Abstract

This is the final report of the project with option pricing with macroeconomic inputs. In this project, we developed an option pricing model with the combination of Black Scholes formula and the forecasting volatility from Garch process added with macroeconomics effects. To be more precisely to observe that how the macroeconomic effects on the volatility as well as the option price, additionally, an Equivalent Martingale Simulation method was induced to ensures that the price estimated by simulation satisfies the rational option pricing bounds.

1 Introduction

Modelling and forecasting stock market volatility has been the subject of vast empirical and theoretical investigation over the past decade or so by academics and practitioners alike. Volatility, as measured by the standard deviation or variance of returns, is often used as a crude measure of the total risk of financial assets. The pricing of market options requires the input of volatility into Black-Scholes formula, thus the modelling of volatility have been great interests for option market participants.

While traditional finance theory assumes assets return follow normal distribution and the variance is homoscedasticity, the assumptions are usually violated.

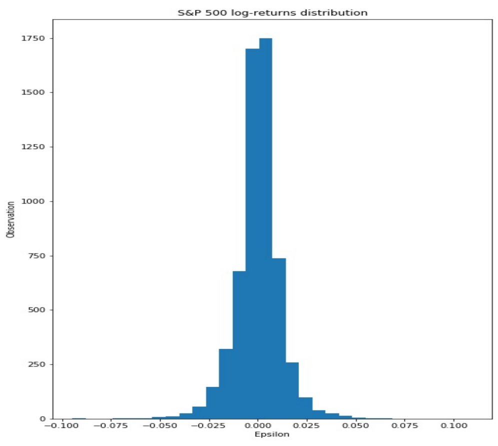

Figure 1: Log return distribution

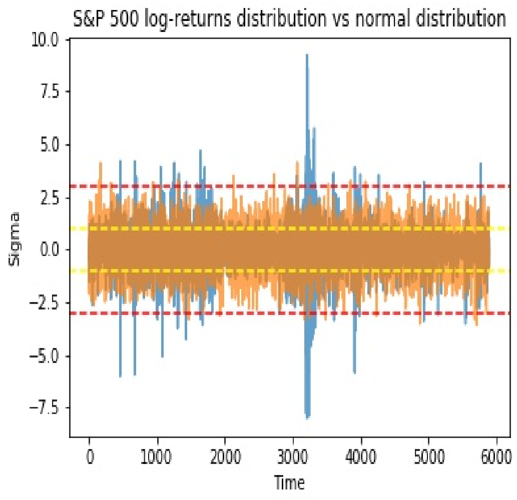

Figure 2: Volatility plotting

As seen in figure 1 and figure 2, the log-returns of the sampled SP 500 index exhibit fat tails and negative skewness in its distributions and their volatility behaviours change through time, exhibits extreme value and clustering effect. This motivates academics and practitioners to model volatility using GARCH models. A GARCH (generalized autoregressive conditional heteroskedasticity) model is used to model conditional variance and allow it to be dependent upon previous legs and the volatility from the previous period from the mean model. The widely used GARCH(1,1) is parsimonious, flexible and free from overfitting, and has various extension based on it. There are empirical evidences that GARCH models could price options with limited errors and capture volatility smiles to a certain extent. Hsieh and Ritchken (2000) compared Duan's NGARCH model and Heston & Nandi Model in their ability in pricing S&P 500 index options, concluded that not only both GARCH models could explain significant amount of maturity and strike price bias from Black Scholes model, but also NGARCH has superiority in pricing out of sample options even when parameters are not re-estimated for quarters. Barone-Adesi (2007) estimate GJR GARCH model with filtered historical innovations methods and demonstrated it could explain implied volatility smiles and negative skewness. Lethnert(2003) applied EGARCH with generalized error distribution and documented significantly small valuation errors especially for deep OTM options.

When market collapses or soars, high trading volume will increase volatilites. During crisis when people rush out of stock market and rush into safe haven, long term treasury bonds yield will be usually brought down, gold prices or the Swiss franc will be pushed up. Lower interest rates will attract more stock tradings, thus more volatility. Those observed market phenomenon also attracts many studies into Marco-economic factors' impacts on stock volatility. There are numerous evidences showing strong linkages between stock market volatility and economic variables.Lamoureax and Lastrapes (1990) indicated the influence of trading volume on the persistence of GARCH effects on the returns of the financial assets. Brailsford (1994) paper presented an empirical analysis of the relationship between trading volume and stock return volatility in the Australian market. Bhowmik (2013) investigated the market volatility interaction among a futures contract written on the monetary policy rate set by the Fed. Kennedy and Nourzad (2016) presented the effect of exchange rate volatility on financial volatility using weekly data for the U.S. covering the period from the first week of 1999 through the third week of 2010. This study finds that increased exchange rate volatility exerts a positive and statistically significant effect on the volatility of market returns.Jubinski and Lipton (2013) indicated a statistically significant and negative relationship between oil futures returns and market implied volatility, however, from 1996-2006, gold volatility does have a positive relation to equity volatility.

Despite that, there is few study that combine Macro-economic factors with GARCH models to jointly explain volatility behaviours and subsequent application of option pricing, this study attempts to fill this gap. In this paper, a self-designed computational scheme has been deployed to estimate augmented GARCH model with S&P 500 index closing prices. The economic variables that are used to augment the GARCH are S&P 500 trading volume, 10 years US Treasury Yield, foreign exchange rate (Japanese Yen/US Dollar), gold futures, and crude oil (West Texas Intermediate) futures. Those economic factors' individual impact on option pricing is investigated one by one, though out of sample model forecasts and simulations. Black Scholes model' option pricing results are used as Benchmarks.

2 Data Collection and Process

2.1 Stocks and Options Data

The date used to estimate the parameters GARCH is S&P 500 index daily closing price from 1996 Jan 1st to 2019 Jun 28th, has a sample size 5898. The logarithm return of the date has a mean 0.0002639 and a variance 0.00014027 (daily). This variance is close to the calculated unconditional variance from selected sample in the later session.

In this paper, we only used 3-month maturity market options from 1996 Jan 1st to 2019 Jun 28th to validate the option pricing ability of the GARCH model . Options that have 86 days to 94 days are supposed to have 3-month maturity because exact 90 days maturity options are rare, which are not enough for regression process and validation. After filtering the options data, 333378 call or put options with different strikes were selected. The detail is shown in the table below.

Table 1: Options data descriptive statistics

Option Type	Call Options	Put Options	Total
In the money	115931	50645	166575
At the money	3	3	6
Out of money	50713	116083	166796
Total	166647	166731	333378

2.2 Macroeconomic Data

In the GARCH process: $\sigma_{t+1}^2 = \beta_0 + \beta_1\sigma_t^2 + \beta_2\sigma t^2(e_t - \lambda)^2 + \sum_{k=1}^{n}\gamma_k X_{kt}$, different types of macroeconomic data in the same period with selected options are added to volatility process to forecast future volatility. Based on studies of Lamoureax and Lastrapes (1990), Brailsford (1994), Bhowmik (2013) Kennedy and Nourzad (2016), we selected S&P 500 trading volume, 10 years US Treasury Yield, foreign exchange rate volatility (Japanese Yen/US Dollar), gold futures, crude oil (West Texas Intermediate) futures, and those are from Investing.com, Yahoo finance, US Department of the Treasury and Wharton Research Data Services. Daily exchange volatility is hard to get, according to Kennedy and Nourzad (2016), daily exchange rate volatility is calculated as $\sqrt{\frac{(high-open)^2+(low-open)^2}{2}}$ for simplification, where 'high' is daily highest price, 'low' is daily lowest price, and 'open' is daily open price. Descriptive statistics, correlation matrix and unit root test of these data are shown in the table below.

It can be noticed from the correlation matrix that 10y treasury yield has relatively strong negative correlation with gold future, oil future and trading volume. The trading volume has strong positive correlation with oil and gold futures. Besides, the oil and gold futures themselves are strongly correlated. Thus only individual economic component is included into GARCH model one at time to avoid multicollinearity.

For unit root, only daily log return of S&P 500 index and exchange rate are stationary.

Table 2: Descriptive statistics of Macroeconomic Data

Variables	Mean	Std Dev	25%	Median	75%
Exchange Rate Daily Volatility (USD/JPY)	0.634475	0.425896	0.378021	0.538029	0.770065
10y Treasury Yield (%)	3.892956	1.472776	2.55	3.92	5.00
Gold Futures Price	826.512365	477.375280	347.525	767.350	1259.775
Oil Futures Price	54.818546	29.025235	28.3975	51.2050	76.1125
S&P 500 Trading Volume (in Billion)	2.660432	1.647126	1.20880	2.77301	3.75089

Table 3: Correlation Matrix of Macroeconomic Data

Correlation Coefficient	Exchange Rate Daily Volatility (USD/JPY)	10y Treasury Yield (%)	Gold Futures Price	Oil Futures Price	S&P 500 Trading Volume (in Billion)
Exchange Rate Daily Volatility (USD/JPY)	1.0000	0.1861	-0.2669	-0.2442	-0.0529
10y Treasury Yield (%)	0.1861	1.0000	-0.8709	-0.5964	-0.7127
Gold Futures Price	-0.2669	-0.8709	1.0000	0.7481	0.7285
Oil Futures Price	-0.2442	-0.5964	0.7481	1.0000	0.6650
S&P 500 Trading Volume (in Billion)	-0.0529	-0.7127	0.7285	0.6650	1.0000

Table 4: Unit Root Test of log S&P500 Index and Macroeconomic data, Augmented Dickey-Fuller (ADF) test statistics, and the p-values

	Log returns S&P 500 Index	Exchange Rate Daily Volatility (USD/JPY)	10y Treasury Yield (%)	Gold Futures Price	Oil Futures Price	S&P 500 Trading Volume (in Billion)
ADF Statistic	-13.74202	-6.463663	-1.426715	-0.416305	-1.836465	-2.13041
p-value	1.08988e-25	0	0.569404	0.907341	0.362514	0.232464

3 Methodology

3.1 Volatilites Dynamics

This paper aims to apply augmented GARCH model (With Macro-economic variables) to price SPX options and intends to capture the volatility smiles consistently observed in the market. Under the physical measure P, the augmented GARCH is:

$$\ln(S_{t+1}) - \ln(S_t) = \mu + \sigma_{t+1}\varepsilon^P_{t+1}, where\ \ \varepsilon_{t+1}|F_t \equiv W^P_{t+1} - W^P_t \sim N(0,1)$$

$$\sigma^2_{t+1} = \beta_0 + \beta_1\sigma^2_t + \beta_2\sigma^2_t(\varepsilon_t - \lambda)^2 + \sum_{k=1}^{n}\gamma_k X_{kt}$$

The stock price follows a Geometric Brownian motion.λ is a important parameter that capture a financial leverage effect so that when price falls too low, i.e.$\varepsilon_t < 0$, then $(\varepsilon_t - \lambda)^2$ becomes larger than when $\varepsilon_t > 0$ and hence next period σ^2_{t+1} is much larger. It matches the negative skewness of financial returns and improves the model's ability to explain volatility smiles. X_t represents Macro-economic variable that are included into GARCH.
It should be noted that the unconditionally variance without economic variables is given as

$$\sigma^2 = \beta_0 / \left[1 - \beta_1 - \beta_2\left(1 + \lambda^2\right)\right]$$

and becomes

$$\sigma^2 = (\beta_0 + \sum_{k=1}^{n}\gamma_k E[X_{kt}]) / \left[1 - \beta_1 - \beta_2\left(1 + \lambda^2\right)\right]$$

if any economic variables are included in the GARCH.

The estimation of augmented GARCH is to apply Maximum Likelihood methods to fit the S&P 500 index daily closing price from 1996 to 2019. The estimation utilizes python programming to maximize the log-likelihood function:

$$L(\theta) = -\frac{T}{2}\log(2\pi) - \frac{1}{2}\sum_{t=1}^{T}\log\left(\sigma^2_t\right) - \frac{1}{2}\sum_{t=1}^{T}\left(\ln\left(\frac{S_t}{S_{t-1}}\right) - \mu\right)^2 / \sigma^2_t$$

$$\theta = \arg\max L(\theta)_{\beta_0,\beta_1,\beta_2,\mu,\lambda,\gamma_1\ldots\gamma_k}$$

The parameters should satisfy the following constraints: $\beta_0, \beta_1, \beta_2 > 0$, $(\beta_0 + \sum_{k=1}^{n}\gamma_k E[X_{kt}]) > 0$,$\beta_1 + \beta_2(1 + \lambda^2) < 1$, in order to avoid Non-stationary in variance as well as to ensure a positive unconditional variance.

The economic variables are included one by one into GARCH model to investigate their individual impact on explaining smiles, compare with a simple GARCH model without any economic inputs. The Black Scholes model result, which is produced from Monte Carlo simulation when $\beta_1 = \beta_2 = 0$ in GARCH model, serves as a benchmark.

After the estimations of GARCH model, it is then applied in a risk neutral option pricing framework by Monte Carlo simulation. When there are options that could be hedged, GARCH model is presented under risk neutral Q measure:

$$\ln(S_{t+1}) - \ln(S_t) = \left(r - \tfrac{1}{2}\sigma^2_{t+1}\right) + \sigma_{t+1}\varepsilon^Q_{t+1}, \text{ where } \varepsilon_{t+1}|F_t \equiv W^Q_{t+1} - W^Q_t \sim N(0,1)$$
$$\sigma^2_{t+1} = \beta_0 + \beta_1\sigma^2_t + \beta_2\sigma^2_t\left(\varepsilon^Q_t - \lambda\right)^2 + \textstyle\sum_{k=1}^{n}\gamma_k X_{kt}$$

The GARCH model under Q measure is different from P measure in terms it is GARCH-in-mean form, but the parameters are the same estimates under P measure. This study focuses on out of sample option pricing and the GARCH-in-mean is used to simulate future volatility. For each pricing date T, a two-year rolling window (252 × 2) is used to on S&P 500 index to estimate GARCH parameters up to T. The Monte Carlo is then applied to simulate future paths with the estimated GARCH augmented by forward looking macro variables to price SPX index options up to 3 months maturity. Time interval is discredited into daily and 10000 simulation paths are generated for each pricing.

In this paper, Empirical martingale simulation (EMS) developed by Duan and Simonato (1998) is adopted as an effective control variate technique for Monte Carlo simulation in option pricing.The system is:

$$S_i^* \left(t_j, n\right) = S_0 \frac{Z_i\left(t_j, n\right)}{Z_0\left(t_j, n\right)},$$

$$where \;\; Z_i\left(t_j, n\right) = S_i^*\left(t_{j-1}, n\right) \frac{S_i\left(t_j\right)}{\delta_i\left(t_{j-1}\right)}, Z_0\left(t_j, n\right) = \frac{1}{n} e^{-rt_j} \sum_{i=1}^{n} Z_i\left(t_j, n\right)$$

Here $\mathrm{n}^{-1}\mathrm{e}^{-(ir)} \sum S_i^*$ converges to S_0 by construction so that Martingale property has been maintained.

3.2 Computational Schemes

The estimations of augmented GARCH and EMS option pricing are all conducted by using python programming. The self-constructed algorithms are implemented to tackle the unconventional GARCH forms and extra economic exogenous variables. Ensure a robust estimation of the augmented GARCH model is essential to produce meaningful on option pricing and smile explanations. While the simple maximum likelihood method has been used to fit the stock data, the estimation of more than 4 variables with different forms make the function high dimensional and non-linear. As a result, a careful design of python programming is required. For GARCH estimation, the steps are followings:

1. Create variance bounds. The straightforward way to estimate a maximum likelihood is direct use of Scipy.optimization. However, the optimizer will encounter negative and infinite values for the unbounded conditional variances during the process, which lead to optimization failure. Proper bounds curb the squared sigma so there will no error during optimization. The bounds based on a moving average of residuals, which multiplies 1e-6 and 1e6 respectively to establish lower and upper limit. The bounds themselves will follow an exponential weighted moving average (EWMA) and will be smoothed away periods by periods.

2. Give a stable initial $\sigma^2(0)$. The first input $\sigma^2(0)$ to start the GARCH estimation process will be a an EWMA of the squared residuals to certain legs. It serves as an approximation of unconditional variance before the unconditional variance can be calculated.

3. Construct negative log-likelihood function. The augmenter GARCH process is established here and stored in Numpy.array, which will be summed up and used to form the negative log-likelihood function. It will be input into Scipy.optimize.minimize for minimization.

4. Select starting values smartly. The selection of starting values sometimes will determine the success of likelihood algorithm, for the reason that multiple local optimas of maximum likelihood function generates difficulties for global optimise search. Thus certain initial guesses for all the variables are chosen (Those guesses for β_0 will be slightly lower than approximated unconditional variance, those for β_1 will be higher than 0.5 and β_2 will be lower than 0.2), which are then used to create random combinations. Those combinations' likelihood functions are then maximized one by one, from which the one gives the highest value will be a good set of initial guesses to start with. This automation process smoothies the optimization process by avoiding the manual inputs of starting values, especially for large scale estimation with automatically rolling window.

5. Implement the optimization algorithm. Compile all the components into optimizer and set essential constraints as well as bounds. The primary methods is 'SLQP' and 'Nelder-Mead', but the

complex nature of the function will raise the difficulties for successful minimization each iteration for a single optimization methods, thus a set of alternative methods will be substituted sequentially if initial method fail. Only when optimization exits successfully and all the parameters satisfy the constraints/bounds, the estimation results are rendered valid.

6. Estimate co-variance matrix. Use the function "approx fprime" from stats model to numerically approximate the Jacobian from log-likelihood function, after the parameters have been obtained from optimizer. The product of transpose of Jacobian and the Jacobian is then inversed, which will produce the BHHH estimator of co-variance matrix, from which the standard errors of estimators can all be obtained. This approach is easier to implement than the inverse of negative expected Hessian since BHHH will ensure the non-negative definite of matrix.

The Monte Carlo simulation process for option pricing is as following:

1. EMS future option pricing. In a rolling window context, (252×2) number of data-sets have been used to train the GARCH model, it is then to price the options 20 working days later(no change of parameters only during the 20 days) hence out of sample. Simulation is conducted at each valuation date, the 3 months maturity is sliced into each day and 10000 paths are created for each time slice. With the estimated Q measure GARCH-in-mean model, the expected future daily stock price will be simulated. For each time interval, a variance will be forecasted by the augmented GARCH model with a forward looking economic component. This variance will then be used to generate next stock price which will be corrected by EMS. At the end of maturity, the option price will be calculated by $max(S_t - K, 0)$ for European calls or $max(K - S_t, 0)$ for European puts, the prices of which will be averaged among 10000 simulation paths. They are then used to compare with market option prices with the same maturity and the implied volatility will be root searched to explain the smiles.

2. Data and error handling. We use European options on th SP 500 Index (symbol:SPX) with maturity of 3 month to test our model.Since out-of-money (OTM) options are more actively traded in-the-money options, so we only use OTM options to avoid the potential issues associated with liquidity problems.Based on literature research, we choose trading volume, US treasury 10-year yield,foreign exchangerate(Yen/US),foreign exchange rate daily volatility, crude oil futures, gold futures as macroeconomic variables included in our GARCH model. The risk-free rate is decompounded from 1 year US treasury rate. The procedure is repeated for each day in the sample. The sample period in our empirical analysis is from 2009 to 2019.To ensure the accuracy of the estimation, wediscard the invalid estimated parameters which are out of the rational boundary

3.3 Model Estimation and Robustness Tests

To validate the estimation methodology could produce robust results and is free from biases, the stability and robustness of the estimated model should be examined. The GARCH model exclusion of economic variables will be the test case here. In Table 1, The full data sample is divided into yearly and two-year's data is used as one sample, within which each year's estimation will be the sub-sample result for comparison. By comparisons, it can be noticed that:

1. The sub-samples' coefficients and unconditional variances are inline with large 2 year samples'. For 1996 to 1997 sample, the β_1 is in between Beta 1 in 1997 and that in 1996, as only β_1 in 1996 is significant, the two-year's sample coefficients are in the same magnitude with 1997's sample, including unconditional variance. There are also similar observations for 1998-1999 sample, 2006-2007 sample, 2008-2009 sample, 2009-2010 sample, 2011-2012 sample, 2012-2013 sample, 2013-2014 sample, 2015-2016 sample, 2016-2017 sample, 2017-2018 sample and 2018-2019 sample, in which coefficients are in the similar magnitude with both sub-sample or in between if the sub-samples have significant coefficients but differ relatively large in magnitudes. If any sub sample has insignificant coefficients, the sample's corresponding parameter will tend to be close with the sub-sample that has the significant coefficients.

2. The two-year samples' unconditional variances are consistent with its sub-sample throughout the whole period if conduct a head to toe comparison. They are the similar magnitude until 2003, from which the stationary decrease, then increase back from 2007. There are also very low unconditional variances from 2012 to 2014, during which full sample and its sub-samples still reveal the same picture. The noticeable high unconditional variance is in 2015 and it differs dramatically from the 2014-2015 or 2015-2016 full sample, but since the β_0, β_2 and λ are not significant in 2015, the high unconditional variance is rendered invalid. Another one is in 2018, it could be attributed to a precision problem since the denominator for unconditional variances will be so small because of the high λ.
3. The unconditional variances can be clustered periodically which is consistent with market observations. The model will be more statistically significant if increase sample size.

The full samples and sub-samples validation indicates there is no noticeable biases in estimations that will be sensitive to the sample changes. All the parameters including stationary variances are intuitive and relatively stable.

Follow the principle of robustness tests by Young and Holsteen (2017), the parameter robustness summary is presented in Table 2, using the same data in Table 1.

1. Compare the statistics between the full sample and sub-samples side by side, all the parameters have close means and standard deviations, except for unconditional variances. However, as mentioned above, there are outlines in individual sub-sample, if eliminate, could produce a mean 0.00069825 with a standard error 0.00159687 for unconditional variance in sub-sample.
2. If by analogy to standard t-statistic, a robustness ratio above 2 may indicate robustness in terms of parameter stability. In sub-sample β_1 is very robust, β_2 and λ are acceptable but can increase robustness if in full sample when size is doubled. However, β_0 is less stable, despite its significance number in full sample. Since volatility is not homoskedastic, which can seen in the high variations of β_0, the instability of β_0 shows the base level of volatility (When$\beta_1 = \beta_2 = 0$) will changes all the times, which in turn explain the variations of unconditional variances.
3. Almost all the β_1 are significant and many β_2 and λ are also significant, those facts reveal that conditional heteroskedasticity of S&P 500 log returns do exists. Combine the results from Table 1, most μ are close to 0 and most λ are close to 1 (starting value of λ for optimization is very low), it may indicate the fact that log returns are drift-less and leverage effect is quite strong for the sample period.

Since the major topic of this paper is to examine whether the macro-economic factors will contribute to superior option pricing, it is also of interest to include another economic variable to test its impact on GARCH. In Table 3, there is statistics summary which compare GARCH estimations differences with and without trading volume, using rolling window to expand the estimation numbers. Figure 3 and Figure 4 presents the parameters distribution of the two types of GARCH.

1. The GARCH without economic variables have the similar statistics with that in Table 2, after a larger scale estimations. However, when trading volume is introduced, most parameters are not statistically significant anymore except β_1, which implies there is only volatility clustering effect left. More μ becomes significant but the magnitude is still very close to 0 judging its mean. The average λ has been brought down resulting less leverage effects. The robustness ratio shows very stable β_1 but high variations for all the other parameters.
2. In figure 3, it could be noticed some variations for β_1, although most are above 0.8. Relatively large variations can be noticed in β_2 distribution, most values are within 0-0.15 range. All the other parameters are very clustering. By contrast, in figure 4, all the parameters except β_1 are clustering around 0, indicating their insignificance's in explaining volatility, which is the consequence of introducing trading volume.

In conclusion, the validation and robustness test results shrink possibilities of the model and algorithm defects to a certain extent, there is no evidence of systematic bias. On one hand, the augmented

GARCH without economic variable is relatively robust; On the hand, introducing economic variables could weaken the robustness of model. Whether introduce economic variables will be incremental or detrimental on augmented GARCH option pricing will be investigated in details in next section.

4 Data Comparison

Table 5: GARCH estimation results with full samples (2-year's index prices) VS sub-samples (1 year's index prices)

	Beta0	Beta1	Beta2	Mu	Lambda	Unconditional Variance		Beta0	Beta1	Beta2	Mu	Lambda	Unconditional Variance
1996-1997***							1996*						
Coef	5.74944E-06	0.760780128	0.095533851	0.000717859	0.999997311	0.000119400	Coef	1.43E-08	0.997371923	0.000299312	0.000820471	0.941956194	6.911039e-06
P-Value	0.002858894	2.59146E-64	9.53749E-07	0.075736834	0.000338679		P-Value	0.980056057	0	0.976362468	0.10339047	0.985221153	
1997-1998***							1997***						
Coef	9.73617E-06	0.705664745	0.120835519	0.000972932	0.99990239	0.00018479	Coef	2.78595E-05	0.488843456	0.161217123	0.000940679	0.999998538	0.000147621
P-Value	3.35577E-05	7.11831E-52	0.000188606	0.038544767	0.00167196		P-Value	0.052151638	0.003530628	0.012204105	0.170479912	0.02574798	
1998-1999***							1998***						
Coef	6.12758E-06	0.790235551	0.087126578	0.00073246	0.99937259	0.000172024	Coef	6.57772E-06	0.743964948	0.111870584	0.001000383	0.999999936	0.000203683
P-Value	0.004914487	5.62588E-65	0.006603614	0.132214828	0.017035462		P-Value	0.010040969	1.37947E-32	0.011828035	0.130613815	0.023027948	
1999-2000***							1999						
Coef	1.25026E-05	0.732328806	0.102142569	4.05968E-06	0.999500353	0.000196929	Coef	0.00012069	0.069232261	0.000711471	0.000695421	0.000706724	0.000129767
P-Value	0.001133535	9.44393E-38	0.005648702	0.994291573	0.009303542		P-Value	0.942481033	0.995711734	0.993830103	0.345852631	0.999992702	
2000-2001***							2000*						
Coef	3.10218E-06	0.803672223	0.091660297	0	1	0.000238497	Coef	0.00014222	0.139949577	0.139949181	0	6.38385E-07	0.000197501
P-Value	0.221079542	2.9223E-122	0.000295764	1	0.001378072		P-Value	0.121461792	0.769736547	0.090317543	1	0.999998284	
2001-2002***							2001*						
Coef	3.15131E-06	0.846513937	0.0676536	1.31381E-10	0.999999972	0.00017335	Coef	4.85971E-06	0.831703419	0.067728341	1.72216E-10	0.999999915	0.000147982
P-Value	0.160500284	1.3577E-177	0.037053236	0.999999826	0.042384456		P-Value	0.292041447	8.14636E-80	0.120487209	0.99999983	0.135005269	
2002-2003***							2002*						
Coef	1.20636E-06	0.897830815	0.047588828	6.82446E-05	0.999852527	0.0001722	Coef	2.68413E-06	0.858163213	0.064360183	3.59157E-10	0.999999766	0.000204638
P-Value	0.23103975	2.9351E-245	0.048825692	0.891766849	0.034800961		P-Value	0.435342045	2.91293E-62	0.245567872	0.999999699	0.242650361	
2003-2004							2003*						
Coef	7.54613E-05	0.058718374	0.000599812	0.00058008	0.000599178	8.02E-05	Coef	1.95322E-12	0.967797455	0.012988009	0.000900803	0.999997068	3.14E-10
P-Value	0.520059585	0.967787321	0.983241389	0.156283178	0.999989843		P-Value	0.999997537	2.6681E-247	0.608868033	0.135553671	0.655245076	
2004-2005*							2004*						
Coef	4.01034E-07	0.980033766	0.010519961	0.000240021	0.010489446	4.25E-05	Coef	3.37743E-06	0.869644263	0.031088229	0.00030278	0.994031368	4.93E-05
P-Value	0.706133139	7.7053E-233	0.415055884	0.429138087	0.991796991		P-Value	0.477913364	3.79428E-12	0.469742688	0.495386303	0.510994069	
2005-2006**							2005*						
Coef	1.90689E-06	0.836285507	0.060523822	0.000251597	0.997388285	4.44E-05	Coef	2.17383E-06	0.835759254	0.058453608	1.34454E-06	0.998678119	4.58E-05
P-Value	0.05696886	2.74616E-49	0.111196449	0.377734077	0.163724074		P-Value	0.407960029	1.8309E-16	0.333102009	0.997387207	0.38535832	
2006-2007***							2006*						
Coef	2.09674E-06	0.839856869	0.06946085	0.000232078	0.997508396	9.72E-05	Coef	1.5745E-06	0.849579181	0.0561289	0.000421584	0.995602826	4.07E-05
P-Value	0.001824568	7.8088E-104	0.00901897	0.525241851	0.027152625		P-Value	0.134508059	1.79317E-27	0.250828323	0.282324694	0.359979976	
2007-2008***							2007**						
Coef	4.1522E-06	0.815556236	0.085901673	0	1	0.000328486	Coef	4.08408E-06	0.834742194	0.068634565	9.37998E-05	0.999815635	0.000145787
P-Value	0.000251363	9.40684E-74	0.001401685	1	0.014522761		P-Value	0.04534022	1.78396E-40	0.114940314	0.893698677	0.229613053	
2008-2009***							2008**						
Coef	3.54316E-06	0.84558587	0.073591938	2.2013E-09	0.999999659	0.000490042	Coef	7.72397E-06	0.857393025	0.134097673	0	0.008741923	0.000908803
P-Value	0.026568481	1.8089E-127	0.000297238	0.999997379	0.005106255		P-Value	0.179380215	2.35596E-51	0.018443029	1	0.973755962	
2009-2010***							2009**						
Coef	3.24354E-06	0.820402665	0.083150554	0.000563245	0.999957101	0.000243814	Coef	1.71464E-07	0.955269848	0.035330277	0.00121556	0.068364389	1.86E-05
P-Value	0.000145577	5.2617E-124	0.000758194	0.245388352	0.005878993		P-Value	0.881146781	0	0.071395825	0.16749943	0.888008131	
2010-2011***							2010***						
Coef	5.0731E-06	0.775487805	0.101341559	0.000221361	0.999999723	0.0002324	Coef	5.49512E-06	0.739282338	0.112311904	0.00073159	0.999999768	0.000152245
P-Value	1.69785E-05	2.18387E-98	0.000251742	0.633721245	0.002555448		P-Value	0.001392537	3.5686E-26	0.038346548	0.236388531	0.111820559	
2011-2012***							2011***						
Coef	3.64314E-06	0.797416907	0.091528263	5.48647E-06	0.999912746	0.000186421	Coef	4.52024E-06	0.810068335	0.089870519	3.5024E-06	0.979084995	0.00032495
P-Value	0.000816527	3.2369E-125	0.001033698	0.989155743	0.003181435		P-Value	0.011939588	5.00768E-80	0.024385429	0.996488886	0.061297948	
2012-2013***							2012**						
Coef	1.01601E-06	0.979626754	0.004077258	0.000821143	0.004131099	6.23E-05	Coef	2.69596E-06	0.839804971	0.061895121	0.000508245	0.99871368	7.37E-05
P-Value	0.05718398	0	0.465673283	0.018567176	0.99821986		P-Value	0.070291602	4.45821E-38	0.172697445	0.299301884	0.155089845	
2013-2014***							2013*						
Coef	1.10396E-05	0.524379966	0.243757592	0.000757055	0.241557897	5.07E-05	Coef	3.93403E-05	0.98004262	0.0100257	0.010062922	0.0099543	0.003961489
P-Value	0.004178955	2.28E-05	0.001046155	0.013746147	0.124494964		P-Value	0.462318598	4.39286E-35	0.987749627	0.090206076	0.999717158	
2014-2015**							2014***						
Coef	1.55457E-07	0.979789094	0.0199681	0.000250706	0.020058839	0.000662163	Coef	3.25991E-06	0.604211299	0.196548795	0.000401166	0.99992988	0.00119908
P-Value	0.688453555	0	0.001776678	0.525663155	0.965430884		P-Value	0.028155892	3.59395E-17	0.005071937	0.34043892	0.002361851	
2015-2016**							2015**						
Coef	7.38509E-05	0.976710338	0.009906645	0.005103939	0.134915307	0.005593623	Coef	5.8111773e-05	0.807861645	0.191725163	0.0096576355	0.0209218265	0.176487471
P-Value	0.047880709	1.67785E-92	0.95914847	0.505159926	0.991911879		P-Value	0.65489919602	0.0014633045	0.5772008004	0.00013469103	0.98557600	
2016-2017***							2016*						
Coef	2.28149E-07	0.979888688	0.006840088	0.000642424	0.006914721	1.72E-05	Coef	6.68834E-05	0.979868304	0.009921356	0.005844448	0.00857681	0.006551024
P-Value	1.34056E-07	0	0.156547595	0.008892491	0.995087984		P-Value	0.611191526	2.14363E-58	0.985551216	0.466431871	0.999680544	
2017-2018***							2017**						
Coef	1.82827E-06	0.69280081	0.152163777	0.000487684	0.992858166	0.000362945	Coef	3.0361E-07	0.983540434	3.56826E-05	0.000786198	1.52124E-08	1.85E-05
P-Value	8.35157E-06	9.3124E-216	1.2871E-07	0.058342684	2.4845E-09		P-Value	0.530063702	1.0368E-223	0.997464454	0.003348747	1	
2018-2019***							2018***						
Coef	3.43431E-06	0.694685042	0.14770953	0.000317076	0.998965998	0.000336658	Coef	3.22101E-06	0.70021257	0.149820569	0.000165387	0.999988762	0.021522209
P-Value	4.74025E-05	3.5885E-164	0.000132785	0.433764022	9.47455E-06		P-Value	0.006935272	3.6519E-131	0.000852828	0.757451291	0.000304341	
							2019***						
							Coef	8.4156E-06	0.47658011	0.284119254	0.00072198	0.796017851	0.000141987
							P-Value	0.075744074	0.002145142	0.027847338	0.24267873	0.001925947	

Note: ***, **, *, indicate more than 2 variables, only 2 variables, only 1 variable are statistically significant at least 10% significance level respectively.

Table 6: Parameter robustness of GARCH estimations exclusion of economic variables (2 year sample VS 1 year sub-sample)

Sample (2 years)						
Parameters	Beta0	Beta1	Beta2	Mu	Lambda	Unconditional Variance
Mean	1.011519824e-05	0.788445691	0.077112289	0.000563889	0.71321138	0.00043862037
Std.E	2.017686083e-05	0.190955808	0.056003002	0.001014649	0.43480015	0.00110931343
Robustness ratio	0.501326659	4.128943218	1.376931357	0.555747560	1.64032200	0.39539805482
Significance out of 23 sample	17	22	17	6	15	
Sub-sample (1 year)						
Parameters	Beta0	Beta1	Beta2	Mu	Lambda	Unconditional Variance
Mean	1.9340437306e-05	0.746730321	0.087631455	0.001067427	0.700006623	1.491778937
Std.E	3.7062980557e-05	0.245961417	0.076669509	0.002196637	0.442427144	7.146775213
Robustness ratio	0.5218262809	3.035965265	1.142976604	0.485937137	1.582196372	0.208734554
Significance out of 24 sample	10	22	11	2	7	

Note 1:Signifiance means parameter's P-Value is less than 10%
Note 2:If adjust outliers, sub-sample unconditional will has a mean 0.00069825 and a standard error 0.00159687

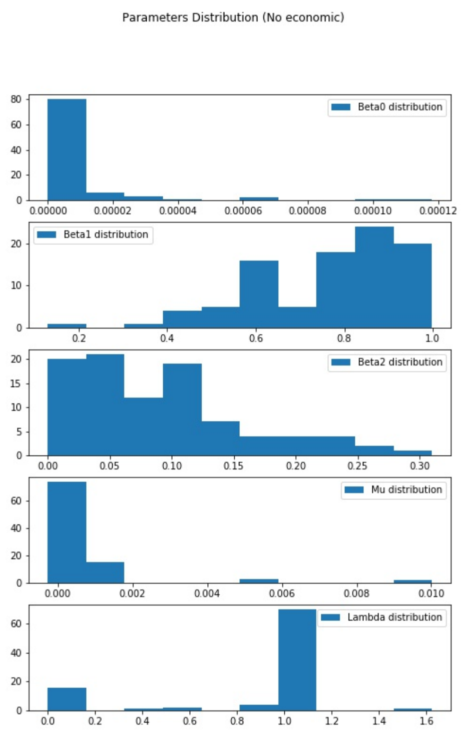

Figure 3: params_dist

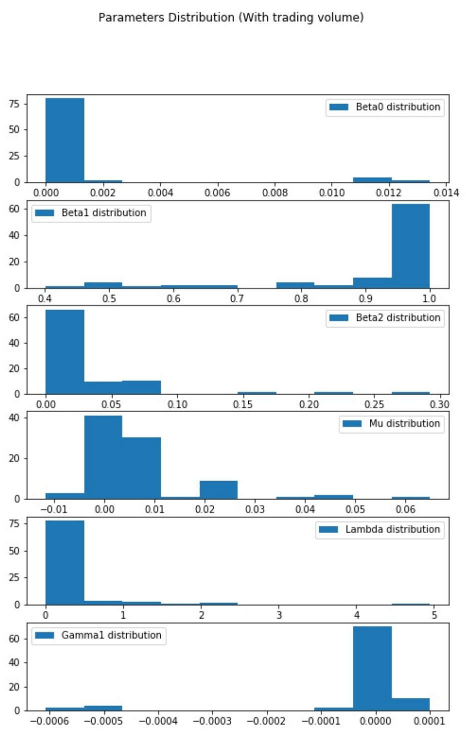

Figure 4: params_vol_dist

5 Empirical Analysis

The objective of our empirical analysis is to validate our pricing model in aspects of parameter analysis, pricing error measurement and implied volatility surface fitting.

5.1 Model performance measurement measure

In the empirical analysis, we compare our GARCH pricing model with Black-Scholes model. The assumed constant volatility we plugged into the model is the intercept term denoted as beta0 in the GARCH model.

We divide the option data into several categories according to the ratio of strike price over the asset price K/S. A put option is said to be deep out-of-money if its moneyness $K/S < 0.85$, and out-of-money if $0.85 \leq K/S < 1$. A call option is said to be out-of-money if $1 \leq K/S < 1.15$, and deep out-of-money if $K/S \geq 1.15$.

The pricing error is measured by calculating the root mean squared error $\text{RMS} = \sqrt{\frac{1}{m}\sum_{i=1}^{m} e_i^2}$. The absolute pricing error presented across different moneyness is computed by:

$$|\text{model price} - \text{market price}| / \text{market price}$$

5.2 Out-of-sample model comparison results

5.2.1 Performance of GARCH pricing model without macroeconomic effects

The below tables and graph show the overall results of the performance of GARCH pricing model without macroeconomic effects.

The statistic summary for the pricing GARCH parameters calibrated across the whole sample period. The low standard deviations and the high percentage of statistically significant parameter estimates proof that the implied volatility smile is a persistent phenomenon in the SPX option market.

From the average pricing error RMS calculation, we can see that the GARCH model outperforms Black scholes model when it price out-of-money put option. The percentage of the case that GARCH model has lower RMS than Black-Scholes model is 45%.

From the observation of option price estimation and implied volatility plotted across the moneyness, the GARCH model tends to overestimate the price of deep out-of-money put and call option, so it can capture the smile of implied volatility but can not exactly capture the skewness of the curve.

Table 7: Summary statistics of parameter estimates without macroeconomic effects

	Mean	SD	Percentage of P-value <0.05
beta0	4.09E-06	2.19E-06	0.92
beta1	0.734834	0.139372	1
beta2	0.094619	0.073631	0.95
mu	0.000171	0.000459	0.08
lambda	1.179225	0.709307	0.77

Table 8: Average pricing error RMS across moneyness without macroeconomic effects

Moneyness	Garch	BS
$K/S<0.85$	6.87	4.42
$0.85 \leq K/S<1$	19.4	29.58
$1 \leq K/S<1.15$	23.07	20.15
$K/S \geq 1.15$	5.55	0.55

Table 9: Average pricing error RMS across years without macroeconomic effects

Year	Moneyness	Garch	BS
2009∼2011	K/S<0.85	9.01	4.78
	0.85≤K/S<1	25.71	28.9
	1≤K/S<1.15	29.66	19.95
	K/S≥1.15	10.56	0.66
2011∼2013	K/S<0.85	3.16	2.32
	0.85≤K/S<1	10.35	24.48
	1≤K/S<1.15	12.81	15.51
	K/S≥1.15	0.63	0.26
2013∼2015	K/S<0.85	1.92	3.29
	0.85≤K/S<1	5.98	26.29
	1≤K/S<1.15	10.23	14.71
	K/S≥1.15	0.97	0.15
2015∼2017	K/S<0.85	11.64	3.02
	0.85≤K/S<1	35.87	23.05
	1≤K/S<1.15	40.62	14.09
	K/S≥1.15	1.08	0.11
2017∼2019	K/S<0.85	7.64	6.36
	0.85≤K/S<1	16.67	40.19
	1≤K/S<1.15	18.91	28.35
	K/S≥1.15	0.35	0.42

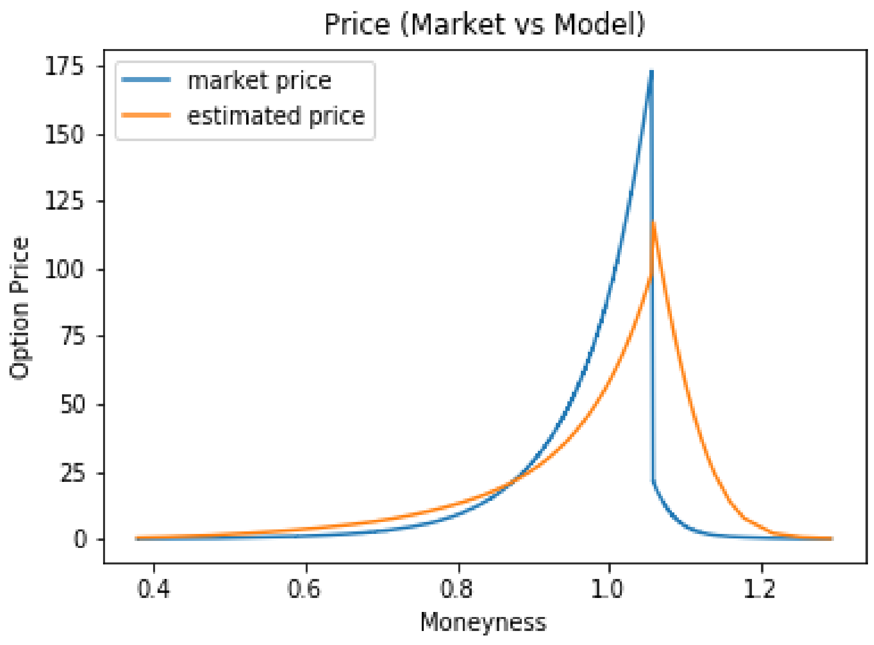

Figure 5: Option pricing without macroeconomic effects

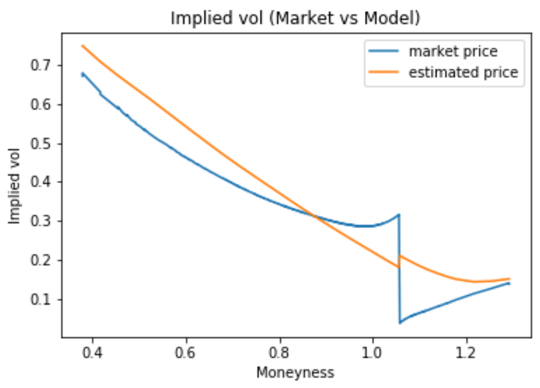

Figure 6: Implied volatility analysis without macroeconomic effects

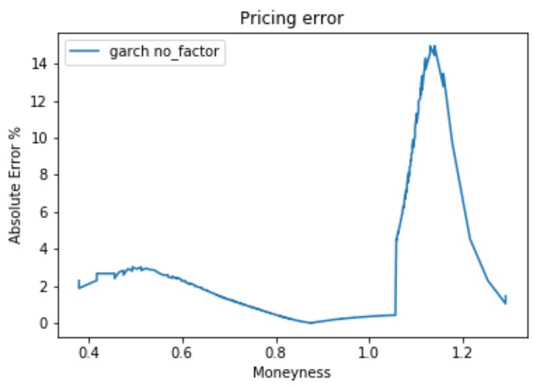

Figure 7: Pricing error analysis without macroeconomic effects

5.2.2 Performance of GARCH pricing model with macroeconomic effects

This section reports the performance of all the GARCH pricing model with different macroeconomic variables.

Through the comparison, the overall conclusion is that, in terms of parameter analysis, the parameter denoted as beta 1 in the GARCH model is significant in each model, further indicating that the conditional volatility process in the option market. The parameter gamma for the macroeconomic variable is rarely significant, so adding this additional variable may not helps a lot in option pricing and implied volatility fitting. However, the significance of other parameters is not such stable.

As for the pricing error, GARCH pricing model can beat the benchmark Black Scholes model generally if the option is not deep out-the-money. And compared with other economic variables, the GARCH pricing model with foreign exchange rate performs the best in views of the highest percentage of RMS smaller than Black-Scholes model across years.

Addionally, only when pricing with trading volume and US treasury the volatility smile can be captured fairly well, the ability of variable foreign exchange rate and foreign exchange daily volatility to explain vilatility smile is weaker because they fail to capture smile if the option is deep out-the-money. Pricing with crude oil futres and gold futures as economic variable totally fail to capture the volatility smile.

The detailed model performance results of each macroeconomic variable are shown as follows.

1. Trading volume

Table 10: Summary statistics of parameter estimates with trading volume

	Mean	SD	Percentage of P-value <0.05
beta0	4.445E-05	7.77E-05	0.04
beta1	0.7640191	0.20437	0.96
beta2	0.1294229	0.103591	0.87
mu	0.0005051	0.000293	0.35
lambda	0.3679165	0.458223	0.39
gamma	-1.821E-06	3.49E-06	0.04

Table 11: Average pricing error RMS across moneyness with trading volume

Moneyness	Garch	BS
$K/S<0.85$	8.13	4.17
$0.85\leq K/S<1$	22.49	25.05
$1\leq K/S<1.15$	23.67	20.46
$K/S\geq 1.15$	11.29	3.46

Table 12: Average pricing error RMS across years with trading volume

Year	Moneyness	Garch	BS
2009∼2011	$K/S<0.85$	13.18	4.54
	$0.85\leq K/S<1$	33.62	20.67
	$1\leq K/S<1.15$	36.04	16.46
	$K/S\geq 1.15$	20.55	3.36
2011∼2013	$K/S<0.85$	1.28	1.82
	$0.85\leq K/S<1$	3.52	21.72
	$1\leq K/S<1.15$	13.11	13.03
	$K/S\geq 1.15$	1.92	0.15
2013∼2015	$K/S<0.85$	4.6	4.71
	$0.85\leq K/S<1$	15.16	29.65
	$1\leq K/S<1.15$	8.22	13.96
	$K/S\geq 1.15$	0.47	0.2
2015∼2017	$K/S<0.85$	5.855	3.8025
	$0.85\leq K/S<1$	16.2	25.7375
	$1\leq K/S<1.15$	16.985	21.0175
	$K/S\geq 1.15$	5.82	3.31
2017∼2019	$K/S<0.85$	4.36	4.14
	$0.85\leq K/S<1$	12.5	30.91
	$1\leq K/S<1.15$	10.57	40.62
	$K/S\geq 1.15$	0.34	9.53

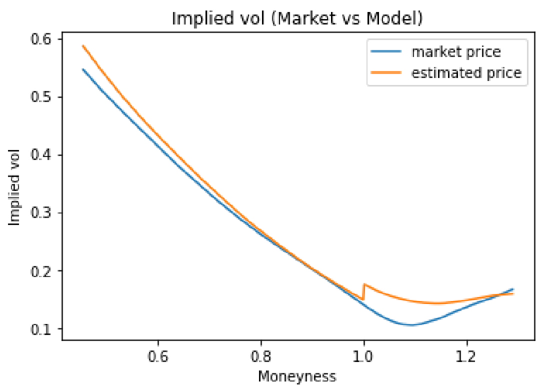

Figure 9: Implied volatility analysis with trading volume

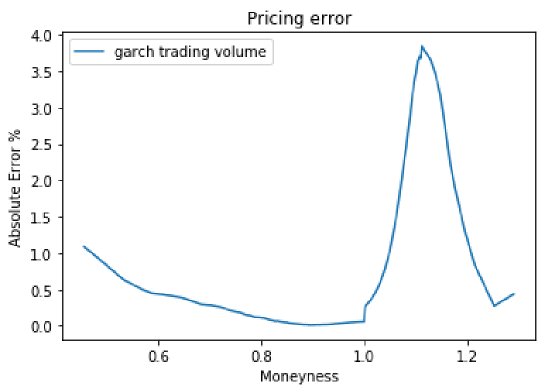

Figure 10: Pricing error analysis with trading volume

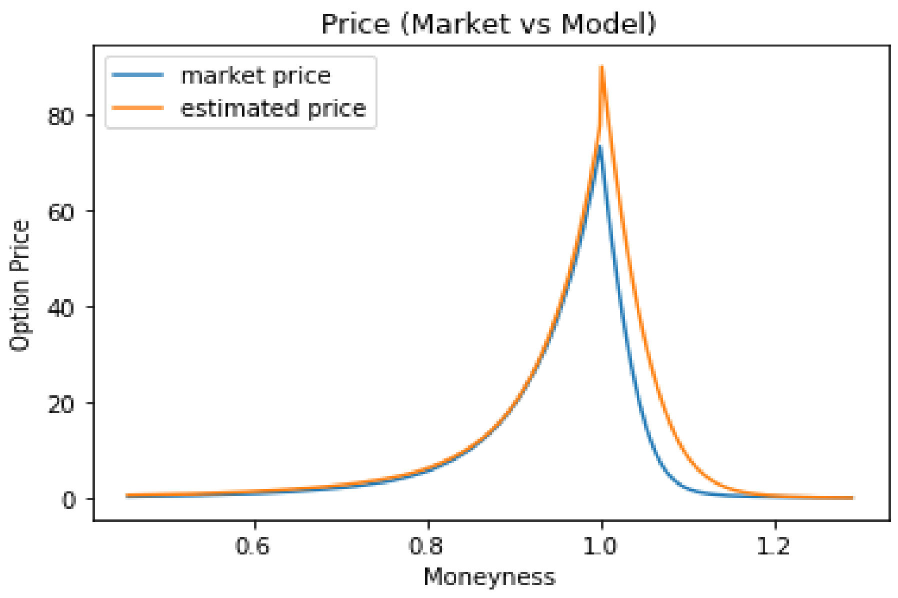

Figure 8: Option pricing with trading volume

2. US treasury 10 year yield

	Mean	SD	Percentage of P-value <0.05
beta0	1.12E-05	1.27E-05	0.53
beta1	0.673429	0.15442	1
beta2	0.12066	0.071202	0.94
mu	0.000234	0.00048	0.21
lambda	1.114548	0.632589	0.8
gamma	-0.00019	0.000271	0.17

Table 13: Summary statistics of parameter estimates with US treasury 10 year yield

Moneyness	Garch	BS
K/S<0.85	5.17	3.96
0.85≤ K/S<1	16.14	26.47
1≤K/S<1.15	20.12	15.58
K/S≥1.15	2.98	0.43

Table 14: Average pricing error RMS across moneyness with US treasury 10 year yield

Table 15: Average pricing error RMS across years with US treasury 10 year yield

Year	Moneyness	Garch	BS
2009∼2011	K/S<0.85	4.38	4.06
	0.85≤K/S<1	15.14	25.58
	1≤K/S<1.15	17.57	15.04
	K/S≥1.15	3.75	0.61
2011∼2013	K/S<0.85	3.5	2.49
	0.85≤K/S<1	11.72	24.27
	1≤K/S<1.15	18.37	14.81
	K/S≥1.15	3.02	0.3
2013∼2015	K/S<0.85	1.47	2.82
	0.85≤K/S<1	6.44	24
	1≤K/S<1.15	11.76	12.69
	K/S≥1.15	0.14	0.2
2015∼2017	K/S<0.85	10.62	2.49
	0.85≤K/S<1	32.14	20.73
	1≤K/S<1.15	39.64	9.76
	K/S≥1.15	5.86	0.15
2017∼2019	K/S<0.85	3.79	7.19
	0.85≤K/S<1	10.93	38.81
	1≤K/S<1.15	9.36	24.16
	K/S≥1.15	0.46	0.42

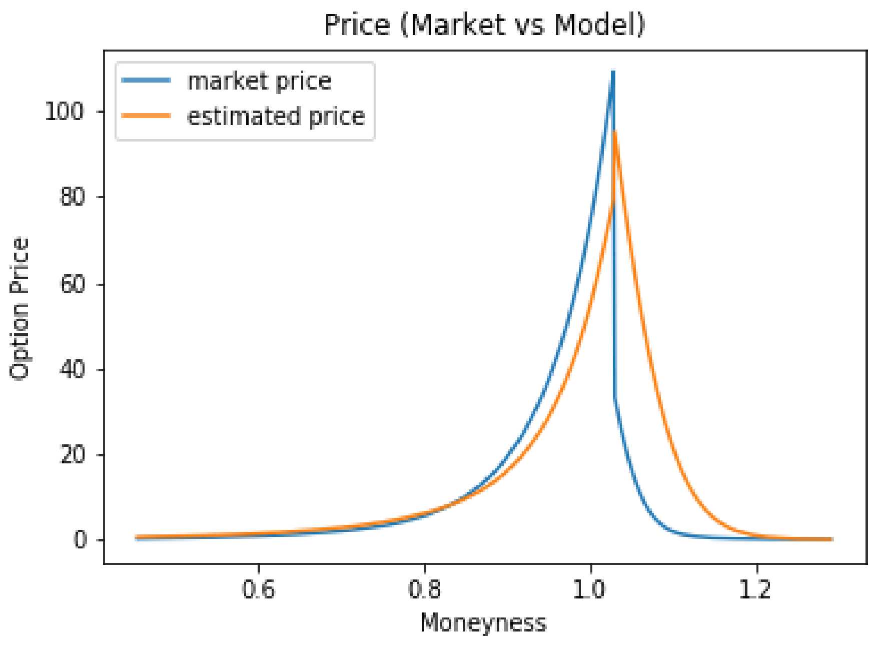

Figure 11: Option pricing with US treasury 10 year yield

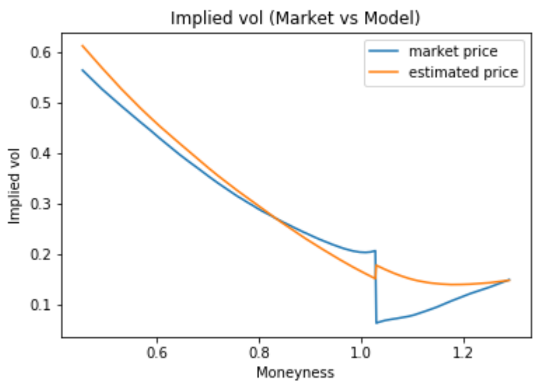

Figure 12: Implied volatility analysis with US treasury 10 year yield

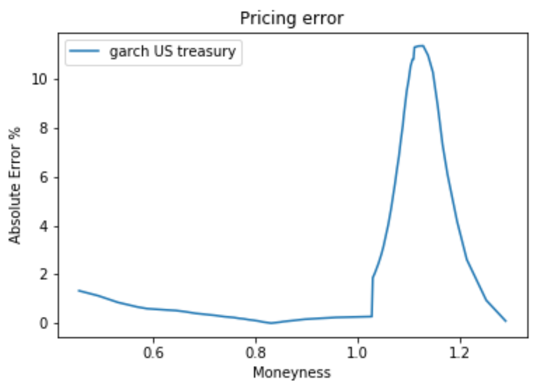

Figure 13: Pricing error analysis with US treasury 10 year yield

3. Foreign exchange rate (Yen/US)

Table 16: Summary statistics of parameter estimates with Foreign exchange rate (Yen/US)

	Mean	SD	Percentage of P-value <0.05
beta0	4.34E-06	6.3E-06	0.07
beta1	0.907996	0.081656	1
beta2	0.045321	0.034418	0.73
mu	0.000468	0.000544	0.27
lambda	0.068763	0.204851	0.07
gamma	-1.6E-08	5.51E-08	0.07

Table 17: Average pricing error RMS across moneyness with Foreign exchange rate (Yen/US)

Moneyness	Garch	BS
$K/S<0.85$	3.41	3.47
$0.85\leq K/S<1$	9.31	27.24
$1\leq K/S<1.15$	14.72	19.43
$K/S\geq 1.15$	3.64	0.45

Table 18: Average pricing error RMS across years with Foreign exchange rate (Yen/US)

Year	Moneyness	Garch	BS
2009∼2011	K/S<0.85	3.26	4.17
	0.85≤K/S<1	7.77	27.49
	1≤K/S<1.15	7.65	17.83
	K/S≥1.15	2.82	0.53
2011∼2013	K/S<0.85	1.77	1.94
	0.85≤K/S<1	4.14	24.55
	1≤K/S<1.15	10.88	18.5
	K/S≥1.15	0.63	0.23
2013∼2015	K/S<0.85	3.16	3.22
	0.85≤K/S<1	12.38	29.54
	1≤K/S<1.15	5.25	18
	K/S≥1.15	0.83	0.17
2015∼2017	K/S<0.85	2.81	3.2575
	0.85≤K/S<1	7.6	28.3325
	1≤K/S<1.15	11.8025	20.4425
	K/S≥1.15	1.44	0.28
2017∼2019	K/S<0.85	3.05	3.7
	0.85≤K/S<1	6.11	31.75
	1≤K/S<1.15	23.43	27.44
	K/S≥1.15	1.48	0.19

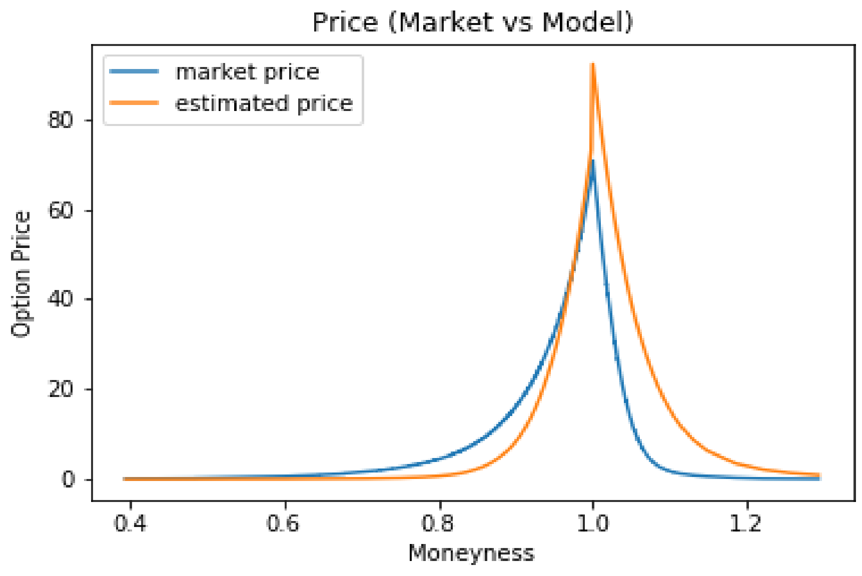

Figure 14: Option pricing with Foreign exchange rate (Yen/US)

4. Exchange rate daily volatility

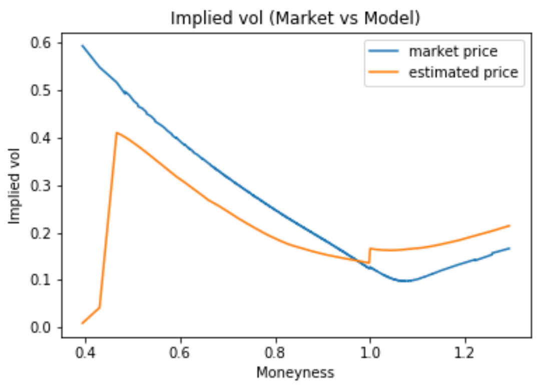

Figure 15: Implied volatility analysis with Foreign exchange rate (Yen/US)

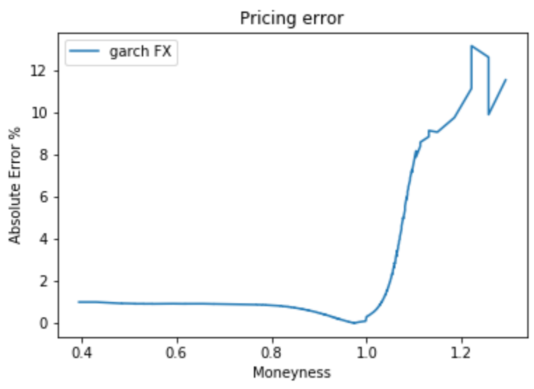

Figure 16: Pricing error analysis with Foreign exchange rate (Yen/US)

Table 19: Summary statistics of parameter estimates with exchange rate daily volatility

	Mean	SD	Percentage of P-value <0.05
beta0	2.26E-06	1.67E-06	0.13
beta1	0.679539	0.241	1
beta2	0.164783	0.122531	0.93
mu	0.000627	0.000227	0.4
lambda	0.420888	0.517223	0.47
gamma	9.73E-06	1.29E-05	0.4

Table 20: Average pricing error RMS across moneyness with exchange rate daily volatility

Moneyness	Garch	BS
K/S<0.85	5.18	4.47
0.85≤K/S<1	14	30.72
1≤K/S<1.15	14.8	21.09
K/S≥1.15	2.07	0.44

Table 21: Average pricing error RMS across years with exchange rate daily volatility

Year	Moneyness	Garch	BS
2011∼2013	K/S<0.85	1.87	2.65
	0.85≤K/S<1	4.93	26.11
	1≤K/S<1.15	12.48	17.1
	K/S≥1.15	2.55	0.18
2013∼2015	K/S<0.85	5.56	2.71
	0.85≤K/S<1	16.84	24.39
	1≤K/S<1.15	20.57	14.79
	K/S≥1.15	3.21	0.16
2015∼2017	K/S<0.85	4.483333	3.48
	0.85≤K/S<1	13.45667	28.88
	1≤K/S<1.15	15.46667	19.53333
	K/S≥1.15	1.98	0.213333
2017∼2019	K/S<0.85	6.02	5.08
	0.85≤K/S<1	18.6	36.14
	1≤K/S<1.15	13.35	26.71
	K/S≥1.15	0.18	0.3

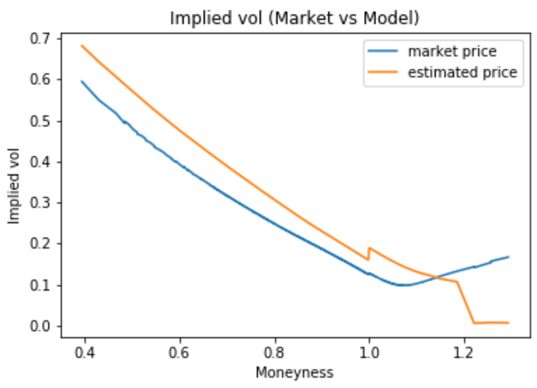

Figure 18: Implied volatility analysis with exchange rate daily volatility

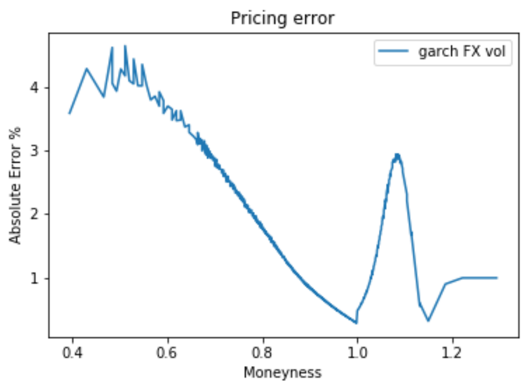

Figure 19: Pricing error analysis with exchange rate daily volatility

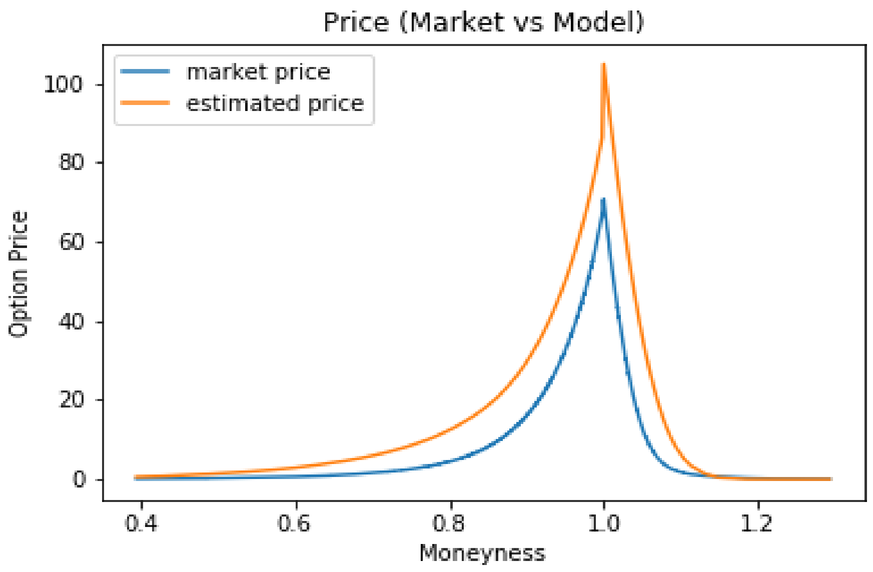

Figure 17: Option pricing with exchange rate daily volatility

5. Crude oil (West Texas Intermediate) futures

Table 22: Summary statistics of parameter estimates with Crude oil (WTI) futures

	Mean	SD	Percentage of P-value <0.05
beta0	9.18E-06	1.36E-05	0.36
beta1	0.871485	0.144015	1
beta2	0.066129	0.073915	0.68
mu	0.000532	0.000193	0.27
lambda	0.143294	0.380773	0.05
gamma	-7.6E-08	1.27E-07	0.23

Table 23: Average pricing error RMS across moneyness with Crude oil (WTI) futures

Moneyness	Garch	BS
K/S<0.85	2.55	2.69
0.85≤K/S<1	8.61	22.17
1≤K/S<1.15	10.21	12.16
K/S≥1.15	0.63	0.18

Table 24: Average pricing error RMS across years with Crude oil (WTI) futures

Year	Moneyness	Garch	BS
2011∼2013	K/S<0.85	1.7	1.82
	0.85≤K/S<1	5.43	23.1
	1≤K/S<1.15	7	14.6
	K/S≥1.15	0.48	0.15
2013∼2015	K/S<0.85	2.44	2.48
	0.85≤K/S<1	9.83	23.55
	1≤K/S<1.15	8.87	14.24
	K/S≥1.15	0.3	0.16
2015∼2017	K/S<0.85	3.32	3.39
	0.85≤K/S<1	9.88	20.28
	1≤K/S<1.15	12.45	8.42
	K/S≥1.15	0.24	0.12

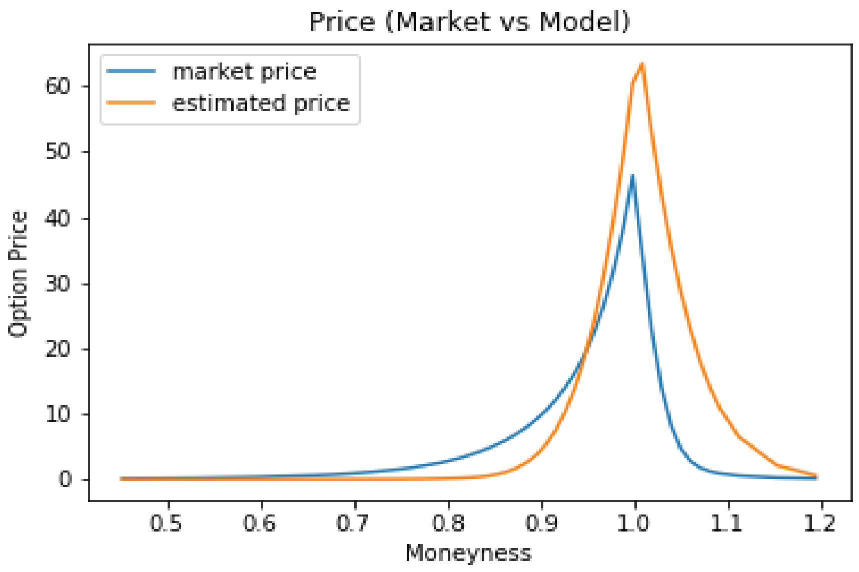

Figure 20: Option pricing with Crude oil (WTI) futures

6. Gold futures

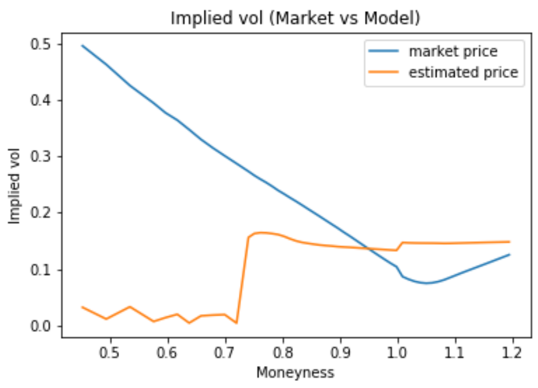

Figure 21: Implied volatility analysis with Crude oil (WTI) futures

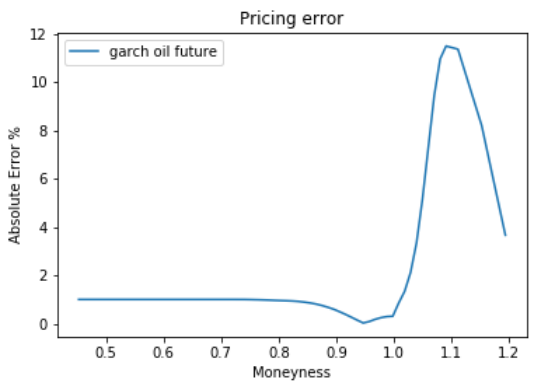

Figure 22: Pricing error analysis with Crude oil (WTI) futures

Table 25: Summary statistics of parameter estimates with Gold futures

	Mean	SD	Percentage of P-value <0.05
beta0	4.34E-06	6.17E-06	0.53
beta1	0.937937	0.045472	1
beta2	0.020904	0.030471	0.5
mu	0.000577	0.000399	0.31
lambda	0.044467	0.084451	0
gamma	-1.7E-09	7.14E-09	0.28

Table 26: Average pricing error RMS across moneyness with Gold futures

Moneyness	Garch	BS
K/S<0.85	12.95	4.47
0.85≤ K/S<1	29.53	29.9
1≤K/S<1.15	34.81	21.94
K/S≥1.15	15.73	0.72

Table 27: Average pricing error RMS across years with Gold futures

Year	Moneyness	Garch	BS
2009∼2011	K/S<0.85	4.55	5.55
	0.85≤K/S<1	17.72	33.63
	1≤K/S<1.15	15.68	23.63
	K/S≥1.15	3.92	1.02
2011∼2013	K/S<0.85	0.85	1.57
	0.85≤K/S<1	10.83	18.55
	1≤K/S<1.15	24.48	13.39
	K/S≥1.15	2.29	0.09
2013∼2015	K/S<0.85	3.32	3.4
	0.85≤K/S<1	10.77	29.1
	1≤K/S<1.15	9.74	20.17
	K/S≥1.15	0.38	0.18
2015∼2017	K/S<0.85	3.1675	3.645
	0.85≤K/S<1	12.3425	28.2125
	1≤K/S<1.15	16.02	20.82
	K/S≥1.15	1.7975	0.415
2017∼2019	K/S<0.85	3.95	4.06
	0.85≤K/S<1	10.05	31.57
	1≤K/S<1.15	14.18	26.09
	K/S≥1.15	0.6	0.37

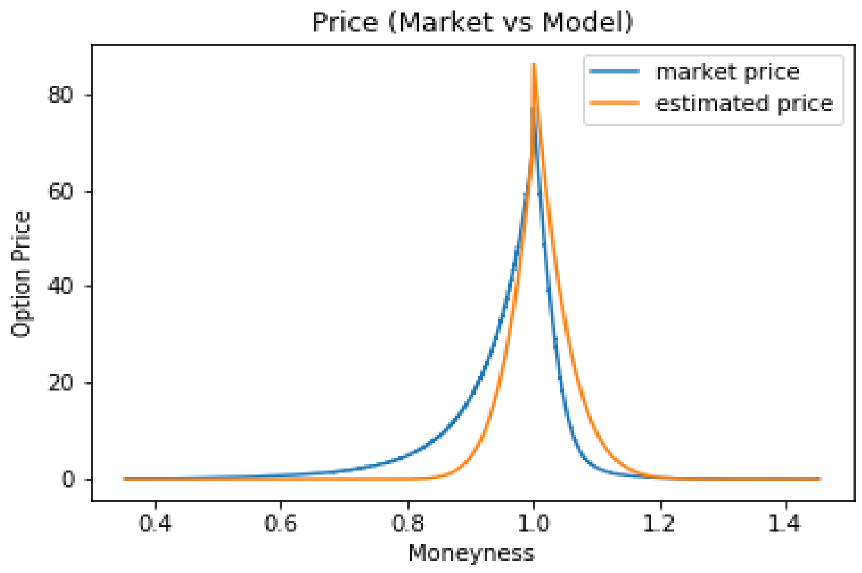

Figure 23: Option pricing with Gold futures

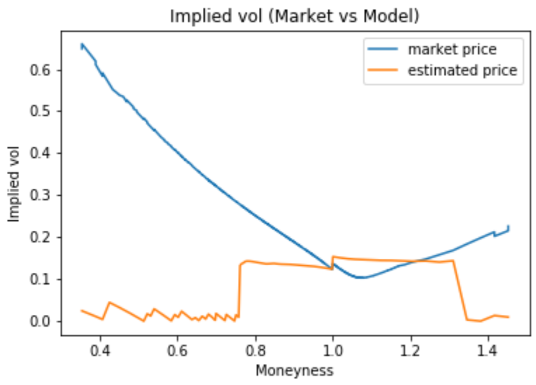

Figure 24: Implied volatility analysis with Gold futures

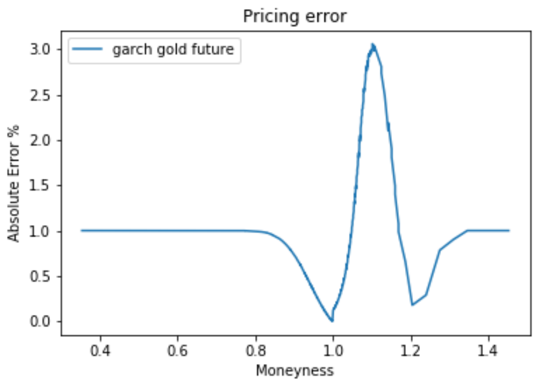

Figure 25: Implied volatility analysis with Gold futures

5.2.3 Simulated stock price distribution

To investigate the reason why implied volatility from GARCH model with different macroeconomic variable behaves differently, we further compare the stock distribution generated at the end of Monte Carlo simulation because the European option price is just the expected value conditional on the stock price when option expires and different series of estimated option price associated with strike price will give corresponding implied volatility curve with different shape.

Based on the assumption of Black Scholes model, the volatility is constant during pricing period. In this case, the stock price will follow log normal distribution and thus fail to capure the volatility smile in the market. If we change the pricing model to GARCH model without macroeconomic variables, the stock distribution evolves far away from the log normal distribution with a wider range of simulated stock price. The stock distribution generated from GARCH model fits the market situation better than Black Scholes model, so GARCH model is able to explain to the volatility smiles. Each economic variables putting into the GARCH model will have specific impact on the form the distribution. Therefore, the ability to capture volatility smile differs among different economic variables. But since the prices from GARCH model are dispersed, much extreme values yield higher price for deep out-the-money option and thus higher implied volatility. As a result, the GARCH model can only capture the smile but not the skewness of the implied volatility curve observed in the option market.

The simulated stock price distributions of each model are plotted as below.

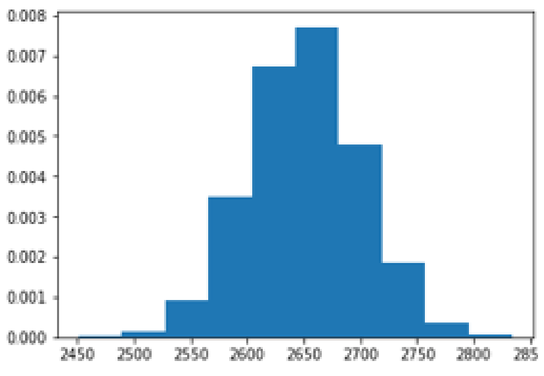

Figure 26: Stock distribtion from Black Scholes model

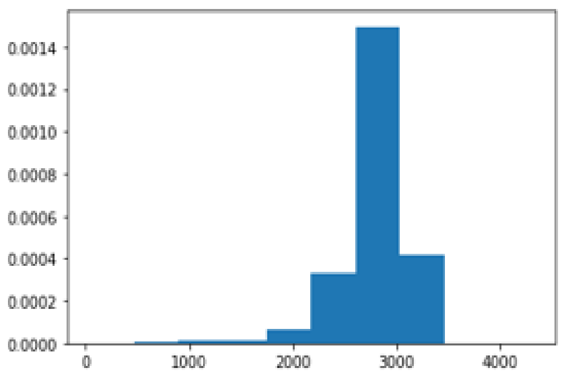

Figure 27: Stock distribtion from GARCH model without macroeconomic variables

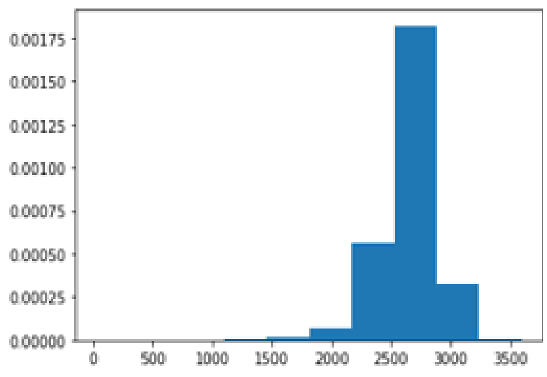

Figure 28: Stock distribtion from GARCH model with trading volume

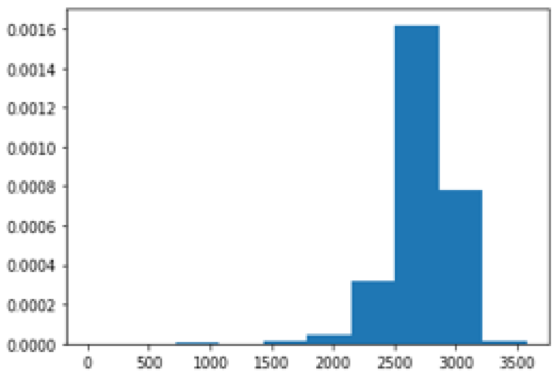

Figure 29: Stock distribtion from GARCH model with US treasury 10 year yield

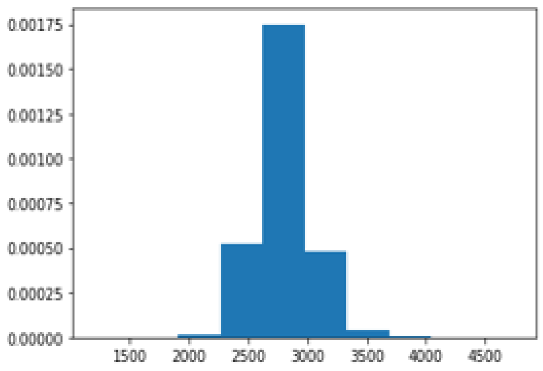

Figure 30: Stock distribtion from GARCH model with Foreign exchange rate (Yen/US)

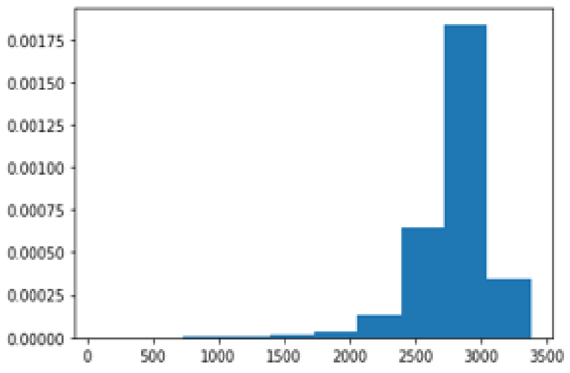

Figure 31: Stock distribtion from GARCH model with exchange rate daily volatility

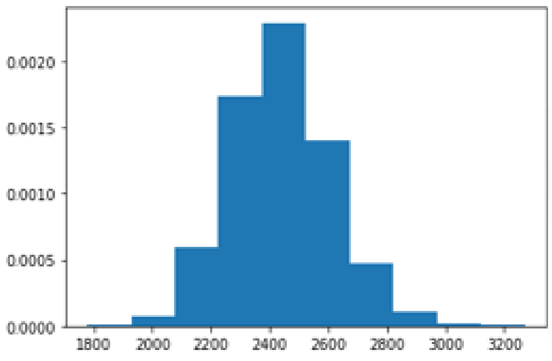

Figure 32: Stock distribution from GARCH model with Crude oil(WTI) futures

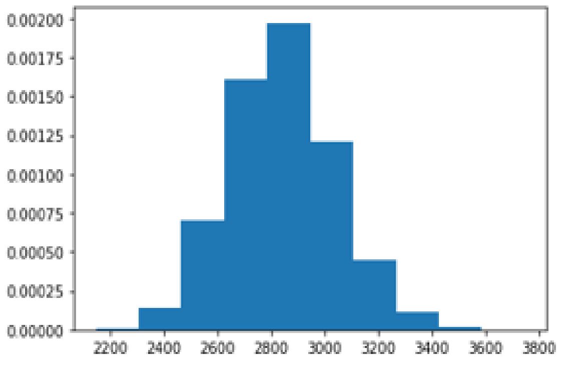

Figure 33: Stock distribution from GARCH model with Gold futures

6 Conclusions and Limitations

Judge from the pricing error and significance percentage from the the above evidences. The GARCH model with the asymmetric effect factor λ is the foundation to perform better in option pricing especially for out of money put. Since it is the λ tends to bring down the price thus could more accurately capture the smile. The other economic variables, not mention themselves are not stationary, will handicap the effect of λ thus poor performance in pricing and smile capture. The good performance of trading volume, foreign exchange daily volatility might due to the significant lambda to certain extent, since their coefficients are not significant most of the times. Oil future behaves like gold future, but has better performance in generally. All in all, the evidences for the positive effect of economic variables in option pricing is not strong.

There are several limitations should be noticed: 1.Unlike Lethnert(2003), Hsieh and Rithchken(2000) and Barone-Adesi (2007) who estimated GARCH through fitting to market option data, this paper only estimate GARCH with index price data first then conduct option pricing subsequently. It may be criticised that this approach could miss essential information only available in the option market (i.e. expectation about future market movements), but the direct fit to option approach may increase the computational burdens to a greater extent.

2.This paper also doesn't take dividends rate into account, which is expected to have impacts on option pricing. However, this paper only targets at 3 months short maturity options, the dividend impacts may be limited.

3.It is also encouraged to investigate the pricing errors on more options (1 month, 2 months etc) as well as more economic variables or combinations of them.

7 References

Bhowmik. D, 2013, Stock Market Volatility: An Evaluation, International Journal of Scientific and Research Publications, Volume 3, Issue 10, October 2013, 71-86.

Brailsford, Timothy J., 1994, The Empirical relationship between trading volume, returns and volatility. Research Paper 94-01, December.

Barone-Adesi, G., Engle, R.F. and Mancini, L., 2008. A GARCH option pricing model with filtered historical simulation. The review of financial studies, 21(3), pp.1223-1258.

Hsieh, K.C. and Ritchken, P., 2005. An empirical comparison of GARCH option pricing models. Review of derivatives research, 8(3), pp.129-150.

Jubinski, D. and Lipton, A.F., 2013. VIX, gold, silver, and oil: how do commodities react to financial market volatility?. Journal of Accounting and Finance, 13(1), pp.70-88.

Kennedy. K. and Nourzad. F., 2016, Exchange rate volatility and its effect on stock market volatility. International Journal of Human Capital in Urban Management, 1(1), 37-46.

Lamoureux, C. G., and W. D. Lastrapes.,1990, Heteroskedasticity in Stock Return Data: Volume versus GARCH Effects. Journal of Finance, 45, 221-229.

Lehnert, T., 2003. Explaining smiles: GARCH option pricing with conditional leptokurtosis and skewness. The Journal of Derivatives, 10(3), pp.27-39.

Young, C. and Holsteen, K., 2017. Model uncertainty and robustness: A computational framework for multimodel analysis. Sociological Methods Research, 46(1), pp.3-40.

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from enum import Enum
from functools import partial
import scipy
from scipy import stats
import statsmodels.api as sm
from statsmodels.stats.stattools import jarque_bera
from statsmodels.base.model import GenericLikelihoodModel
from arch import arch_model
from statsmodels.tools.numdiff import approx_hess,approx_fprime
from arch.univariate import ConstantMean
import itertools
import datetime
from scipy.stats import norm
from scipy.optimize import brentq

import logging


logger = logging.getLogger('EMS_fx')
logger.setLevel(logging.DEBUG)
fh = logging.FileHandler('EMS_fx.log')
ch = logging.StreamHandler()
formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)
                                        s')
fh.setFormatter(formatter)
ch.setFormatter(formatter)
logger.addHandler(fh)
logger.addHandler(ch)



class PayoffType(Enum):
    Call=0
    Put=1

class EuropeanOption():
    def __init__(self,expiry,strike,payoffType):
        self.expiry = expiry
        self.strike = strike
        self.payoffType = payoffType
    def payoff(self,S):
        if self.payoffType == PayoffType.Call:
            return max(S-self.strike,0)
        elif self.payoffType == PayoffType.Put:
            return max(self.strike-S,0)
        else:
            raise Exception('payoffType not supported:',self.payoffType)

    def valueAtNode(self,t,S,continuation):
        return continuation


def garch_ems(sigma0,dw,beta0,beta1,beta2,gamma1,volume,lbd):
    return beta0 + beta1 * sigma0 + beta2 * sigma0 * (dw - lbd)**2 + gamma1 *
                                            volume


def ems(S0,sigma0_2,r,mIntervals,nPaths,trade,beta0,beta1,beta2,gamma1,volume,lbd)
                                        :
    "Empirical Martingale simulation for option pricing."

    value = 0
    value1 = 0
    hsquare = 0
    hsquare1 = 0
    sigma_2 = sigma0_2*np.ones((nPaths))
    dw = np.random.normal(0,1,nPaths)
    Sbar = np.zeros((mIntervals,nPaths))
    Sbar[0] = S0 * np.ones((nPaths))
    Sast = np.zeros((mIntervals,nPaths))
    Sast[0] = Sbar[0]
    sigmalist = np.zeros((mIntervals,nPaths))
    sigmalist[0] = sigma_2
    for m in range(1,mIntervals):
        sigma_2 = garch_ems(sigma_2,dw,beta0,beta1,beta2,gamma1,volume[m-1],lbd)
        sigmalist[m] = sigma_2
        dw = np.random.normal(0,1,nPaths)
        S = Sbar[m-1] * np.exp(r - 0.5 * sigma_2 + np.sqrt(sigma_2) * dw)
        Sbar[m] = S
        Z = Sbar[m] / Sbar[m-1] * Sast[m-1]
        Z0 = np.exp(-m * r) * sum(Z) / nPaths
        Sast[m] = S0 * Z/ Z0
    for s in Sast[-1]:
        h = trade.payoff(s)
        value += h
        hsquare += h**2
    for s in Sbar[-1]:
        h = trade.payoff(s)
        value1 += h
        hsquare1 += h**2

    pv1 = np.exp(-r * mIntervals) * value1 / nPaths
    pv = np.exp(-r * mIntervals) * value / nPaths
    stderr = np.sqrt((hsquare / nPaths - (value / nPaths)**2) / nPaths)
    stderr1 = np.sqrt((hsquare1 / nPaths - (value1 / nPaths)**2) / nPaths)
    return Sbar, Sast, pv, pv1, stderr, stderr1, sigmalist


#Under P measure, the stock process ln(S(t+1)/S(t))=mu+sigma_t+1*eplison_t+1,
                                            GARCH process is
#sigma(t+1)^2=beta0+beta1*sigma(t)^2+beta2*(sigma_t*eplison_t)^2+sum(gamma(t)*X(t)
                                            ).
#stock log return is observable

def backcast(resids) -> float:
    " used to generate inital sigma value for GARCH recursion,\
    it is the weighted average of maximum 75 approximated residuals \
    with a expotential smoothing factor 0.94."

    power = 2
    tau = min(75, resids.shape[0])
    w = 0.94 ** np.arange(tau)
    w = w / sum(w)
    backcast = np.sum((abs(resids[:tau]) ** power) * w)

    return float(backcast)

def ewma_recursion(
    lam: float, resids, sigma2, nobs: int, backcast: float):

    "Exponentially Weighted Moving-Average Variance process."

    # Throw away bounds
    var_bounds = np.ones((nobs, 2)) * np.array([-1.0, 1.7e308])

    parameters = np.array([0.0, 1.0 - lam, lam])

    p, q= 1, 1
    fresids = resids **2

    for t in range(nobs):
        loc = 0
        sigma2[t] = parameters[loc]
        loc += 1
        for j in range(p):
            if (t - 1 - j) < 0:
                sigma2[t] += parameters[loc] * backcast
            else:
                sigma2[t] += parameters[loc] * fresids[t - 1 - j]
            loc += 1
        for j in range(q):
            if (t - 1 - j) < 0:
                sigma2[t] += parameters[loc] * backcast
            else:
                sigma2[t] += parameters[loc] * sigma2[t - 1 - j]
            loc += 1
        sigma2[t] = bounds_check(sigma2[t], var_bounds[t])

    return sigma2


def variance_bounds(logS,theta):
    "Create variance bounds to cap squared sigma,ensure optimizer doesn't \
    produce invalid values, use EWMA to smooth out the bounds periodically."

    resids = logS - theta[3]
    nobs = resids.shape[0]
    tau = min(75, nobs)
    w = 0.94 ** np.arange(tau)
    w = w / sum(w)
    var_bound = np.zeros(nobs)
    initial_value = w.dot(resids[:tau] ** 2.0)
    ewma_recursion(0.94, resids, var_bound, resids.shape[0], initial_value)

    var_bounds = np.vstack((var_bound / 1e6, var_bound * 1e6)).T
    var = resids.var()
    min_upper_bound = 1 + (resids ** 2.0).max()
    lower_bound, upper_bound = var / 1e8, 1e7 * (1 + (resids ** 2.0).max())
    var_bounds[var_bounds[:, 0] < lower_bound, 0] = lower_bound
    var_bounds[var_bounds[:, 1] < min_upper_bound, 1] = min_upper_bound
    var_bounds[var_bounds[:, 1] > upper_bound, 1] = upper_bound

    return np.ascontiguousarray(var_bounds)

def bounds_check(sigma2: float, var_bounds):
    if sigma2 < var_bounds[0]:
        sigma2 = var_bounds[0]
    elif sigma2 > var_bounds[1]:
        if not np.isinf(sigma2):
            sigma2 = var_bounds[1] + np.log(sigma2 / var_bounds[1])
        else:
            sigma2 = var_bounds[1] + 1000
    return sigma2

def squared_sigmas_Pmeasure(logS,volume,theta,var_bounds,backcast):
    "Use empirical data to formalize recursive GARCH process with exogenous\
    variables."

    beta0 = theta[0]
    beta1 = theta[1]
    beta2 = theta[2]
    mu = theta[3]
    lbd = theta[4]
    gamma1 = theta[5]

    var_bounds = var_bounds(logS,theta)
    backcast = backcast(logS-mu)
    n = len(logS)
    sigma2 = np.ndarray(n)

    for t in range(n):
        if (t -1) < 0:
            sigma2[t] = beta0 + beta1 * backcast + beta2 * backcast
        else:
            sigma2[t] = beta0 + beta1 * sigma2[t-1] + beta2 * sigma2[t-1] *\
    ((logS[t-1] - mu) / np.sqrt(sigma2[t-1]) - lbd)**2 + gamma1 * volume[t]

        sigma2[t] = bounds_check(sigma2[t], var_bounds[t])

    return sigma2

def log_likelihood(logS,volume,var_bounds,backcast,theta, individual=False):
    n = len(logS)
    mu = theta[3]
    sigma_sqd = squared_sigmas_Pmeasure(logS,volume,theta,var_bounds,backcast)
    fun = [-np.log(np.sqrt(2 * np.pi))-((logS[t]-mu) ** 2)/(2 * sigma_sqd[t])-\
         0.5 * np.log(sigma_sqd[t]) for t in range(n)]
    if individual:
        return np.array(fun)
    else:
        return -sum(fun)

def starting_values(logS,volume,ngamma,var_bounds,backcast):
    "To pick up close-optimal inital guesses by creating certain combinations of \
    random parameters to input into log-likelihood funcionts. The pair with \
    highest likelihhod value will be the good starting values for optimizer."

    p, q = 1,1
    mu,lbd = 1,1
    power = 2
    alphas = np.linspace(0.01, 0.2,3)
    alphas1= alphas
    alphas2= alphas
    betas = alphas
    zetas = [alphas for i in range(ngamma)]
    abg = [0.5, 0.7, 0.9, 0.98]
    abgs = list(itertools.product(*([alphas, abg, betas, alphas1, alphas2]+(zetas)
                                            )))

    resids=logS-np.mean(logS)
    target = np.mean(abs(resids) ** power)
    svs = []
    llfs = np.zeros(len(abgs))
    for i, values in enumerate(abgs):
        alpha, agb, *zeta = values
        sv = (1.0 - agb) * target * np.ones(p + mu + lbd + q + 1+ngamma)
        if q > 0:
            sv[1 : 1 + p] = agb
        if mu > 0:
            sv[3 : 3 + mu] = alpha
        if lbd >0:
            sv[4 : 4 + lbd] = alpha
        if p > 0:
            sv[1 + p  : 1 + p + q] = alpha
        svs.append(sv)
        sigma2=squared_sigmas_Pmeasure(logS,volume,sv,var_bounds,backcast)
        resids=logS-sv[3]
        lls=-0.5 * (np.log(2 * np.pi) + np.log(sigma2) + resids ** 2.0 / sigma2)
        llfs[i] =sum(lls)
    loc = np.argmax(llfs)

    return svs[int(loc)]

def params_estimate(logS,volume,var_bounds,backcast,sv,method,xmean):
    cons=(\
          {'type':'ineq','fun':lambda x:np.array([1-x[1]-x[2]*(1+x[4]**2)])},
           {'type':'ineq','fun':lambda x:np.array(x[1])},
           {'type':'ineq','fun':lambda x:np.array(x[2])},
           {'type':'ineq','fun':lambda x:x[0]+x[5]*xmean})

    resids = logS - np.mean(logS)
    v= np.mean(abs(resids) ** 2)

    objective = partial(log_likelihood,logS,volume,var_bounds,backcast,individual=
                                            False)

    bnds = ((0,10 * v),(0,1),(0,1),(0,1),(0,1),(None,None))

    result = scipy.optimize.minimize(objective,sv,
                               method = method,
                               bounds = bnds,
                               constraints = cons)

    return result
def Sts_interference(func, result, names:list ) -> pd.DataFrame:

    " With the numercial estimation of Jacobian, use BHHH estimator to calcualte\
    covariance matrix, with the virtue of non-negative definite, so that standard\
    errors, T-statistic and P-values could all be calculated."

    params = result.x
    jac = approx_fprime(params,func)
    btt = jac.T.dot(jac)
    inv_btt = np.linalg.inv(btt)
    stderr = np.sqrt(np.diag(inv_btt))
    params_df = pd.Series(params, index = names, name='Coef')
    stderr_df = pd.Series(stderr, index = names, name="std_err")
    tvalues_df  = pd.Series(params / stderr, index=names, name='t')
    pvalues_df = pd.Series(stats.norm.sf(np.abs(params / stderr)) * 2,
                        index = names, name="P>|t|")

    return pd.concat([params_df, stderr_df, tvalues_df, pvalues_df],axis=1)



if __name__=='__main__':

#    GARCH estimation & backtesting using market option price

    path='SPX Option/SPX stock prices/'
    year=[1996+i*1 for i in range(24)]
    stockp=[]
    volume=[]
    for y in year:
        stockp.append(pd.read_csv(path+'SPX'+' '+str(y)+' '+'Stock Prices.csv',
                                          header=0,index_col=0)['Close'])
        volume.append(pd.read_csv(path+'SPX'+' '+str(y)+' '+'Stock Prices.csv',
                                          header=0,index_col=0)['Volume'])
    np.random.seed(2)


    #get raw data

    #vol of return
    stockdf = pd.concat(stockp,axis=0)

    #macro factor
    volumedf = pd.concat(volume,axis=0)

    interest_rate=pd.read_excel("interest_rate.xlsx",parse_dates=True,header=0,
                                        index_col=0)
    treasury_yield=pd.read_csv("Quandl Zero Curve FED-SVENY.csv",parse_dates=True,
                                        header=0,index_
    col=0,usecols=["Date","SVENY10"])/100
    foreign_exchange=pd.read_excel("foreign_exchange.xlsx",parse_dates=True,header
                                        =0,index_col=0).dropn
    a()

    gold_future=pd.read_excel("gold futures.xlsx",parse_dates=True,header=0,
                                        index_col=0,usecols=["Date",
    "Price"])[::-1]
    oil_future=pd.read_excel("Crude Oil WTI Futures.xlsx",parse_dates=True,header=
                                        0,index_col=0,us
    ecols=["date","price"])

    stock=pd.DataFrame(stockdf,index=pd.to_datetime(stockdf.index))
    vol=pd.DataFrame(volumedf,index=pd.to_datetime(volumedf.index))

    fx_vol=pd.read_excel("exchange_vol.xlsx",parse_dates=True,header=0,index_col=0
                                        ,usecols=["date",
    "exchange rate daily vol"])[::-1]

    dic_factor={"volume":vol,"federal":interest_rate,"treasury":treasury_yield,"fx
                                        ":foreign_exchange,\
                "gold_future":gold_future,"oil_future":oil_future,"fx_vol":fx_vol}


    #interest rate
    df_rate=pd.read_csv("Quandl Zero Curve FED-SVENY.csv",parse_dates=True,header=
                                        0,index_col=0,usecols
    =["Date","SVENY01"])/100
```

```
#option price
df_market=pd.read_csv("market_data.csv",parse_dates=True,header=0,index_col=0)

#data processing (get parameter calculation sample,return sample array)
def get_sample(rtdf,factordic,factorname,startdate,window,pricing_date=""):

    factordf=factordic[factorname]
    rtdf["logstock"]=np.log(rtdf/rtdf.shift())[1:]

    start=pd.to_datetime(startdate)
    l_date=list(rtdf.index)
    i=l_date.index(start)
    threshold=len(l_date)-window
    if i>=threshold:

        end=pd.to_datetime(pricing_date)
        start=end+datetime.timedelta(-window)

    else:
        end=l_date[i+window]

    logstock=rtdf[~rtdf.isin([-np.inf,np.inf])]
    logstock=logstock[(rtdf.index>=start)&(rtdf.index<=end)]["logstock"].
                                             dropna()
    factor_sub=factordf[(factordf.index>=start)&(factordf.index<=end)]

    df_merge=pd.merge(logstock,factor_sub,left_index=True,right_index=True)

    if factorname=="volume":
        logS=df_merge.iloc[:-1,0].values
        factor=np.log(df_merge.iloc[1:,1]).values
    else:
        logS=df_merge.iloc[:,0].values
        factor=df_merge.iloc[:,1].values
    return logS,factor,end

'''
Backtesting
'''
'''
overall result
'''

df_parameter=pd.DataFrame()
df_option_sample=pd.DataFrame()
#report daily rms(total,call,put)
df_pricing_rst=pd.DataFrame(columns=["g_put_d","g_put_nd","g_call_d",
"g_call_nd","bs_put_d","bs_put_nd","bs_call_d","bs_call_nd"])

#get sample for garch parameter estimation
l_date=list(stockdf.index)
threshold=len(l_date)-252*2
count=0
n=0
macro_factor="fx"
start_date=l_date.index("2007-01-03")
for date in stockdf.index[start_date::20]:
    starttime=datetime.datetime.now()
    if l_date.index(date)<threshold:
        sample=get_sample(stock,dic_factor,macro_factor,date,window=252*2,
                                          pricing_date="")

    else:
        try:
            end_date_index=l_date.index(date)+20*count
            end_date=l_date[end_date_index]
            sample=get_sample(stock,dic_factor,macro_factor,date,window=252*2,
                                                pricing_date=end_date)
            count=count+1
        except:
            break

    logS=sample[0]
```

```
factor=sample[1]
end_date=sample[2]
xmean=np.mean(factor)
#start estimate parameters in garch
sv = starting_values(logS,factor,1,variance_bounds,backcast)
status=0

result=params_estimate(logS,factor,variance_bounds,backcast,sv,'Nelder-
                                          Mead',xmean)
beta0,beta1,beta2,lbd,gamma=result.x[0],result.x[1],result.x[2],result.x[4
                                          ],result.x[5]

try:
    #get statistic test result
    names = ['Beta0','Beta1', 'Beta2', 'Mu', 'Lambda', 'Gamma1']
    func = partial(log_likelihood,logS,factor,variance_bounds, backcast,
                                          individual=True)
    Model_results_df = Sts_interference(func,result, names)
    p_value=Model_results_df.iloc[:,-1]
    p_beta0,p_beta1,p_beta2,p_mu,p_lambda,p_gamma=p_value[0],p_value[1],
                                          p_value[2],\
                                                           p_value[3],









except:
    continue

if result.success==True and beta0>0 and beta1 >0 and beta2 >0 and lbd >0
                                          and 1-beta1-beta2*(1+lbd
**2)>0:
    status=1
    finalresult=result

    print("get parameter")

else:
    print('Invalid estimation')


if status==0:
    continue

#get final estimated parameter
beta0,beta1,beta2,mu,lbd,gamma=finalresult.x[0],finalresult.x[1],
                                          finalresult.x[2],finalresult.x[3]
,finalresult.x[4],result.x[5]
df_parameter=df_parameter.append([{"date":date,"beta0":beta0,"beta1":beta1
                                          ,"beta2":beta2,\
                                  "mu":mu,"lbd":lbd,"gamma":gamma,\
                                "p_beta0":p_beta0,"p_beta1":p_beta1,
                                "p_beta2":p_beta2,"p_mu":p_mu,
                                "p_lambda":p_lambda,"p_gamma":p_gamma}])

sigma0_garch=abs((beta0+gamma*xmean)/(1-beta1-beta2*(1+lbd**2)))
sigma_bs=beta0
#start option pricing test after parameter estimation updated

for date_p in pd.date_range(end_date,end_date+datetime.timedelta(19)):

    try:
        test=df_market.loc[end_date,:].sort_values(by="Strike")
    except:
        continue
    #daily pricing error list by category
    print("start option pricing")
```

```
#risk free rate
try:
    r=(1+df_rate.loc[date_p].values[0])**(1/360)-1
except:
    r=r


df_daily_error=pd.DataFrame(columns=["g_put_d","g_put_nd","g_call_d","
                                         g_call_nd",\
                                     "bs_put_d","bs_put_nd","bs_call_d
                                                                           "
                                                                           ,


                                     "bs_call_nd"])

trade=EuropeanOption(90/360,1000,PayoffType.Put)
S0=test["stock market price"][0]

ems_garch=ems(S0,sigma0_garch,r,90,10000,trade,beta0,beta1,beta2,gamma
                                         ,factor,lbd)
state_price_garch=ems_garch[1]

ems_bs=ems(S0,sigma_bs,r,90,10000,trade,beta0,0,0,0,factor,0)
state_price_bs=ems_bs[1]

#get risk free rate & select out-the-money option
for d,v in test.iterrows():

    strike,price=v["Strike"],v["Option Price"]

    if strike< S0:
        if v["Type"]=="P":
            est_garch_price=np.exp(-r * 90)*np.mean([max(i,0) for i in
                                                      strike-
                                                     state_price_gar
            ch[-1,:]])
            est_bs_price=np.exp(-r * 90)*np.mean([max(i,0) for i in
                                                     strike-
                                                     state_price_bs[-1
                                                     ,:]])

        else:
            continue

    if strike>=S0:
        if v["Type"]=="C":
            est_garch_price=np.exp(-r * 90)*np.mean([max(i,0) for i in

                                                     state_price_garch
                                                     [-1,:]
            -strike])
            est_bs_price=np.exp(-r * 90)*np.mean([max(i,0) for i in
                                                     state_price_bs[-1
                                                     ,:]-strike])
        else:
            continue

    df_option_sample=df_option_sample.append([v])
    error_garch=(est_garch_price-price)**2
    error_bs=(est_bs_price-price)**2
    moneyness=strike/S0

    if moneyness<0.85:
        df_daily_error=df_daily_error.append(pd.DataFrame([{"g_put_d":
                                                  error_garch,"bs_put_d
                                                  ":
        error_bs}]))
    if moneyness>=0.85 and moneyness<1:
        df_daily_error=df_daily_error.append(pd.DataFrame([{"g_put_nd"
                                                  :error_garch,"
                                                  bs_put_nd":
        error_bs}]))
```

```
                if moneyness>=1 and moneyness<1.15:
                    df_daily_error=df_daily_error.append(pd.DataFrame([{"g_call_nd
                                                                        ":error_garch,"
                                                                        bs_call_nd":
                    error_bs}]))
                if moneyness>=1.15:
                    df_daily_error=df_daily_error.append(pd.DataFrame([{"g_call_d"
                                                                        :error_garch,"
                                                                        bs_call_d":
                    error_bs}]))


            dic_error={}
            for col in df_daily_error.columns:
                rms=np.sqrt(np.mean(df_daily_error[col].dropna()))
                dic_error[col]=rms

            df_pricing_rst=df_pricing_rst.append(pd.DataFrame([dic_error],index=[
                                                    date_p]))
            endtime=datetime.datetime.now()
            time=endtime-starttime
            n=n+1
            print(time.seconds,"parameter estimation date",date,"pricing date",
                                                    date_p,"{}/{}".format
            (n,len(stockdf.index)))
            logger.info("parameter estimation date {} , pricing date{}".format(
                                                    date,date_p))

    '''
    Statistic summary
    '''

    eco_factor="fx"


    #summary of parameter
    summary=df_parameter.set_index(["date"]).loc[:,"beta0":"gamma"]
    df_summary_parameter=pd.DataFrame([summary.mean(),summary.std()]).T

    df_pvalue=df_parameter.loc[:,"p_beta0":"p_gamma"]
    dic_pvalue={}
    for col in df_pvalue.columns:
        df_sub=df_pvalue[col].copy()
        ratio=len(df_sub[df_sub<0.05])/len(df_sub)
        dic_pvalue[col]=np.round(ratio,2)

    df_summary_parameter["significant ratio"]=list(dic_pvalue.values())
    df_summary_parameter=df_summary_parameter.rename(columns={0:"Mean",1:"SD"})
    df_summary_parameter.to_excel("parameter_summary_{}.xlsx".format(eco_factor))

    #overall result

    #group by option
    rms_mean_option=np.round(df_pricing_rst.mean().values,2)
    df_summary_option=pd.DataFrame({"Garch":list(rms_mean_option[[0,1,3,2]]),"BS":
                                            list(rms_me
    an_option[[4,5,7,6]])})
    df_summary_option.index.set_names(["Moneyness"],inplace=True)
    df_summary_option.rename(index={0:"K/S<0.85",1:"0. 8 5 K /S<1",2:" 1 K /S<1.15"
                                            ,3:"K/ S 1 .15"},inplace=True)
    df_summary_option.to_excel("rms_option_{}.xlsx".format(eco_factor))


    #group by year
    df_summary_year=np.round(df_pricing_rst.resample("2A").mean()[1:],2)
    df_summary_year.fillna(df_summary_year.mean(),axis=0,inplace=True)

    garch_rms=df_summary_year.loc[:,["g_put_d","g_put_nd","g_call_nd","g_call_d"]]
                                            .values.reshape(-1,1)
    bs_rms=df_summary_year.loc[:,["bs_put_d","bs_put_nd","bs_call_nd","bs_call_d"]
                                            ].values.reshape(-1,1)

    index_year=["2009~2011"]*4+["2011~2013"]*4+["2013~2015"]*4+["2015~2017"]*4+["
```

```
                                                  2017~2019"]*4
     index_moneyness=["K/S<0.85","0. 8 5 K /S<1"," 1 K /S<1.15","K/ S 1 .15"]*5

     df_summary_year_rst=pd.DataFrame({"Garch":garch_rms[:,0],"BS":bs_rms[:,0],"
                                                  Year":index_year,
     "Moneyness":index_moneyness}).set_index(["Year","Moneyness"])

     df_summary_year_rst.to_excel("rms_year_{}.xlsx".format(eco_factor))



     """
     Test on a particular date
     """

     #get parameter on a given date
     para_date="2017-02-15"
     test_date="2019-02-20"



     beta0,beta1,beta2,mu,lbd,gamma=df_parameter.set_index("date").loc[para_date,"
                                                  beta0":"gamma"].values

     #test market sample
     sample_test=df_market.loc[pd.to_datetime(test_date),:].sort_values(by="Strike"
                                                  )
     factor_test=get_sample(stock,dic_factor,eco_factor,pd.to_datetime(para_date),
                                                  window=252*2,pricing_date="")[1]

     xmean_test=np.mean(factor_test)

     #risk free rate
     try:
         r=(1+df_rate.loc[pd.to_datetime(test_date)].values[0])**(1/360)-1
     except:
         r=r

     # test pricing error
     l_error=[]
     l_strike=[]
     l_price=[]
     l_est_price=[]
     sigma0_garch=abs((beta0+gamma*xmean_test)/(1-beta1-beta2*(1+lbd**2)))

# test implied vol skewness
     def BSModelCall(S,K,r,sigma,T):
         d1=(np.log(S/K)+(r+sigma**2/2)*T)/(sigma*np.sqrt(T))
         d2=d1-sigma*np.sqrt(T)
         Vc=S*norm.cdf(d1)-K*np.exp(-r*T)*norm.cdf(d2)
         return Vc

     def BSModelPut(S,K,r,sigma,T):
         d1=(np.log(S/K)+(r+sigma**2/2)*T)/(sigma*np.sqrt(T))
         d2=d1-sigma*np.sqrt(T)
         Vp=K*np.exp(-r*T)*norm.cdf(-d2)-S*norm.cdf(-d1)
         return Vp

     # calculate implied voliatility
     def BSImpliedVol(func,S,K,r,T,price):
         impliedvol=brentq(lambda x:func(S, K, r, x, T)-price,-1,1)
         return impliedvol


     l_vol_mkt=[]
     l_vol_model=[]

     trade=EuropeanOption(90/360,1000,PayoffType.Put)
     S0_test=sample_test["stock market price"][0]

     ems_garch=ems(S0,sigma0_garch,r,90,10000,trade,beta0,beta1,beta2,gamma,
                                                  factor_test,lbd)
     state_price_garch=ems_garch[1]
```

```
    #get risk free rate & select out-the-money option
    for d,v in sample_test.iterrows():

        strike,price=v["Strike"],v["Option Price"]

        if strike< S0:
            if v["Type"]=="P":
                est_garch_price=np.exp(-r * 90)*np.mean([max(i,0) for i in strike-
                                                         state_price_garch[-1,:]])
                vol_mkt=BSImpliedVol(BSModelPut,S0,strike,r,90/360,price)
                vol_model=BSImpliedVol(BSModelPut,S0,strike,r,90/360,
                                                         est_garch_price)

            else:
                continue

        if strike>=S0:
            if v["Type"]=="C":
                est_garch_price=np.exp(-r * 90)*np.mean([max(i,0) for i in
                                                         state_price_garch[-1,:]-
                                                         strike])
                vol_mkt=BSImpliedVol(BSModelCall,S0,strike,r,90/360,price)
                vol_model=BSImpliedVol(BSModelCall,S0,strike,r,90/360,
                                                         est_garch_price)

            else:
                continue

        error_garch=abs((est_garch_price-price)/price)
        l_error.append(error_garch)
        l_strike.append(strike)
        l_price.append(price)
        l_est_price.append(est_garch_price)
        l_vol_mkt.append(vol_mkt)
        l_vol_model.append(vol_model)


    l_strike=list(np.array(l_strike)/S0_test)
    fig, ax = plt.subplots()
    plt.plot(l_strike,l_price,label="market price")
    plt.plot(l_strike,l_est_price,label="estimated price")
    plt.xlabel("Moneyness")
    plt.ylabel("Option Price")
    plt.title("Price (Market vs Model)")
    plt.legend()
    plt.savefig("Price_{}.png".format(eco_factor))

    fig, ax = plt.subplots()
    plt.plot(l_strike,l_vol_mkt,label="market price")
    plt.plot(l_strike,l_vol_model,label="estimated price")
    plt.xlabel("Moneyness")
    plt.ylabel("Implied vol")
    plt.title("Implied vol (Market vs Model)")
    plt.legend()
    plt.savefig("Vol_{}.png".format(eco_factor))


    fig,ax=plt.subplots()
    plt.plot(l_strike,l_error,label="garch "+"FX")
    plt.xlabel("Moneyness")
    plt.ylabel("Absolute Error %")
    plt.title("Pricing error")
    plt.legend()
    plt.savefig("Error_{}.png".format("FX"))


'''
stock price distribution
'''
plt.hist(state_price_bs[-1],density=True)
plt.hist(state_price_garch[-1],density=True)
plt.title("State price at maturity predicted with treasury")
```

```
#      beta0,beta1,beta2,mu,gamma1=0.00001,0.7,0.2,0,0
#      model=arch_model(logstock[1:700].values).fit()
#      print(model.summary())


# # Option pricing


#      gamma1=0
#      sigma0_21=abs(beta0/(1-beta1-beta2))
# #    sigma0_2=(logstock[2]/np.random.normal(0,1))**2
#      S0=stockdf.iloc[2]
#      r=(1+0.1)**(1/360)-1
# #     lbd=0.01
#      lbd2=np.mean((logstock[1:]-mu)/np.std(logstock[1:]))
#      trade=EuropeanOption(0.25,1100,PayoffType.Call)
#      volume=np.log(volumedf[1:253].values)
#      volume=factor[1:253]
#      Sbar,Sast,callems,callcrude,stderr_ems,stderr_crude,slist=ems(1300,sigma0_21
                                          ,r,90,10000,
trade,beta0,beta1,beta2,gamma,volume,lbd)

#      np.sqrt((np.mean(np.sum(slist,axis=0)))*12/9)

#      ems(1300,sigma0_21,r,90,10000,trade,beta0,beta1,beta2,gamma,volume,lbd)[2]
#      ems(1300,sigma0_21,r,90,10000,trade,beta0,beta1,beta2,gamma,volume,lbd)[2]



# # ploting and test on normality
#      testy=(logstock[1:]-mu)/np.std(logstock[1:])
#      plt.hist(logstock[1:].values,bins=30)
#      plt.xlabel('Sigma')
#      plt.ylabel('Observation')
#      plt.title('log-returns sigma distribution')
#      plt.savefig('sigma distribution.jpg')
#      jarque_bera(logstock[1:].values)

#      y2 = np.random.normal(0, 1, 3514)
#      both = np.matrix([testy, y2])
#      plt.plot(both.T, alpha=.7);
#      plt.axhline(y2.std(), color='yellow', linestyle='--')
#      plt.axhline(-y2.std(), color='yellow', linestyle='--')
#      plt.axhline(3*y2.std(), color='red', linestyle='--')
#      plt.axhline(-3*y2.std(), color='red', linestyle='--')
#      plt.xlabel('time')
#      plt.ylabel('Sigma')
#      plt.title('Log-returns vs normal distribution')
#      plt.savefig('compare.jpg')
```