Pricing Asian Options on Commodities with GARCH Model

Lue Shen¹, Leheng Chen¹, Chu Song¹ and Jing Qian¹

¹Department of Mathematics, the Hong Kong University of Science and Technology, Hong Kong SAR, China

May 14th, 2019

This report studies log return time series for 16 commodities, including energy, agriculture and metals. Basic time series analysis, i.e. statistics and correlation analysis, are conducted for all commodity series. Binomial tree method, Monte Carlo simulations with log-normal assumptions and Monte Carlo simulations with GJR-GARCH model are introduced to price Asian options on crude oil, ethanol, natural gas, gold and silver. Multiple conclusions are derived based on the pricing results. Future work is also discussed at the end of this report.

1 Introduction

Asian option, also called average value option, is essentially an exotic option, whose payoff depends on the path of the underlying price. As presented in equation 1, the payoff is determined by the strike price and the average underlying price (either arithmetic or geometric) during the option’s life at the maturity. This averaging feature makes it popular among derivative traders, because the payoff will not be affected dramatically from unpredictable oscillations caused by the underlying’s market performance close to maturity, and these kinds of options also provide a nice hedging choice to companies exposed to average prices, say oil production firms and farm producers. Most common underlying for Asian option are commodities and foreign exchange rate. In this report, Asian options on commodities with high liquidity will be utilized to construct numerical examples and examine proposed pricing methods.

Since their invention in late seventies, several pricing methods have been proposed, and generally speaking, they can be classified into three categories: semi-analytical, approximation and Monte Carlo. The semi-analytical pricing approach assumes the underlying price following a geometric Brownian motion (GBM). Taking logarithms, the average value of the GBM can be expressed in terms of the sum of normal random variables. Using fast Fourier transform (FFT) and convolutions, the final payoff can be calculated in an analytical way. (Carverhill and Clewlow, 1990) Benhamou applied the same method for discrete Asian options on underlying with non-normal returns in 2000. (Benhamou, 2000) Approximation methods approximate the real distribution of the average underlying price at the maturity with tractable ones, such as Edgeworth series expansion (Turnbull and Wakeman, 1991) and lognormal distributions (Levy, 1992). The Monte Carlo approach is a combination of stochastic models and Monte Carlo simulations. Kemna and Vorst proposed a Monte Carlo simulation scheme for the arithmetic average, as well as a variance reduction technique in their paper (Kemna and Vorst, 1990).

This report introduces an innovative Monte Carlo scheme to price a discretely monitored Asian options (average value options) on commodities by assuming that log returns of the underlying price follow GJR-GARCH model. Other non-GARCH models, i.e. binomial tree model and Monte Carlo simulations with constant volatility, are also implemented as a comparison to the proposed pricing method.

\[
C_T = (A_T(t, T) - K)^+ \\
\frac{P_T = (K - A_T(t, T))^+}{(1)}
\]
2 Theory

2.1 Commodities

A commodity, such as crude oil or wheat, is a basic good used in commerce that is interchangeable with other commodities of the same type. Commodities can be classified into a few categories according to their usages and values, say raw materials, basic resources, agricultural, and mining products, etc. The price of a commodity good is typically determined as a function of its market as a whole: established physical commodities possess actively traded spot and derivative markets.

Commodity futures are agreements to buy or sell a raw material at a specific date in the future at a particular price. The intrinsic value of one commodity future, at any time during its life, can be expressed by equation 2. Since its intrinsic value is a linear function of its underlying price and the value of the contract is zero, we assume that the future price is also a linear function of the underlying price with a coefficient close to 1 and it contains all the volatility and underlying price information for us to price their options. Due to data source limitations, prices of commodities are not available to us, hence we study their future prices as an alternative data source for underlying prices, regardless of its minor delays.

\[ F_t = S_t - f_t \]  
(2)

In this project, 16 commodities from 3 broad categories, namely energy, agriculture and metals, are selected for analysis. For each of them, price data of the most liquid/major future contract is collected for analysis. Details as follow:

**Energy:**
- Crude Oil: WTI Financial Futures
- Natural Gas: Henry Hub Natural Gas Futures
- Refined Products: RBOB Gasoline Futures
- Biofuels: Chicago Ethanol (Platts) Futures
- Coal: Coal (API2) CIF ARA (ARGUS-McCloskey) Futures

**Agriculture:**
- Corns: Corns Futures
- Wheats: Chicago SRW Wheat Futures
- Soybean: Soybean Futures
- Soybean Meal: Soybean Meal Futures
- Soybean Oil: Soybean Oil Futures
- Livestock: Live Cattle Futures
- Livestock: Lean Hog Futures

**Metals:**
- Gold: Gold Futures
- Silver: Silver Futures
- Platinum: Platinum Futures
- Copper: Copper Futures

2.2 Asian Options

Asian option, also named average value option, is one of the most popular path dependent options among derivative traders. Its payoff is determined by the average underlying price over some pre-set period of time. The average price of the underlying asset can either determine the underlying settlement price or the option strike price. This is different from the case of usual European options and American options, whose payoff depends on underlying price at the exercise date. Hence Asian option is a typical form of exotic options. These exotic features provide risk reduction of market manipulation of the underlying instrument close to maturity (Kemna and Vorst, 1990). On top of that, the cost of Asian options is usually much lower than European or American options with same strike and maturity.

Suppose \( S_t \) is the underlying price at time \( t \), \( A_T(0, T) \) is the arithmetic average underlying price at maturity \( T \), then we have

\[ A_T(0, T) = \frac{1}{T} \sum_{t=0}^{T} S_t \]

in discrete form, or in continuous form like this

\[ A_T(0, T) = \frac{1}{T} \int_{0}^{T} S_t dt \]

For geometric average underlying price, say \( \tilde{A}_T(0, T) \), can be expressed as

\[ \tilde{A}_T(0, T) = \left( \prod_{t=0}^{T} S_t \right)^{\frac{1}{T}} \]

in discrete form, or in continuous form like this

\[ \tilde{A}_T(0, T) = \exp\left( \frac{1}{T} \int_{0}^{T} \log S_t dt \right) \]

For a European Asian call or put option with strike price \( K \), its payoff becomes

\[ C_T = (A_T(0, T) - K)^+ \]

or

\[ P_T = (K - A_T(0, T))^+ \]

In this report, only arithmetic average underlying price in discrete form is taken into considerations for final payoff. We use strike prices and maturities from 5 commodity option contracts to construct numerical examples, say European Asian call/put options, for proposed pricing models. Considering different specifications, there are 319 option contracts in total, with maturity varying from 5 days to 5 years. Their contract specifications, together with latest settlement prices, are collected as reference. Here is the list:

- WTI Crude Oil Asian Option
• Chicago Ethanol (Platts) Asian Option
• Gold American Option
• Silver American Option
• Natural Gas European Option

2.3 Forward Risk Free Interest Rates

In any pricing model based on numerical methods, the risk free rate always plays an important part and will affect the final pricing results significantly. Since all of the interested options as well as their underlyings are traded in US exchanges, US treasury yield curve on the day we collect price data will be regarded as risk free rate in numerical models. To implement risk free rate and simulate movements of underlying prices, the forward curve can be constructed as the following:

\[
f(t_i, t_{i+1}) = \log\left(\frac{e^{\rho \Delta t_i + t_{i+1}}/e^{\rho \Delta t_i}}{t_{i+1} - t_i}\right) = \frac{r_{t_i} + t_{i+1} - r_{t_i} t_i}{t_{i+1} - t_i}
\]  

(3)

In equation 3, \(r_t\) is the yield of a treasury instrument maturing at \(t_i\), and it can be regarded as the risk free rate for \((0, t_i)\). \(f(t_i, t_{i+1})\) is the forward rate in \((t_i, t_{i+1})\) derived by two treasury instruments with adjacent maturity dates.

2.4 Asian Option Pricing Models

2.4.1 Binomial Tree Model

The binomial tree model uses tree structure to simulate the path of the underlying asset over the whole life of Asian option and applies the forward shooting grid method to avoid the exponential growth of average price. In this case, we add extra grids to record the possible average price and restrict the number of possible average price to be linear growth. To update the average value when the asset value jumps, we use grid function to determine the updated average price.

Under Black-Scholes world, we assume that the volatility of asset value is constant. For the binomial tree, let the jump factor \(u, d\) be to be \(u = \exp(\sigma \sqrt{\Delta t}), d = \exp(-\sigma \sqrt{\Delta t})\) where \(\sigma\) is constant volatility, and \(\Delta t\) is the time step.

Denote \(S^n_k\) and \(A^n_k\) to be the asset value jumping upward \(j\) times and average price with index \(k\) at \(n\)-th time level, respectively. To restrict the possible values for \(F\) to a certain set of predetermined values, we limit the number of averaging values to some multiple of the number of values assumed by the asset price: Assume the coefficient to be \(1/\rho \in \mathbb{N}\). For a given time step \(\Delta t\), we let the asset value \(S^n_k\) and the average value \(A^n_k\) to be:

\[
S^n_k = S_0 e^j \Delta W, \quad A^n_k = S_0 e^k \Delta Y, \quad \Delta W = \sigma \sqrt{\Delta t}, \quad \Delta Y = \rho \Delta W, \quad k \in \mathbb{N} \tag{4}
\]

Then consider at \((n, j)\) node, for a upward jump from \((n, j)\) to \((n+1, j+1)\), the asset price will change from \(S^n_k\) to \(S^{n+1}_{j+1}\). Let \(A^n_{k+1}\) to be the updated value changing from \(A^n_k\) by the upward move. Then by usual computation of updating average value, we have:

\[
A^{n+1}_{k+1} = \left\{ (n+1) A^n_k + S^{n+1}_{j+1} \right\} / n + 2 \tag{5}
\]

Note that \(A^{n+1}_{k+1}\) do not coincide with \(A^{n+1}_{k+1} = S_0 e^{k \Delta Y}\) for some \(k' \in \mathbb{N}\) in general, indicating that \(k^\pm(j)\) may not be integer. Recall \(A^{n+1}_{k+1} = S_0 e^{k^{\pm(j)} \Delta Y}\) and \(S^{n+1}_{j+1} = S_0 e^{(p+1) \Delta W}\), by (5), we equate the two parts and have the grid function:

\[
k^{\pm(j)} = \ln\left\{ \left(\frac{n+1}{n+2}\right) \exp(j \Delta Y) + \exp((j+1) \Delta W) \right\} / \Delta Y \tag{6}
\]

Let \(K\) denote a subset of integer s.t. \(A^{n+1}_k \leq A^{n+1}_k\). Define \(k^{\pm(j)}\) s.t. \(A^{n+1}_{k^{\pm(j)}} = A^{n+1}_k\). In other words, \(k^{\pm(j)} = \lfloor k^{\pm(j)} \rfloor\), then \(k^{\pm(j)} - k^{\pm(j)}\) is an integer. Define \(k^{\pm(j)} = \lfloor k \rfloor\) and \(k^{\pm(j)} = k^{\pm(j)} + 1\). Define \(k^{\pm(j)}\) s.t. \(k^{\pm(j)} = k^{\pm(j)} + 1\). At \(n\)-th step, we have \(A_t = [S^n_{n+1}, S^n_{n+1}]\) and thus \(-n/\rho \leq k \leq n/\rho\). This helps us to restrict the size of \(k\). For a \(k \leq S^n_{n+1}\), we have \(A^n_k = [A^n_k, A^n_{k+1}]\) and \(-n/\rho \leq k^{\pm(j)} < k^{\pm(j)} \leq n/\rho\) at \(n\)-th time step.

To solve the problem that that \(k^{\pm(j)}\) may not be integer, we define the linear interpolation formula that will be used in our FSG method.

Let \(c^n_{j,1}\), denote the numerical approximation to the Asian call value at \((n, j)\) node with the averaging state variable assuming the value \(A^n_k\). Similar notations for \(c^n_{j,1}\), and \(c^n_{j,1}\),. For \(j \in \mathbb{R}\), the \(c^n_{j,1}\) will be approximated by linear interpolation:

\[
c^n_{j,1} = c^n_{j,1} + \epsilon_t (c^n_{j,1} - c^n_{j,1}) \tag{7}
\]

where \(\epsilon_t\) is the fraction of one step \(\Delta Y\) between \(A^n_{j,1}\) and \(A^n_{j,1}\):

\[
\epsilon_t = \frac{A^n_{j,1} - A^n_{j,1}}{\Delta Y}, A^n_{j,1} = A^n_{j,1} e^{\epsilon_t \Delta Y} \tag{8}
\]

Finally we have the pricing formula under a binomial tree scheme is given by:

\[
c^n_{j,k} = \frac{1}{R} \left\{ \rho c^n_{j+1,k} + (1 - \rho) c^n_{j-1,k} \right\} 
\]

\[
\approx \frac{1}{R} \left\{ \rho \left[ \left(1 - \epsilon_t \right) c^n_{j+1,k+1} + \left(1 + \epsilon_t \right) c^n_{j+1,k} \right] + (1 - \rho) \left[ \left(1 - \epsilon_t \right) c^n_{j-1,k-1} + \left(1 + \epsilon_t \right) c^n_{j-1,k} \right] \right\} \quad n = N - 1, \ldots, 0, j = -n, -n + 2, \ldots, n, \quad k \in \mathbb{N} \bigcap \left[ -\frac{n}{\rho}, \frac{n}{\rho} \right] \tag{9}
\]

where the risk neutral probability of jump upward is:

\[
p = \frac{R - d}{u - d}, \quad R = e^{\rho (T - t)} \tag{10}
\]
where \( r_t \) is the forward rate at time \( t \). Finally, the terminal payoff of put and call are:
\[
\begin{align*}
    p^N_{j,k} &= \max(K - S_0e^{k\Delta Y}, 0) \\
    c^N_{j,k} &= \max(S_0e^{k\Delta Y} - K, 0),
\end{align*}
\] (11)

### 2.4.2 Monte Carlo Simulations with Constant Volatility

In this method, the underlying commodity price \( S_t \) is assumed to follow the stochastic process below:
\[
dS_t = (r_t - \frac{1}{2}\sigma^2)S_t dt + \sigma S_t dW_t
\] (12)

where \( W_t \) is a standard Wiener process, \( r_t \) is the risk free interest rate at time \( t \), \( \sigma \) is a constant, representing the volatility of the underlying price's log return.

With US treasury yield curve, the forward the rate at time \( t \), \( \bar{r}_t \) is assumed to follow the stochastic process below:
\[
\bar{r}_t = \frac{1}{N-1}\sum_{i=1}^{N}(r_i - \bar{r})^2
\] (13)

Given a commodity start price \( S_0 \), the underlying price movements can be simulated by the dynamics of its log returns \( \{r_t\} \):
\[
dS_t = (r_t - \frac{1}{2}\sigma^2)S_t dt + \sigma S_t dW_t
\]
\[
d\log S_t = (r_t - \frac{1}{2}\sigma^2)dt + \sigma dW_t
\] (14)
\[
r_t = r_{t-1} + (r_t - \frac{1}{2}\sigma^2)dt + \sigma dW_t
\]
\[
S_{t_i} = S_{t_i-1}\exp(r_t\Delta t) = S_0\exp\left(\sum_{k=0}^{N} r_k\Delta t\right)
\]

The average price \( A_T(0, T) \) at the maturity is essentially:
\[
A_T(0, T) = \frac{1}{N}\sum_{i=0}^{N} S_{t_i}
\]

where \( t_0 = 0, t_N = T \).

Without loss of generality, Asian call option is used to illustrate the Monte Carlo scheme, and the put option can be easily derived thereafter. The Asian call option payoff becomes
\[
V_T = (A_T(0, T) - K)^+
\]

With \( n \) paths of simulated underlying prices, the fair payoff at the maturity can be expressed as an average of the payoff generated from each path.

\[
\tilde{V}_T = \frac{1}{n}\sum_{i=1}^{n} V^i_T
\]

Multiplying with corresponding discount factor, the value of the contract at present can be derived:
\[
df(0, T) = \exp\left(-\sum_{i=0}^{N-1} r_t\Delta t\right)
\]
\[
\tilde{V}_0 = df(0, T)V_T
\] (15)

where \( r_t \) is the forward risk free interest rate at \( t \), and \( \Delta t = 1/252 \).

Since the underlying is only traded on business days, BUS/BUS date convention is implemented in the simulation scheme.

### 2.4.3 Monte Carlo Simulations with GJR-GARCH model

As a comparison to the constant volatility MC scheme, GJR-GARCH model defines the log return of underlying price as a GARCH process. For a given time series \( \{S_t\} \),
\[
r_t = \log S_t - \log S_{t-1}
\]
\[
r_t = \mu + \epsilon_t
\]
\[
\epsilon_t = \alpha_t Z_t
\] (16)

where \( Z_t \) follows a standard normal distribution and
\[
\sigma_t^2 = \omega + \sum_{i} \alpha_i \epsilon_{t-i}^2 + \sum_{j} \beta_j \sigma_{t-j}^2
\]
\[+ \sum_{k} \gamma_k \epsilon_{t-k}^2 \text{ if } t_k < 0 \] (17)

This model will illustrate the volatility behavior of the time series, and since the volatility is the source engine to generate simulated price paths, we assume that a more subtle volatility model will contribute to more accurate pricing results.

In this report, for each commodity, the best model configuration, say the number of \( \alpha, \beta, \gamma \) and whether using zero mean, is selected based on the following steps.

**Step one: zero mean vs constant mean**

Constant mean model will be considered at the first place. If the p-value of \( \mu \) is larger than 0.05, then the mean is considered as insignificant, and it will be removed to refit the model again as a consequence.

**Step two: Ljung-Box test**

The fitted model will be plugged into historical data, and the series of \( \{Z_t\} \) is derived. If \( \{Z_t\} \) passes Ljung-Box test at a significance level of 0.05, then it proves that the model well explains the historical time series. Otherwise, the model will be rejected.

**Step three: BIC**

Bayesian information criterion (BIC) is implemented as a criterion for model selection, instead of Akaike information criterion (AIC). This is because BIC has
more punishments on number of parameters when the data set is large comparing AIC.

\[ AIC = 2k - 2\ln(\hat{L}) \]
\[ BIC = \ln(n)k - 2\ln(\hat{L}) \]  

(18)

where \( \hat{L} \) is the maximized value of the likelihood function of the model, \( k \) is the number of parameters of the model to estimate, and \( n \) is the sample size.

If one model has a less BIC and less number of parameters than the other one, it will be selected as the temporary best model. Alternatively, if BIC of one model is 5% less than the other one, it will also win the selection.

With the best fitted GJR-GARCH model, the volatility of future log returns will be predicted. A series of \( \{\sigma_0(t)\} \) can be derived, since

\[ \sigma_0(t) = E_0[\sigma(t)] \]

and then a path of underlying price can be derived from the generated log return series \( r_t \), where

\[ r_t = \mu + \sigma_0(t)Z_t \]

The rest pricing procedure is the same as the constant volatility scheme.

3 Process

3.1 Data Source

There are two types of data involved in this project, say commodity future price and option specifications. All of commodity future price data are retrieved from investing.com, a global financial portal owned by Fusion Media Limited. The time span starts from 2000-01-03 and ends at 2019-03-22. However, some futures, like silver and corn, may not have such a long price history. In this case, we take as earlier as we can, and all the future prices are available after year 2009. On the other hand, the option contract specifications and corresponding latest settle prices are recorded manually from CME group website, and there are 319*2=638 options in total (319 call put pairs).

3.2 Data Preprocess

The commodity prices are retrieved one by one, hence to further analyze or utilize them, one must concat price series into one data table with respect to dates. The concatenation generates plenty of missing values due to the difference in trading day conventions. For example, futures of silver, platinum and US soybean oil can be traded on Sundays, and all other future contracts will have missing prices on those days. Besides, the time span also varies from one commodity to another, which produces missing values for those with shorter time span before their first date point.

Two preprocessing rules are developed, one for time series analysis while the other for pricing. **Preprocessing for time series analysis:**

When two time series are selected for correlation analysis or collinearity analysis, only data points when both have data will be kept and all the date points with missing values will be removed.

**Preprocessing for pricing:**

Since Asian options only have one underlying, every historical time series will be handled independently. Thus date points with missing values will be dropped in this case.

For option specification data, the currency unit for strike price is different from each other, some are using USD cent, some are 0.001 USD. A currency divisor dictionary is specially created, and the strike price will all be converted to USD before pricing.

3.3 Correlation Analysis

Correlation analysis is a method of statistical evaluation used to study the strength of a relationship between two, numerically measured, continuous variables. In this step, the correlation coefficients among future prices and price log returns are calculated respectively by equation 19. Heatmaps are used to reflect their relationships. The results are shown in section 4.1.

\[ \rho_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{(n-1)s_x s_y} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2\sum_{i=1}^{n}(y_i - \bar{y})^2}} \]  

(19)

3.4 Non-GARCH Model Pricing

This part illustrates the process of Non-GARCH model pricing.

3.4.1 Binomial Tree Method

We use binomial tree model to price the current Asian option value. We first calculate the historical volatility of log return. Then we compute the forward rate based the U.S. treasury yield curve. We will use the historical volatility and forward rate as the volatility and risk free rate of the model. For choosing the time step, we consider the total number of business days from the spot date to maturity date. If the time length reaches over 1 year, we restrict the time steps to 252, so that the computational speed could be faster. Since the underlying asset is futures and each futures contract contains more than one unit of goods, hence we choose the price the option that the underlying asset contains 1 unit good.
3.4.2 Monte Carlo Simulations

After data preprocessing, the option specifications are stored in one data table, and the underlying table is stored in the other. To price one option with Monte Carlo simulations, the option specifications are retrieved as strike price $K$, start date $t$, maturity date $T$, and underlying name.

Using the underlying name, the fixed volatility $\sigma$ can be derived from historical underlying price data, as mentioned in section 2. The forward risk free interest rates are also calculated from treasury yield curve before simulations. The step size of Monte Carlo simulation is one business date, so the number of steps can be derived from Julian date difference of $t$ and $T$.

For every simulation, the underlying price path can be generated by following equation 14. Then the average price at the maturity, say $A_T(0,T)$ can be calculated based on the prices along the path. And the final discounted final payoff can be derived correspondingly; the average value of which will be the value priced by Monte Carlo simulation method.

3.5 GARCH Model Pricing

Before pricing the option, the best GARCH model needs to be selected from 200 possible models $(2 \times 10 \times 10 \times 10)$. There are 10 possible values, say from 0 to 9, for each of $\alpha$, $\beta$ and $\gamma$, and the model can be either zero mean or constant mean. The details about model selection criterion is mentioned in section 2.4.3. With the best fitted GARCH model, predictions for mean of volatilities can be derived from the model. Since the step size is one business day, the number of predictions equal to the number of business days from commencing date and the maturity of the option.

The rest pricing procedure are the same as those of Monte Carlo simulations mentioned in last section, except for the simulation scheme for underlying prices, which follows in section 2.4.3.

4 Results

4.1 Correlation Results

The heap maps are shown in the appendices.

Figure 1 is the correlation coefficient heat map of future log returns. It is clear that there are many strong positive linear relationships between some contracts, such as crude oil and gasoline or silver and platinum or soybean products. There is almost no strong negative linear relationships, but two commodities, say natural gas and live cattle, are less correlated to all other commodities. Except for those commodities made from same materials and those with competitive relationships, most of correlation coefficients are very high, say close to right color in heat map. These present randomness of correlations in commodity prices.

By contrast, figure 2 shows the correlation coefficient heat map of future log returns. Strong correlations are only presented between commodities within the same category, especially in agriculture or metals, and the log returns in same category are positively correlated.

4.2 Model results

Table 1 shows the model we fit based on BIC criteria. The p-value of Ljung-Box test shows that the GJR-GARCH model for Crude Oil, Gold, Natural gas and Silver is adequate but the model for Ethanol is not. The order of first four GJR-GARCH models are all less than 1, indicates that the selected model is simple. The order of the last model(Silver) is (1, 0, 5), indicating that the volatility of Silver contract has long memory compared to other underlying futures.

Convergence of the Monte Carlo simulation Figure 3, 4, 6, 5 shows the sample mean of the call option payoff on Monte-Carlo simulation. The x-axis shows the simulation times and y-axis shows the estimated option value. These figures shows that the option value convergences to a single value and hence the option value from the Monte-Carlo simulation is rational. The prediction of the volatility is shown in appendix.

For the three pricing model(GARCH-MC, MC, BT), we use the following criterion to evaluate the performance of the model:

ARE We use ARE to evaluate the difference between market price and fair price derived from the model. The formula of ARE is:

$$ARE = \frac{1}{N} \sum_{j=1}^{N} \frac{|V_{model} - V_{market}|}{V_{market}} \times 100$$

where $N$ is the total number of the samples, $V_{model}$ and $V_{market}$ are the fair price from the model and the market price of the $j$-th sample, respectively. The results are shown in table 3 (Zhu and Ling, 2015).

Compared to market price, GARCH-MC model outperforms to other model when the underlying assets are Ethanol and Silver, and call option on Gold, put option on Crude Oil. The MC model outperform on call option on Crude Oil, put option on Natural gas. The BT model outperforms only on call option on Natural gas. Therefore, the fair price form GARCH-MC model is the closest price to the market price.
Fair price of European option Instead of market price, the European option price calculated from BS model can also be used as criterion. The analytical value calculated from the BS formula is also a fair price under BS world. Since Asian option uses the average price of the whole path to compute the payoff while European option only consider the asset price on maturity, the risk of holding one unit of European option should be larger than holding one unit of Asian option. Hence the fair price of the Asian option should be smaller than the European option value with the same specification of options.

Denote \( N_s \) as the total number that the fair price of Asian option is smaller than the corresponding European option with the same specification and \( N \) as the total number of samples. We use the proportion \( PS = N_s/N \) as the statistic to evaluate the model, the result is:

<table>
<thead>
<tr>
<th>Models</th>
<th>( PS_{call} )</th>
<th>( PS_{put} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>GARCH-MC</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MC</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>BT</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 2: Proportion of number of Asian option value smaller than European option value with same specification

From table 2, we can conclude that all the value from our model are rational.

5 Discussions

5.1 Data Limitations

Commodity futures are the assessment of raw materials trading in open market. Future prices are essentially estimates for the value of future commodities, instead of the spot price. The difference between two prices, say future price and spot price, will be affected by delivery time, risk-free rate, storage cost and convenience yield. Therefore, it will bring inaccuracies into the pricing process when using futures price not the commodity spot prices.

Besides, only WTI Crude Oil Asian option and Chicago Ethanol (Platts) Asian option are Asian options and all the rest options are either American or European options, specifications of which are used to construct numerical examples for the proposed pricing models. In future studies, more traded Asian options settings should be collected to examine these models.

5.2 Model Limitations

Chicago Ethanol (Platinum) futures has many duplicate prices on adjacent dates in the early stage, resulting in zero log returns. Moreover, there is no fitted GJR-GARCH model for it, and GJR-GARCH(1,1,1) is used to fit its time series of log returns. However, ARIMA(0,0,1) can sufficiently explain the time series, we may consider applying ARIMA GARCH to fit the historical data and embedding into Monte Carlo scheme in further studies.

5.3 Monte Carlo Limitations

In Monte Carlo GARCH method, although Monte Carlo simulations are executed 10000 times, the standard deviation of payoffs generated is relatively large compared to constant volatility Monte Carlo method. This indicates that the number of paths might not be enough. Hence in GARCH Monte Carlo simulations, the final payoff could be a partial solution, and it may take million paths to obtain an output with lower variance. On top of that, the payoff variance from the proposed GARCH Monte Carlo simulation might underestimate the real variance, especially when the number of simulated path is insufficient. In this case, probability bounds analysis (PBA) can be implemented as an examination for partial information. Using PBA, the sparse simulations will provide far-apart upper and lower bounds, and this will well estimate the risk in pricing results.

5.4 Advanced GARCH Models

Affine GARCH model is described as the following:

\[
\begin{align*}
    r_t &= r - \frac{1}{2} \sigma_t^2 + \sigma_t Z_t \\
    \sigma_t^2 &= \omega + \alpha_1 (Z_{t-1} - \lambda \sigma_{t-1})^2 + \beta_1 \sigma_{t-1}^2
\end{align*}
\]  

(21)

According to Lorenzo, a semi-analytical formula for geometric Asian options can be provided when the underlying follows an affine GARCH process and the extreme asymmetry of this kind of models with non-normal innovations will price more accurately for options with very short time to maturity(Lorenzo, 2011). Hence, affine GARCH might provides a better pricing result if it is implemented to Monte Carlo simulation scheme.

6 Conclusions

In this report, 16 commodity future prices are studied, and 5 options of them are priced by three pricing methods, say binomial tree model, Monte Carlo simulations with constant volatility, and Monte Carlo simulations with GJR-GARCH volatility. Strong sector correlations are discovered in the 16 time series of commodity log returns. According to pricing result analysis, the ARE criterion shows that GJR-GARCH model is the most appropriate pricing model in three models implemented in this report. However, there are also a few limitations in this report, say the data limitations and Monte Carlo convergence limitations. There are also other advanced
GARCH models, say affine GARCH, to examine in the future studies.

References


7 Appendices

7.1 Figures

Figure 1: Correlation Analysis of Commodity Prices
Figure 2: Correlation Analysis of Commodity Log Returns
Figure 3: Convergence Plot for a Crude Oil Asian Option

Figure 4: Convergence Plot for an Ethanol Option
Figure 5: Convergence Plot for a Natural Gas Asian Option

Figure 6: Convergence Plot for a Silver Asian Option
Figure 7: Volatility Prediction on Crude Oil

Figure 8: Volatility Prediction on Ethanol
Figure 9: Volatility Prediction on Gold

Figure 10: Volatility Prediction on Natural Gas
Figure 11: Volatility Prediction on Silver
### 7.2 Tables

<table>
<thead>
<tr>
<th>Models</th>
<th>Crude Oil WTI put</th>
<th>Crude Oil WTI call</th>
<th>Ethanol put</th>
<th>Ethanol call</th>
<th>Gold put</th>
<th>Gold call</th>
<th>Silver put</th>
<th>Silver call</th>
<th>Natural gas put</th>
<th>Natural gas call</th>
</tr>
</thead>
<tbody>
<tr>
<td>MC</td>
<td>83.121</td>
<td>122.238</td>
<td>56.035</td>
<td>73.732</td>
<td>52.874</td>
<td>91.129</td>
<td>64.379</td>
<td>89.397</td>
<td>64.891</td>
<td>84.962</td>
</tr>
<tr>
<td>BT</td>
<td>402.088</td>
<td>819.106</td>
<td>45.110</td>
<td>62.466</td>
<td>106.801</td>
<td>149.124</td>
<td>83.969</td>
<td>107.557</td>
<td>264.242</td>
<td>72.697</td>
</tr>
</tbody>
</table>

**Table 3:** ARE for the Asian put and call option for each underlying futures.
### 7.3 Scripts

#### 7.3.1 corr.py

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

Created on Sun April 12 15:53:00 2019
@author: jingqian

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

data = pd.read_csv("../Cleaned Data/Cleaned Underlying Data.csv", index_col=0)
newCols = ['Crude Oil WTI', 'Natural Gas', 'Gasoline RBOB', 'Ethanol', 'Coal',
'Lean Hogs', 'Live Cattle', 'Gold', 'Silver', 'Platinum', 'Copper']

price = data[newCols]
corr_price = price.corr()
corr_logRt = price.corr()

for col1 in newCols:
    for col2 in newCols:
        tmpdf = data[[col1, col2]].dropna()
corr_price[col1][col2] = tmpdf[[col1]].corrwith(tmpdf[col2]).values[0]

logdf = data[[col1, col2]].dropna()
logdf[col1] = (np.log(logdf[col1]) - np.log(logdf[col1].shift(1))) * 100
logdf[col2] = (np.log(logdf[col2]) - np.log(logdf[col2].shift(1))) * 100
logdf = logdf.dropna()
corr_logRt[col1][col2] = logdf[[col1]].corrwith(logdf[col2]).values[0]

plt.subplots(figsize=(12, 12))
sns_plot = sns.heatmap(corr_price, annot=True, vmin=-1, vmax=1, square=True, cmap="RdBu_r")
fig = sns_plot.get_figure()
fig.savefig('corr_price.jpg')

plt.subplots(figsize=(12, 12))
sns_plot = sns.heatmap(corr_logRt, annot=True, vmin=-1, vmax=1, square=True, cmap="RdBu_r")
fig = sns_plot.get_figure()
fig.savefig('corr.jpg')
```

#### 7.3.2 binomialTreePrice.py

```python
# -*- coding: utf-8 -*-

Created on Thu April 14 10:23:02 2019
@author: Leheng Chen

import numpy as np

class asianOptionBinomialTree:
    def __init__(self, num_steps, volatility, time_period, oneOverRho, interest_rate):
        self.num_steps = num_steps
        self.volatility = volatility
        self.time_period = time_period
        self.oneOverRho = oneOverRho
        self.interest = np.array(interest_rate)
        self.discount_factor = np.exp(-1 * self.interest * self.time_period)
```
self.half_len_grid = self.num_steps * self.oneOverRho

self.averagePriceTree = np.zeros(2 * self.num_steps + oneOverRho + 1)
self.assetPriceTree = np.zeros((self.num_steps + 1, self.num_steps + 1))
self.optionPriceTree = np.zeros((self.num_steps + 1, 2 * self.num_steps + self.
oneOverRho + 1))

def forwardInduction(self):
    self.up_factor = np.exp(self.volatility * np.sqrt(self.time_period))

    for i in range(self.num_steps + 1):
        lower_bound = -i
        for j in range(i + 1):
            self.assetPriceTree[i, j] = self.init_price * (self.up_factor ** lower_bound)
            lower_bound = lower_bound + 2

    for j in range(2 * self.num_steps + oneOverRho + 1):
        jump = j - self.half_len_grid
        self.averagePriceTree[j] = self.init_price * (self.up_factor ** (jump / self.
oneOverRho))

    for s in range(self.num_steps + 1):
        for k in range(2 * self.num_steps + oneOverRho + 1):
            self.optionPriceTree[s, k] = max(self.averagePriceTree[k] - self.strike, 0)


def grid(self, n, k, j, plus):
    numerator = np.zeros((len(j), len(k)))
    denominator = self.volatility * np.sqrt(self.time_period) / self.oneOverRho

    for jj in j:
        numerator[jj] = (n + 1) * self.up_factor ** (k / self.oneOverRho) + self.up_factor ** (jj + plus)
        numerator[jj] = np.log(numerator[jj] / (n + 2))

    return numerator / denominator


def backwardInduction(self):
    delta_y = self.volatility * np.sqrt(self.time_period) / self.oneOverRho
    proba_up = (1 / self.discount_factor - 1 / self.up_factor) / (self.up_factor - 1 / self.up_factor)

    for n in reversed(range(self.num_steps)):
        print(np.round(self.optionPriceTree))

        k_idx = np.array([k for k in range(- n * self.oneOverRho, n + self.oneOverRho + 1)])
        j_idx = np.array([j for j in range(n + 1)])
        k_up = self.grid(n, k_idx, j_idx, 1)
        k_down = self.grid(n, k_idx, j_idx, -1)

        j_idx_ext = np.repeat(j_idx[:, np.newaxis], len(k_idx), axis=1)
        k_up_floor = np.maximum(np.floor(k_up + self.half_len_grid).astype(int), 0)
        k_upceil = np.minimum(k_up_floor + 1, self.half_len_grid + 2)

        average_price_up = ((n + 1) * self.averagePriceTree[n, k] + self.assetPriceTree[
n + 1, i + 1]) / (n + 2)
        average_price_down = self.init_price * self.up_factor ** (k_up / self.oneOverRho)
        factor_interpolation_up = np.log(average_price_up / self.averagePriceTree[k_up_
floor]) / delta_y

        option_price_up = self.factor_interpolation_up[0:] * self.optionPriceTree[j_idx_ext[0:], k_upceil[0:]] + 
        (1 - self.factor_interpolation_up[0:]) * self.optionPriceTree[j_idx_ext[0:], k_up_
floor[0:]]

        k_down_floor = np.maximum(np.floor(k_down + self.half_len_grid).astype(int), 0)
        k_downceil = np.minimum(k_down_floor + 1, self.half_len_grid + 2)

        average_price_down = ((n + 1) * self.averagePriceTree[n, k] + self.
assetPriceTree[n + 1, i - 1]) / (n + 2)
        factor_interpolation_down = np.log(average_price_down / self.averagePriceTree[k_
downfloor]) / delta_y

        # assert self.averagePriceTree[n + 1, k_down_floor] != 0

        # for k in range(2 * self.num_steps + oneOverRho + 1):
        #     jump = j - self.half_len_grid
        #     self.averagePriceTree[j] = self.init_price * (self.up_factor ** (jump / self.
oneOverRho))

        # for s in range(self.num_steps + 1):
        #     for k in range(2 * self.num_steps + oneOverRho + 1):
        #         self.optionPriceTree[s, k] = max(self.averagePriceTree[k] - self.strike, 0)
```python
option_price_down = factor_interpolation_down[: (n + 1)] * self.optionPriceTree[j_idx_ext[: (n + 1)]] + \
(1 - factor_interpolation_down[: (n + 1)]) * self.optionPriceTree[j_idx_ext[: (n + 1)], k_down_floor[: (n + 1)]]
self.optionPriceTree[j_idx_ext[: (n + 1)], k_idx + self.half_len_grid] = proba_up[n] * option_price_up + (1 - proba_up[n]) * option_price_down
self.optionPriceTree[j_idx_ext[: (n + 1)], k_idx + self.half_len_grid] *= self.discount_factor[n]
# print(np.round(self.optionPriceTree[j_idx_ext[: (n + 1)], k_idx + self.half_len_grid]))

def getOptionPrice(self, init_price, strike):
    self.init_price = init_price
    self.strike = strike
    self.forwardInduction()
    self.backwardInduction()
    return self.optionPriceTree[0, self.half_len_grid]
```

---

### 7.3.3 biTreePriceSimulation.py

```python
# -*- coding: utf-8 -*-

# Created on Thu May 9 13:46:08 2019
# Author: Leheng Chen

from binomialTreePricer import asianOptionBinomialTree
import pandas as pd
import numpy as np
from datetime import datetime, timedelta

uly_names = ['Crude Oil WTI', 'Ethanol', 'Gold', 'Silver', 'Natural Gas']
uly_init = df_uly[uly_names].tail(1)
df_opt['bdays'] = 1 + np.busday_count(df_opt['Start Date'].values.astype('datetime64[D]'), df_opt['Maturity Date'].values.astype('datetime64[D]'))
df_uly_vol = df_uly[uly_names].std(skipna=True)
oneOverRho = 3
df_vols = pd.DataFrame([[0.3, 0.01, 0.4, 0.1, 0.001]], columns = uly_names)
df_units = pd.DataFrame([[0.01, 0.0001, 1, 0.001, 0.01]], columns = uly_names)
bdays_year = 252

# -------------------------------------------------------------------------------------------------
# Define risk free rate, reference to US treasury yield curve as of 20190322
# 1m, 2m, 3m, 6m, 1y, 2y, 3y, 5y, 7y, 10y, 20y, 30y
# -------------------------------------------------------------------------------------------------
# Define risk free rate according to US

# Derive forward rates from US treasury yield curve
```
Pricing Asian Options on Commodities with GARCH Model

curvePoints = ['2019-03-22'] + list(yieldCurveDict.keys())

forwardCurveDict = {}
for i in range(len(yieldCurveDict)):
    datePoint1 = curvePoints[i]
    datePoint2 = curvePoints[i+1]
    if (datePoint1 == curvePoints[0]):
        forwardCurveDict[datePoint2] = yieldCurveDict[datePoint2]
    else:
        yieldAtDate1 = yieldCurveDict[datePoint1]
        yieldAtDate2 = yieldCurveDict[datePoint2]
        busDateDiff1 = np.busday_count(curvePoints[0], datePoint1)
        busDateDiff2 = np.busday_count(curvePoints[0], datePoint2)
        forwardCurveDict[datePoint2] = float((yieldAtDate2 * busDateDiff2 - yieldAtDate1 * busDateDiff1) / (busDateDiff2 - busDateDiff1))

# Function to get risk free rate given a date (datetime.date object)
def getRiskFreeRate(inputDate):
    input_date = inputDate.date()
    for i in range(len(forwardCurveDict)):
        datePoint1 = datetime.strptime(curvePoints[i], '%Y-%m-%d').date()
        datePoint2 = datetime.strptime(curvePoints[i+1], '%Y-%m-%d').date()
        if (input_date >= datePoint1 and input_date < datePoint2):
            return forwardCurveDict[curvePoints[i+1]]
    return 0

for row in df_opt.index:
    # Retrieve the name of the underlying
    tmp_uly = df_opt['Underlying'][row] [-8:
    tmp_strike = df_opt['Strike'][row]
    tmp_maturity = df_opt['Maturity Date'][row]
    tmp_steps = df_opt['bdays'][row]
    if tmp_steps > bdays_year:
        tmp_steps = bdays_year
    tmp_init = uly_init[tmp_uly][0]
    tmp_time_period = 1 / bdays_year
    tmp_vol = df_uly_vol[tmp_uly]
    tmp_ir = get_interest_rate(tmp_steps)
    tmp_rates = [getRiskFreeRate(tmp_maturity - timedelta(d)) for d in range(tmp_steps)]
    tmp_call = df_opt['Call'][row]
    tmp_unit = df_units[tmp_uly][0]
    pricer = asianOptionBinomialTree(tmp_steps, tmp_vol, tmp_time_period, oneOverRho, tmp_rates)
    sim = pricer.getOptionPrice(tmp_init, tmp_strike * tmp_unit)
    print('underlying : %s, bdays : %d, strike : %6.3f, init : %6.3f --> simulate : %6.3f; actual call : %6.3f' %
        (tmp_uly, tmp_steps, tmp_strike * tmp_unit, tmp_init, sim, tmp_call))
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
warnings.simplefilter("ignore")

# Define risk free rate, reference to US treasury yield curve as of 20190322
# 1m, 2m, 3m, 6m, 1y, 2y, 3y, 5y, 7y, 10y, 20y, 30y

# Define risk free rate according to US
yieldCurveDict = {
    '2019-04-22': 2.49,
    '2019-05-22': 2.48,
    '2019-06-22': 2.46,
    '2019-09-22': 2.48,
    '2020-03-22': 2.45,
    '2021-03-22': 2.31,
    '2022-03-22': 2.24,
    '2024-03-22': 2.24,
    '2026-03-22': 2.34,
    '2029-03-22': 2.44,
    '2039-03-22': 2.69,
    '2049-03-22': 2.88
}

# Derive forward rates from US treasury yield curve
curvePoints = ['2019-03-22'] + list(yieldCurveDict.keys())

forwardCurveDict = {}
fwdCurveDict = {0:0}
for i in range(len(yieldCurveDict)):
datePoint1 = curvePoints[i]
datePoint2 = curvePoints[i + 1]
busDateDiff1 = np.busday_count(curvePoints[0], datePoint1)
budDateDiff2 = np.busday_count(curvePoints[0], datePoint2)
if (datePoint1 == curvePoints[0]):
    forwardCurveDict[datePoint2] = yieldCurveDict[datePoint2]
fwdCurveDict[busDateDiff2] = yieldCurveDict[datePoint2]
else:
yieldAtDate1 = yieldCurveDict[datePoint1]
yieldAtDate2 = yieldCurveDict[datePoint2]
forwardCurveDict[datePoint2] = float((yieldAtDate2 * busDateDiff2 - yieldAtDate1 * busDateDiff1) / (busDateDiff2 - busDateDiff1))
fwdCurveDict[busDateDiff2] = float((yieldAtDate2 * busDateDiff2 - yieldAtDate1 * busDateDiff1) / (busDateDiff2 - busDateDiff1))

# Function to get risk free rate given a date (datetime.date object)
deﬁn getRiskFreeRateByDate(inputDate):
    for i in range(len(forwardCurveDict)):
        datePoint1 = datetime.strptime(curvePoints[i], '%Y-%m-%d').date()
datePoint2 = datetime.strptime(curvePoints[i + 1], '%Y-%m-%d').date()
        if (inputDate >= datePoint1 and inputDate < datePoint2):
            return forwardCurveDict[curvePoints[i + 1]]
    return 0

# Function to get risk free rate given a business date count from 20190322
def getRiskFreeRate(dayCounts):
    dayCountPoints = list(fwdCurveDict.keys())
    for i in range(len(dayCountPoints)-1):
        dayCount1 = dayCountPoints[i]
dayCount2 = dayCountPoints[i + 1]
        if (dayCounts >= dayCount1 and dayCounts < dayCount2):
            return fwdCurveDict[dayCount2]
    return 0

# convenience yield - storage cost
commodityYieldDict = {
    'Crude Oil WTI': 0.01,
    'Ethanol': 0.01,
    'Gold': 0.01,
    'Natural Gas': 0.01,
}
```python
# Utility functions
# =============================================================================
commodityCurrDict = {
    'Crude Oil WTI': 0.01,
    'Ethanol': 0.0001,
    'Gold': 1,
    'Natural Gas': 0.001,
    'Silver': 0.01
}

def get_currency_divisor(commodity):
    return commodityCurrDict[commodity]

# Get best GJR GARCH model for the log return series
# Make sure the model pass Ljungbox Test, and fit the time series
if pvalue[19] > 0.05:
    if best_bic / tmp_bic > 1.05:
        best_bic = tmp_bic
        best_order = [pValue, oValue, qValue]
    else:
        best_bic = best_bic
        best_order = [pValue, oValue, qValue]
except:
    continue
```
Pricing Asian Options on Commodities with GARCH Model

```
best_mdl = tmp_mdl.fit(update_freq=5, disp='off')

# Remove mean if it's not significant
if best_mdl.pvalues['mu'] > 0.05:
    isZeroMean = True
    tmp_mdl = arch_model(y = logRtSeries,
                       mean = 'Zero',
                       p = 1,
                       o = 1,
                       q = 1,
                       dist = 'Normal')
    best_mdl = tmp_mdl.fit(update_freq=5, disp='off')

best_bic = best_mdl.bic
best_order = [1, 1, 1]

# Test for first 20-lag
wn_test = best_mdl.resid / best_mdl._volatility
[lbvalue, pvalue] = acorr_ljungbox(wn_test, lags = 20)

output = {}
output['Zero Mean Model'] = isZeroMean
output['Best BIC'] = best_bic
output['Best Order'] = best_order
output['Best Model'] = best_mdl
volForecasts = best_mdl.forecast(horizon=predictDays)
output['Vol Predictions'] = np.sqrt(volForecasts.residual_variance.iloc[-1].values)
output['Ljungbox Test Statistics'] = lbvalue[19]
output['Ljungbox Test pvalue'] = pvalue[19]
return output

# Get affine garch model [not implemented yet]
def get_affine_garch(logRtSeries):
    tmp_mdl = arch_model(y = logRtSeries,p = 1,q = 1,dist = 'Normal')
    tmp_res = tmp_mdl.fit(update_freq=5, disp='off')
    tmp_bic = tmp_res.aic

    output = {}
    output['Best AIC'] = tmp_bic
    output['Best Model'] = tmp_res
    return output

# Use GARCH vol to price Asian option (Monte Carlo)
def garchPricer(startPrice, strikePrice, garchModel, expBusDays, numPath):
    sumCallPrice = 0
    sumPutPrice = 0
    res = garchModel['Best Model']
    mu = 0
    if garchModel['Zero Mean Model'] == False:
        mu = res.params['mu']
    vol = garchModel['Vol Predictions']
    simulatedUlyPrices = []
    callPrices = []
    putPrices = []
    for i in range(numPath):
        simulatedUlyPrice = [startPrice]
        ulyPrice = startPrice
        sumUlyPrice = 0
        dt = 1 / 252
        randomGenerator = np.random.normal(0, 1, expBusDays)
        discountRate = 0
        # Simulated one path of underlying price
        for j in range(expBusDays):
            zt = randomGenerator[j]
            rt = getRiskFreeRate(j) / 100
            dLogSt = mu + vol[j] / 100 * zt
            discountRate += rt
            ulyPrice *= np.exp(dLogSt)
            simulatedUlyPrice.append(ulyPrice)
```
def garchPricer(startPrice, strikePrice, vol, costYield, expBusDays, numPath):
    sumCallPrice = 0
    sumPutPrice = 0
    simulatedUlyPrices = []
    callPrices = []
    putPrices = []

    for i in range(numPath):
        simulatedUlyPrice = [startPrice]
        ulyPrice = startPrice
        sumUlyPrice = 0
        dt = 1 / 252
        randomGenerator = np.random.normal(0, np.sqrt(dt), expBusDays)
        discountRate = 0

        # Simulated one path of underlying price
        for j in range(expBusDays):
            dWt = randomGenerator[j]
            rt = getRiskFreeRate(j) / 100
            dLogSt = (rt - costYield - (vol / 100)**2 / 2) * dt + vol / 100 * dWt
            discountRate += rt
            ulyPrice *= np.exp(dLogSt)
            simulatedUlyPrice.append(ulyPrice)
        avgUlyPrice = sumUlyPrice / expBusDays
        simulatedUlyPrices.append(simulatedUlyPrice)
        callPrice = max(avgUlyPrice - strikePrice, 0) * np.exp(-discountRate * dt)
        putPrice = max(strikePrice - avgUlyPrice, 0) * np.exp(-discountRate * dt)
        callPrices.append(callPrice)
        putPrices.append(putPrice)
        sumCallPrice += callPrice
        sumPutPrice += putPrice

    output = {
        'Call': sumCallPrice / numPath,
        'Put': sumPutPrice / numPath,
        'Call Prices': callPrices,
        'Put Prices': putPrices,
        'Call STD': statistics.stdev(callPrices),
        'Put STD': statistics.stdev(putPrices),
        'Simulated Price': simulatedUlyPrices
    }

    return output
# Data Preprocessing
# -------------------------------------------------------------------------------
# Load underlying data from git
df_uly = pd.read_csv("../Underlying Data/Underlying Data.csv", sep=',')
# Preprocess dataframe, set up index, fill nan with latest previous values
df_uly.index = pd.to_datetime(df_uly['Date']).dt.date
df_uly = df_uly.drop('Date', axis=1)
# Load option data from git
df_opt = pd.read_csv("../Option Price Data/Option Data.csv", sep=',')
# Preprocess dataframe, convert dates, calculate days to maturity
df_opt.columns = ['Start Date', 'Maturity Date', 'Strike', 'Put', 'Call', 'Underlying']
df_opt['Maturity Date'] = pd.to_datetime(df_opt['Maturity Date']).dt.date
df_opt['Start Date'] = '2019-3-25'
df_opt['Start Date'] = pd.to_datetime(df_opt['Start Date']).dt.date
try:
df_opt['Exp BusDays'] = np.busday_count(df_opt['Start Date'], df_opt['Maturity Date']) + 1
except:
    # Sometimes line above may not work, use method below as an alternative
    tmp_list = []
    for i in range(len(df_opt['Start Date'])):
        tmp_list.append(np.busday_count(df_opt['Start Date'][i], df_opt['Maturity Date'][i]) + 1)
df_opt['Exp BusDays'] = tmp_list
# Get best GARCH model and other info for underlyings whose options we will price later
masterObj = {}
ulyList = list(np.unique(df_opt['Underlying']))
for underlying in progressbar.progressbar(ulyList):
    tmp_uly = underlying[:8]
    TS_uly = df_uly[tmp_uly].dropna()
    TS_logRt = (np.log(TS_uly) - np.log(TS_uly.shift(1))).dropna() * 100
    TS_logRt = TS_logRt[TS_logRt > 0]
    max_expDays = max(df_opt[df_opt['Underlying'] == underlying]['Exp BusDays'])
    masterObj[tmp_uly] = {
        'Start Price': TS_uly[-1],
        'Volatility': statistics.stdev(TS_logRt),
        'Garch Model': get_best_model(TS_logRt, 10, 10, 10, max_expDays)
    }
# Execute garchPricer and collect results
# -------------------------------------------------------------------------------
nonGarchFairPriceCall = []
nonGarchFairPricePut = []
garchFairPriceCall = []
garchFairPricePut = []
nonGarchCallStd = []
nonGarchPutStd = []
garchCallStd = []
garchPutStd = []
resultsObj = {}
# Number of Monte Carlo simulated paths
numPath = 10000
# Loop through options
for row in progressbar.progressbar(df_opt.index):
    # Retrieve the name of the underlying
    tmp_uly = df_opt['Underlying'][row][:8]
tmp_strike = df_opt['Strike'][row] * get_currency_divisor(tmp_uly)
tmp_maturity = df_opt['Maturity Date'][row]
tmp_expBusDays = df_opt['Exp BusDays'][row]
    # Retrieve the underlying historical data
    tmp_s0 = masterObj[tmp_uly]['Start Price']
tmp_vol = masterObj[tmp_uly]['Volatility']
tmp_model = masterObj[tmp_uly]['Garch Model']

nonGarchResults = nonGarchPricer(tmp_s0, tmp_strike, tmp_vol, commodityYieldDict[tmp_uly],
                              tmp_expBusDays, numPath)
nonGarchFairPriceCall.append(nonGarchResults['Call'])
nonGarchFairPricePut.append(nonGarchResults['Put'])
nonGarchCallStd.append(nonGarchResults['Call STD'])
nonGarchPutStd.append(nonGarchResults['Put STD'])
garchResults = garchPricer(tmp_s0, tmp_strike, tmp_model, tmp_expBusDays, numPath)
garchFairPriceCall.append(garchResults['Call'])
garchFairPricePut.append(garchResults['Put'])
garchCallStd.append(garchResults['Call STD'])
garchPutStd.append(garchResults['Put STD'])

resultsObj[row] = {
    'GARCH MC': garchResults,
    'Non-GARCH MC': nonGarchResults
}

df_opt['Put (MC non-GARCH)'] = nonGarchFairPricePut
df_opt['Put (MC non-GARCH) STD'] = nonGarchPutStd
df_opt['Call (MC non-GARCH)'] = nonGarchFairPriceCall
df_opt['Call (MC non-GARCH) STD'] = nonGarchCallStd
df_opt['Put (MC GARCH)'] = garchFairPricePut
df_opt['Put (MC GARCH) STD'] = garchPutStd
df_opt['Call (MC GARCH)'] = garchFairPriceCall
df_opt['Call (MC GARCH) STD'] = garchCallStd