Monte Carlo Simulation

Rupesh Poudel, MEMS, 614779

Seminar Paper:
Numerical Introductory Course

Lecturer: Prof. Dr. Brenda López Cabrera

March 11, 2022

Abstract

Monte Carlo Simulation uses random numbers to perform repeated sampling of a complex process. This paper generates uniformly distributed random sequence of numbers using Linear Congruential Generator and Lagged Fibonacci Generator, and converts the uniformly distributed random sequence of numbers into the standard normally distributed random sequence of numbers by Box-Müller Method, Inversion method, and the Marsaglia method. These random sequence of numbers, along with the default random sequence of numbers generated from Python's Numpy library will be used to carry out the Monte Carlo Simulation to predict the price of the stock of the Apple Inc. 6 months into the future. The sensitivity of the simulation with respect to a change in the random number generator, the number of trials, and the computational time of the simulation are reported. MSE, RMSE, random number generation time, and 95 % Confidence Intervals are also reported. The Kernel Density Estimators is drawn for each variation.

Contents

1 Motivation

In a financial market, any information about the value a stock price of a particular company prevailing in the future is highly sought after. Investors can buy shares, even with high leverage, if the shares are expected to rise in the future; and offload their position in addition with short selling if the shares are expected to fall. Complex financial derivatives like futures and options, whose payoffs are based on the future price of the underlying, can allow investors to gain the profits with relatively less initial investment.
Different private and institutional investors have their own future price prediction model which they use to pick stocks to buy. The prediction power of the model determines the success of an investor. It is therefore important to find a suitable model that outperforms the market and hopefully the competitors. This paper will discuss about Monte Carlo Simulation, a statistical method of repeated sampling and aggregation, as an initial estimate for such prediction.
Monte Carlo Simulation is a tool that can be used to solve problems numerically when it is too complex to be solved analytically. It can solve higher dimensional problems whose numerical evaluation is computationally engaging and tedious. Furthermore, when the underlying input for the estimation, forecasting, or some decision process is stochastic in nature, Monte Carlo Simulation can provide with a range of choice for that input.
Scientists working for the "Manhattan Project", namely Stanislaw Ulam, John von Neumann, and Nicholas Metropolis, are credited for the invention of Monte Carlo Simulation. It all started when Ulam wished to calculate the probability of him winning any hand of Solitaire, a popular card game, analytically. Unable to find a solution, Ulam hypothesised whether playing, or equivalently simulating the game for sufficiently large number of trials and counting the number of wins would be a reasonable alternative, when combinatorics involved in the game is too complex. The same approach could be replicated in numerous other sectors, including nuclear physics, where they had to "solve the problem of neutron diffusion in fissionable material." Neumann, after being suggested the approach, wrote a detailed, computer execution-able set of instructions for neutron diffusion to be executed in the ENIAC, a first generation computer. Metropolis suggested the name "Monte Carlo", referencing to Ulam's uncle and his gambling tendencies and frequent visits to the casinos of Monte Carlo, Monaco. Metropolis (1987) outlines the aforementioned origin story and formalises the method of Monte Carlo Simulation.
Uniformly distributed (pseudo) random numbers are generated using an algorithm which in our previous story is the middle square digits. These random numbers are then converted to a suitable distribution that best matches the problem under consideration. The inverse of the cumulative distribution function of our desired distribution is used for the conversion. These transformed random numbers are used to carry out instances of deterministic computation. Appendix 1 shows an example of a coin toss experiment which can serve as a primer for the way in which Monte Carlo Simulation works, and Appendix 2 outlines the middle square digits method of generation of random sequence of numbers.

2 Relation to existing literature or methods

Monte Carlo Simulation draws upon the Bernouli's law of large numbers. When an experiment is conducted multiple times, as the number of times the experiment is performed increases, the mean of the results from those experiments will converge to the true (theoretical) mean. A fair coin tossed countably infinite times will see the probability of each of its outcome {Heads or Tails} approach 0.5. A fair six-sided dice will witness the probability of each of its side to face upwards after a spin converge to 1/6.
Estember and Maraña (2016) use Monte Carlo Simulation to model future stock prices and compare the Geometric Brownian Motion (GBM) vs the Artificial Neural Network (ANN) method. They show GBM method to be more effective in predicting the future stock price movement. This paper will also follow suit, and simulate price paths of trials using a given mean and volatility of returns.

3 Application for Finance and Statistics

3.1 Application for Statistics

Statistics is among the many fields where the use cases of the Monte Carlo Simulation is plentiful. Monte Carlo Simulation can be used to provide an efficient random estimate for the Hessian matrix. Yu (2016) uses automatic differentiation under Monte Carlo setting to compute estimate of the Hessian Matrix. The Hessian matrix can be used in Fisher Scoring, and subsequently the Fisher Information matrix. Das, Spall, & Ghanem (2010) show the use of the resampling algorithm, a Monte Carlo technique, for computing Fisher Information Matrix. The analytical determination of the Fisher Information Matrix in non linear models is difficult because of the intractable higher dimension integration and modelling requirement.
Similarly, Bayesian Statistics is making use of Monte Carlo Simulation. Chen, Shao, Ibrahim (2012) develop advanced Monte Carlo methods by using posterior distribution's sample to compute posterior qualities, namely: marginal posterior densities, marginal likelihood, Bayesian Credible Intervals and so on. Monte Carlo Simulation is used in statistical interference as well. Hypothesis testing, comparison of variance of two estimators, evaluating higher dimension integrals and parametric bootstraps are some uses identified in Gentle (2009, Chapter 11).

3.2 Application for Finance

Hull (2018) suggests Monte Carlo Simulation for option pricing. Path dependent options such as Asian options, and options with many stochastic variables can use Monte Carlo Simulation. Hull(2018, pg 511) also suggests Monte Carlo Simulation to calculate the dollar change in portfolio in a day, or VaR. The value of derivative such as options can be calculated by sampling the random path of an underlying in a risk neutral world, calculating the payoff for the option, and run sufficiently high number of trials so that average of the sample payoff can be calculated to get the expected payoff in the risk neutral world. The option is now discounted expected payoff at the risk free rate. (Pg. 469). The performance of stop loss hedging can be seen by using Monte Carlo Simulation.

In addition, an investor can analyse the default risk of a company or credit. The cash flow analysis of an uncertain project can be carried out. Instead of a point estimate, a confidence interval can be computed which guides the investors to make informed decisions. Furthermore, the evolution of a stock price, and consequently the investment portfolio of an investor can be modelled. This paper will carry out an analysis on the evaluation of the Apple share price in the coming sections.

4 Formalization of the method

The paper starts with a very generic description of a stock price process as listed below:

$$dS_t = \mu S_t dt + \sigma S_t dZ_t$$

To make sure that the stock process is a martingale, and that the next value in the sequence is equal to present value in terms of conditional expectation, change in measure is required. The risk neutral price process, without taking into account the cost of carry, changes into the following.

$$dS_t = r S_t dt + \sigma S_t dZ_t$$

This was the same process represented in Black, Scholes, Merton (1973). From here on, I will include the cost of carry. Apple paid 88 cents of dividend in 2021 where its ending price was $ 158.58. This is a rate of 0.55 percent. Treasury bill of 6 month maturity was trading for 0.6 percent in March 1, 2022. So, to assume the cost of carry similar to a risk free rate is not a far fetched assumption. The Geometric Brownian Motion now becomes:

$$dS_t = (r - \delta) S_t dt + \sigma S_t dZ_t = \sigma S_t dZ_t. \;\; for \;\; r = \delta$$

$$dS_t / S_t = \sigma dZ_t$$

Assuming that, for the purpose of valuation of the stock, the only change in the future are driven by the innovation terms σ, we can simplify the prediction. Also, the diffusion term, or the volatility of the process is assigned the value of the historical volatility of log returns. Then random numbers which follow standard normal distribution are generated and then transformed into random numbers with the aforementioned historical volatility and 0 mean. To predict N days into the future, N random numbers that adhere to zero mean and the historical volatility are drawn. The result of which will guide the daily movement of the stock each day for N days. Plotting the value of the stock from initial price to the price prevailing in the day N into the future will plot one price path for 1 trial of the simulation.
Performing the simulation for M times will lead to M prices for day N. As random sequences of numbers were generated for each price path, they will lead to M different prices for day N value of the Apple share. These M different prices for day N value are stored in a different list, or a price series, to observe its behavior. One can compute the mean of the price series, the minimum and the maximum value of the price series. A Kernel Density Estimator, similar to a histogram, can be plotted to see how the final day price series are lined up in a plot. From there, one can not only see the point estimate for the future value of the share, but also the range of the values.

In the section below, I will discuss about random numbers. I explained where and how the random number was used. The following section explains why random numbers are needed in Monte Carlo Simulation.

4.1 Random Numbers

Random numbers, or more accurately, a random sequence of numbers, are essential for Monte Carlo Simulation. In this paper, random numbers are required to dictate where the stocks are supposed to move on any given day for any given trial. Without sufficiently randomized sequence of numbers, the estimates of the inputs will be biased. Biased inputs will lead to less trustworthy outputs. To get unbiased estimates, different attempts have been made to construct algorithms that produce uniformly distributed and independent random numbers. Mersenne Twister algorithm by Matsumoto & Nishimura(1998) is one such attempt. Random numbers must adhere to some conditions for it to be usable. Güneş(2013) presents some tests for randomness. Each numbers in the range must be equally likely, the random numbers must be continuous and not discrete. The mean and variance must neither be too high or too low. Similarly, there should not be significant auto-correlation between numbers. There must be no seeming pattern between numbers. A period of a sequence of random number reflects the number of terms in the sequence before the sequence repeats. An appropriate PRNG must consist of very high period.
For the purpose of this paper, once I generate the uniformly distributed random sequence of numbers, I test for their randomness. I plot the scatterplot of two consecutive random numbers to check whether there is a pattern. The points in scatterplot which is all over the graph shows that there is no apparent pattern. I also compute the variance covariance matrix of the random numbers to check whether there is an association in the random sequence of numbers generated from different algorithm/transformation. A kernel density estimator is also plotted to show the pdf of the estimated random sequence of numbers, be it uniform or standard normal.

4.2 Pros and Cons of the Method

Pros: Deterministically complex problems can be solved numerically. As was with the coin toss, nuclear fission, solitaire, and stock price estimation, Monte Carlo method uses random number based numerical approach to provide estimate for an answer. For non- invertible functions, Monte Carlo can use reverse search to plot the graph of the inverse when possible. One can easily visualise at which domain of the inputs the function becomes unbounded, and where the function can be tamed. Monte Carlo method is a powerful tool for solving any problem with uncertainty involved. When random numbers with an appropriate distribution is used as an input to replace the uncertain factors, the solution has reasonable accuracy with surprisingly low variance.
Cons: Considerable computational time and resources are consumed by Monte Carlo Simulation. Reverse search to invert a non-invertible function will make the computer evaluate the function under consideration at many points for a given range of inputs. The evaluation will most likely be discrete with spacing of the input predetermined by the user. As it is normal for Monte Carlo simulation to involve more than a million simulation, this leads to a same set

of computation being performed N (number of simulation) times. Each simulation replicates a process. This process itself will include some computation M. So, carrying M computations N times means MN computation, which takes processing speed away from the computer. Sometimes, a computer cannot sustain request to perform simulation citing low available resources and the device crashes. The choice of programming language, the efficiency of algorithm, the choice of storage of data, the allocation of bits for storage of calculations all play a part in determining the computational time. Moreover, If our input or parameters are garbage, our output will be garbage too. This includes the domain of input and the underlying probability distribution.
Monte Carlo method does not factor in irrationality and behaviour aspects of "dismal science" field of finance and economics very well. Monte Carlo methods do not accurately predict tail events. I did state that the evolution of the stock price is determined solely by innovation terms. But, major shocks to the industry, such as a possible war breakout, and supply chain issues like global chip shortage, has to be specifically modelled and Monte Carlo Method might need to incorporate jumps. The model I will use is a basic model which has major drawback, and is far from reality, specially when reality itself is deviating from the norm on a daily basis. Use of random numbers and Monte Carlo simulation itself, that can have extreme maximum and extreme minimum values, does replicate tail event in some of its simulation. However, this extreme is a result of experimental design of Monte Carlo Simulation as a statistical tool and is found even when no tail event occurs in the real life.

5 Data Description, Simulation Settings

5.1 Data Description

Figure 1: Share Price History of Apple Inc.

I am using Monte Carlo Simulation to predict the future share price of Apple Inc. I wish to concentrate the analysis on the COVID-era. I set the timeframe for the data to "2020-1-1" to "2022-02-28". I chose the python library "yfinance"

to download the data. The library "yfinance" allows the user to extract the share price history of Apple Inc. from the Yahoo Finance database. Apple trades on NASDAQ under the ticker AAPL. I specified the ticker symbol and the timeframe of the price series to download the data of Apple Share under 6 headings: "Open", "High", "Low", "Close", "Adj Close", "Volume". I took "Adj Close", short for "Adjusted Close" and plotted the progression of the adjusted closing price with respect to time. Figure 1 is that plot. Looking at the figure, Apple has seen a rapid post-COVID boom. The market crash brought about by COVID appears to be a very small price fall in comparison to the rise thereafter.
S_t/S_{t-1} gives the return of the price series. Here S_t is the price of the stock at time t, and S_{t-1} is the price of the stock a day before. I will call this a return series. From this returns series, the natural logarithm of stock returns from day $t-1$ to t is calculated. The paper assumes that the natural logarithm of the stock follows normal distribution with given mean and standard deviation. Figure 2 below gave a mean and standard deviation of 0.00148 and 0.023318 respectively. To assume a mean of 0 is then reasonable. This standard deviation of "log returns", the natural logarithm of daily stock returns, is around 2.33 percent. From the visual inspection of Figure 2, the log returns seem to have heteroskedastic error terms. They are not stationary. Non-parametric models or variants of GARCH are appropriate to model this type of series. I choose to assume the volatility of 2.33 percent to persist in the next 6 months as well. Six months equals 0.5 * 252 = 126 trading days.

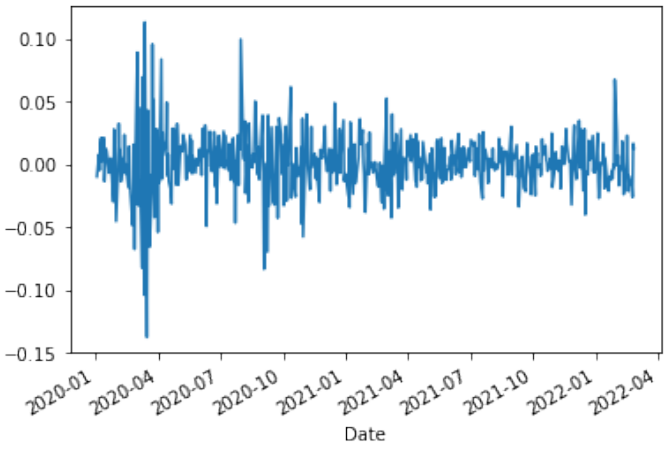

Figure 2: Returns History of Apple Inc.

Linear Congruential Generator, Lagged Fibonacci Generator, and Numpy's default pseudo uniform random number generator generates uniformly distributed random sequence of numbers. These sequence is thereafter transformed into standard normal distribution using the Box-Müller Method, Inversion method, and Marsaglia method, details of which is found in Franke, Härdle, & Hafner (2019, Chapter 6). These standard normal random numbers are converted into normally distributed random numbers with the mean of 0 and a standard deviation of 2.33 percent. This way, I fulfilled another criteria of Monte Carlo

simulation outlined by Metropolis (1987), i.e. convert uniformly distributed random number I generated into a suitable distribution that best matches the problem under consideration. The suitable distribution for log returns is normal distribution with mean around zero and standard deviation of around 2.33 percent.

5.2 Simulation Setting

For the initial run, a matrix of size [**126,100000**] consisting of normally distributed random numbers with the desired mean and dispersion is generated using Numpy's default random number generator. As I wish to predict the price path for 126 trading days ahead and run the Monte Carlo Simulation 100,000 times, the matrix has that specific dimension. For comparability and results reproduction, the matrix is stored under a separate variable so that the same random sequence of numbers can be used to check the sensibility of the final results on a reduced number of simulations. In case of other random numbers, an array of 12.6 million normally distributed random sequence of numbers with the same mean and standard deviation are generated. The array is then reshaped into a matrix of size [**126,100000**]. Different set of matrices representing different pseudo random number generators and consequently different conversion methods will follow the same procedure.

6 Results of Empirical Analysis

6.1 Initial Run

As stated above, 100 thousand simulations for 126 trading days was carried out. The following figure to the left shows how the Apple stock is predicted to progress for the next 126 trading days for 100,000 simulations. The following figure to the right is a kernel density estimator drawn from the 100,000 prices predicted by the model of the Apple share for the final day. The kernel density estimator has a positive skewness of 0.801 and an excess kurtosis of 1.135. The mean of the final day series is $170.512 and the standard deviation is $45.362. In this model, the maximum simulated stock price reaches upto $525.35 and the minimum simulated stock price reached upto $50.82. The 95 percent confidence interval for mean with 99,999 degrees of freedom is [170.23119, 170.79350]. This is to say that I am 95 percent confident that the true population mean lies within the range specified before. This is an increase from the initial price of $ 164.85. So my initial recommendation, within the scope of my model, is to buy the Apple share, as the share price is expected to appreciate in the future.

6.2 Sensitivity: Change in Simulation Frequency

Once I have the future price of the Apple Inc. share generated from the Monte Carlo simulation. I check for the effect that the number of simulations has on the price, confidence interval and other values described above. For this check, I extract the first 10,000 random numbers I used in the simulation above. Then, I repeat the procedure from above for 10,000 simulations. The following figure on the left shows the progression of the stock price for 10,000 simulations; and

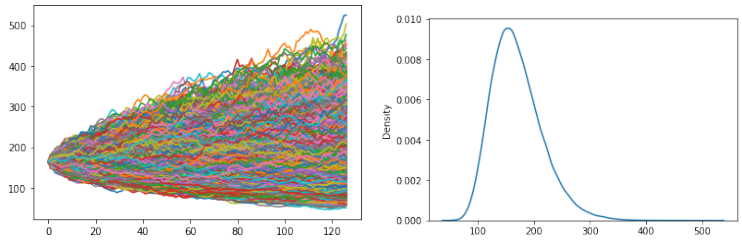

Figure 3: 100,000 simulations for 126 days and its KDE plot

the figure on the right shows the Kernel density estimate associated with the 10,000 prices for the 126^{th} day for 10,000 simulations.

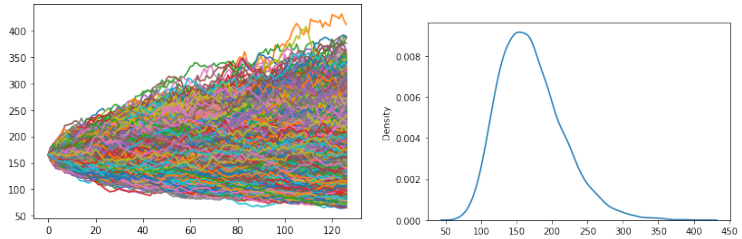

Figure 4: 10,000 simulation for 126 days and its KDE plot

In this simulation, the mean of the simulated values of the 10,000 prices for the final day is \$170.50998 and the standard deviation is \$45.36971. When this number is compared to the benchmark with 100,000 simulations. There is very small difference. In fact, this difference would be barely visible, if I had rounded the answer to euros and cents only. The range of values in the final day of the prediction is narrower than the benchmark. The maximum price was \$411.37 and the minimum price is \$65.02. It is to be expected when 90 percent of random numbers used in the benchmark are taken away for the new simulations, then the numbers that would lead to extreme value in either side can disappear. The skewness is 0.753 and the excess kurtosis is 0.827. The distribution has less skew than the benchmark distribution and approach normality in comparison to the benchmark. Another difference comes in the form of a 95 percent confidence interval. The 95 percent CI for 10,000 simulations is given by (\$169.62064,\$171.39932). This is wider than the benchmark value.
I ran a 1000 simulations to see if the similarity would still remain and whether the differences would even widen. For 1000 simulations, the mean of the price series is equal to \$171.94 and the standard deviation is equal to \$45.84. The price range is still only \$1.43 greater than the benchmark. Assuming that 100,000 simulations would give the true future value of the price, then to be able to predict within 1 dollar 43 cents is a good news who are willing to sacrifice accuracy for computational resources consumed. I no longer had to list upto 5 decimal places to show that the price predicted were indeed different. The skewness was 0.889 and the excess kurtosis was 1.473. They did not continue to decrease. Had that happened, that would actually indicate the poor quality of random numbers. The range of prices were less wide. The maximum value was

$411.37 and the minimum was $77.48. The maximum was still the same but the minimum changed. This means, the set of random numbers that produced the maximum value still remained within the first 1000 numbers whereas the set of random numbers that produced the minimum value in 10,000 simulations was not in the first 1000 rows of the random number matrix. The gap widens in the confidence interval yet again. The formula for the confidence interval clarifies the reason:

$$CI = \left[\mu - z * \frac{\sigma}{\sqrt{n}}, \mu + z * \frac{\sigma}{\sqrt{n}}\right]$$

In all three of the cases, I found the near, if not similar, values of μ and σ. The z-score for 95 percent confidence interval is approximately 1.959, which remains the same for all 3 cases. This means, the only real change comes from the number of simulations. 1000 simulation leads to the denominator of $\sqrt{1000}$, 10000 simulation leads to a denominator of $\sqrt{10000}$ and so on. For larger value of n, the term that gets added and subtracted from the mean $z * \sigma/\sqrt{n}$ gets smaller and the confidence interval gets narrower. For the 1000 simulations, the smallest n among three cases presented, the 95 percent confidence interval was [$162.94, $180.93]. The figure below shows the price progression for 1000 simulation on the left and its KDE plot on the right. Even after choosing the appropriate kernel by the seaborn's default setting, there is still roughness in the KDE plot. In the top, there is a M-shape present. The plots with less number of simulation is less refined in comparison to the plots with more number of simulations.

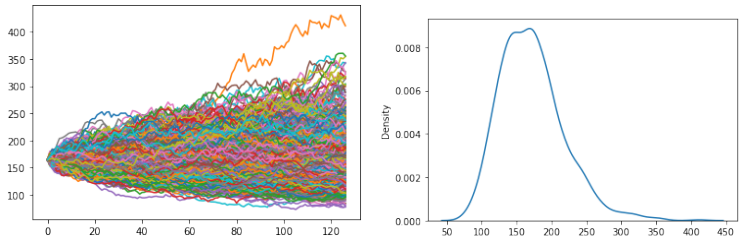

Figure 5: 1000 simulations and its KDE plot

6.3 Sensitivity: Different Random Numbers

I now wish to check whether using different random numbers can have an effect on the price and the confidence intervals. I will be using 100,000 simulations. I will report more decimal places to show the differences when necessary. For the first random number, I used Lagged Fibonacci generator for generating 12.6 million uniformly distributed random sequence of numbers and transformed these numbers into standard normal distribution using Box-Müller method. I reshaped the numbers into [126, 100000] shape and used this matrix instead of Numpy's 2 dimensional matrix of the random numbers from the initial run before. The process was repeated like the benchmark process before. The mean of the price series obtained this way is $170.495 and the standard

deviation is $45.343. The benchmark predicted $170.512 with a standard deviation of $45.362. From now on, I will report the value associated with the benchmark inside a () right next to the number. The skewness is 0.805(0.801) and the excess kurtosis is 1.144(1.135). The maximum is $494.46($525.35) and the minimum is $53.89($50.82). The 95 percent confidence interval is [$170.21,$170.77]($170.23,$170.79). One observation is that the numbers are all slightly different. The most important numbers, the 95% CI and the mean were very close.

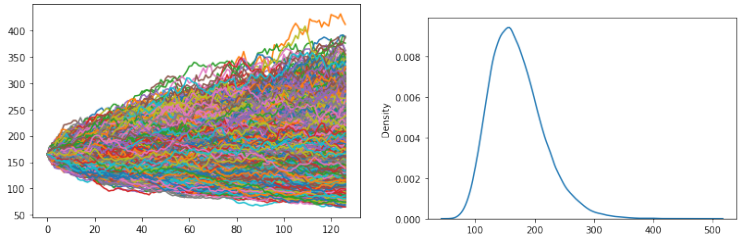

Figure 6: Lagged Fibonacci, Box-Müller, 100,000 simulations and its KDE plot

For the next evaluation, I let Numpy's default command generate the uniform random numbers and then converted these uniformly distributed random sequence of numbers into normally distributed random numbers by the use of Marsaglia method. A reshaping of 12.6 million numbers into appropriate dimension led to a unique matrix which replaced the random number matrix from the benchmark evaluation. Similar process as mentioned above was carried out to get the following results. The mean was calculated to be $170.65($170.51) and the standard deviation was calculated to be $45.34($45.36). The maximum and minimum was $548.854($525.35) and $55.25 ($50.82) respectively. The skewness and kurtosis of 0.814(0.801) and 1.201(1.135) was almost the same as well. The 95 % CI was [$170.37,$170.93] ($170.23,$170.79). The values were not the same but very close to each other. The figure below on the left shows the path the stock took for each simulation and the figure below on the right shows the kernel density estimation plot for the final day prices.

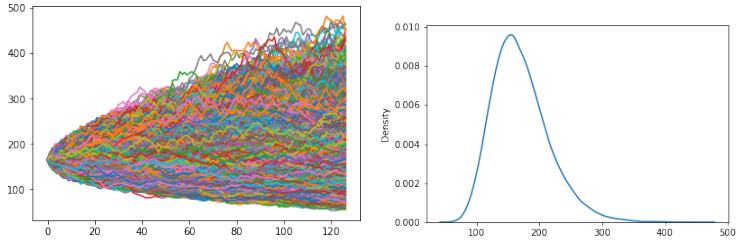

Figure 7: Numpy, Marsaglia, 100,000 simulations and its KDE plot

For the third and final comparison, I choose to generate the uniformly distributed random sequence of numbers using Linear Congruential Generators, and use the inversion method to convert uniformly distributed random sequence of numbers into normally distributed random sequence of numbers. Inversion method maps the point in x-axis that would lead to the CDF with value equal to

the uniformly distributed random number. 12.6 million random numbers was reshaped into a matrix and this matrix replaced the random number matrix from the benchmark to produce the Figure 8.

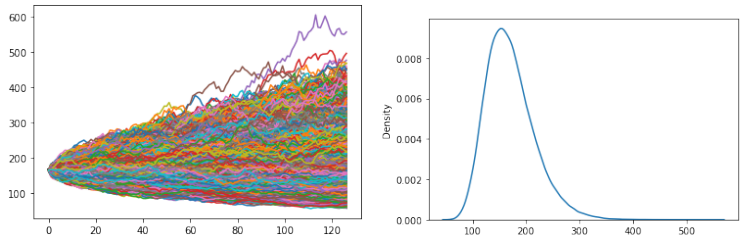

Figure 8: LCG, Inversion, 100,000 simulations and its KDE plot

To the left, as stated before, is the 100,000 simulated paths of the Monte Carlo simulation for Apple share price till 126 days to the future. To the right is the kernel density estimation plot for the stock prices prevailing in each path at the end of the simulation. The skewness is 0.826(0.801) and the excess kurtosis is 1.259(1.135). The maximum value among the final day price series is \$557.32(\$525.35) and the minimum value is \$57.56(\$50.82). The 95% CI was [170.26,170.82] (\$170.23,\$170.79) with a price series mean of \$170.54 and a standard deviation of \$45.49.

7 Observation

7.1 Random Numbers

So, after testing 3 other random number matrices to acquire the results, I come to a conclusion that the choice of random number generator did not make a big impact on the final results. The skewness, kurtosis, maximum and minimum were the property of random numbers. Their differences was what made the evaluation worthwhile. A commonality among all variants of Monte Carlo simulation for this problem was that the skewness was positive and the excess kurtosis was also positive. The values were all different. Despite the differences in aforementioned values, the values for confidence interval, mean and standard deviation came very close to each other. This is partly possible due to the quality of random number generated. Even though Linear Congruential Generator and Lagged Fibonacci Generator have their own drawbacks, they provided me with sufficiently random numbers and I used sufficiently large number of simulations to postulate my results.
The picture above compares the initial price at the start of the simulation (red line) with the future price predicted by the benchmark model (black line). The black line lying to the right of the red line implies the future appreciation of the stock price. The light blue area represents 1 standard deviation left and right to the mean of the predicted price series. Together with dark blue area, the figure only accounts for 3 standard deviation left and right, or 99.7% of all the observations. Here, outliers that go beyond either side are ignored, and a normal curve is drawn. This figure would be more interesting if the red line did not fall

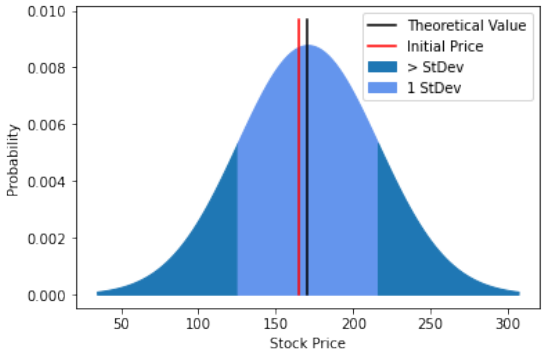

Figure 9: A different representation of the PDF

on the light blue area. That would indicate one of the two possible scenario: error in modelling, or an opportunity to gain from the financial market.

7.2 An overview of findings

Table 1 below lists the expected price, the standard deviation, the maximum predicted price, the minimum predicted price, the skewness in the prediction, the excess kurtosis in the prediction, the 95% Confidence Intervals with (nSim - 1) degrees of freedom, and the number of simulations(nSim) respectively. The unit of measurement for mean, standard deviation, maximum, minimum and the confidence intervals is USD.
Model 1 refers to the initial model run with Numpy's random numbers and 100,000 simulation. This serves as a benchmark model to compare the rest of the models. Model 2 takes only first 10,000 observations into consideration to do the analysis, and Model 3 only takes 1000 simulations. Model 4 takes 100,000 simulation but uses the lagged fibonacci random number generation algorithm to generate uniformly distributed random sequence of numbers and Box-Müller transformation to convert these uniform random numbers to normally distributed random sequence of numbers. Model 5 uses Numpy's own random number generator to generate uniformly distributed random numbers, and converts these numbers to normally distributed random numbers using the Marsaglia method. The number of simulations is still 100,000. Model 6 uses the linear congruential generator to generate uniformly distributed random numbers and then uses inversion method to convert the numbers to normal distribution. Model 6 also uses 100,000 simulations.
Table 2 outlines the mean squared error, the root mean squared error, the Anderson-Darling test statistic for normality, the p-value for Kolmogorov-Smirnov test, the time taken for generation of pseudo random numbers, and the computational time for each simulation. For the MSE and the RMSE, I used the values for the predicted prices for the final day together with a sequence I generated which was normally distributed with the same mean and standard deviation as the predicted price series. Then, I sorted the two arrays. Using scipy.stats in Python, I computed these two values. The tests for normality was also carried out from scipy.stats library in Python. Time is measured in seconds. The models 1 to 6 is the same as the models in Table 1.

S.N	mean	std	max	min	Skew	Kurt	95 % -CI	nSim
1	170.512	45.362	525.354	50.820	0.801	1.135	(170.23,170.79)	100,000
2	170.509	45.369	411.378	65.027	0.753	0.827	(169.62,171.39)	10,000
3	171.940	45.848	411.378	77.488	0.889	1.473	(162,94,180.93)	1,000
4	170.495	45.343	494.462	53.897	0.805	1.144	(170.21,170,77)	100,000
5	170.655	45.345	548.854	55.252	0.814	1.201	(170.37,170.93)	100,000
6	170.544	45.494	557.325	57.562	0.826	1.259	(170.26,170.82)	100,000

Table 1: Summary

Model	MSE	RMSE	AD T-stat	KS p-value	t(PRNG)	t(simulation)
1	69.307	8.325	659.50	0.0	0.319	55.754
2	59.991	7.745	64.06	0.0	0.0319	5.206
3	86.750	9.313	7.22	0.0	0.00319	0.591
4	69.708	8.349	653.53	0.0	136.46	61.825
5	70.793	8.413	669.95	0.0	83.58	64.495
6	72.895	8.537	685.93	0.0	26.709	61.263

Table 2: Error, Normality, and Computation Times

The first observation I wish to make is a p-value of 0.0 for the Kolmogorov-Smirnov test statistic. They are all 0. This means that the price series for my model are not normally distributed. This is reinforced by the fact that Anderson-Darling test statistic has very high value. A normally distributed series would have 0.787 or less as the test statistic for an $\alpha = 0.05$.
The pseudo random numbers generated and transformed by myself took more time than the PRNG of Numpy. This is because, their random number generators are more efficiently coded than mine. The simulation time decreased for decreased number of simulations, partly due to the fact that extraction, rather than generation of random numbers was done. But, even after taking this into account, the decrease in simulation time from Model 1 to 3 is apparent.
The Mean Squared Errors and the Root Mean Squared Errors are high in this model. An ideal value for Mean Squared Error and Root Mean Squared Error is 0, which signifies a perfect fit of true and predicted values. MSE and RMSE are common model selection criteria in academia. Following that trend, Model 2 with 10,000 simulations, has the least MSE and RMSE number.

8 Conclusion

This seminar paper briefly discusses Monte Carlo Simulation, a popular statistical tool with many use cases in finance, statistics, and even nuclear physics. Monte Carlo simulation uses random numbers to replace uncertain factors and run a complex process for numerous times, and the results are aggregated to provide an alternative or initial estimate of the solution. Originated in an era where computation were done via vacuum tubes, computational time, resources and set of instruction supported by the typical Monte Carlo Simulation were limited. Present day microprocessors and parallel processing GPUs allow for manyfold processing capacity and therefore its applications have increased in a

similar fashion. I used Monte Carlo simulation in this seminar to price Apple Inc. stock 6 months into the future. According to the model, 126 trading days from March 1, the Apple stock is expected to be trading around $170.51. I drew the kernel density estimate to visualise how the results were distributed around the mean. I used Seaborn's default function to plot the kernel density estimate. The default setting allows for an appropriate kernel, and chooses bandwidth according to the data. A bigger than appropriate bandwidth results in oversmoothing whereas smaller than appropriate bandwidth results in undersmoothing, where plots have rough edges and noises. In the stock price prediction, the most appropriate kernel is the Gaussian Kernel. The bandwidth is chosen so that the mean squared error of the kernel density estimator is minimized. I have left this to the seaborn's kdeplot function itself.

Other than that, this paper checked the sensitivity of the results with regards to the change in number of simulations and change in the pseudo random number generator. For decreasing number of simulations performed, the gap between the lower bound and the upper bound in the confidence interval kept widening. The confidence interval widens for decreasing number of simulation because the gap from the mean is inversely proportional to the square root of the number of simulations performed. More simulations indicate less gaps. The mean squared errors were similar, and the distribution of price series were not normal for any variation. The mean and the standard deviation were similar for 100,000 simulations and 10,000 simulations. For 1000 simulations, the mean was off by 143 cents only. For a different random number generator, 3 different sets of random numbers were checked. One uniformly distributed random number generator and one transformation method led to one set of random number to replace the default normally distributed random numbers. Lagged Fibonacci for Uniform Distribution and Box Müller Transformation made the first set. Numpy's default uniform random number and Marsaglia method transformation made the second set. Linear Congruential generator and inversion method made the third set. In all three sets of random numbers, for 100000 simulations, there was very little difference in the confidence intervals generated and the value of the mean. The quality of all three of the random number generated was checked separately in the form of scatterplot. The correlation between random number generated was also tested. Proper random number generator will lead to similar results. Small difference is tolerable as the nature of Monte Carlo Simulation is to use random sequence of numbers, which are different from each other to generate some results. To get desired results in terms of accuracy, number of simulations must be increased. Such simulation takes times and resources.

In conclusion, Monte Carlo Simulation is a statistical tool which performs repeated sampling with random numbers to obtain the solution. It can analytically solve higher dimensional problems, and problems with uncertainty involved. From the thermonuclear fission to option pricing to stock price prediction, Monte Carlo simulation can offer an initial estimate of the solution. Any estimate comes with some degree of variance. Variance can be reduced by some variance reduction techniques, by increasing the number of simulation, or by using quasi-random numbers. In this paper, I increased the number of simulation to reduce variance. More appropriately, I reduced the number of simulation to increase the variance in the form of gap of the 95 percent confidence interval.

Monte Carlo simulation only gives us a proper result if the mechanism in place are correct. A Monte Carlo Simulation simulates a complex process. It can

only replace the uncertain factors. It will still need a sound process on the evolution of the input factors with respect to time, for example. A Geometric Brownian Motion, a heat equation, the Solow-Swan model of economic growth are some examples of sound processes that dictate the nature of evolution of the input factors. Only when such specific mechanisms are in place, can Monte Carlo Simulation make use of random numbers to provide the user with a point estimate and a corresponding confidence interval around which proper decisions can be made, as I did with the stock price of Apple Inc. 6 months into the future.

9 References

Baumgärtner, A., Binder, K., Hansen, J. P., Kalos, M. H., Kehr, K., Landau, D. P.,& Weis, J. J. (2013). Applications of the Monte Carlo method in statistical physics (Vol. 36). Springer Science Business Media.

Chen, M. H., Shao, Q. M., & Ibrahim, J. G. (2012). Monte Carlo methods in Bayesian computation. Springer Science Business Media.

Das, S., Spall, J. C., & Ghanem, R. (2010). Efficient Monte Carlo computation of Fisher information matrix using prior information. Computational Statistics Data Analysis, 54(2), 272-289.

Eckhardt, R., Ulam, S., & Von Neumann, J. (1987). the Monte Carlo method. Los Alamos Science, 15, 131.

Estember, R. D., & Maraña, M. J. R. (2016, March). Forecasting of stock prices using Brownian motion–Monte Carlo simulation. In International Conference on Industrial Engineering and Operations Management (pp. 8-10).

Franke, J., Härdle, W.K., & Hafner, C.M. (2019). Statistics of Financial Markets: An Introduction. (5th edition). Springer.

Gentle, J. E. (2009). Monte carlo methods for statistical inference. In Computational Statistics (pp. 417-433). Springer, New York, NY.

Güneş, M. (2013) Chapter 6: Random Number Generation. FU Berlin. Accessed at 11.03.2022 https://www.mi.fu-berlin.de/inf/groups/ag-tech/teaching/2013_SS/L_19540_Modeling_and_Performance_Analysis_with_Simulation/06.pdf

Harris, C.R., Millman, K.J., van der Walt, S.J. et al.(2020). Array programming with NumPy. Nature 585, 357–362. DOI: 10.1038/s41586-020-2649-2. (Publisher link).

Harrison, R. L. (2010, January). Introduction to monte carlo simulation. In AIP conference proceedings (Vol. 1204, No. 1, pp. 17-21). American Institute of Physics.

Hull, J. (2018) Options, Futures and Other Derivatives. 9th Global Edition, Prentice Hall, Upper Saddle River.

Hunter, J.D.(2007) "Matplotlib: A 2D Graphics Environment", Computing in Science Engineering, vol. 9, no. 3, pp. 90-95

Matsumoto, M., & Nishimura, T. (1998). Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. ACM Transactions on Modeling and Computer Simulation (TOMACS), 8(1), 3-30.

Metropolis, N. (1987). The beginning of the Monte Carlo method. Los Alamos Science, 15(584), 125-130.

Riffenburgh, R. H. (2006). Methods You Might Meet, But Not Every Day. Statistics in Medicine, 521-529.

Waskom, M., Botvinnik,O.,O'Kane, D., Hobson,P., Lukauskas, S., Gemperline, D.C., ... Qalieh,A. (2017). mwaskom/seaborn: v0.8.1 (September 2017). Zenodo. https://doi.org/10.5281/zenodo.883859

Van Groenendaal, W. J., Kleijnen, J. P. (1997). On the assessment of economic risk: factorial design versus Monte Carlo methods. Reliability Engineering System Safety, 57(1), 91-102.

Virtanen, P., Gommers, R. Oilphant, T.E., Haberland, M., Reddy, T., Cournapeau, D. ... van Mulbregt, P.(2020) SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. Nature Methods, 17(3), 261-272.

Yu, X. (2016). Efficient Computation of Hessian Matrices in a Monte Carlo Setting using Automatic Differentiation. [Master's Thesis, University of Waterloo]. https://uwaterloo.ca/computational-mathematics/sites/ca.computational-mathematics/files/uploads/files/xixuan_cm_essay.pdf

10 Appendix

10.1 Appendix 1: The Coin Toss Experiment

Let there be two players: A B. A and B will participate in a coin toss game where a fair coin will be tossed 100 times. The outcome of each tosses will be recorded. Any "Heads" will entitle A a balance transfer of 1 from B and "Tails" will endow B with 1 from the bank balance of A. Alternatively, the winnings can be recorded and a single transfer to offset the payables can be carried out after 100 tosses. This is a setup of an ordinary random walk with two nodes. Franke, Härdle, & Hafner (2019, Chapter 4) outlines the ordinary random walk in the following way:

$$X_t = X_0 + \sum_{k=1}^{t} Z_t, \quad t = 1, 2, 3...$$

where $X_0, Z_1, Z_2....$ are independent. $P(Z_k = 1) = p$, and $P(Z_k = -1) = 1 - p$ for all k . There is a source of uncertainty in this problem. The value of at the end of each coin toss will not be known beforehand. Monte Carlo will make use of random number it generates for each trial to suggest the value for Z_t. As the heads and tails are equally likely, any symmetric distribution is a suitable distribution, provided that the experimental design is unbiased towards either outcome. Using the uniformly generated random numbers for example, one can fairly allocate the random numbers if one allocates even numbers for head and odd numbers for tail when choosing for integers between 0 and 100. One can achieve the same results by allocating the numbers greater or equal to 0.50 as heads and the remaining half as tails when choosing the numbers in range [0,1). Once the heads and tails are determined by the use of uniformly distributed random numbers, can be computed. The coin toss experiment was settled without tossing a single coin. Such is the use of Monte Carlo Simulation.

10.2 Appendix 2: Middle Square Digits Algorithm

John von Neumann, in his first Monte Carlo code ever written, proposed the middle square digits algorithm for random number generation. In this algorithm, an arbitrary number of N digits is chosen. A 4 digit number 5555 for illustration purposes will show the workings. The initial number from which a random number generator starts to generate subsequent random numbers is called the seed of the algorithm. Starting with the same seed will help replicate the sequence of pseudo random numbers for future use. The square of 5555 is 30858025. This is an eight digit number. The middle 4 digits, 8580 is the new random number and the process is repeated for the generation of subsequent random numbers.

10.3 Appendix 3: Some external links

US Interest rates in https://www.federalreserve.gov/releases/h15/
Github for my codes for this paper:
Monte Carlo Simulation