredibly complex and are the reason why there is a whole field a mathematics dedicated to it. Understanding the limitations of the models, such as the constant volatility of an option, and how to circumvent them can lead to clean, profitable trading.

Geometric Brownian motion and the Black-Scholes Model are the cornerstones of quantitative finance and an introduction into how mathematicians and physicists took over the financial industry. I have completed my overall goal and acquired skills that will be very practical for my pursuit of a career in the quantitative finance field.

1. Joel Liden, Stock Price Predictions using Geometric Brownian Motion, UPPSALA Universitet (2018)
2. Jorgen Veisdal, Brownian Motion in Financial Markets, **https://www.cantorsparadise.com/brownian-motion-in-financial-markets-ea5f02204b14**
3. Umut Yildiz, Simulating Stock Prices in Python Using Geometric Brownian Motion, **https://towardsdatascience.com/simulating-stock-prices-in-python-using-geometric-brownian-motion-8dfd6e8c6b18**