Lab 1 : Univariate volatility models

Enterprise Risk Management
Spring 2021

Supervised by:
Markus Ådahl

October 14, 2021

Umeå University
Department of Mathematics and Mathematical Statistics

Filip Forsgren (fifo0010@student.umu.se)

Contents

1 Task 1, Daily volatility forecast using MA and EWMA

1.1 Method

1.1.1 Model Variance: 60 day Moving Average (MA)

Given a vector Y where y_t is the daily log return of a stock at time t and $t = -1, -2 \ldots, -60$ the volatility in time t is estimated as:

$$\hat{\sigma}_t = \sqrt{\frac{1}{60} \sum_{i=1}^{60} y_{l-i}^2}$$

See Sections 6.1.1 and 6.1.2 for python implementation.

1.1.2 Model Variance: 60 day Exponentially Weighed Moving Average (EWMA)

Given a vector Y where y_t is the daily log return of a stock at time t and $t = -1, -2 \ldots, -60$. The volatility in time t is estimated as:

$$\hat{\sigma}_t = \sqrt{\frac{1-\lambda}{\lambda\left(1-\lambda^{\top}\right)} \sum_{i=1}^{60} \lambda^i \cdot y_{t-i}^2}$$

Where $\lambda = 0.96$. See Sections 6.1.1 and 6.1.3 for python implementation.

1.2 Result

in Figure 1 the MA and EWMA daily volatility estimations of S&P500 from 2017 to 2021 are plotted together with the price of S&P500 for reference.

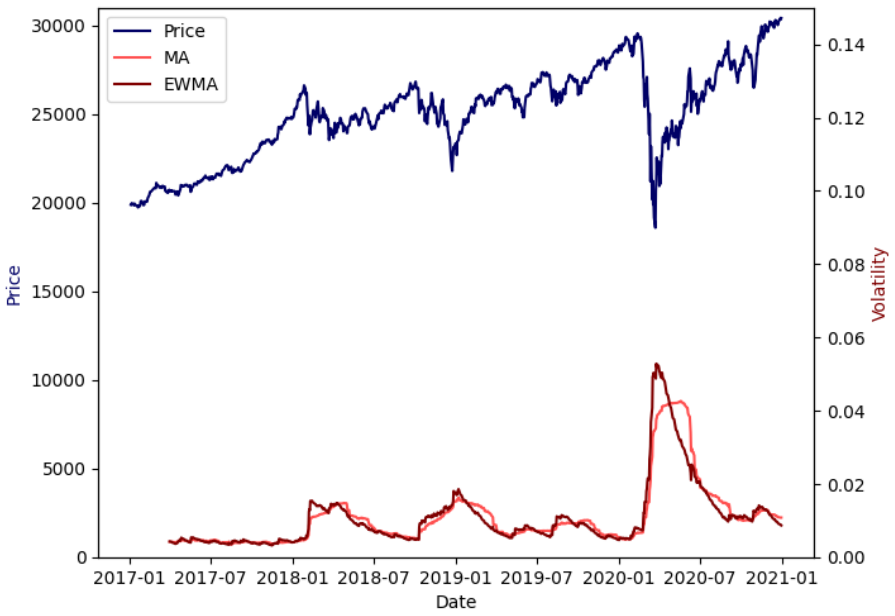

Figure 1: Volatility Estimations of SP500 using MA and EWMA plottet together with the price of S&P500.

1.3 Discussion

As can be seen in Figure 1 EWMA estimations have higher and earlier peaks compared to MA estimation. MA Does often not spike as high as EWMA but the increases *lingers* for a longer time period until it dips dramatically. The reason for this is that MA have equal weight to a price change 60 days ago as an price change yesterday. Hence, for example, small fluctuation 50-60 days ago counteracts high fluctuations the past 5 days and wise versa. Resulting in less extreme high points and less rapid decrees after a high point. However, when the extreme fluctuation is no longer in the 60-day window MA estimation decreases rapidly compared to EWMA where the extreme observations effect eases out.

2 Task 2, Daily volatility forecast using GARCH

2.1 Method

2.1.1 Estimating GARCH Parameters α, β, ω

Given a vector Y where y_t is the daily log return of a stock at time t and $t = 1, 2 \ldots, T$. The Parameters α, β and ω are estimated by maximizing following log-likelihood function with respect to α, β and ω:

$$\ell(\omega, k, \beta) = -\frac{1}{2} \sum_{i=100}^{T} term_i \tag{1}$$

Where $term_i$ is formulated as following:

$$term_i = \ln\left(\sigma_{i+1}^2\right) + \frac{y_{i+1}^2}{\sigma_{i+1}^2}$$

Where σ_{i+1} is estimated by:

$$\sigma_{i+1}^2 = \omega + \alpha y_i^2 + \beta \sigma_i^2$$

Where the initial variance σ_1^2 is estimated by:

$$\sigma_1^2 = \frac{1}{T} \sum_{t=1}^{T} (y_t - \bar{y})^2$$

Where $\bar{y}$ is the mean of the log return sample.

Since variance has to be recursively calculated, an algorithm to iterate through combinations of parameters α, β and ω are implemented as following:

1 Log Returns are multiplied with 100.

2 Set $i = 1$

3 set $range_{\alpha 0} = 0.0, 0.1 \ldots, 1$, $range_{\beta 0} = 0.0, 0.1 \ldots, 1$, $range_{\omega 0} = 0.0, 0.1 \ldots, 1$.

4 Log-likelihood is calculated for all combinations of $\alpha = range_{\alpha i}$, $\beta = range_{\beta i}$ and $\omega = range_i$

5 The parameter α, β and ω is set to the values that maximizes Log-likelihood function (1).

6 Set $range_{ki} = k - 10^{-(i)}, k - 10^{-i} + 10^{-(i+1)}, \ldots, k + 10^{-(i)}$ Where $k = \alpha, \beta, \omega$

7 Increase i by 1 ($i+ = 1$)

8 If $i < 5$ repeat 4 to 7. Else return α, β and $\omega/10000$

See Section **??** for python implementation.

2.1.2 Estimating Volatility using GARCH

Given the parameters derived from implementing the method described in Section 2.1.1 on the period 2006 to 2017 the volatility of S&P500 for the period 2017 to 2021 is estimated by:
Where σ_{i+1} is estimated by:

$$\sigma_{i+1} = \sqrt{\omega + \alpha y_i^2 + \beta \sigma_i^2}$$

Where the initial variance σ_1^2 is estimated by:

$$\sigma_1^2 = \frac{1}{T} \sum_{t=1}^{T} (y_t - \bar{y})^2$$

Where $\bar{y}$ is the mean of the 2006-2017 log return sample.

See Section 6.2.1 for python implementation.

2.2 Result

The GARCH Parameter estimates is $\alpha = 0.1236$, $\beta = 0.8527$ and $\omega = 0.0279$

In Figure 2 the MA, EWMA and GARCH daily volatility estimations of S&P500 from 2017 to 2021 are plotted together with the price of S&P500 for reference.

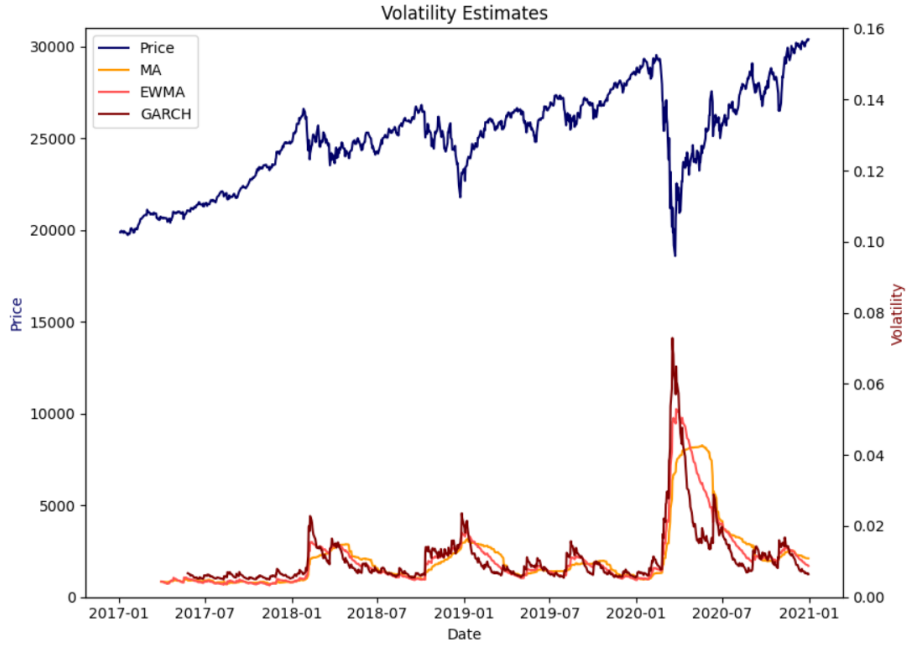

Figure 2: Volatility Estimations of SP500 using MA, EWMA and GARCH plottet together with the price of S&P500.

2.3 Discussion

Referring to previous discussion in Section 1.3 as the same arguments can be applied here. GARCH has an even smaller window of consideration when it comes to historical fluctuations.

It only takes into account the event from previous day, given the previous days volatility have in turn have been estimated trough it's previous day's fluctuation. Hence there are some correlation to days before previous day. But, as can be seen in Figure 2, the GARCH-estimated volatility is even more sensitive to extreme near past fluctuation and peaks even higher than EWMA. Hence, also decreases most rapid of all estimation methods presented here.

3 Task 3, GARCH Model Diagnostics

3.1 Method

See Section 6.3.1 for python implementation.

3.1.1 Unconditional Volatility

$$\sigma_t^2 = \omega + \alpha\varepsilon_{t-1}^2 + \beta\sigma_{t-1}^2$$

Where ε_{t-1}^2 is the log return yesterday, σ_{t-1}^2 is yesterdays variance, α and β are weights and σ can also be denoted as:

$$\gamma V_L$$

Where γ is the weight and V_L is the long run variance or *Unconditional Volatility*. Hence the sum of α, β and γ must equal 1, the following can be derived:

$$V_L = \frac{\omega}{1 - \alpha - \beta}$$

After performing the algorithm described in Section 2.1.1 α, β and γ are known. Hence, V_L can be derived.

3.1.2 Auto Correlation Function (AFC)

Auto correlation (AC) is used to describe the dependency of an event happening in time t and $t - k$. It's the ratio between the variance between time t and $t - k$ and the variance for the whole sample. The Auto Correlation Function (AFC) is as follows:

$$p_k = \frac{c_k}{c_0}$$

Where c_k is the variance between time period t and $t - k$

$$c_k = \frac{1}{T}\sum_{t=1}^{T-k}(y_+ - \bar{y})(y_{t+k} - \bar{y})$$

and c_0 is the variance of the sample:

$$c_0 = \frac{1}{T}\sum_{t=1}^{T}(y_t - \bar{y})^2$$

Here y is an arbitrary event and $\bar{y}$ is the mean of the events.

3.2 Result

3.2.1 Unconditional Volatility

The Unconditional Volatility estimate is derived to be approximately 1.08%

3.2.2 Log Return AC

In Figures 3 and 4 the AC for log return and squared log returns are presented for lag 1 to 200.

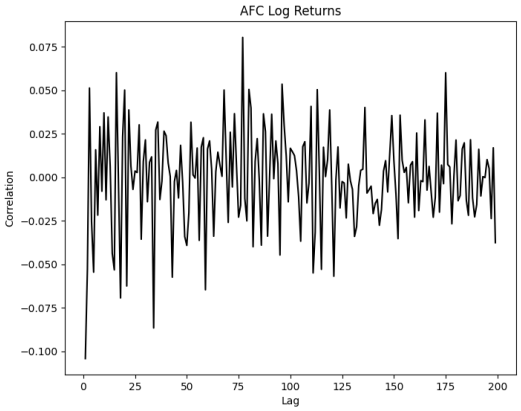

Figure 3: AC for log return are presented for lag 1 to 200.

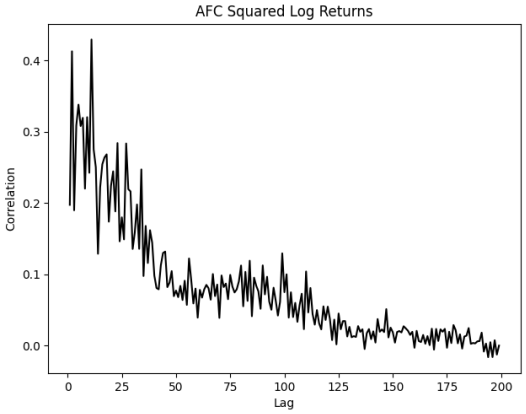

Figure 4: AC for squared log return are presented for lag 1 to 200.

3.2.3 Residual AC

In Figures 5 and 6 the AC for residuals and squared residuals are presented for lag 1 to 200.

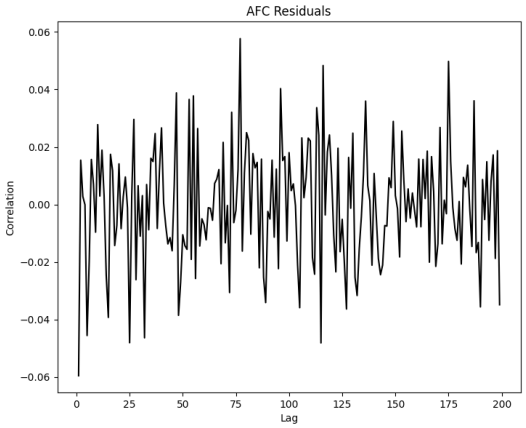

Figure 5: AC for residuals are presented for lag 1 to 200.

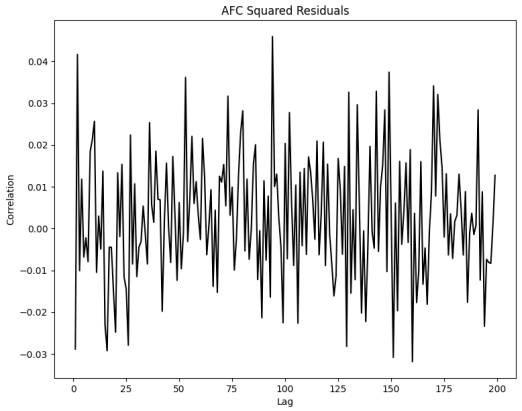

Figure 6: AC for squared residuals are presented for lag 1 to 200.

3.2.4 Residual Distribution

In Figure 7 the distribution of the residuals from GARCH estimated volatility is presented.

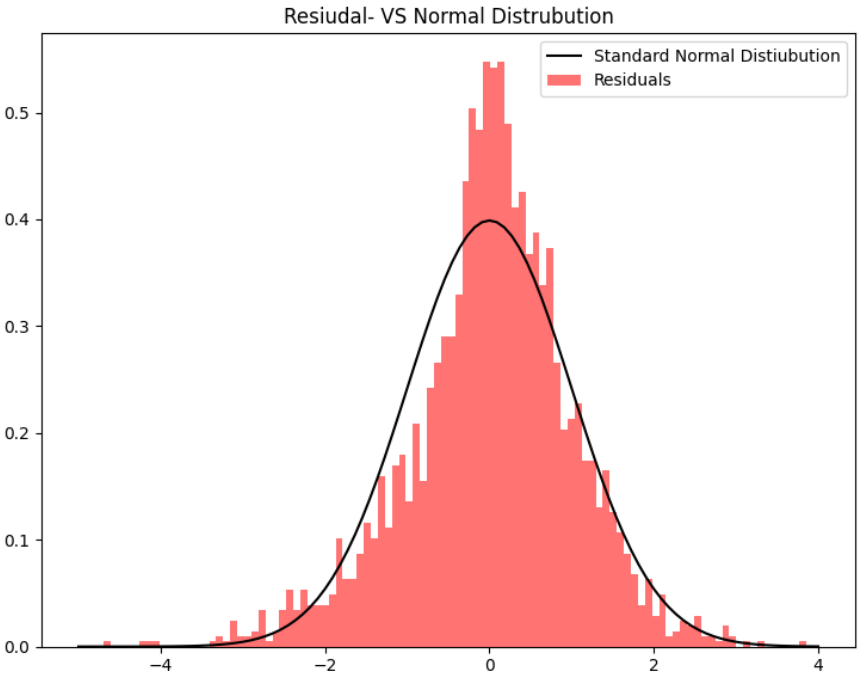

Figure 7: Residuals Distribution compared to normal distribution

In Figure 8, 9, 10, 11, 12 and 13 QQ-Plots are presented where residuals actual quantiles are plotted against theoretical quantiles of various t-distributions and normal distrubution.

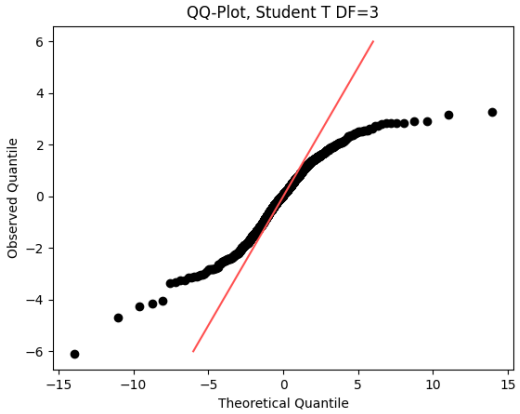

Figure 8: QQ-Plot over residuals quantiles against t-distribution with 3 degrees of freedom.

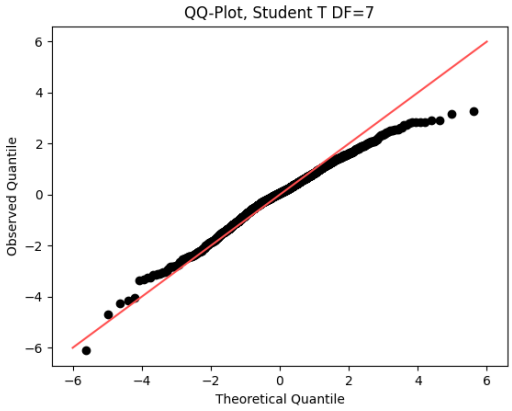

Figure 9: QQ-Plot over residuals quantiles against t-distribution with 7 degrees of freedom.

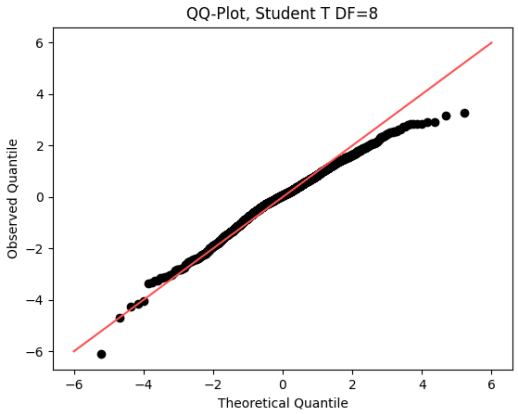

Figure 10: QQ-Plot over residuals quantiles against t-distribution with 8 degrees of freedom.

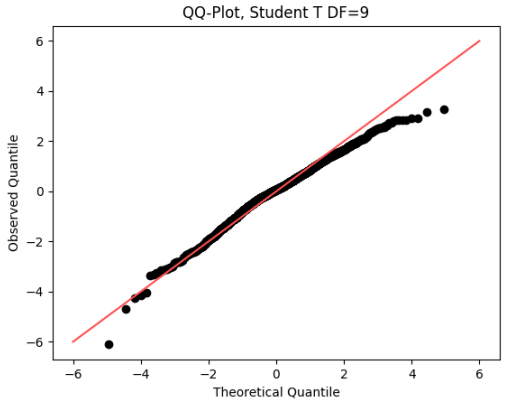

Figure 11: QQ-Plot over residuals quantiles against t-distribution with 9 degrees of freedom.

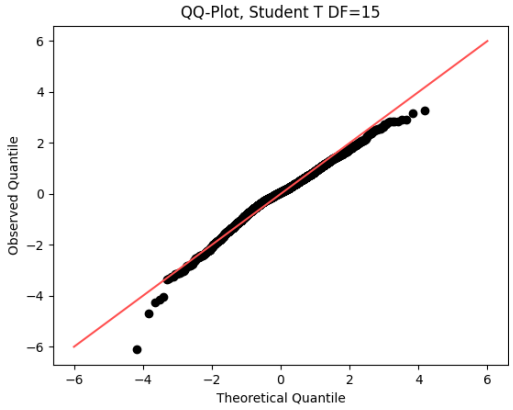

Figure 12: QQ-Plot over residuals quantiles against t-distribution with 15 degrees of freedom.

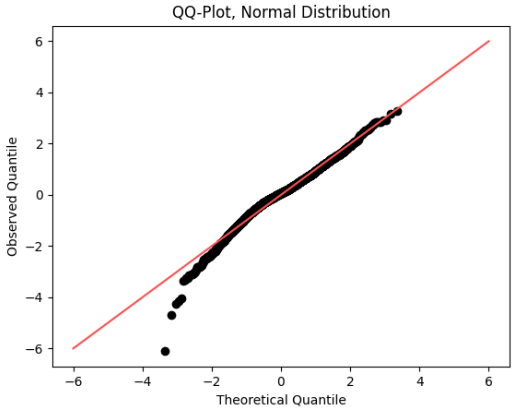

Figure 13: QQ-Plot over residuals quantiles against standard normal distribution.

3.3 Discussion

In Section 3.1.1 we can see that the estimated unconditional volatility is approximately 1% i.e. the volatility tend to be 1% past events are not taken into consideration. Which seems to be reasonable.

In Sections 3.2.2 and 3.2.4 the dependencies between the log returns and residuals between the log returns and residuals days before respectively are captured. As for log returns we can see in Figure 4 that there seems to be a lower correlation the the larger the lag is. However we can not see a particular pattern that positive returns correlates with future positive returns and vise versa. In Figure 4 it is even more evident that near past events have stronger correlation with today's event. This suggests that the size, of the return is dependent on near past returns i.e. they are not independent.

In Figures 5 and 6 we can see that the residuals produced by the GARCH model have

no correlation, they are random. More so than the shocks being assumed to be random, they are further also often assumed to be a *normal random variable*. As can be seen in Figure 7 this assumption seems to not be entirely true. Even tho it's near normal distribution we can observe *small shoulders* and *fat tails*, mainly on the negative side. Also the distribution is shifted to the positive side.
Investigating the residuals distribution further with QQ-Plots (See Figure 8 through 13) we can see that a t-distribution with to low degree of freedom (DF) 3 results in a distribution that over estimates the probability of extreme positive and negative returns. t-distribution with 7-9 DF seems to have a pretty accurate way of approximating the large negative returns. The case for higher DF, 15 up to normal distribution, contrasts low DF t-distribution as it under estimates chance of extreme low returns. More so under estimates the risk of *extreme* low returns

4 Task 4, Simple Historical Simulation

4.1 Method

4.1.1 Basic Historical Simulation

Given a vector Y where y_t is the daily log return of a stock at time t and $t = -1, -2 \ldots, -T$, a simulation of future log returns are predicted using the following algorithm:

1 Set t = 1 and Y_{sim} as an empty array

2 A uniform random variable r from 1 to T is generated.

3 Next days log return y_t is set as $y_t = y_{-r}$.

4 Set $Y = y_t, y_{t-1}, y_{t-2}, \ldots, y_{-(T-t)}$ and append y_t to T_{sim}

5 Increase t by 1, $(t+ = 1)$

6 If t $<$ P repeat step 2 to 5, else return α quantile of $-Y_{sim}$

In this specific case the algorithm is used for $T = 500, 1000$, P is the number of daily returns we want to simulate (1006) and the whole algorithm is performed 1 000 iterations to get a stable result.

See Section 6.4.1 for python implementation.

4.1.2 Result

In Table 1 the VaR and Violation Ration for window 500 and 1000 is presented.

Window	**Violations**	**Violation Ratio**
500	22	2.187
1000	26	2.585

Table 1: Performance of Simple Historical Simulation Estimates for VaR

In Figures 14 the violations for respective estimation of VaR can be viewed.

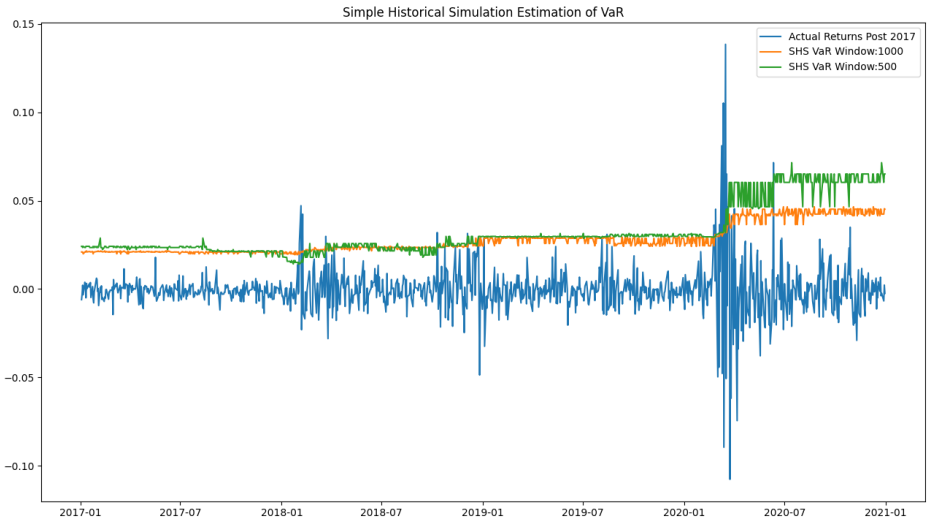

Figure 14: Actual log returns and estimated VaR using Simple Historical Simulation.

In Figure 15 the price for S&P500 over the period 2013-2017 (estimation period) and 2017-2021 (prediction period) is presented as an reference.

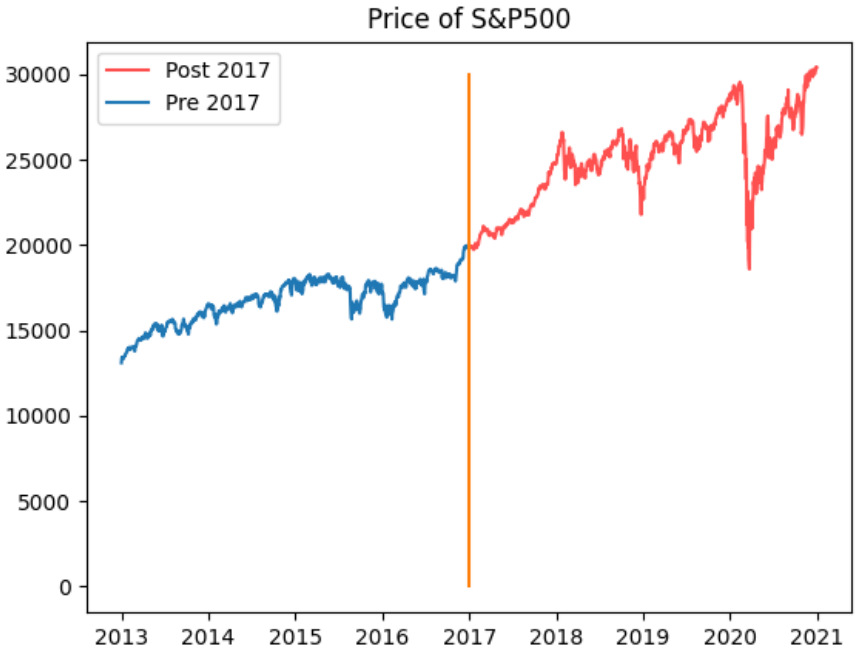

Figure 15: price for S&P500 over the period 2013-2017 (estimation period) and 2017-2021 (prediction period).

4.1.3 Discussion

Using Figure 15 as reference, we can observe that the volatility seems to be higher 4 years after 2017 compared to 4 years before 2017. This Exposes Simple Historical Simulations weakness. That is, we have a set of discrete daily returns and we cant consider returns outside this set. Thus by running a large number of iterations should converge to A VaR of

the α quantile smallest return. We can also see that what window we choose effects the VaR estimate.

5 Task 4, VaR forcasting using GARCH

5.1 Method

5.1.1 VaR forcasting using GARCH

Given a vector Y where y_t is the daily log returns, α, β and ω and a vector Z where z_t is the residuals derived from GARCH model described in Section 2.1.1 where $t = -1, -2 \ldots, -T$. A simulation of future log returns are predicted using the following algorithm:

1 Set t = 1, $\sigma_0 = \frac{1}{T}\sum_{t=1}^{T}(y_t - y)^2$ and Y_{sim} as an empty array

2 A uniform random variable r from 1 to T is generated.

3 Set $\sigma_t^2 - \omega + \alpha y_{t-1}^2 + \beta\sigma_{t-1}^2$.

4 If using FHS set $z_t = z_{-r}$, else set z_t standard normal random variable

5 Set $y_t = \sigma_t \times z_t$

6 Set $Y = y_t, y_{t-1}, y_{t-2}, \ldots, y_{-(T-t)}$ and append y_t to T_{sim}

7 Increase t by 1, $(i+ = 1)$

8 If t $<$ P repeat step 2 to 5, else return α quantile of $-Y_{sim}$.

P is the number of daily returns we want to simulate (1006) and the whole algorithm is performed 1 000 iterations to get a stable result.

See Section 6.5.1 for python implementation.

5.1.2 Result

In Table 2 the VaR and Violation Ration using GARCH model with assumed normal distribution (GARCHN) och using GARCH model with filtered historically residuals as random variables (FHS).

Window	**Violations**	**Violation Ratio**
GARCHN	23	2.286
FHS	15	1.49

Table 2: Performance of Simple Historical Simulation Estimates for VaR

In Figures 16 the violations for respective estimation of VaR can be viewed.

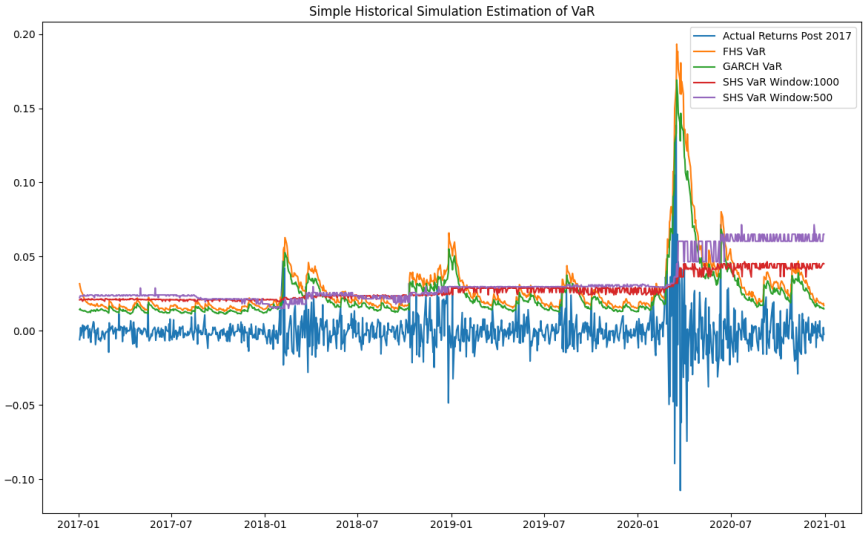

Figure 16: Actual log returns and estimated VaR using Simple Historical Simulation and GARCH Simulations.

5.1.3 Discussion

The GARCH Models can for one simulate continuous future log returns unlike the simple models described in Section 4. Also, there is an dependency between volatility's, meaning, there are times where fluctuations are low and other times when fluctuations are higher. This allows for higher peaks of volatility and consequently a higher probability to have large negative returns in one day. This is has shown to more accurately depict the real world. However there is some downsides. The GARCHN model assumes normally distributed *chocks* on the log returns. For a stock or index where this does not hold true (as it often does not), the VaR can be miss leading. Further, as can be seen in Table 2, the GARCHN model under estimates the VaR. Using FHS this problem is solved, historical residuals have a distribution with fat tails and seemingly predicts daily VaR very accurately.

There is no way to be certain when making models to predict the future. Afterall *It's impossible to predict the infinitely many outcomes of the future using the finite knowledge from the past.* But... GARCH Models seems more robust than Simple Historical Simulation.

6 Appendix

6.1 Task 1

6.1.1 Main

```
# -*- coding: utf-8 -*-
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors
import numpy as np
from MA import MA
from EWMA import EWMA

## Import from csv
df = pd.read_csv('DJI2017_2020.csv')
df['Date'] = pd.to_datetime(df.Date)
df['LogReturn'] = np.log(df['Close']).diff()

## Calculate Volatility
N = 60

ma = MA(df['LogReturn'],N)
df['VMA'] = ma

ewma = EWMA(df['LogReturn'],N,0.96)
df['VEWMA'] = ewma

## Plot
fig, price = plt.subplots()
yVol = price.twinx()
price.set_ylim(0, 31000)
yVol.set_ylim(0, 0.15)
price.set_ylabel("Price")
yVol.set_ylabel("Volatility")
price.set_xlabel("Date")

red = mcolors.to_rgb('#ff5050')
darkRed = mcolors.to_rgb('#800000')
blue = mcolors.to_rgb('#000066')

p1, = price.plot(df['Date'],df['Close'], color=blue, label="Price")
p2, = yVol.plot(df['Date'][N:len(df['VMA'])],df['VMA']...
    [N:len(df['VMA'])],color=red, label="MA")
p3, = yVol.plot(df['Date'][N:len(df['VEWMA'])],df['VEWMA']...
    [N:len(df['VEWMA'])], color=darkRed, label="EWMA")
```

```
lns = [p1, p2, p3]
price.legend(handles=lns, loc='best')

price.yaxis.label.set_color(blue)
yVol.yaxis.label.set_color(darkRed)

plt.show()
```

6.1.2 MA

```
# -*- coding: utf-8 -*-
def MA(array,N):
    import math
    '''input an n lenght array with prices. supports python lists,
    pandas columns numpy array.
    N-day Simple Moving average is calculated for from t=N+1 to t=n.
    Return an n lenght array (python list) with N-day moving averages
    and n-N zeros.
    '''
    maArray = []
    for i in range(N):
        maArray.append(0)
    for i in range(N,len(array)):
        maArray.append(math.sqrt(sum(array[i-N:i]**2)/N))
    return maArray
```

6.1.3 EWMA

```
# -*- coding: utf-8 -*-
def EWMA(array,N,lambd):
    import math
    '''input an n lenght array with prices. supports python lists,
    pandas columns numpy array.
    N-day Exponantially Weighted Moving average is calculated for
    from t=N+1 to t=n.
    Return an n lenght array (python list) with N-day moving
    averages and n-N zeros.
    '''
    EWMAArray = []
    for i in range(N):
        EWMAArray.append(0)
    for i in range(N,len(array)):
        window = []
        for j in range(1,N+1):
            window.append((lambd**j) * (array[i-j]**2))
```

```
        EWMAArray.append(math.sqrt(((1-lambd)/(lambd*(1-lambd**N)))...
            * sum(window)))

    return EWMAArray
```

6.2 Task 2

6.2.1 Main

```
variance = np.var(df['LogReturn'][1:len(df['LogReturn'])].to_numpy())
garch = [variance, variance]
var = variance
for i in range(2,len(df['LogReturn'])):
    var = alpha*(df['LogReturn'][i-1]**2) + beta*var + omega
    garch.append(var)
df['GARCH'] = np.sqrt(garch)
```

6.2.2 Main

```
def GARCHParameters(logReturns):
    import numpy as np
    '''
    Solves for variance in time t using GARCH(1.1).
    i.e based on log return and the variance in t-1 to t
    '''
    logReturns = logReturns*100
    variance = np.var(logReturns)
    start = [0,0,0]
    end = [1.1,1.1,1.1]
    length = 0.1
    alpha = np.arange(start[0], end[0], length)
    beta = np.arange(start[1], end[1], length)
    omega = np.arange(start[2], end[2], length)
    logLikelyhoods = []
    indexes = []

    for l in range(5):
        if l > 0:
            if  0 < alpha and alpha < 1 :
                start[0] = alpha - length
                end[0] = alpha + length
            elif alpha == 0:
                start[0] = 0
                end[0] = alpha + length
            else:
                start[0] = alpha - length
                end[0] = 1

            if  0 < beta and beta < 1 :
                start[1] = beta - length
                end[1] = beta + length
            elif beta == 0:
                start[1] = 0
                end[1] = beta + length
            else:
                start[1] = beta - length
                end[1] = 1

            if  0 < omega and omega < 1 :
                start[2] = omega - length
                end[2] = omega + length
            elif omega == 0:
                start[2] = 0
                end[2] = omega + length
```

```
            else:
                start[2] = omega - length
                end[2] = 1
            length = length/10
            alpha = np.arange(start[0], end[0] + length, length)
            beta = np.arange(start[1], end[1] + length, length)
            omega = np.arange(start[2], end[2] + length, length)
        logLikelyhoods = []
        indexes = []
        var = variance

        for i in range(len(alpha)):
            for j in range(len(beta)):
                if alpha[i] + beta[j] < 1:
                    for k in range(len(omega)):
                        logLikelyhood = 0
                        for t in range(1,len(logReturns)):
                            var = omega[k] + alpha[i]*(logReturns[t-1]...
                                **2) + beta[j]*var + 0.00000000000000001
                            term = np.log(var) + (logReturns[t]**2)/var
                            if t >= 100:
                                logLikelyhood -= term
                        logLikelyhoods.append(logLikelyhood)
                        indexes.append([i,j,k])

        alpha = alpha[indexes[logLikelyhoods.index...
            (max(logLikelyhoods))][0]]
        beta = beta[indexes[logLikelyhoods.index...
            (max(logLikelyhoods))][1]]
        omega = omega[indexes[logLikelyhoods.index...
            (max(logLikelyhoods))][2]]

    return alpha, beta, omega
```

6.3 Task 3

6.3.1 Main

```
# -*- coding: utf-8 -*-
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors
import numpy as np
import math
from scipy.stats import norm, t

alpha = 0.12360000000000002
beta = 0.8527399999999997
omega = .01**2 * 0.027929999999999993

#Import from csv
df = pd.read_csv('DJI2007_2016.csv')
df['Date'] = pd.to_datetime(df.Date)
# Calculate log-returns
df['LogReturn'] = np.log(df['Close']).diff()

variance = np.var(df['LogReturn'][1:len(df['LogReturn'])].to_numpy())
garch = [math.sqrt(variance), math.sqrt(variance)]
var = variance
for i in range(2,len(df['LogReturn'])):
    var = alpha*(df['LogReturn'][i-1]**2) + beta*var + omega
    garch.append(var)
df['GARCH'] = np.sqrt(garch)

K = 200
red = mcolors.to_rgb('#ff5050')
#a)
longtermVariance = omega/(1-(alpha+beta))
print('Unconditional Volatility:', math.sqrt(longtermVariance))

#b)
acfReturns = []
returns = df['LogReturn'][1:len(df['LogReturn'])].to_numpy()
meanReturn = np.mean(df['LogReturn'][1:len(df['LogReturn'])].to_numpy())
variance = np.var(df['LogReturn'][1:len(df['LogReturn'])].to_numpy())
for k in range(1,K):
    ck = []
    for t in range(len(returns)-k):
        ck.append((returns[t]-meanReturn)*(returns[t+k]-meanReturn))
    acfReturns.append((sum(ck)/len(df['LogReturn']))/variance)
fig, acf = plt.subplots()
```

```
black = mcolors.to_rgb('#000000')
p1, = acf.plot(range(1,K),acfReturns, color=black, label="Afc")
plt.title('AFC Log Returns')
plt.ylabel('Correlation')
plt.xlabel('Lag')
plt.show()


squaredReturns = df['LogReturn'][1:len(df['LogReturn'])].to_numpy()**2
acfSquaredReturns = []
meanSquaredReturns = np.mean(squaredReturns)
varianceSquaredReturns = np.var(squaredReturns)
for k in range(1,K):
    ck = []
    for t in range(len(squaredReturns)-k):
        ck.append((squaredReturns[t]-meanSquaredReturns)*(squaredReturns[t+k
    acfSquaredReturns.append((sum(ck)/len(df['LogReturn']))/varianceSquaredR
fig, acf = plt.subplots()
p1, = acf.plot(range(1,K),acfSquaredReturns, color=black, label="Afc")
plt.title('AFC Squared Log Returns')
plt.ylabel('Correlation')
plt.xlabel('Lag')
plt.show()



#c)
residuals = []
for i in range(1,len(df['LogReturn'])):
    residuals.append(df['LogReturn'][i]/df['GARCH'][i])
acfResiduals = []
meanResidual = np.mean(residuals)
varianceResidual = np.var(residuals)
for k in range(1,K):
    ck = []
    for t in range(100,len(residuals)-k):
        ck.append((residuals[t]-meanResidual)*(residuals[t+k]-meanResidual))
    acfResiduals.append((sum(ck)/len(residuals))/varianceResidual)
fig, acf = plt.subplots()
p1, = acf.plot(range(1,K),acfResiduals, color=black, label="Afc")
plt.title('AFC Residuals')
plt.ylabel('Correlation')
plt.xlabel('Lag')
plt.show()


squaredResiduals = np.array(residuals)**2
```

```
acfSquareResiduals = []
meanSquareResidual = np.mean(squaredResiduals)
varianceSquareResidual = np.var(squaredResiduals)
for k in range(1,K):
    ck = []
    for t in range(100,len(squaredResiduals)-k):
        ck.append((squaredResiduals[t]-meanSquareResidual)*(squaredResiduals
    acfSquareResiduals.append((sum(ck)/len(squaredResiduals))/varianceSquare
fig, acf = plt.subplots()
p1, = acf.plot(range(1,K),acfSquareResiduals, color=black, label="Afc")
plt.title('AFC Squared Residuals')
plt.ylabel('Correlation')
plt.xlabel('Lag')
plt.show()



#d)
mu = 0
variance = 1
sigma = math.sqrt(variance)
x = np.linspace(mu - 5*sigma, mu + 4*sigma, 100)
plt.plot(x, norm.pdf(x, mu, sigma), label='Standard Normal Distiubution', co
plt.hist(residuals[100:-1], 100, color=red, label="Residuals", alpha=0.8, de
plt.legend(loc='best')
plt.title('Resiudal- VS Normal Distrubution')
plt.show()



#e)
kurt = pd.DataFrame(residuals).kurtosis()
print(kurt[0])

residuals.sort()

red = mcolors.to_rgb('#ff5050')
black = mcolors.to_rgb('#000000')
theorethicalQ = np.arange(1,len(residuals)+1)/(len(residuals))
x = [-6,6]
y = x

theorethicalV = t.ppf(theorethicalQ, 3, loc=0, scale=1)
plt.scatter(theorethicalV,residuals, color=black)
plt.plot(x,y, color=red)
plt.title('QQ-Plot, Student T DF=3')
plt.xlabel('Theoretical Quantile')
```

```
plt.ylabel('Observed Quantile')
plt.show()

theorethicalV = t.ppf(theorethicalQ, 7, loc=0, scale=1)
plt.scatter(theorethicalV,residuals, color=black)
plt.plot(x,y, color=red)
plt.title('QQ-Plot, Student T DF=7')
plt.xlabel('Theoretical Quantile')
plt.ylabel('Observed Quantile')
plt.show()

theorethicalV = t.ppf(theorethicalQ, 8, loc=0, scale=1)
plt.scatter(theorethicalV,residuals, color=black)
plt.plot(x,y, color=red)
plt.title('QQ-Plot, Student T DF=8')
plt.xlabel('Theoretical Quantile')
plt.ylabel('Observed Quantile')
plt.show()

theorethicalV = t.ppf(theorethicalQ, 9, loc=0, scale=1)
plt.scatter(theorethicalV,residuals, color=black)
plt.plot(x,y, color=red)
plt.title('QQ-Plot, Student T DF=9')
plt.xlabel('Theoretical Quantile')
plt.ylabel('Observed Quantile')
plt.show()

theorethicalV = t.ppf(theorethicalQ, 15, loc=0, scale=1)
plt.scatter(theorethicalV,residuals, color=black)
plt.plot(x,y, color=red)
plt.title('QQ-Plot, Student T DF=15')
plt.xlabel('Theoretical Quantile')
plt.ylabel('Observed Quantile')
plt.show()

theorethicalV = norm.ppf(theorethicalQ, loc=0, scale=1)
plt.scatter(theorethicalV,residuals, color=black)
plt.plot(x,y, color=red)
plt.title('QQ-Plot, Normal Distribution')
plt.xlabel('Theoretical Quantile')
plt.ylabel('Observed Quantile')
plt.show()
```

6.4 Task 4

6.4.1 Main

```
# -*- coding: utf-8 -*-
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors
import numpy as np
import math

iterations = 10000
VaRAlpha = 0.01
cutOff = pd.to_datetime('2017-01-01')

#Import from csv
df = pd.read_csv('DJI2013_2020.csv')
df['Date'] = pd.to_datetime(df['Date'])
df['LogReturn'] = np.log(df['Close']).diff()

period = len(df['LogReturn'][df['Date'] > cutOff])
futureReturns = df['LogReturn'][df['Date'] > cutOff].values.tolist()

windowSize = 1000

rollingReturnWindow = df['LogReturn'][df['Date'] < cutOff][-(windowSize+1):
VaR = np.zeros(period)
for t in range(period):
    simulatedReturns = np.zeros(iterations)
    for i in range(iterations):
        pick = int(np.floor(np.random.rand(1)*windowSize)[0])
        simulatedReturns[i] = -rollingReturnWindow[pick]
    simulatedReturns.sort()
    VaR[t] = simulatedReturns[math.floor(((len(simulatedReturns)-1)*(1-VaRA
    rollingReturnWindow = np.append(rollingReturnWindow,futureReturns[t])
    rollingReturnWindow = np.delete(rollingReturnWindow, 0)

violations = 0
for i in range(len(VaR)):
    if VaR[i] < -futureReturns[i]:
        violations += 1

violationsPrc = violations/len(VaR)
violationRatio = violationsPrc/VaRAlpha

print([violations,violationRatio])
```

```
windowSize = 500

rollingReturnWindow = df['LogReturn'][df['Date'] < cutOff][-(windowSize+1):
VaR1 = np.zeros(period)
for t in range(period):
    simulatedReturns = np.zeros(iterations)
    for i in range(iterations):
        pick = int(np.floor(np.random.rand(1)*windowSize)[0])
        simulatedReturns[i] = -rollingReturnWindow[pick]
    simulatedReturns.sort()
    VaR1[t] = simulatedReturns[math.floor(((len(simulatedReturns)-1)*(1-VaR
    rollingReturnWindow = np.append(rollingReturnWindow,futureReturns[t])
    rollingReturnWindow = np.delete(rollingReturnWindow, 0)

violations = 0
for i in range(len(VaR1)):
    if VaR1[i] < -futureReturns[i]:
        violations += 1

violationsPrc = violations/len(VaR1)
violationRatio = violationsPrc/VaRAlpha

print([violations,violationRatio])

red = mcolors.to_rgb('#ff5050')

plt.plot(df['Date'][df['Date'] > cutOff], -df['LogReturn'][df['Date'] > cut
plt.plot(df['Date'][df['Date'] > cutOff],VaR, label='SHS VaR Window:1000')
plt.plot(df['Date'][df['Date'] > cutOff],VaR1, label='SHS VaR Window:500')
plt.title('Simple Historical Simulation Estimation of VaR')
plt.legend(loc='best')
plt.show()
```

6.5 Task 4

6.5.1 Main

```
# -*- coding: utf-8 -*-
import pandas as pd
import matplotlib.pyplot as plt
from scipy.stats import norm
import numpy as np
import math
import random

## Settings
iterations = 10000
cutOff = pd.to_datetime('2017-01-01')


# GARCH Parameters (estimation window 2007-2016)
alpha = 0.12360000000000002
beta = 0.8527399999999997
omega = .01**2 * 0.027929999999999993


# VaR Parameter
VaRAlpha = 0.01

## Import Data
df = pd.read_csv('DJI2007_2020.csv')
df['Date'] = pd.to_datetime(df.Date)
df['LogReturn'] = np.log(df['Close']).diff()

historicLogReturns = df['LogReturn'][df['Date'] < '2017-01-01'][1:-1].to_num
period = len(df['LogReturn'][df['Date'] > cutOff])

# Generate Residuals from Historical Data
variance = np.var(historicLogReturns)
variances = [variance]
var = variance
for i in range(1,len(historicLogReturns)):
    var = alpha*(historicLogReturns[i-1]**2) + beta*var + omega
    variances.append(var)

residuals = []
for i in range(len(historicLogReturns)):
    residuals.append(historicLogReturns[i]/np.sqrt(variances[i]))

mu = 0
```

```
variance = 1
sigma = math.sqrt(variance)
x = np.linspace(mu - 5*sigma, mu + 4*sigma, 100)
plt.plot(x, norm.pdf(x, mu, sigma), label='Standard Normal Distiubution')
plt.hist(residuals, 100, label="Residuals", alpha=0.8, density=True)
plt.legend(loc='best')
plt.title('Resiudal- VS Normal Distrubution')
plt.show()

futureLogReturns = df['LogReturn'][df['Date'] > '2017-01-01'].to_numpy()

# Generate Future residuals
variance = np.var(futureLogReturns)
variances = [variance]
var = variance
for i in range(1,len(futureLogReturns)):
    var = alpha*(futureLogReturns[i-1]**2) + beta*var + omega
    variances.append(var)

futResiduals = []
for i in range(len(futureLogReturns)):
    futResiduals.append(futureLogReturns[i]/np.sqrt(variances[i]))
futResiduals.sort()

mu = 0
variance = 1
sigma = math.sqrt(variance)
x = np.linspace(mu - 5*sigma, mu + 4*sigma, 100)
plt.plot(x, norm.pdf(x, mu, sigma), label='Standard Normal Distiubution')
plt.hist(futResiduals, 100, label="Residuals", alpha=0.8, density=True)
plt.legend(loc='best')
plt.title('Resiudal- VS Normal Distrubution')
plt.show()


## FHS; Simulate LogReturns with GARCH (volatility) and Randomly Drawns Resi

FHSsimulatedReturns = np.zeros(iterations)
FHSVaR = np.zeros(period)
logReturn = historicLogReturns[-1]
var = np.var(historicLogReturns)
for t in range(period):
    var = alpha*logReturn**2 + beta*var + omega
    for i in range(iterations):
        z = residuals[random.randint(0,len(residuals)-1)]
        logReturn = math.sqrt(var)*-z
        FHSsimulatedReturns[i] = logReturn
```

```
    FHSsimulatedReturns.sort()
    FHSVaR[t] = np.quantile(FHSsimulatedReturns, 1-VaRAlpha)
    logReturn = futureLogReturns[t]

violations = 0
for i in range(len(FHSVaR)):
    if FHSVaR[i] < -futureLogReturns[i]:
        violations += 1

violationsPrc = violations/len(FHSVaR)
violationRatio = violationsPrc/VaRAlpha

print([violations,violationRatio])

## Simulate LogReturns with GARCH (volatility) and Normal Random Variable (n
mu, sigma = 0, 1
GARCHsimulatedReturns = np.zeros(iterations)
GARCHVaR = np.zeros(period)
logReturn = historicLogReturns[-1]
var = variances[-1]
for t in range(period):
    var = alpha*logReturn**2 + beta*var + omega
    for i in range(iterations):
        z = np.random.normal(mu, sigma)
        logReturn = math.sqrt(var)*-z
        GARCHsimulatedReturns[i] = logReturn

    GARCHsimulatedReturns.sort()
    GARCHVaR[t] = np.quantile(GARCHsimulatedReturns, 1-VaRAlpha)

    logReturn = futureLogReturns[t]

violations = 0
for i in range(len(GARCHVaR)):
    if GARCHVaR[i] < -futureLogReturns[i]:
        violations += 1

violationsPrc = violations/len(GARCHVaR)
violationRatio = violationsPrc/VaRAlpha

print([violations,violationRatio])

plt.hist(-df['LogReturn'][df['Date'] > cutOff], 100, density=True, alpha=0.6
plt.plot(x,y,color=red, label='GARCH VaR')
plt.plot(x1,y, label='FHS VaR')
plt.plot(x2,y, label='SHS 500 VaR')
```

```
plt.plot(x3,y, label='SHS 1000 VaR')
plt.legend(loc='best')
plt.title('VaR estimates ant actual Returns')
plt.show()
```