FINAL REPORT

CSC 521

CRYPTO CURRENCY

MEMBERS:

Tyler Boudart, Byeongho Choi, Augustine Chiu, Gerardo Palacios

Contents

Non-Technical Summary

This paper aims to analyze and generate a risk assessment for cryptocurrencies when holding the asset or valuing its derivatives. This is accomplished by using a Monte Carlo algorithm which leverages the law of large numbers and the central limit theorem as its core reasoning for interpreting its results. A Monte Carlo simulation models a system by identifying random variables within it and using their respective distributions in a series of simulations. The simulation draws a random value for each random variable based on the variable's underlying distribution. The random numbers are used within the model to predict the final value for the process. Thus, the simulation is repeated many times using new random numbers for each iteration of the simulation but keeping the same underlying model. After conducting many simulations, the results' descriptive statistics can be used to get an estimated sense of the true value.

We simulated a 365-day Bitcoin price trajectory from its last known observation and valued its profit and loss from holding or buying an Asian style derivative. The average loss across all the simulations was **-$1,247.63**. Its 95% value at risk was **$-17,794.46.** In other words, in 95% of generated simulations the investment will have an estimated floor with a loss of **$-17,794.46**. Lastly, it had a 95% confidence interval -$1,304.82 to - $1,190.44, meaning we are 95% confident the true loss on the investment will be in that range. We also simulated the amount of profit an investor would earn with an Asian Option. The results are based on the quarterly geometric average with an arbitrary contract price of $5.00. The average profit for the option is **$2,415.58.** The 95% value at risk was $-5.00 meaning 95% of the time we will not lose less than $5.00. The 95% confidence interval was between $2,392.33 - $2,438.82. We are 95% confidence the true profit from the option will be in that range.

The results indicate that Bitcoin is a highly volatile and very risky asset. In either case, the probability of making any sort of return was around 50%, which is pure chance. In most instances an investor would tend to lose all or a portion of their investment in Bitcoin. With that said, it appears that by purchasing its derivatives Bitcoin may yield less loss, as the only loss incurred is the price of the initial contract. However, the probability of making a return is still roughly 50%.

Introduction

What is currency? Currency is a medium of exchange for goods and services. The notion of currency has changed throughout history. It was food, gold, and paper bills, but a new form of currency has entered the vocabulary: cryptocurrency. Cryptocurrency has several important features. It is secured by cryptography which ensures the virtual currency cannot be counterfeit or double-spent. Cryptocurrency is a decentralized structure that exists outside the control of governments and central authorities. It can be mined or purchased from cryptocurrency exchanges. The core of cryptocurrency is blockchain technology. Blockchain technology is essentially a set of connected blocks or an online ledger. Each block has a set of transactions independently verified by each member of the network. Every block generated must be verified by each node before being confirmed. These days, cryptocurrency is very popular. Cryptocurrency exchanges reported more than $15.8 trillion in trading volumes in the year 2021 which is a 567% increase compared to 2020.

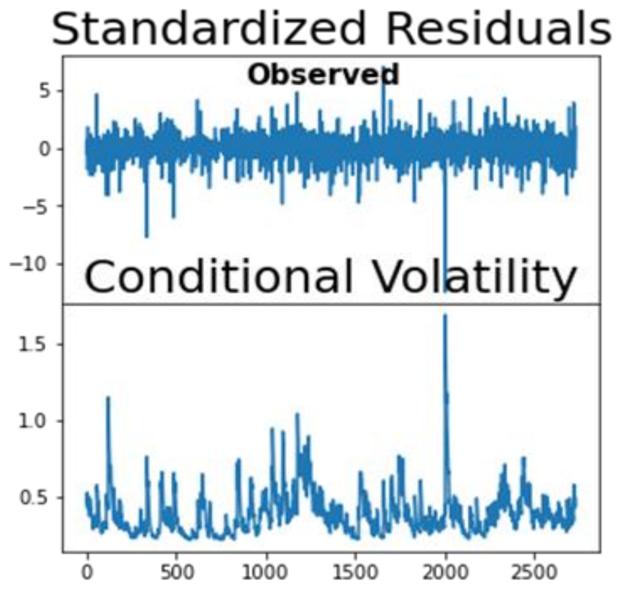
The largest cryptocurrency is Bitcoin by market capitalization. The closing price of Bitcoin is $39,640 as of this writing, which is way higher than any other cryptocurrency. It is not rare to hear that the Bitcoin price has dropped 30% or increased 20% within a short period. Because of this extreme volatility, it is important to be careful when investing. For these reasons, this paper investigates Bitcoin using a Monte Carlo algorithm. The data we used for this project was daily Bitcoin adjusted closing prices from September 2014 to February 2022. The data was pulled from the Yahoo Finance API. There were 2,695 days in the data set.

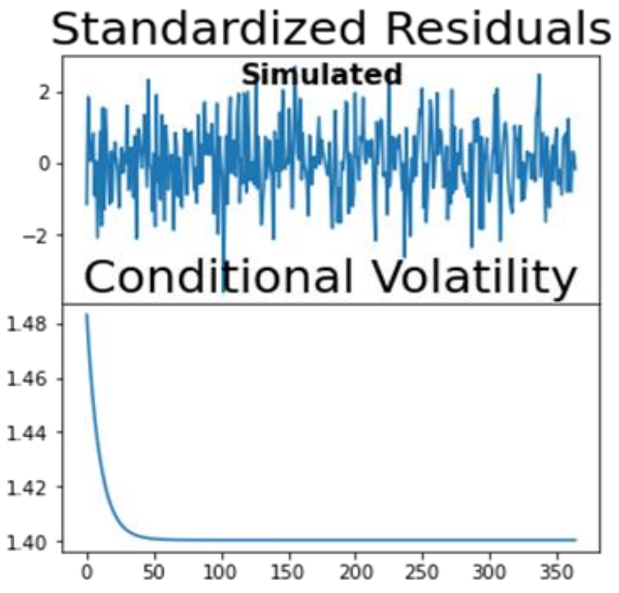
Methodology

In order to simulate and assess cryptocurrencies, the algorithm modeled the price trajectory of Bitcoin over the course of 365 days. In order to create a realistic price trajectory, we incorporated a time series analysis approach in that we model a stationary (constant mean and variance) series that follows a near normal distribution. To achieve this, Bitcoin's price was transformed into its log returns. At its core, the methodology used in this simulation regarded fitting an ARMA-GARCH model to the log-returns of Bitcoin. However, we had two methodologies attempted in this paper. Both methodologies are similar except for how many times a model is fitted. Our first methodology was as follows:

1. Transform series into its log returns
2. Build ARMA-GARCH model from the log returns
3. Forecast 1-day out volatility
4. Generate a new price
5. Add the new price to the end of the series
6. Repeat steps 1-5

We repeated this process 365 times. This was a very computationally expensive process in which our initial simulations took nearly 11 hours to complete. We tried to find ways of optimizing. The first option we considered was to use batching techniques. However, we did not think it was possible to batch fitted ARIMA models. The next option we considered was to use the GPU which would run our code much faster. This was quickly deemed to be an unrealistic option as we would have had to manually recreate the packages and functions, we used in NumPy as Numba does not allow the use of third-party packages. The third and final option we considered was to change the number of days we would forecast at each iteration. If we were to use something like ten days as the forecast number, we would significantly cut down on the time it took for the code to run as well as preserve volatility. As an added check of due diligence, we compared two GARCH models, one of the observed series and the other from a randomly selected simulated price trajectory, hypothesizing that if the series was realistic the conditional volatility of both models would be identical. Unfortunately, this was not the case. As shown below, the volatilities between the observed and simulated are drastically different. This was indicative that the simulated series was not truly behaving as the observed series.





Our second methodology improved on the first by both efficiency and relevance. The biggest change was building the ARMA-GARCH model **only once using the observed log returns and leveraging Numba**. In this second iteration, the fitted model's coefficients were used to forecast the volatility and generate a return.

1. Transform **observed series** into its log returns
2. Build ARMA-GARCH model from the log returns

3. Use GARCH formula to generate volatility, with Numba
4. Generate random return
5. Create a new price
6. Repeat steps **3 - 5**

The forecasted volatility was generated based on the $Return_{t-1}$ and $Volatility_{t-1}$. following the garch (1, 1) formula:

$$Volatility_t = \sqrt{\omega + \alpha_1 Return^2{}_{t-1} + \beta_1 Volatility^2{}_{t-1}}$$

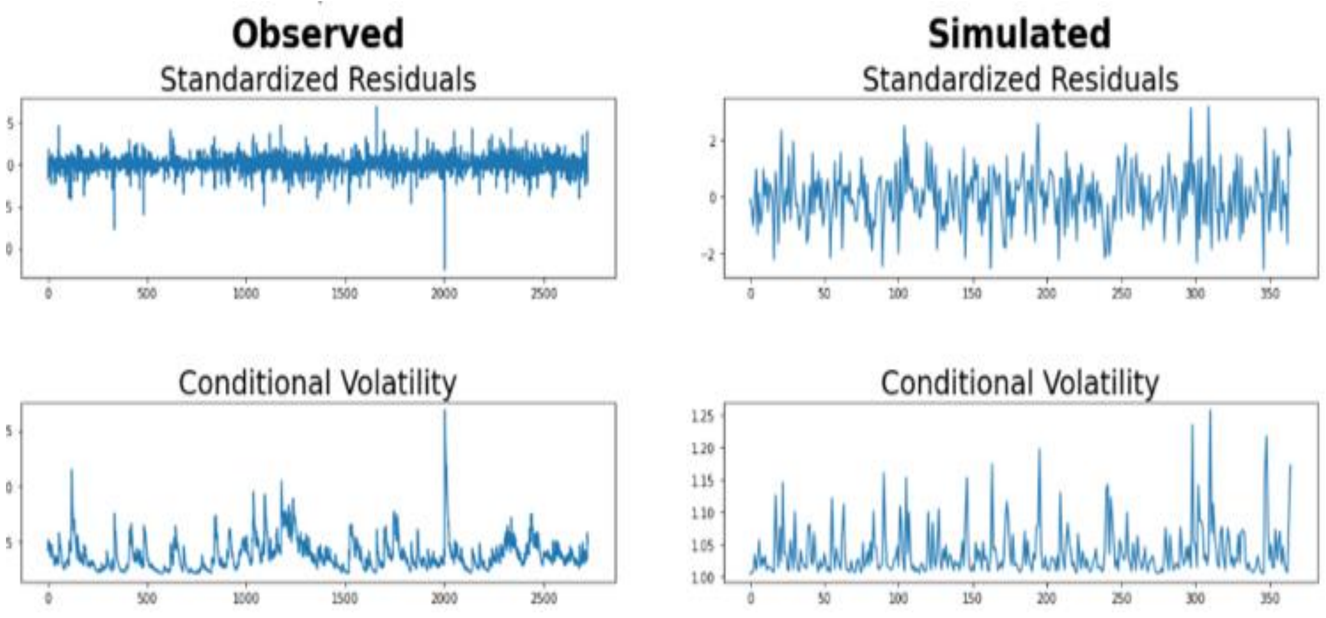
The random return was generated using a risk-free rate and accounting for any potential drag effects. The return was based off a normal distribution with a mean and sigma described as:

$$mean = (risk\ free\ rate - 0.5 * Volatility^2{}_t)^{\frac{1}{365}}$$

$$sigma = Volatility * \sqrt{\frac{1}{365}}$$

$$random\ return = Normal(mean, sigma)$$

This process was repeated to generate a 365-day price trajectory from its last observed price. This process was much faster, as one hundred simulations took less than a minute to run and the conditional volatilities of the observed vs simulated were more in line with each other.

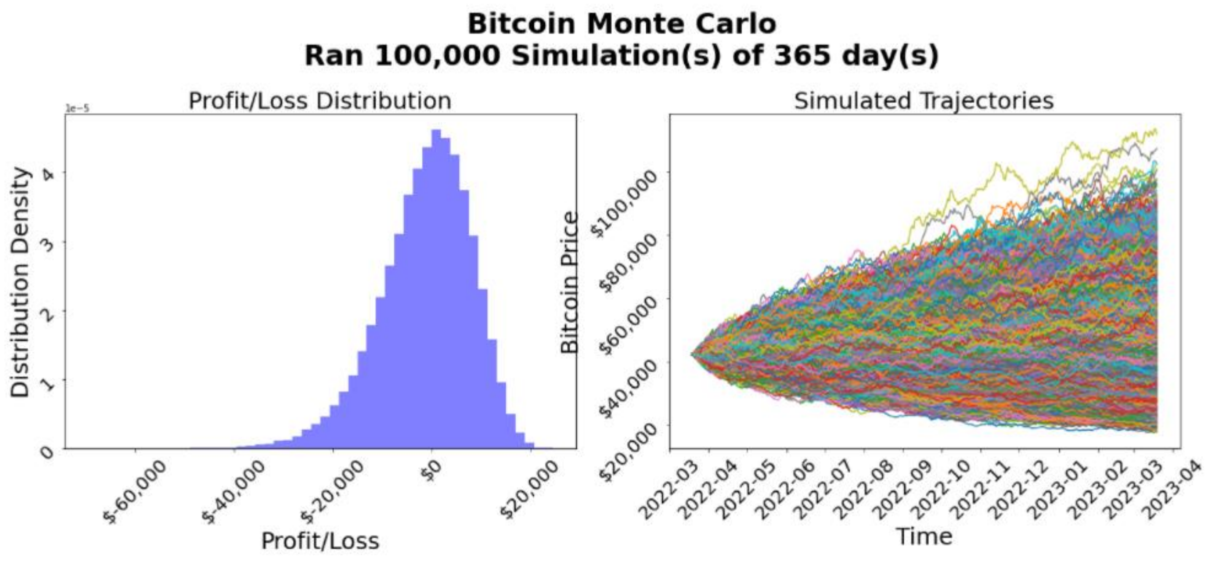


Results

Returns

The following show the results after modeling the returns of Bitcoin. The returns were calculated by taking the last simulated price less the last observed price. The difference would represent the profit or loss of the asset if held for the duration of 365 days from the last known price. As shown below, the distribution of returns is slightly left-skewed with an average at -$1,247.63. This indicates that the tendency of Bitcoin would lose its value over time given its volatility. There is a great propensity to lose the investment. At 100,000 simulations, the probability to make any profit is at 48%, which pure random chance.

```
100,000 Simulation(s) Completed in 0 hour(s), 2 minute(s), and 45.90 second(s)
-----------------------------------------------------------------------------
Reviewing 7 year(s), 6 month(s), and 5 day(s) of historical data
Average Profit/Loss: $-1,247.63
Profit/Loss Ranges from $-69,557.62 - $24,498.16
95% Confidence Interval Range: from $-1,304.82 - $-1,190.44
Probability of Earning a Return = 48.78%
The VaR at 95% Confidence is: $-17,794.46
-----------------------------------------------------------------------------
```

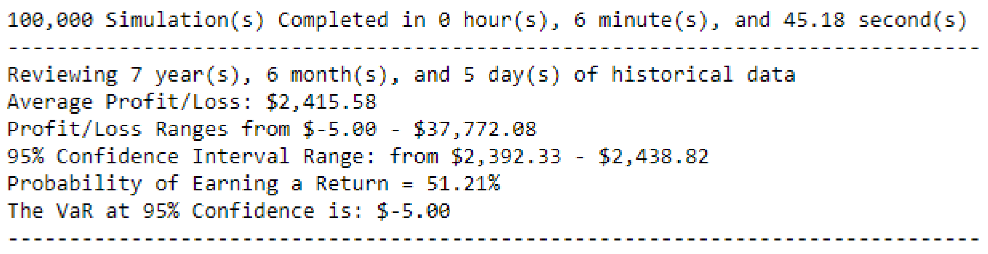


Asian Options

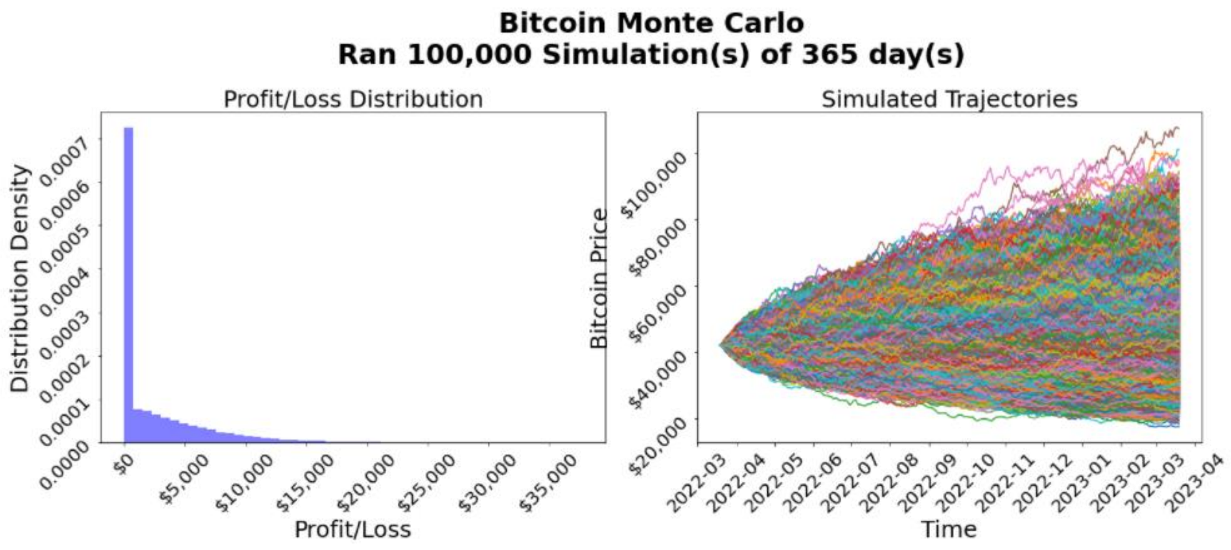
The Asian option is a financial product or contract that gives an investor the opportunity, but does not mandate, to buy or sell an asset. With Asian options, a call option gives the buyer the choice to buy a stock at its average price over a defined period using a defined method for averaging. A put option gives the buyer of the option the choice to sell a stock at its average price. A call option value is equal to the current price of Bitcoin minus the average price because the investor could buy Bitcoin at its average price then instantly sell it at the spot price. The option will expire worthless if the spot price is less than the average price because the investor could buy the stock cheaper at the spot and would buy through the normal market instead of exercising the option and buying at the higher average price. The put option works similarly to the call option but has opposite equations. The value of a put option is the average price minus the spot price because the investor could buy Bitcoin on the market at the spot price and instantly sell it by exercising their option to sell at the higher average price.

Since we dealt with European options in the class lecture, we decided to investigate the Asian option price of bitcoin. Asian option uses the average price offer specified time horizon as the strike price. For example, the average price of the 1-year option where value is based on the quarterly period can be calculated as (PQ1 + PQ2 + PQ3 + PQ4)/4. Then, the investor’s profit at the end of a call option would be current price - average price - contract cost. In addition to specifying a time horizon, an average method needs to be specified as well, such as arithmetic or geometric mean.

To run the option simulation, the user needs to specify the option type in terms of European vs Asian and call vs put. Additionally, the user enters a contract price and strike price for European options. Based on the input information, we could estimate the average profit of the option by running 100,000 simulations along with a confidence interval.



```
100,000 Simulation(s) Completed in 0 hour(s), 6 minute(s), and 45.18 second(s)
--------------------------------------------------------------------------------
Reviewing 7 year(s), 6 month(s), and 5 day(s) of historical data
Average Profit/Loss: $2,415.58
Profit/Loss Ranges from $-5.00 - $37,772.08
95% Confidence Interval Range: from $2,392.33 - $2,438.82
Probability of Earning a Return = 51.21%
The VaR at 95% Confidence is: $-5.00
--------------------------------------------------------------------------------
```



Conclusion

The results indicate that Bitcoin is a highly volatile and very risky asset. The simulations on both holding and derivatives have a probability of 50%, which is pure chance. In most cases an investor would lose a portion of their investment in Bitcoin. On average over 100,000 simulations, the profit/loss of Bitcoin at the end of a 365-day period is -$1,247.63 with a 95% confidence interval range: from $-1,304.82 - $-1,190.44. On the other hand, the average profit/loss for the Bitcoin Option was $2,415.58 with 95% confidence interval range: from $2,392.33 - $2,438.82. With that said, it appears that by purchasing Bitcoin Options may yield less overall loss albeit with the same amount of risk, as the only loss incurred is the price of the initial contract. However, the probability of making a return is still roughly 50%.

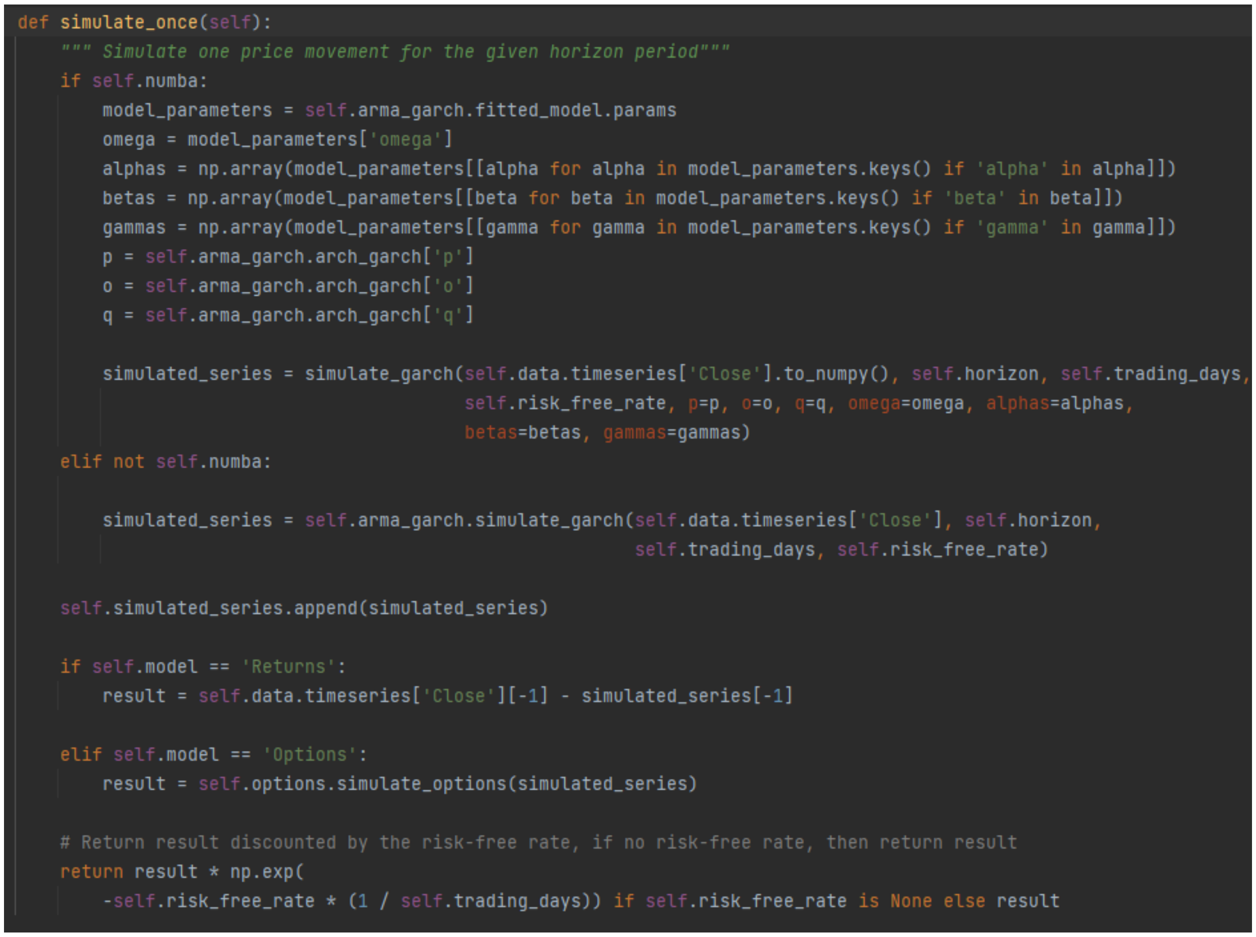
For future work, there could be further due diligence on the efficacy of the simulations. One path would be to do a holdout method and compare the results of the hold out series with the simulated series. If the two would share similar conditional volatilities and means, it would add evidence that our methodology of simulating the price is appropriate.

Appendix

The full code and notebooks can be found on the GitHub repository here, however core functions are shown below.

Simulate_Once

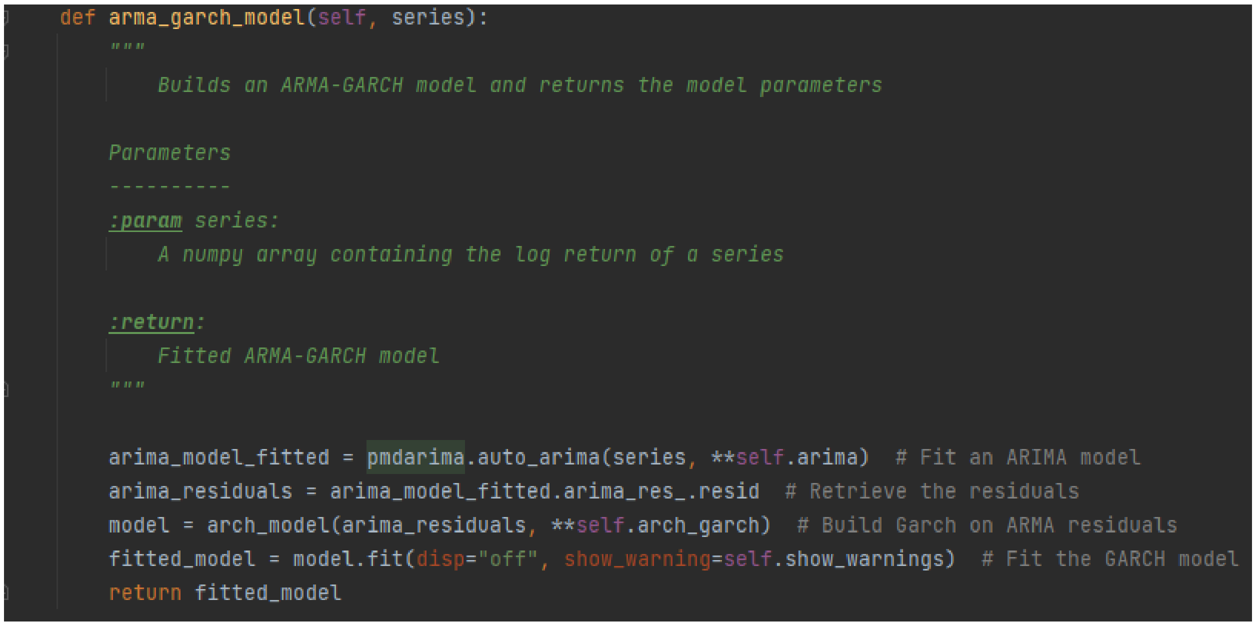
The following figure shows the simulate_once() function. This was a key function because it took the timeseries and generated a simulated trajectory which was then used to value its profit/loss or its derivative value.



```python
def simulate_once(self):
    """ Simulate one price movement for the given horizon period"""
    if self.numba:
        model_parameters = self.arma_garch.fitted_model.params
        omega = model_parameters['omega']
        alphas = np.array(model_parameters[[alpha for alpha in model_parameters.keys() if 'alpha' in alpha]])
        betas = np.array(model_parameters[[beta for beta in model_parameters.keys() if 'beta' in beta]])
        gammas = np.array(model_parameters[[gamma for gamma in model_parameters.keys() if 'gamma' in gamma]])
        p = self.arma_garch.arch_garch['p']
        o = self.arma_garch.arch_garch['o']
        q = self.arma_garch.arch_garch['q']

        simulated_series = simulate_garch(self.data.timeseries['Close'].to_numpy(), self.horizon, self.trading_days,
                                          self.risk_free_rate, p=p, o=o, q=q, omega=omega, alphas=alphas,
                                          betas=betas, gammas=gammas)
    elif not self.numba:

        simulated_series = self.arma_garch.simulate_garch(self.data.timeseries['Close'], self.horizon,
                                                          self.trading_days, self.risk_free_rate)

    self.simulated_series.append(simulated_series)

    if self.model == 'Returns':
        result = self.data.timeseries['Close'][-1] - simulated_series[-1]

    elif self.model == 'Options':
        result = self.options.simulate_options(simulated_series)

    # Return result discounted by the risk-free rate, if no risk-free rate, then return result
    return result * np.exp(
        -self.risk_free_rate * (1 / self.trading_days)) if self.risk_free_rate is None else result
```

Arma_Garch_Model

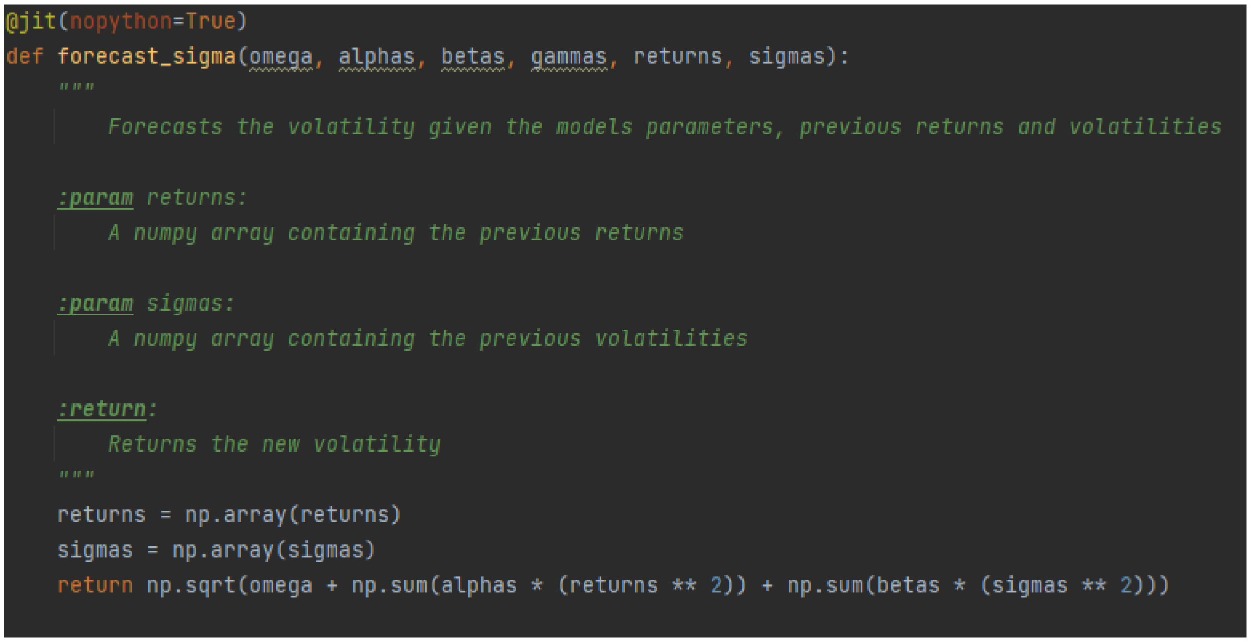
The next key function was that of the arma_garch_model. This was important as it was the model that was fitted and coefficients used to predict the next day's volatility.



```python
    def arma_garch_model(self, series):
        """
            Builds an ARMA-GARCH model and returns the model parameters

        Parameters
        ----------
        :param series:
            A numpy array containing the log return of a series

        :return:
            Fitted ARMA-GARCH model
        """

        arima_model_fitted = pmdarima.auto_arima(series, **self.arima)  # Fit an ARIMA model
        arima_residuals = arima_model_fitted.arima_res_.resid  # Retrieve the residuals
        model = arch_model(arima_residuals, **self.arch_garch)  # Build Garch on ARMA residuals
        fitted_model = model.fit(disp="off", show_warning=self.show_warnings)  # Fit the GARCH model
        return fitted_model
```

Forecast_Sigma

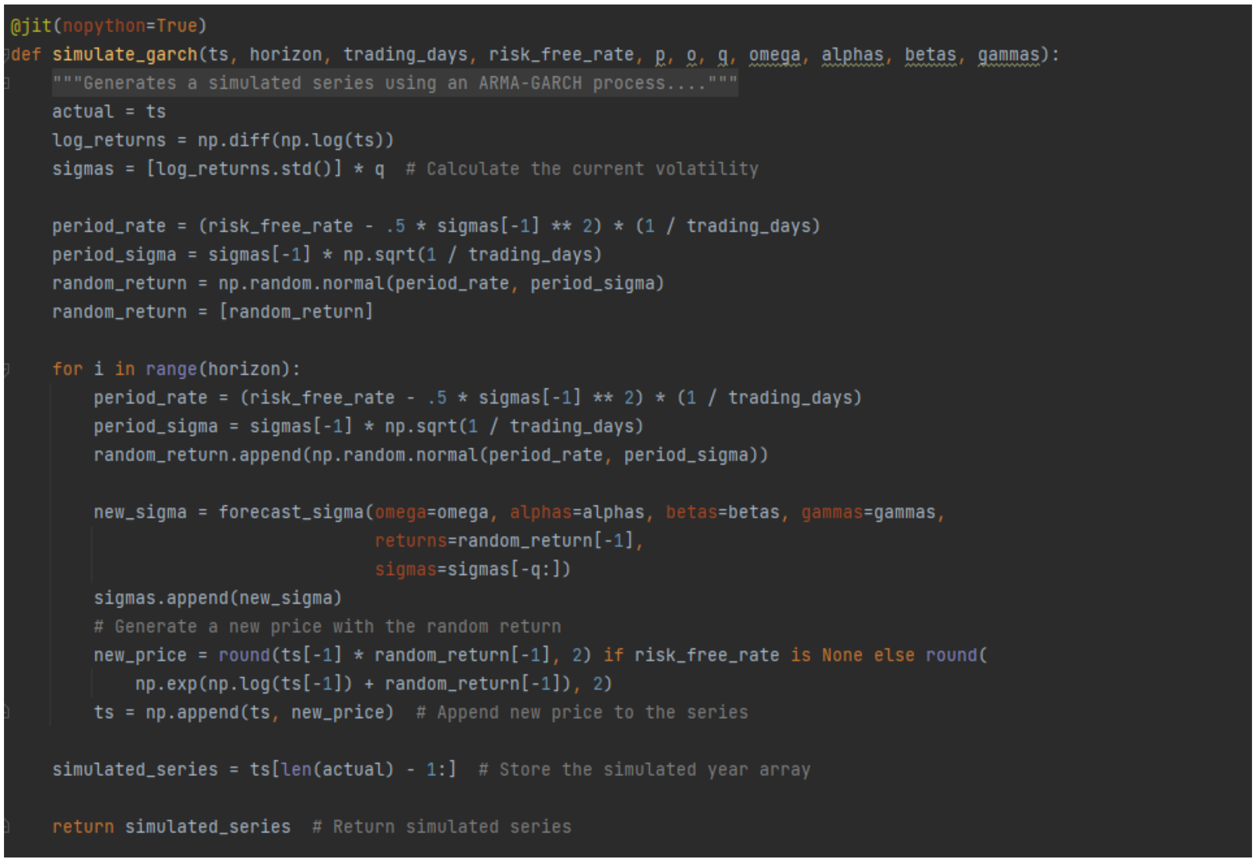
Next is the forecast_sigma() function. This used the GARCH(1,1) model formula to estimate the new volatility. Notice that this is also using the @jit decorator which leverages the use of Numba to pre-compile the code and improve performance.



```python
@jit(nopython=True)
def forecast_sigma(omega, alphas, betas, gammas, returns, sigmas):
    """
        Forecasts the volatility given the models parameters, previous returns and volatilities

    :param returns:
        A numpy array containing the previous returns

    :param sigmas:
        A numpy array containing the previous volatilities

    :return:
        Returns the new volatility
    """
    returns = np.array(returns)
    sigmas = np.array(sigmas)
    return np.sqrt(omega + np.sum(alphas * (returns ** 2)) + np.sum(betas * (sigmas ** 2)))
```

Simulate_Garch

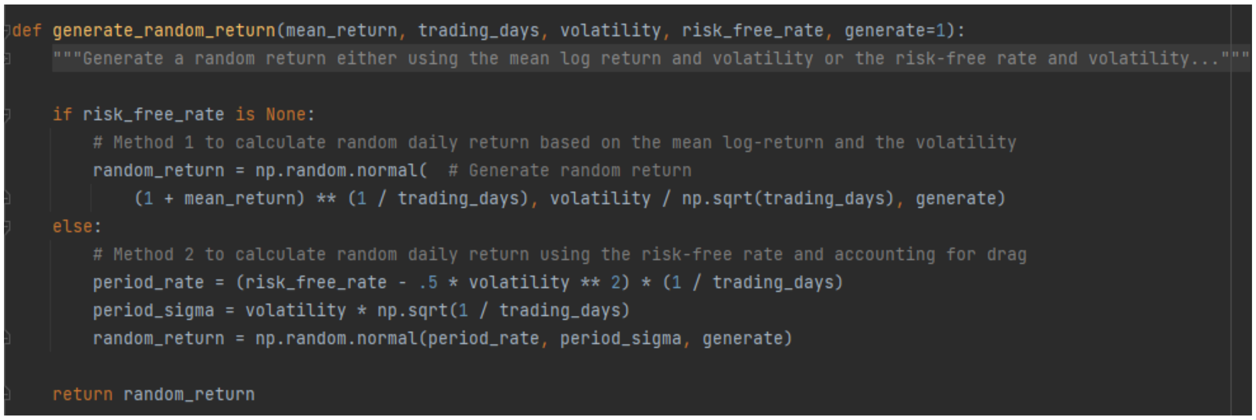
Next, the simulate_garch() function. This function was key for generating the complete 365-day trajectory for Bitcoin price. At each iteration, a new volatility and random return were generated which were used to create a new price. In addition, this also uses the @jit decorator to pre-compile the code and improve performance.



```python
@jit(nopython=True)
def simulate_garch(ts, horizon, trading_days, risk_free_rate, p, o, q, omega, alphas, betas, gammas):
    """Generates a simulated series using an ARMA-GARCH process...."""
    actual = ts
    log_returns = np.diff(np.log(ts))
    sigmas = [log_returns.std()] * q  # Calculate the current volatility

    period_rate = (risk_free_rate - .5 * sigmas[-1] ** 2) * (1 / trading_days)
    period_sigma = sigmas[-1] * np.sqrt(1 / trading_days)
    random_return = np.random.normal(period_rate, period_sigma)
    random_return = [random_return]

    for i in range(horizon):
        period_rate = (risk_free_rate - .5 * sigmas[-1] ** 2) * (1 / trading_days)
        period_sigma = sigmas[-1] * np.sqrt(1 / trading_days)
        random_return.append(np.random.normal(period_rate, period_sigma))

        new_sigma = forecast_sigma(omega=omega, alphas=alphas, betas=betas, gammas=gammas,
                                   returns=random_return[-1],
                                   sigmas=sigmas[-q:])
        sigmas.append(new_sigma)
        # Generate a new price with the random return
        new_price = round(ts[-1] * random_return[-1], 2) if risk_free_rate is None else round(
            np.exp(np.log(ts[-1]) + random_return[-1]), 2)
        ts = np.append(ts, new_price)  # Append new price to the series

    simulated_series = ts[len(actual) - 1:]  # Store the simulated year array

    return simulated_series  # Return simulated series
```

Generate_Ramdom_Return

This was the methodology of generating a random return given the risk-free-rate. There was another method used too but we chose for this simulation to use the risk-free-rate. This is because it is not only discounted appropriately but also accounts for any potential drag effects.



```python
def generate_random_return(mean_return, trading_days, volatility, risk_free_rate, generate=1):
    """Generate a random return either using the mean log return and volatility or the risk-free rate and volatility..."""

    if risk_free_rate is None:
        # Method 1 to calculate random daily return based on the mean log-return and the volatility
        random_return = np.random.normal(  # Generate random return
            (1 + mean_return) ** (1 / trading_days), volatility / np.sqrt(trading_days), generate)
    else:
        # Method 2 to calculate random daily return using the risk-free rate and accounting for drag
        period_rate = (risk_free_rate - .5 * volatility ** 2) * (1 / trading_days)
        period_sigma = volatility * np.sqrt(1 / trading_days)
        random_return = np.random.normal(period_rate, period_sigma, generate)

    return random_return
```

Individual Reports

Augustine Chiu Individual Report

For our project, we wanted to use Monte Carlo simulation to look at cryptocurrency returns. More specifically, we planned to use Bitcoin log returns. Prior to this project, I had some prior knowledge of Bitcoin and cryptocurrencies, but not enough to confidently talk about them. I did some basic research and also read through Gerardo's time series course project which explored Bitcoin.

For this project, I first worked on building a simple Monte Carlo simulation in Python that would take the log return series and build a garch model and forecast out 365 days. The first thing I had to do was lookup how to do time series in Python as my previous experience was entirely in R. I ended up using arch_model to begin messing around with the data and trying to get something.

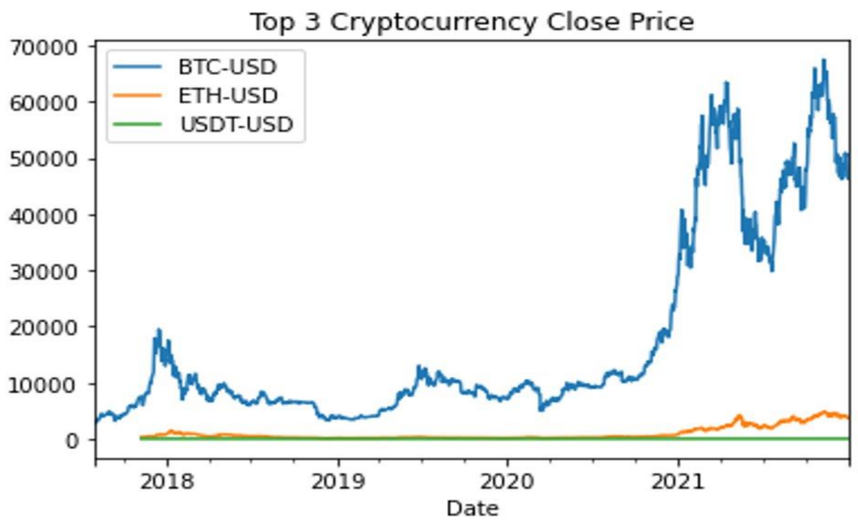
I then compared what I had done with Gerardo, and we saw that our code was very similar with one key difference. I was only building the model just once and forecasting out 365 days with that model, while Gerardo was forecasting out only 1 day and then building another model for every new series. After discussing with the group and Gerardo then going to office hours, it was decided that we would be going forward with Gerardo's version.

The next issue we faced was that generating a new model for every single day was extremely time consuming. I then had to think about ways that we could optimize the process. We looked into a few options including trying to use the GPU but realized that they weren't really feasible. I eventually suggested that instead of building the model every day or just building the model once and forecasting out 365 days, we could forecast out in smaller intervals. We could generate a model and then forecast out something like 10 days, and then generate a new model on that new series. This did cut down on the time significantly and we felt that it would still capture all the volatility.

A major takeaway that I got from the project was working with times series in Python. I had only worked with times series using R from the prior course, so this was a new experience and I walked away preferring R much more. I also just gained some more experience with Monte Carlo simulations. The homeworks mostly included creating simulations based on some given distributions or probabilities, but this project used real data.

Byeongho Choi Individual Report

In this project, we decided to investigate Cryptocurrency by using the Monte Carlo method. I've always had an interest in Cryptocurrency, but I do not have much knowledge of it. Therefore, I research Cryptocurrency, and could make an introduction to the report. I started to think about the fundamentals of currency. It was very fun to think about the meaning of the currency, and how it affects the world. It led me to think about the reliability of currencies. We decided to investigate bitcoin for several reasons. Bitcoin is the largest cryptocurrency by market capitalization and extremely volatile.



For this project, I decided to investigate the option price of bitcoin since I did not have knowledge of the Garch and Arch model. It was a very good experience to collaborate the parts with the parts you don't know. I guess it is a valuable experience because you might have to collaborate with people with different expertise. We decided to investigate the asian option price. It was interesting to calculate the asian option price since asian option uses the average price offer specified time horizon as strike price. Using the average price offer specified time horizon as strike price makes asian options reduce the volatility. Reducing volatility makes option prices lower than European options. I recalled we simulated the option price with different volatility in the class. As we saw, the volatility of stock decided not only the chance of the profit of stocks but also option price.

Through this class and project, I could simulate many things and contemplate the Monte Carlo Method. I could learn what Monte Carlo is, how the Monte Carlo simulation consists of, how to implement the simulation on my own theory and work with people with different expertise.

Tyler Boudart Individual Report

I contributed to the group’s success in several ways and learned several concepts throughout the project. My contributions include participating in group discussions, assisting with completing group Milestones, researching Asian options, helping code the options class, debugging the code, creating slides for the presentation, and writing parts of the final report. Additionally, I scheduled our group meeting with Dr. McDonald for Milestone 3 on behalf of the group. Some of the key takeaways I had included how to structure classes, use GARCH in Monte Carlo simulation, and the power of the numba library.

One of my larger contributions included my work on the Asian options aspect of the project. I researched Asian options to learn how to model them with Monte Carlo. The Investopedia article, Asian Option, served as the basis for my understanding of Asian options. I wrote a draft of the code for Asian options based on what I learned from the article. I learned from this project Asian options are like European options except using the average price over a specific period instead of a set strike price. I helped debug issues with the final simulation. I made sure the code was written as intended by asking my team members questions about their work. Another contribution I made was to the presentation. I created a variety of slides on time series concepts and how they related to our simulation. Further, I helped explain some time series techniques to some in our group and the presentation. Lastly, I contributed to the final report by writing the non-technical summary and adding information, editing, or providing feedback to other sections.

There are several key takeaways I had after completing the project. I learned how important it was to structure classes in a well-thought-out manner. The group discussed how we should structure the class and the potential implications of different ideas throughout the planning process. We were able to achieve a good structure simulator because of our valuable conversations that I think are important to have in other projects I do in the future. The group researched and learned how to incorporate GARCH into a Monte Carlo simulator. I found the integration interesting and hope to use it at work someday. Lastly, one of the major takeaways I learned from this project is the power of the numba library. I helped speed up our simulation tremendously. I implemented numba for the options portion of our code for the last homework assignment. I was happy to learn how to make numba work. I think numba will be a useful tool in my toolbelt in the future and at work.

Gerardo Palacios Individual Report

I mainly contributed to the group but primarily coding and organizing the algorithm logic. This included finding the necessary python packages, reading the documentation of the packages in order to implement them into the simulation. This included packages such as the arch and pmdarima package which were used to fit an ARMA-GARCH model. I was able to modify the examples from class and extend it by writing additional functions and classes that encapsulate the process we were trying to simulate. I also contributed by participating in weekly group discussions, following up on code changes and suggestions, setting up a GitHub repository in order to keep track of all the code changes. I was the member with the highest number of commits to the code. In addition, I also scheduled one-on-one meetings with Dr. McDonald to touch base on the efficacy of the methodologies in use as well receiving guidance towards the end results.

One of my biggest, most recent contributions was through the implementation of the Numba library. Although in my initial logic, I was using the third-party packages to continually rebuild the model after adding the new simulated price to the series. With this method, using Numba was not possible because Numba does not allow any third-party packages except for NumPy. As explain in the paper, I then implemented a second methodology where I only build the ARMA-GARCH model once and use the fitted model's coefficients to generate a new volatility. In this way, I no longer needed to rely on the third-party packages to forecast a new price. With that in mind, I was able to implement Numba into the simulation which drastically improved the computation time on the simulations. To put into perspective, before implementing Numba, 100 simulations would roughly take 31 seconds, after implementing Numba, 100,000 simulations would take 2 minutes and 45 seconds. This allowed us to take our simulations further by achieving a higher number of results which would, in theory, be closer to the true value of the series.

There were many key takeaways from this project. First was the learning what and how to implement a Monte Carlo simulation. I was not only pleasantly surprised how flexible and applicable this technique can be across many industries, but I also found it very fun to experiment and build a simulated process. Also, I am very happy to have been able to apply what I had learned the Timeseries course in this one. I thought it was a natural progression for the use of the techniques learn in class. Second, was how to leverage Numba to achieve a drastically exponential improvement in computation time. This was an exciting and immediate improvement in our simulations. I can definitely see myself applying this library at work and in other projects.