HW2

Alessandra Campanella, Giuseppina Orefice

2024-12-23

In the ACI paper (Zaffran paper) the author provides a general description of both the conformal prediction (CP) and the adaptive conformal prediction or interval (ACI) focusing on their application to uncertainty quantification. Conformal prediction provides a framework for constructing prediction interval that guarantee a statistical property known as **validity coverage**.

Formally, given past observations $Z_t = (X_r, Y_r)_{1 \leq r \leq (t-1)}$ the goal is to predict Y_t using X_t. The obtained prediction interval $\hat{C}_t$ is designed to satisfy:

$P(Y_t \in \hat{C}_t \geq 1 - \alpha)$ where $\alpha \in [0, 1]$ is the user-specified miscoverage rate, typically chosen to be 0.1 or 0.05.

This framework is based on the assumption of exchangeability, which requires that the data sequence have to remain invariant under permutations. Specifically, given a dataset $(V_1, V_2, .. V_n)$, exchangeability implies that for any permutation π, $(V_1, V_2, .. V_n) =^d (V_\pi 1, V_\pi 2, .. V_\pi n)$.

In other words the joint distribution of the data does not depend on the ordering of the observations, that is why exchangeability is crucial to guarantee the validity of conformal prediction intervals.

So, given the value of Y_t, we want to predict it using X_t.

Let y be a candidate for Y_t, we construct a predictor $\hat{\mu}(X)$ (remember that in conformal prediction we care about the functional form of the predictor and it can assume whichever distribution we want) and we need to evaluate the conformity scores.

The conformity scores are defined as: $S(X_t, y) = |\hat{\mu}(X) - y|$

which measures how well y aligns with the observed data.

For instance, in the previous case, $\hat{\mu}(X)$ is chosen as the mean predictor. Alternatively, a quantile regression can be used to achieve conditional coverage (since with the predictor above we can achieve marginal coverage), which provides better conformal intervals by adapting them to the heteroscedasticity of the data.

Specifically, quantile regression estimates the quantile function $\hat{q}(X; p)$ where p is p-th quantile of the conditional distribution $Y \mid X$.

Quantile regression allow us to compute conformal intervals while accounting for the conditional distribution of Y and it improves the precision and efficiency of the prediction intervals.

Split Conformal Prediction

Assuming that the training and calibration dataset are both exchangeable, then we can perform the the split conformal prediction for the quantile regression as follows.

The estimation of the upper quantile is basically given by $\hat{q}(X_t; \frac{\alpha}{2})$ and instead for the lower quantile we have $\hat{q}(X_t; \frac{1-\alpha}{2})$. Then, these quantiles are calculated on the training set. The conformity scores on the calibration set are defined as:

$S(X_t, y) = max(\hat{q}(X_t; \frac{\alpha}{2}) - y; y - \hat{q}(X_t; \frac{1-\alpha}{2}))$

These scores calculated on the calibration set ensure symmetry and equal weighting across all data points, if we use these scores we are able to compute the critical quantile. This step finalizes the prediction interval construction, satisfying the desired validity guarantees.

Critical quantile computation:

$$\hat{Q}(p) := \inf\left\{ s: \left(\frac{1}{|D_{cal}|}\right) \sum_{(X_r, Y_r) \in D_{cal}} \mathbf{1}\{S(X_r, Y_r) \leq s) \geq p \right\}$$

This quantile defines the smallest score s such that the fraction of conformity scores in the calibration dataset D_{cal} less than or equal to s reaches the p-th quanatile.

In simpler terms, the crictial quantile is the score threshold ensuring that the prediction interval aligns with the specified coverage level.

Equivalently the critical quantile is equal to:

$S_{\lfloor (1-\alpha)(n_c+1) \rfloor}$

where $n_{cal} = |D_{cal}|$ is the size of the calibration dataset and this is the score at the $(1 - \alpha)$ -th quantile in D_{cal}. An alternative formulation of the prediction interval's validity is given by:

$$P(Y_t \in \hat{C}_t) = P(S(X_t, Y_t) \leq \hat{Q}(1 - \alpha)) = \frac{\lfloor |D_{cal}|(1-\alpha) \rfloor}{|D_{cal}| + 1} \$$$

Adaptive Conformal Prediction

In real-world applications, the assumption of **exchangeability** often fails compared to financial applications, specially under the covariate or distribution shifts. To address this limitation, **Adaptive Conformal Inference (ACI)** was introduced. ACI is able to ensure the validity through an

online forecasting, meaning that we construct conformal inference adapted to the changes in the data when they occur and then it should be re-estimated to align with the most recent observations. This approach is simple, because it requires estimating only a single parameter α_t, making it computationally efficient, another advantage is the it can integrate any machine learning model for point prediction.

The adaptive conformal prediction method involves estimating the score function $S_t()$ and updating the critical quantile $\hat{Q}_t$. In addition we need to estimate the miscoverage rate of prediction given by:

$$M_t(\alpha) := P(S_t(X_t, Y_t) > \hat{Q}_t(1-\alpha)$$

where $M_t(\alpha)$ represents the proportion of scores exceeding the critical quantile, which should ideally approximate α. If this does not hold, then a corrected value α_t in (0,1) is introduced, such that $M_t(\alpha)$ is approximated to α. Basically we work by looking at the miscoverage rate of prediction and we need to define an indicator function for the errors. $\text{err}_t := \begin{cases} 1, & \text{if } Y_t \notin \hat{C}_t(\alpha_t), \\ 0, & \text{otherwise.} \end{cases}$

where the prediction interval is given by: $\hat{C}_t(\alpha_t) := \{y : S_t(X_t, y) \leq \hat{Q}_t(1-\alpha_t)\}$.

This will tell us that if the errors are equal to 1 then the confidence interval is too short and it is not able to capture all the errors; on the other hand if it is equal to 0, then the confidence interval is too long and it is not able to exclude the errors equal to α.

In the experiment presented in the paper we start with $\alpha_1 = \alpha$, this is the starting point of the ACI, then it update α_t recursively as:
$\alpha_{t+} = \alpha_t + \gamma(\alpha - err_t)$ where $\gamma > 0$ is a step-size parameter balancing adaptability and stability. If γ is very high, we need to adapt more because of the greater change in the distribution.

If it is too short, we need to adapt less.

If the err_t is 0, it basically means that the interval is too short and we need to increase it to take more errors; instead, if it is 1, the interval was too long and we need to short it An extended version incorporates weighted historical errors: $\alpha_{t+1} = \alpha_t + \gamma\left(\alpha - \sum_{s=1}^{t} w_s, \text{err}s\right)$

where $w_{s_{(1 \leq s \leq t)}}$ is a sequence of weights summing to 1, emphasizing recent errors.

Point 1.2

Focusing on the GARCH(1,1) and apARCH(1,1) in this point we are going to evaluate the first strategy, which is: standard (non-adaptive) CP strategy with both simple (non-normalized) and studentized or normalized absolute residual scores as reported in Section 5 of the ACI paper (and as implemented in Section 2.2).

In section 5 the authors explain the impact of the choice of the conformity score St(.) and in particular the types of score discussed are the non-normalized absolute residuals $|e_t|$ and the normalized (studentized) residuals $|e_t| / \sigma_t$. In this section the non-adaptive and adaptive conformal prediction methods are evaluated based on the conformity scores computed on the residuals of the GARCH(1,1) model and apARCH(1,1)

First we have downloaded the Amazon financial data.

```
library(rugarch)
```

```
## Warning: il pacchetto 'rugarch' è stato creato con R versione 4.3.3
```

```
## Caricamento del pacchetto richiesto: parallel
```

```
##
## Caricamento pacchetto: 'rugarch'
```

```
## Il seguente oggetto è mascherato da 'package:stats':
##
##     sigma
```

```
library(quantmod)
```

```
## Warning: il pacchetto 'quantmod' è stato creato con R versione 4.3.3
```

```
## Caricamento del pacchetto richiesto: xts
```

```
## Warning: il pacchetto 'xts' è stato creato con R versione 4.3.3
```

```
## Caricamento del pacchetto richiesto: zoo
```

```
## Warning: il pacchetto 'zoo' è stato creato con R versione 4.3.3
```

```
##
## Caricamento pacchetto: 'zoo'
```

```
## I seguenti oggetti sono mascherati da 'package:base':
##
##     as.Date, as.Date.numeric
```

```
## Caricamento del pacchetto richiesto: TTR
```

```
## Warning: il pacchetto 'TTR' è stato creato con R versione 4.3.3
```

```
## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
```

```
library(quantreg)
```

```
## Warning: il pacchetto 'quantreg' è stato creato con R versione 4.3.3
```

```
## Caricamento del pacchetto richiesto: SparseM
```

```
## Warning: il pacchetto 'SparseM' è stato creato con R versione 4.3.3
```

```
##
## Caricamento pacchetto: 'SparseM'
```

```
## Il seguente oggetto è mascherato da 'package:base':
##
##     backsolve
```

```
## Warning in .recacheSubclasses(def@className, def, env): sottoclasse non
## definita "ndiMatrix" della classe "replValueSp"; definizione non aggiornata
```

```
library(zoo)
library(dplyr)
```

```
## Warning: il pacchetto 'dplyr' è stato creato con R versione 4.3.3
```

```
##
## ########################## Warning from 'xts' package ##########################
## #                                                              #
## # The dplyr lag() function breaks how base R's lag() function is supposed to  #
## # work, which breaks lag(my_xts). Calls to lag(my_xts) that you type or       #
## # source() into this session won't work correctly.                            #
## #                                                              #
## # Use stats::lag() to make sure you're not using dplyr::lag(), or you can add #
## # conflictRules('dplyr', exclude = 'lag') to your .Rprofile to stop           #
## # dplyr from breaking base R's lag() function.                                #
## #                                                              #
## # Code in packages is not affected. It's protected by R's namespace mechanism #
## # Set `options(xts.warn_dplyr_breaks_lag = FALSE)` to suppress this warning.  #
## #                                                              #
## ################################################################################
```

```
##
## Caricamento pacchetto: 'dplyr'
```

```
## I seguenti oggetti sono mascherati da 'package:xts':
##
##     first, last
```

```
## I seguenti oggetti sono mascherati da 'package:stats':
##
##     filter, lag
```

```
## I seguenti oggetti sono mascherati da 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(opera)
```

```
## Warning: il pacchetto 'opera' è stato creato con R versione 4.3.3
```

```
getSymbols("AMZN", from = "2016-12-31", to = "2024-12-31")
```

```
## [1] "AMZN"
```

Then we have inspected in them.

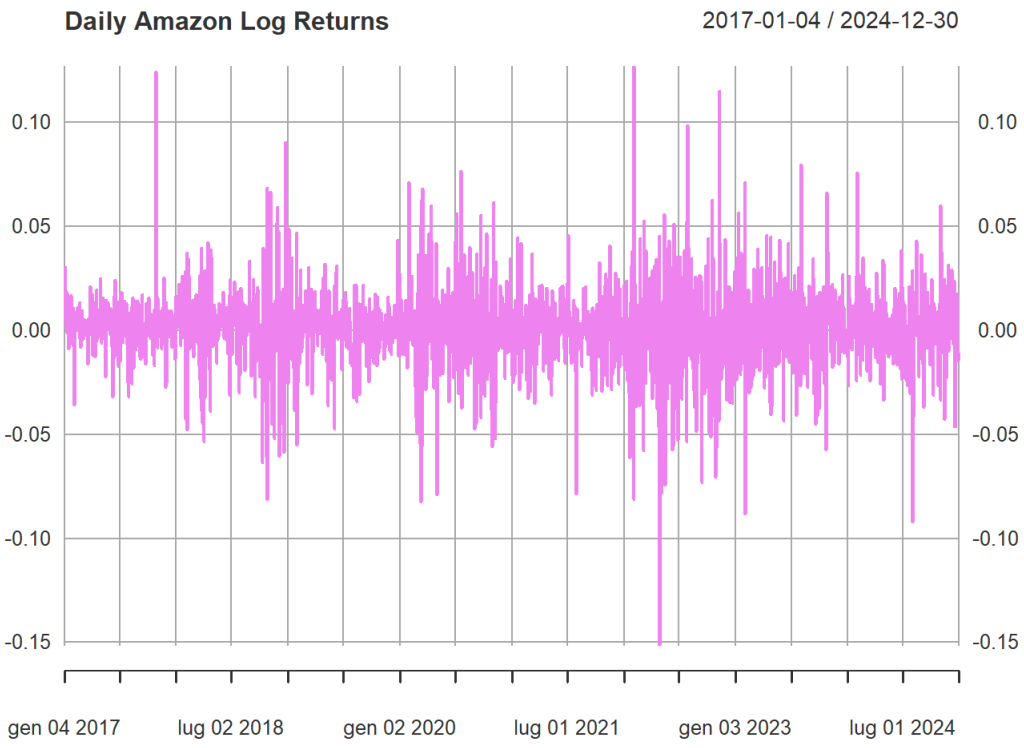
```
head(AMZN)
```

```
##            AMZN.Open AMZN.High AMZN.Low AMZN.Close AMZN.Volume AMZN.Adjusted
## 2017-01-03   37.8960   37.9380  37.3850    37.6835    70422000      37.6835
## 2017-01-04   37.9195   37.9840  37.7100    37.8590    50210000      37.8590
## 2017-01-05   38.0775   39.1200  38.0130    39.0225   116602000      39.0225
## 2017-01-06   39.1180   39.9720  38.9240    39.7995   119724000      39.7995
## 2017-01-09   39.9000   40.0885  39.5885    39.8460    68922000      39.8460
## 2017-01-10   39.8300   39.9000  39.4770    39.7950    51168000      39.7950
```

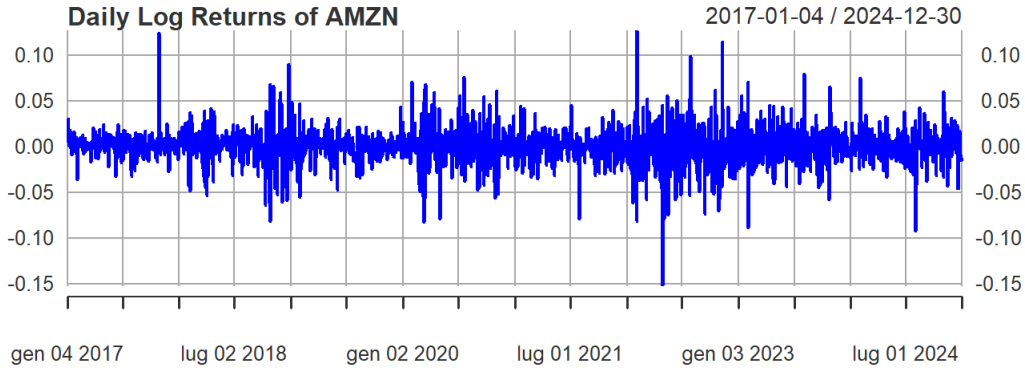
The authors apply ACI to market volatility prediction using a GARCH(1,1) model to form point predictions. They compare the adaptive algorithm with a non-adaptive alternative that keeps α_t fixed, measuring performance through local coverage frequencies. The results show that the adaptive conformal inference maintains the desired coverage even during significant events such as the 2008 financial crisis, while the non-adaptive method fails.

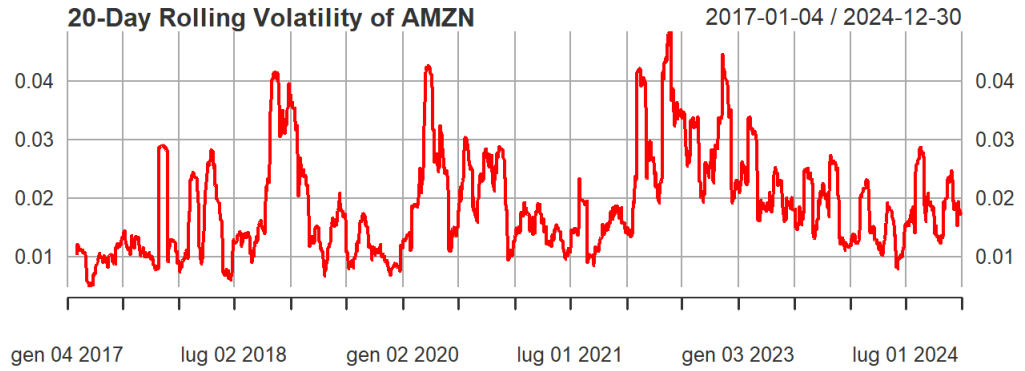
To prepare the data we have computed the log returns and the volatility of Amazon's prices:

```
prices=Cl(AMZN)
returns=diff(log(prices))
returns=na.omit(returns)

plot(returns, main= "Daily Amazon Log Returns", col="violet")
```




```
#Compute 20-day volatility

volatility=rollapply(returns, width=20, FUN=sd, align="right", fill=NA)

par(mfrow = c(2, 1))
plot(returns, main = "Daily Log Returns of AMZN", col = "blue")
plot(volatility, main = "20-Day Rolling Volatility of AMZN", col = "red")
```





In addition, we have proposed a standard GARCH model compared to the asymmetric power ARCH that takes into account the leverage effect of the returns. We recall that leverage effect is a stylized fact and it shows that the negative events impact more than the positive ones in the market, ending up in an asymmetric distribution with heavy left tail.

The standard GARCH model is defined as $r_t = \sigma^2_{t|t-1}\epsilon_t$ where $\epsilon_t = N-(0,1)$. The conditional variance is defined as $\sigma^2_{t|t-1} = \omega + \alpha r^2_{t-1} + \beta\sigma^2_{t-1|t-2}$. As we can see, it looks at the returns only by checking the magnitude, but it does not see whether the return is negative or positive. For that reason it is not really good for modelling the leverage effect. For that reason a valid choice is the APARCH, which takes into account the negative case. There are also some tests to see whether the leverage effect exists in the time series or not, the so called **sign bias** test that checks as null hypothesis if there is no asymmetry, but the main goal of this project is looking at the adaptive conformal prediction and for that reason we have assumed a priori both cases without implementing the sign bias test.

```
garch11_spec=ugarchspec(variance.model = list(model = "sGARCH", garchOrder = c(1, 1)),
mean.model = list(armaOrder = c(0, 0), include.mean = TRUE),distribution.model = "std")
garch11_fit=ugarchfit(spec = garch11_spec, data = returns)
```

Then, we have defined the apARCH(1,1) model (asymmetric power ARCH). Here the conditional variance is calculated as $\sigma^{\delta}_{t|t-1} = \omega + \alpha_i(|r_{t-i} - \theta_i r_{t-i})^{\delta} + \sum B_j\sigma^{\delta}_{t-j}$. In this model we are not taking into account the magnitude of the returns, but now we are looking at the negative returns through the hyperparameter θ; δ, instead, determines the size of the leverage effect. Higher θ and higher δ determine a huge leverage effect! θ is between -1 and 1; δ is greater than zero. When θ is -1 and δ is 2, we have a standard GARCH. Here, we have let the standard choice of R.

```
summary(returns)
```

```
##      Index              AMZN.Close
##  Min.   :2017-01-04   Min.   :-0.1513979
##  1st Qu.:2019-01-03   1st Qu.:-0.0089061
##  Median :2020-12-30   Median : 0.0012257
##  Mean   :2020-12-31   Mean   : 0.0008807
##  3rd Qu.:2022-12-28   3rd Qu.: 0.0116255
##  Max.   :2024-12-30   Max.   : 0.1269489
```

```
any(is.na(returns))
```

```
## [1] FALSE
```

```
any(!is.finite(returns))
```

```
## [1] FALSE
```

```
aparch_spec = ugarchspec(variance.model = list(model = "apARCH", garchOrder = c(1, 1)),
mean.model = list(armaOrder = c(0, 0), include.mean = TRUE), distribution.model = "norm")
aparch_fit = ugarchfit(spec = aparch_spec, data = returns)


any(is.na(sigma(garch11_fit)))
```

```
## [1] FALSE
```

```
any(!is.finite(sigma(garch11_fit)))
```

```
## [1] FALSE
```

```
any(is.na(sigma(aparch_fit)))
```

```
## [1] FALSE
```

```
any(!is.finite(sigma(aparch_fit)))
```

```
## [1] FALSE
```

```
volatility_garch = sigma(garch11_fit)
ylim_garch = range(volatility_garch, na.rm = TRUE)


volatility_aparch = sigma(aparch_fit)
ylim_aparch = range(volatility_aparch, na.rm = TRUE)

show(garch11_fit)
```

```
##
## *---------------------------------*
## *          GARCH Model Fit        *
## *---------------------------------*
##
## Conditional Variance Dynamics
## -----------------------------------
## GARCH Model  : sGARCH(1,1)
## Mean Model   : ARFIMA(0,0,0)
## Distribution : std
##
## Optimal Parameters
## ------------------------------------
##         Estimate  Std. Error  t value Pr(>|t|)
## mu       0.00142    0.000333   4.2617 0.000020
## omega    0.00001    0.000006   1.6755 0.093835
## alpha1   0.11800    0.028061   4.2052 0.000026
## beta1    0.86818    0.020572  42.2013 0.000000
## shape    5.00485    0.729871   6.8572 0.000000
##
## Robust Standard Errors:
##         Estimate  Std. Error  t value Pr(>|t|)
## mu       0.00142    0.000312  4.55795 0.000005
## omega    0.00001    0.000017  0.56539 0.571808
## alpha1   0.11800    0.075835  1.55602 0.119703
## beta1    0.86818    0.031176 27.84792 0.000000
## shape    5.00485    1.575595  3.17648 0.001491
##
## LogLikelihood : 5219.948
##
## Information Criteria
## ------------------------------------
##
## Akaike       -5.1890
## Bayes        -5.1751
## Shibata      -5.1890
## Hannan-Quinn -5.1839
##
## Weighted Ljung-Box Test on Standardized Residuals
## ------------------------------------
##                         statistic p-value
## Lag[1]                  0.0000754  0.9931
```

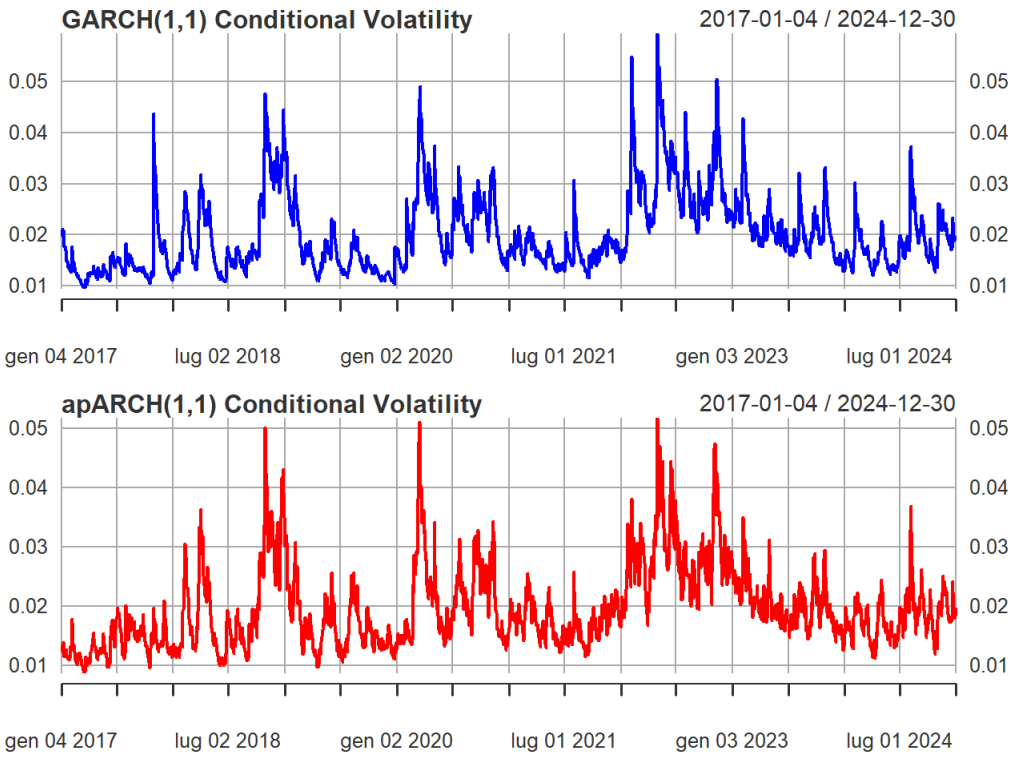
```
## Lag[1]                     0.0000754  0.9931
## Lag[2*(p+q)+(p+q)-1][2] 0.8178584  0.5613
## Lag[4*(p+q)+(p+q)-1][5] 1.6598060  0.7006
## d.o.f=0
## H0 : No serial correlation
##
## Weighted Ljung-Box Test on Standardized Squared Residuals
## ------------------------------------
##                         statistic p-value
## Lag[1]                    0.02944  0.8638
## Lag[2*(p+q)+(p+q)-1][5]   0.23626  0.9897
## Lag[4*(p+q)+(p+q)-1][9]   0.85717  0.9914
## d.o.f=2
##
## Weighted ARCH LM Tests
## ------------------------------------
##             Statistic Shape Scale P-Value
## ARCH Lag[3]    0.1137 0.500 2.000  0.7360
## ARCH Lag[5]    0.2887 1.440 1.667  0.9432
## ARCH Lag[7]    0.6184 2.315 1.543  0.9664
##
## Nyblom stability test
## ------------------------------------
## Joint Statistic:  10.4044
## Individual Statistics:
## mu     0.2494
## omega  0.6463
## alpha1 0.5383
## beta1  1.2284
## shape  0.7475
##
## Asymptotic Critical Values (10% 5% 1%)
## Joint Statistic:          1.28 1.47 1.88
## Individual Statistic:     0.35 0.47 0.75
##
## Sign Bias Test
## ------------------------------------
##                    t-value   prob sig
## Sign Bias          0.895692 0.3705
## Negative Sign Bias 0.004189 0.9967
## Positive Sign Bias 0.357950 0.7204
## Joint Effect       1.132625 0.7692
##
##
## Adjusted Pearson Goodness-of-Fit Test:
## ------------------------------------
##   group statistic p-value(g-1)
## 1    20     17.40       0.5626
## 2    30     27.07       0.5677
## 3    40     34.78       0.6629
## 4    50     45.77       0.6048
##
##
## Elapsed time : 1.804338
```

```
show(aparch_fit)
```

```
##
## *----------------------------------*
## *          GARCH Model Fit         *
## *----------------------------------*
##
## Conditional Variance Dynamics
## ------------------------------------
## GARCH Model  : apARCH(1,1)
## Mean Model   : ARFIMA(0,0,0)
## Distribution : norm
##
## Optimal Parameters
## ------------------------------------
##         Estimate  Std. Error  t value Pr(>|t|)
## mu      0.001016    0.000290   3.4990 0.000467
## omega   0.002550    0.001352   1.8862 0.059273
## alpha1  0.120998    0.018226   6.6386 0.000000
## beta1   0.862929    0.021116  40.8659 0.000000
## gamma1  0.469520    0.087909   5.3410 0.000000
## delta   0.749591    0.135610   5.5275 0.000000
```

```
##
## Robust Standard Errors:
##         Estimate  Std. Error  t value Pr(>|t|)
## mu      0.001016    0.000255   3.9818 0.000068
## omega   0.002550    0.001712   1.4893 0.136407
## alpha1  0.120998    0.027204   4.4478 0.000009
## beta1   0.862929    0.031250  27.6135 0.000000
## gamma1  0.469520    0.180568   2.6002 0.009316
## delta   0.749591    0.161586   4.6390 0.000004
##
## LogLikelihood : 5153.216
##
## Information Criteria
## ------------------------------------
##
## Akaike       -5.1216
## Bayes        -5.1049
## Shibata      -5.1216
## Hannan-Quinn -5.1155
##
## Weighted Ljung-Box Test on Standardized Residuals
## ------------------------------------
##                         statistic p-value
## Lag[1]                    0.01454  0.9040
## Lag[2*(p+q)+(p+q)-1][2]   0.59583  0.6503
## Lag[4*(p+q)+(p+q)-1][5]   1.58273  0.7195
## d.o.f=0
## H0 : No serial correlation
##
## Weighted Ljung-Box Test on Standardized Squared Residuals
## ------------------------------------
##                         statistic p-value
## Lag[1]                    0.04964  0.8237
## Lag[2*(p+q)+(p+q)-1][5]   0.21008  0.9918
## Lag[4*(p+q)+(p+q)-1][9]   0.74102  0.9946
## d.o.f=2
##
## Weighted ARCH LM Tests
## ------------------------------------
##             Statistic Shape Scale P-Value
## ARCH Lag[3]   0.08832 0.500 2.000  0.7663
## ARCH Lag[5]   0.14609 1.440 1.667  0.9776
## ARCH Lag[7]   0.46488 2.315 1.543  0.9815
##
## Nyblom stability test
## ------------------------------------
## Joint Statistic:  1.4461
## Individual Statistics:
## mu     0.26114
## omega  0.07305
## alpha1 0.09039
## beta1  0.10216
## gamma1 0.35286
## delta  0.07269
##
## Asymptotic Critical Values (10% 5% 1%)
## Joint Statistic:          1.49 1.68 2.12
## Individual Statistic:     0.35 0.47 0.75
##
## Sign Bias Test
## ------------------------------------
##                    t-value   prob sig
## Sign Bias           0.4482 0.6541
## Negative Sign Bias  0.1409 0.8880
## Positive Sign Bias  0.9667 0.3338
## Joint Effect        0.9739 0.8076
##
##
## Adjusted Pearson Goodness-of-Fit Test:
## ------------------------------------
##   group statistic p-value(g-1)
## 1    20     60.53    3.193e-06
## 2    30     72.87    1.217e-05
## 3    40     81.70    7.433e-05
## 4    50     95.87    7.161e-05
##
##
## Elapsed time : 19.54339
```

```
par(mfrow = c(2, 1))
plot(sigma(garch11_fit), main = "GARCH(1,1) Conditional Volatility", col = "blue", type = "l", ylim = ylim_garch)
plot(sigma(aparch_fit), main = "apARCH(1,1) Conditional Volatility", col = "red", type = "l", ylim = ylim_aparch)
```



After fitting both GARCH(1,1) and apARCH(1,1) model and plotted their volatilities we implement the Non-adaptive conformal prediction strategy with the residual scores from the model. First we extract the residuals:

```
resd_garch= residuals(garch11_fit, standardize = FALSE)
resd_aparch=residuals(aparch_fit, standardize = FALSE)

sigma_garch =sigma(garch11_fit)
sigma_aparch = sigma(aparch_fit)
```

Then for both models we compute:

- Simple non-normalized absolute residuals $|e_t|$
- Studentized normalized residuals: $|e_t| / \sigma_t$

```
abs_res_garch=abs(resd_garch)
norm_res_garch=abs_res_garch / sigma_garch

abs_res_aparch=abs(resd_aparch)
norm_res_aparch=abs_res_aparch / sigma_aparch
```

We use these residual scores to estimate the quantile for prediction intervals:

```
alpha=0.05

q_abs_garch=quantile(abs_res_garch, probs=1 - alpha)

q_norm_garch=quantile(norm_res_garch, probs= 1- alpha)


q_abs_aparch=quantile(abs_res_aparch, probs=1 - alpha)

q_norm_aparch=quantile(norm_res_aparch, probs= 1- alpha)

#GARCH Prediction intervals
obs_ret=returns[-1]

low_garch_abs= rep(-q_abs_garch, length(obs_ret))
up_garch_abs = rep(q_abs_garch, length(obs_ret))

low_garch_norm= -q_norm_garch*sigma_garch
up_garch_norm=q_norm_garch*sigma_garch

low_garch_norm=low_garch_norm[-1]
up_garch_norm=up_garch_norm[-1]


#apARCH
low_aparch_abs= rep(-q_abs_aparch, length(obs_ret))
up_aparch_abs= rep(q_abs_aparch, length(obs_ret))

low_aparch_norm=-q_norm_aparch*sigma_aparch
up_aparch_norm=q_norm_garch*sigma_aparch

up_aparch_norm=up_aparch_norm[-1]

if (length(low_aparch_norm) != length(obs_ret)) {
  low_aparch_norm = low_aparch_norm[-length(low_aparch_norm)]
}
if (length(up_aparch_norm) != length(obs_ret)) {
  up_garch_norm = up_garch_norm[-length(up_garch_norm)]
}
```

Let's check if the observed returns are covered by the intervals.

```
obs_ret=returns[-1]

#GARCH
cover_garch_abs= mean(obs_ret>=low_garch_abs&obs_ret<=up_garch_abs)

cover_garch_norm= mean( obs_ret>=low_garch_norm&obs_ret<=up_garch_norm)

#apARCH
cover_aparch_abs= mean( obs_ret>=low_aparch_abs&obs_ret<=up_aparch_abs)

cover_aparch_norm= mean( obs_ret>=low_aparch_norm&obs_ret<=up_aparch_norm)

cat("GARCH(1,1) Non-Normalized Coverage:", cover_garch_abs, "\n")
```

```
## GARCH(1,1) Non-Normalized Coverage: 0.9497262
```

```
cat("GARCH(1,1) Normalized Coverage:", cover_garch_norm, "\n")
```

```
## GARCH(1,1) Normalized Coverage: 0.9512195
```

```
cat("apARCH(1,1) Non-Normalized Coverage:", cover_aparch_abs, "\n")
```

```
## apARCH(1,1) Non-Normalized Coverage: 0.9497262
```

```
cat("apARCH(1,1) Normalized Coverage:", cover_aparch_norm, "\n")
```

```
## apARCH(1,1) Normalized Coverage: 0.9477092
```

```r
library(xts)
time_index= index(returns[-1])
stopifnot(length(time_index) == length(obs_ret))


length(time_index)
```

```
## [1] 2009
```

```r
length(obs_ret)
```

```
## [1] 2009
```

```r
length(low_garch_abs)
```

```
## [1] 2009
```

```r
length(up_garch_abs)
```

```
## [1] 2009
```

```r
length(low_garch_norm)
```

```
## [1] 2009
```

```r
length(up_garch_norm)
```

```
## [1] 2009
```

```r
length(low_aparch_abs)
```

```
## [1] 2009
```

```r
length(up_aparch_abs)
```

```
## [1] 2009
```

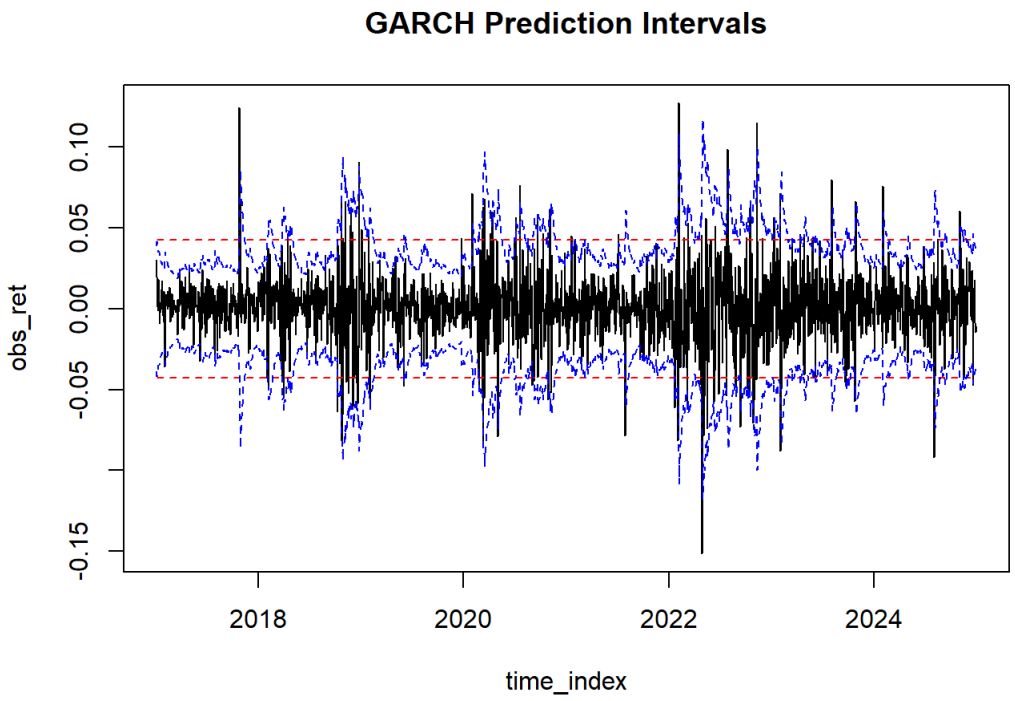
```r
length(low_aparch_norm)
```

```
## [1] 2009
```

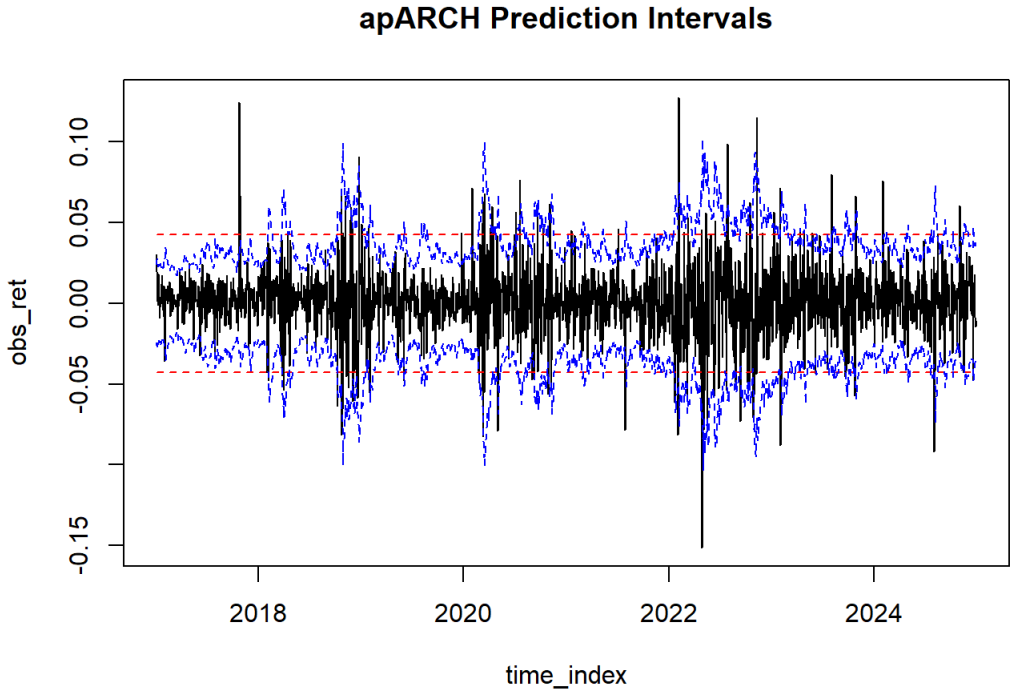
```r
length(up_aparch_norm)
```

```
## [1] 2009
```

```r
#Plot GARCH
plot(time_index, obs_ret, type = "l", col = "black", main = "GARCH Prediction Intervals")
lines(time_index, low_garch_abs, col = "red", lty = 2)
lines(time_index, up_garch_abs, col = "red", lty = 2)
lines(time_index, low_garch_norm, col = "blue", lty = 2)
lines(time_index, up_garch_norm, col = "blue", lty = 2)
```




```
#Plot apARCH
plot(time_index, obs_ret, type = "l", col = "black", main = "apARCH Prediction Intervals")
lines(time_index, low_aparch_abs, col = "red", lty = 2)
lines(time_index, up_aparch_abs, col = "red", lty = 2)
lines(time_index, low_aparch_norm, col = "blue", lty = 2)
lines(time_index, up_aparch_norm, col = "blue", lty = 2)
```



In both cases we have achieved good intervals, especially for the apARCH case. The normalized scores work in an adaptive way and they have reached the conditionl coverage as well. In same cases the interval is too large: we know the larger is the interval, the less informative is the prediction! In order to achieve the efficiency, we need to restrict the bounds around the time series.

Point 1.2 , 1.3 , 1.4

Since we are using a time series and the data are not exchangeable, the paper suggests to use the “adaptive” conformal prediction. We are going to make the first prediction by using the classical α and the we adapt this value step by step when new data arrives. The first adaptive method is the simplest by using $\alpha_{t+1} = \alpha_t + \gamma(\alpha - \text{err})$.

The second method is the momentum, that is $\alpha_{t+1} = \alpha_t + \gamma(\alpha - \sum w_s \text{err}_s)$. \w$_s$ is a sequence of increasing weights between 0,1 with $\sum ws = 1$. In that framework we can compute the miscoverage frequency directly, without using the error indicator. We apply it for a non normalized and a normalized conformal prediction. A non-normalized conformal prediction follows the calculation done in the point 1 for both the scores and the quantile and it reaches the coverage, but that coverage is a marginal coverage. Instead, the normalized one performs the calculation of the scores by normalizing it with the variance and then also the quantile are calculated by multiplying them to the variance. In mathematical words the

predictor is always $\hat{f}_n$, then the scores are evaluated as $S_t = \frac{|V_t^2 - \hat{f}_n|}{\sigma^2}$. The confidence interval is calculated as $\hat{C}_{1-\alpha, n}=\hat{f}_n +/- \hat{Q}_{1-\alpha, n} * \sigma^2$.

Data preparation Let:

- P_t denote the daily open price of a stock at time t.
- $R_t = \frac{P_t - P_{t-1}}{P_{t-1}}$ be the return at time t.

- $V_t = R_t^2$ be the realized volatility at time t.

The GARCH(1,1) model is defined as: $R_t = \sigma_t \epsilon_t$ with $\epsilon_t \sim N(0, 1)$ $\sigma_t^2 = \omega + \tau V_{t-1} + \beta \sigma_{t-1}^2$ where σ_t^2 represents the conditional variance at time t, and ω, τ, β are the model parameters.

To account for shifting market dynamics, the GARCH(1,1) model is re-fitted every day using the most recent 1250 trading days (approximately 5 years of data).

Our point prediction for the realized volatility at time t is: $\hat{\sigma}_t^2 = \hat{\omega}_t + \hat{\tau}_t V_{t-1} + \hat{\beta}_t \hat{\sigma}_{t-1}^2$

R application

We have defined a general function with different arguments:

1. returns of our choice;
2. α and γ;
3. the horizon of our estimation;
4. look back are the number of steps back that we perform;
5. model_spec and score_type are the ones that split the normalized to the non-normalized and the standard GARCH and the asymmetric GARCH;
6. garchP is the arch order and garchQ is the GARCH order (in order to avoid misunderstanding connected to the order);
7. startUp is the starting point of our prediction;
8. updateMethod is used to exploit which method we are going to use to adapt our α.

We have not performed a split in the data (train and calibration data) because the paper suggested to use the entire data set, but we recall that there is the possibility to split the data in order to use the split conformal prediction, even for the returns.

For each step we have included a comment inside the box.

```
garchConformalForecasting <- function(
    returns,
    alpha = 0.05,
    gamma = 0.001,
    lookback = 1250,
    garchP = 1,
    garchQ = 1,
    startUp = 100,
    verbose = FALSE,
    updateMethod = "Momentum",
    momentumBW = 0.95,
    scoreType = "normalized", # Options: "normalized", "non-normalized"
    modelType = "sGARCH",     # Options: "sGARCH", "apARCH"
    forecastSteps = 1         # Number of forecasting steps ahead
) {
  myT <- length(returns)
  T0 <- max(startUp, lookback)

  # Specify GARCH model based on user input
  garchSpec <- ugarchspec(
    mean.model = list(armaOrder = c(0, 0), include.mean = FALSE),
    variance.model = list(model = modelType, garchOrder = c(garchP, garchQ)),
    distribution.model = "norm"
  )

  alphat <- alpha

  # Initialize data storage variables
  errSeqOC <- rep(0, myT - T0 + 1)
  errSeqNC <- rep(0, myT - T0 + 1)
  alphaSequence <- rep(alpha, myT - T0 + 1)
  scores <- rep(0, myT - T0 + 1)
  low_PI <- rep(0, myT - T0 + 1)
  high_PI <- rep(0, myT - T0 + 1)
```

```
 for (t in T0:myT) {
  if (verbose) {
   print(t)
  }

  # Fit GARCH model
  garchFit <- ugarchfit(garchSpec, returns[(t - lookback + 1):(t - 1)], solver = "hybrid")
  garchForecast <- ugarchforecast(garchFit, n.ahead = forecastSteps)
  sigmaNext <- sigma(garchForecast)

  # Compute scores based on score type
  for(i in 1:forecastSteps){
  if (scoreType == "normalized") {
   scores[t - T0 + 1] <- abs(returns[t]^2 - sigmaNext[i]^2) / sigmaNext[i]^2
  } else if (scoreType == "non-normalized") {
   scores[t - T0 + 1] <- abs(returns[t]^2 - sigmaNext[i]^2)
  }}

  recentScores <- scores[max(t - T0 + 1 - lookback + 1, 1):(t - T0)]

  # Compute errors for both methods
  errSeqOC[t - T0 + 1] <- as.numeric(scores[t - T0 + 1] > quantile(recentScores, 1 - alphat))
  errSeqNC[t - T0 + 1] <- as.numeric(scores[t - T0 + 1] > quantile(recentScores, 1 - alpha))
  sorted= sort(scores)
  quantile_c= sorted[ceiling((1-alphat)* length(sorted+1))]


  # Update alphat
  alphaSequence[t - T0 + 1] <- alphat
  if (updateMethod == "Simple") {
   alphat <- alphat + gamma * (alpha - errSeqOC[t - T0 + 1])
  } else if (updateMethod == "Momentum") {
   w <- rev(momentumBW^(1:(t - T0 + 1)))
   w <- w / sum(w)
   alphat <- alphat + gamma * (alpha - sum(errSeqOC[1:(t - T0 + 1)] * w))
  }
  if (scoreType == "non-normalized") {
   low_PI[t - T0 + 1] <- sigmaNext^2 - quantile_c
   high_PI[t - T0 + 1] <- sigmaNext^2 + quantile_c
  } else if (scoreType == "normalized") {
   low_PI[t - T0 + 1] <- sigmaNext^2 - quantile_c *sigmaNext^2
   high_PI[t - T0 + 1] <- sigmaNext^2 + quantile_c*sigmaNext^2
  }

  if (t %% 100 == 0) {
   print(sprintf("Done %g steps", t))
  }
 }


 return(list(alphaSequence = alphaSequence, errSeqOC = errSeqOC, errSeqNC = errSeqNC, scores = scores, low_PI=low_PI, high_PI=high_PI, sigmaN
ext=sigmaNext, quantile_c=quantile_c))
}
```

Basically in the function above we have used `ugarchfit` in order to fit our model: we have inserted the garchspec of above for choosing between the standard GARCH and the apARCH. Then we have performed the forecasting through `ugarchforecast()` by inserting the garchFit and the numbers ahead of the prediction.

It's time for the scores and we have performed them by subtracting the squared returns to the squared sigmaNext calculated above all in absolute value. The loop is done by looking at the number of step in `n.ahead`, in that way we are looking at the starting point of the prediction plus the `n.ahead` indicated then. In the recent scores we are going to store all the scores calculated on the `n.ahead`. In this approach is necessary the error that comes from the calculation because in an adaptive way we can change our alpha and taking more or less scores, depending on the results. So, it is time for the adaptive alpha. We have chosen the alpha-t that changes with the respect the parameter gamma and the alpha-t that changes with the respect the momentum. In order to plot the figure, we need the lower bound and the upper bound of our calculation with always a differentiation among the normalized scores and the non-normalized.

We have started with the aPARCH(1,1) with the simple method.

```
aparch_nonnorm <- garchConformalForecasting(
  returns = returns,
  alpha = 0.1,
  lookback = 1250,
  garchP = 1,
  garchQ = 1,
  gamma = 0.001,
  startUp = 100,
  scoreType = "non-normalized",
  modelType = "apARCH",
  forecastSteps = 5,
  updateMethod = "Simple"
)
```

```
## [1] "Done 1300 steps"
## [1] "Done 1400 steps"
## [1] "Done 1500 steps"
## [1] "Done 1600 steps"
## [1] "Done 1700 steps"
## [1] "Done 1800 steps"
## [1] "Done 1900 steps"
## [1] "Done 2000 steps"
```

```
aparch_norm <- garchConformalForecasting(
  returns = returns,
  alpha = 0.1,
  lookback = 1250,
  garchP = 1,
  garchQ = 1,
  gamma = 0.001,
  startUp = 100,
  scoreType = "normalized",
  modelType = "apARCH",
  forecastSteps = 5,
  updateMethod = "Simple"
)
```

```
## [1] "Done 1300 steps"
## [1] "Done 1400 steps"
## [1] "Done 1500 steps"
## [1] "Done 1600 steps"
## [1] "Done 1700 steps"
## [1] "Done 1800 steps"
## [1] "Done 1900 steps"
## [1] "Done 2000 steps"
```

Then, we have fitted the standard GARCH model.

```
garch_norm <- garchConformalForecasting(
  returns = returns,
  alpha = 0.1,
  lookback = 1250,
  garchP = 1,
  garchQ = 1,
  gamma = 0.001,
  startUp = 100,
  scoreType = "normalized",
  modelType = "sGARCH",
  forecastSteps = 5,
  updateMethod = "Simple"
)
```

```
## [1] "Done 1300 steps"
## [1] "Done 1400 steps"
## [1] "Done 1500 steps"
## [1] "Done 1600 steps"
## [1] "Done 1700 steps"
## [1] "Done 1800 steps"
## [1] "Done 1900 steps"
## [1] "Done 2000 steps"
```

```r
garch_nonnorm <- garchConformalForecasting(
  returns = returns,
  alpha = 0.1,
  lookback = 1250,
  garchP = 1,
  garchQ = 1,
  gamma = 0.001,
  startUp = 100,
  scoreType = "non-normalized",
  modelType = "sGARCH",
  forecastSteps = 5,
  updateMethod = "Simple"
)
```

```
## [1] "Done 1300 steps"
## [1] "Done 1400 steps"
## [1] "Done 1500 steps"
## [1] "Done 1600 steps"
## [1] "Done 1700 steps"
## [1] "Done 1800 steps"
## [1] "Done 1900 steps"
## [1] "Done 2000 steps"
```

```r
aparch_norm_mom <- garchConformalForecasting(
  returns = returns,
  alpha = 0.1,
  lookback = 1250,
  garchP = 1,
  garchQ = 1,
  gamma = 0.001,
  startUp = 100,
  scoreType = "normalized",
  modelType = "apARCH",
  forecastSteps = 5,
  updateMethod = "Momentum"
)
```

```
## [1] "Done 1300 steps"
## [1] "Done 1400 steps"
## [1] "Done 1500 steps"
## [1] "Done 1600 steps"
## [1] "Done 1700 steps"
## [1] "Done 1800 steps"
## [1] "Done 1900 steps"
## [1] "Done 2000 steps"
```

Once we have performed the application of the different models, we need to understand which one has performed well. The paper suggests to calculate the local coverage, since our aim is always $P(Y_t \in \hat{C}_t \geq 1 - \alpha)$

To evaluate the performance of these methods over time, we compute their **local coverage frequencies**. This metric represents the average coverage rate over the most recent two years, providing a measure of how well the method maintains its target coverage in the short term.

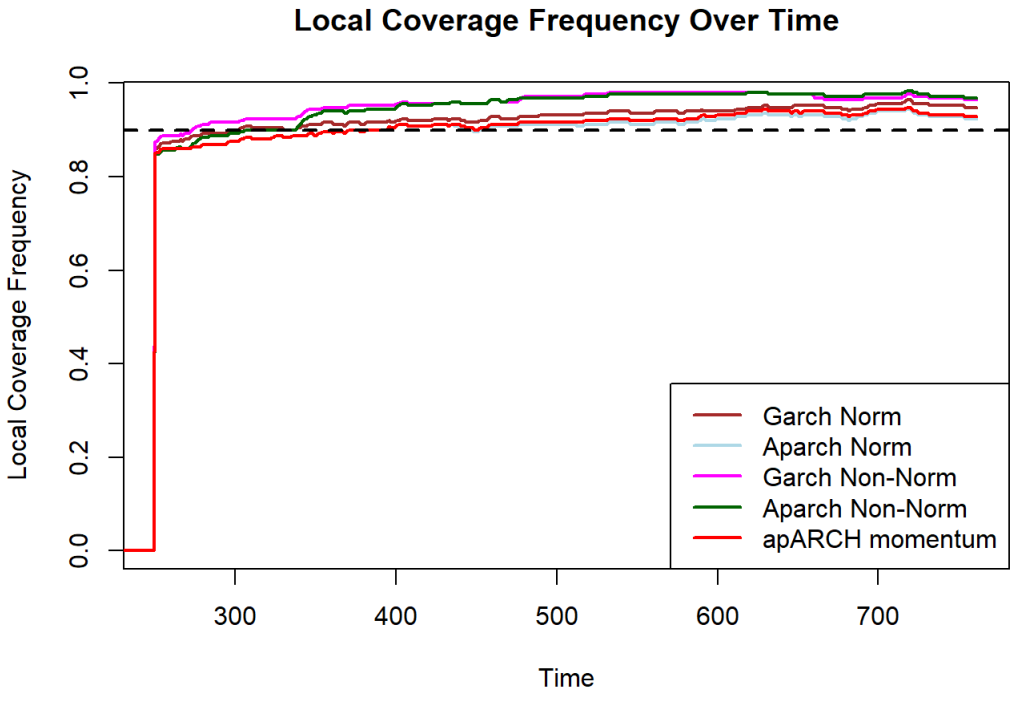
The local coverage frequency at time t, denoted as localCov_t, is defined as: $\text{localCov}_t := 1 - \frac{1}{150} \sum_{r=t-250+1}^{t+250} \text{err}_r$

Here: - $ _r$ is the error term or deviation from the target coverage at time r, - The summation spans the most recent two years, assuming 250 trading days per year (i.e., 500 days total).

If the methods perform well, the local coverage frequency should remain close to the target value of $1 - \alpha$ across all time points. This ensures that the algorithm achieves consistent coverage over time, aligning with the specified level of confidence.

The local coverage frequency is crucial for assessing the temporal stability of a method's performance. By focusing on recent data, this metric provides insights into how effectively the method adapts to changing conditions or maintains its expected coverage level.

```r
calculateLocalCoverage <- function(errSeqOC, windowSize = 250) {
  n <- length(errSeqOC)
  localCoverage <- numeric(n)

  for (t in seq(windowSize, n)) {
    localCoverage[t] <- 1 - sum(errSeqOC[(t - windowSize + 1):t]) / windowSize
  }

  return(localCoverage)
}

# Apply the function to different sequences
localCovt_garch_norm <- calculateLocalCoverage(garch_norm$errSeqOC, windowSize = 250)
localCovt_aparch_norm <- calculateLocalCoverage(aparch_norm$errSeqOC, windowSize = 250)
localCovt_garch_nonnorm <- calculateLocalCoverage(garch_nonnorm$errSeqOC, windowSize = 250)
localCovt_aparch_nonnorm <- calculateLocalCoverage(aparch_nonnorm$errSeqOC, windowSize = 250)
localCovt_aparch_mom <- calculateLocalCoverage(aparch_norm_mom$errSeqOC, windowSize = 250)

# Define the target value (1 - alpha)
target_value <- 1 - 0.10

# Plot the results
plot(localCovt_garch_norm, col = "brown", type = "l",
     main = "Local Coverage Frequency Over Time",
     xlab = "Time", ylab = "Local Coverage Frequency",
     lwd = 2, xlim = c(251, length(garch_norm$errSeqOC)))

lines(localCovt_aparch_norm, col = "lightblue", lwd = 2)
lines(localCovt_garch_nonnorm, col = "magenta", lwd = 2)
lines(localCovt_aparch_nonnorm, col = "darkgreen", lwd = 2)
lines(localCovt_aparch_mom, col = "red", lwd = 2)
# Add the target value as a horizontal line
abline(h = target_value, col = "black", lty = 2, lwd = 2)  # Target value line

# Add a legend
legend("bottomright",
       legend = c("Garch Norm", "Aparch Norm", "Garch Non-Norm", "Aparch Non-Norm", "apARCH momentum"),
       col = c("brown", "lightblue", "magenta", "darkgreen", "red"),
       lwd = c(2, 2, 2, 2,2), lty = c(1, 1, 1, 1,1))
```

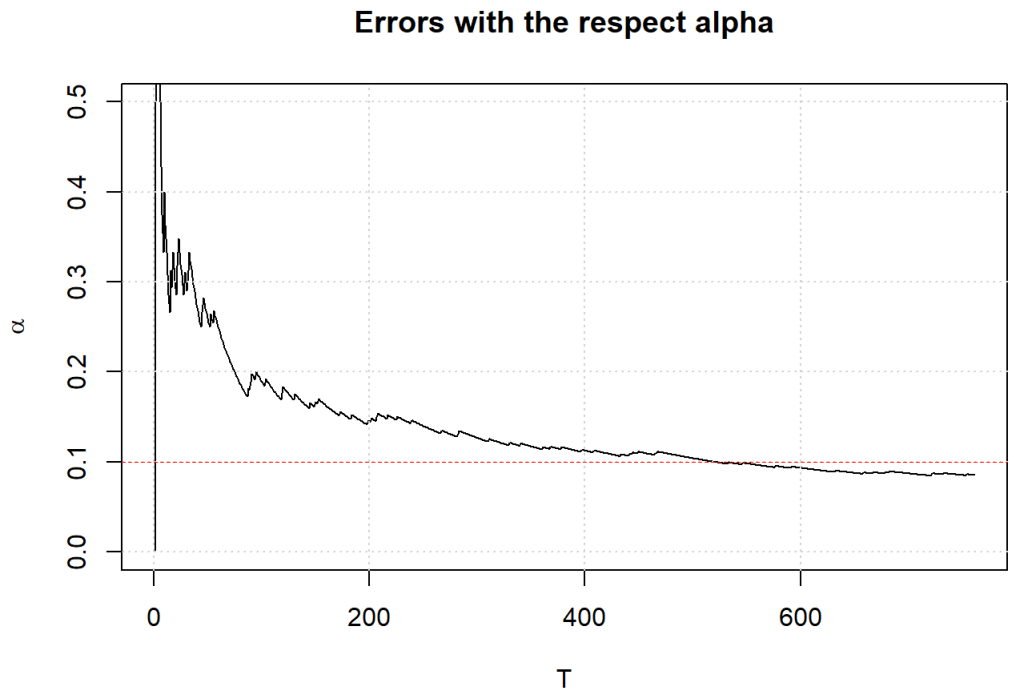


As we can see from the plot above, the apARCH normalized is the one that mantains well the stability over the time. Sharp deviations above or below 0.90 indicate periods where the coverage rate is significantly different from the target. It's important to note whether such deviations are temporary or persistent, as persistent deviations might require model tuning or a different approach. Slight fluctuations around 0.90 are often expected and may indicate that the model's error terms are well-behaved but still subject to some random noise.

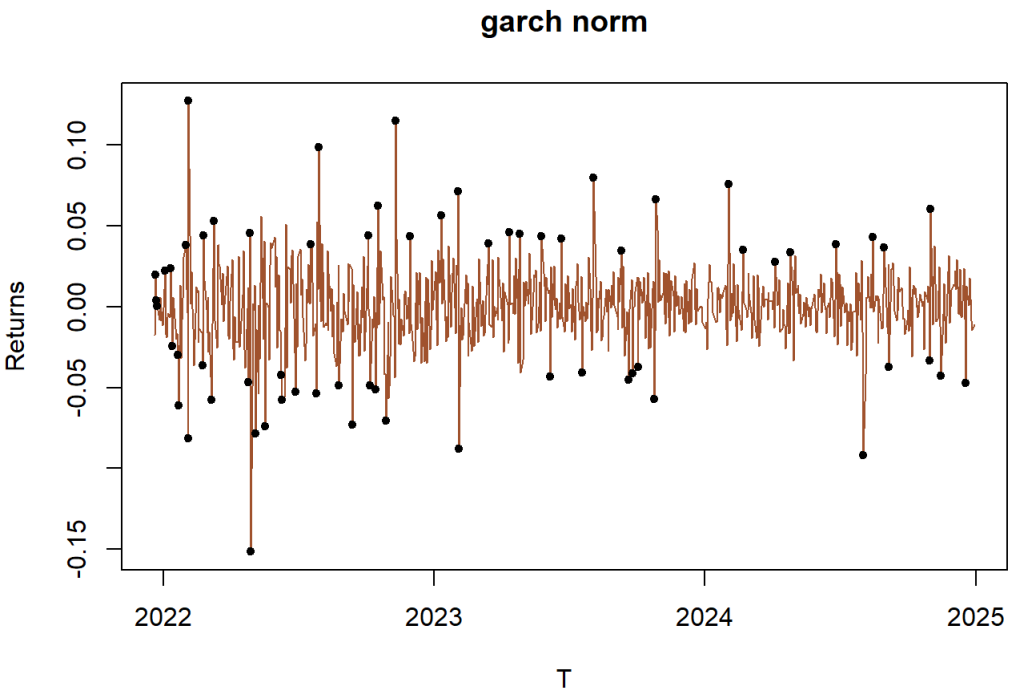
```r
date <- index(returns)[1250:length(index(returns))]
alphaSequence <- garch_norm$alphaSequence
errSeqOC <- garch_norm$errSeqOC ; mean(garch_norm$errSeqOC)
```

```
## [1] 0.08541393
```

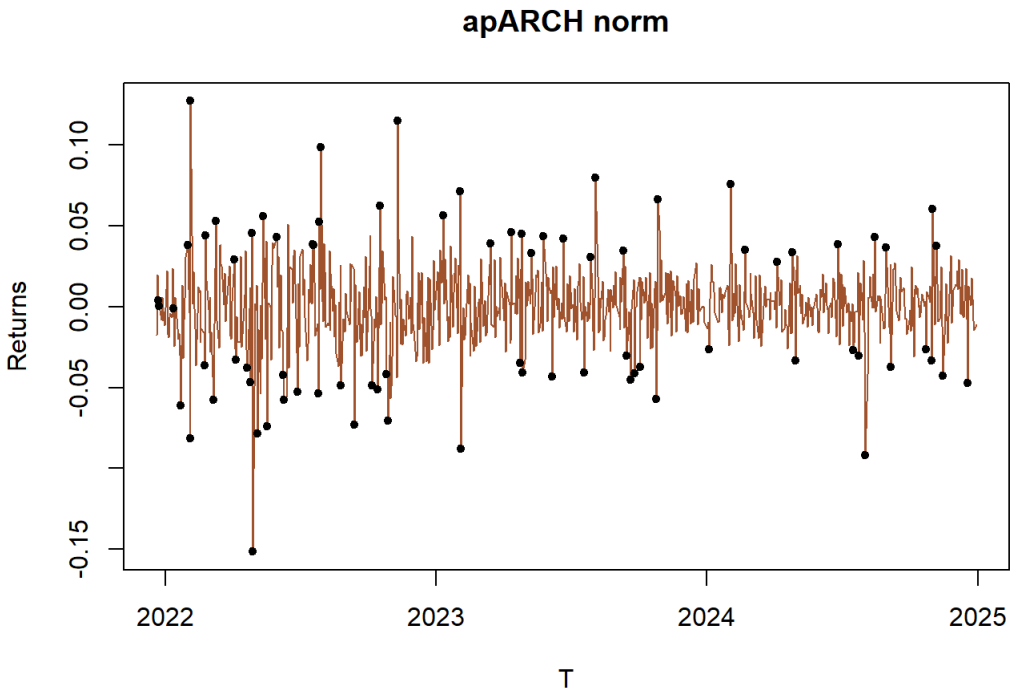
```
plot(cummean(garch_norm$errSeqOC), ylim = c(0, 0.5), type= "l" , col= "black",
    main = "Errors with the respect alpha", xlab = "T", ylab = expression(alpha));grid()
abline(h = 0.10, col = "red", lwd = 0.3, lty = 2)
```




```
plot(date, returns[1250:length(index(returns))], type= "l", col= "sienna",
    main = "garch norm", xlab = "T", ylab = "Returns")
mycol = ifelse(garch_norm$errSeqOC==0, "lightblue", "black")
mycex = ifelse(garch_norm$errSeqOC==0, 0, 1)
points(date, pch = 20, cex = mycex, as.numeric(returns[1250:length(index(returns))]), col=mycol)
```




```
plot(date, returns[1250:length(index(returns))], type= "l", col= "sienna",
    main = "apARCH norm", xlab = "T", ylab = "Returns")
mycol = ifelse(aparch_norm$errSeqOC==0, "lightblue", "black")
mycex = ifelse(aparch_norm$errSeqOC==0, 0, 1)
points(date, pch = 20, cex = mycex, as.numeric(returns[1250:length(index(returns))]), col=mycol)
```



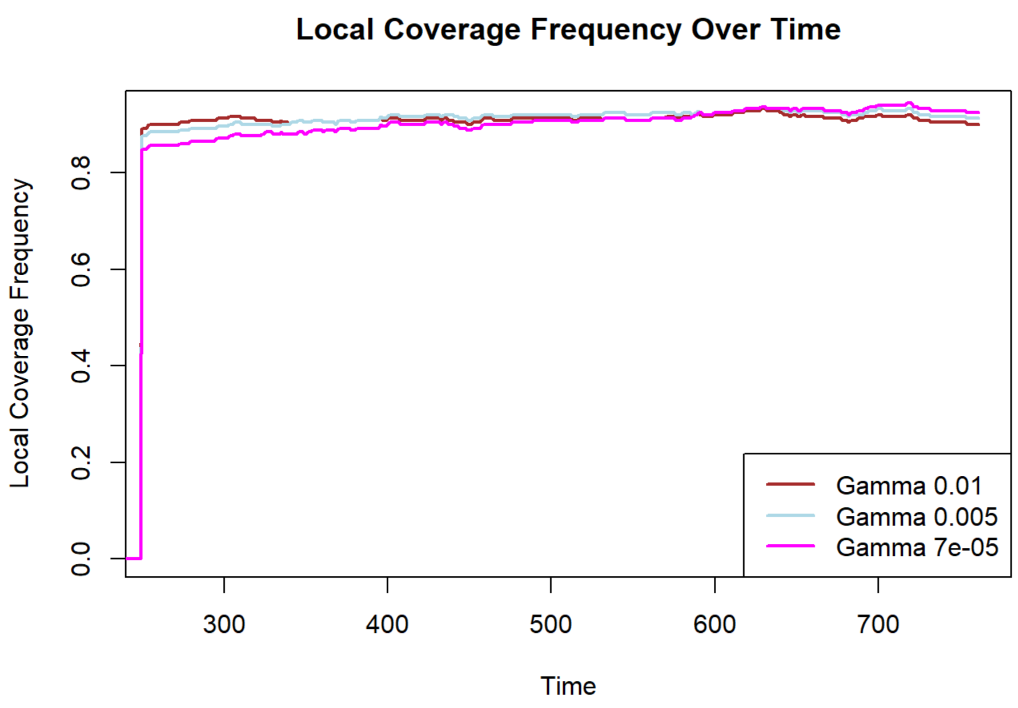
In addition from the plot above we can see how the apARCH norm can perform well with the respect the Garch norm. This is because the lower and the upper bounds are more adapted to the time series in the second case. Unfortunately, we are still not able to capture the highest picks and the lowest downturn movements! Maybe, this issue could be addressed to the adaptation rate gamma as well, because in the first applications we have considered a low value for gamma. Once we have understood the main differences of the models, we are going to perform one of it with different gammas in order to capture how it changes and if it improves with a particular gamma. The adaptation rate is chosen between 0 and 0.01. Higher is gamma, more we want to adapt our interval because of the higher abrupt changes in the time series. Otherwise, we need to adapt less.

```
gamma_list = c(0.01, 0.005, 0.00007)
results <- list()

for (x in gamma_list) {
  aparch_norm_list <- garchConformalForecasting(
    returns = returns,
    alpha = 0.1,
    lookback = 1250,
    garchP = 1,
    garchQ = 1,
    gamma = x,
    startUp = 100,
    scoreType = "normalized",
    modelType = "apARCH",
    forecastSteps = 5,
    updateMethod = "Simple"
  )

  results[[paste("gamma", x, sep = "_")]] <- aparch_norm_list
}
```

```
## [1] "Done 1300 steps"
## [1] "Done 1400 steps"
## [1] "Done 1500 steps"
## [1] "Done 1600 steps"
## [1] "Done 1700 steps"
## [1] "Done 1800 steps"
## [1] "Done 1900 steps"
## [1] "Done 2000 steps"
## [1] "Done 1300 steps"
## [1] "Done 1400 steps"
## [1] "Done 1500 steps"
## [1] "Done 1600 steps"
## [1] "Done 1700 steps"
## [1] "Done 1800 steps"
## [1] "Done 1900 steps"
## [1] "Done 2000 steps"
## [1] "Done 1300 steps"
## [1] "Done 1400 steps"
## [1] "Done 1500 steps"
## [1] "Done 1600 steps"
## [1] "Done 1700 steps"
## [1] "Done 1800 steps"
## [1] "Done 1900 steps"
## [1] "Done 2000 steps"
```

For each adaptation rate we need to calculate the coverage rate.

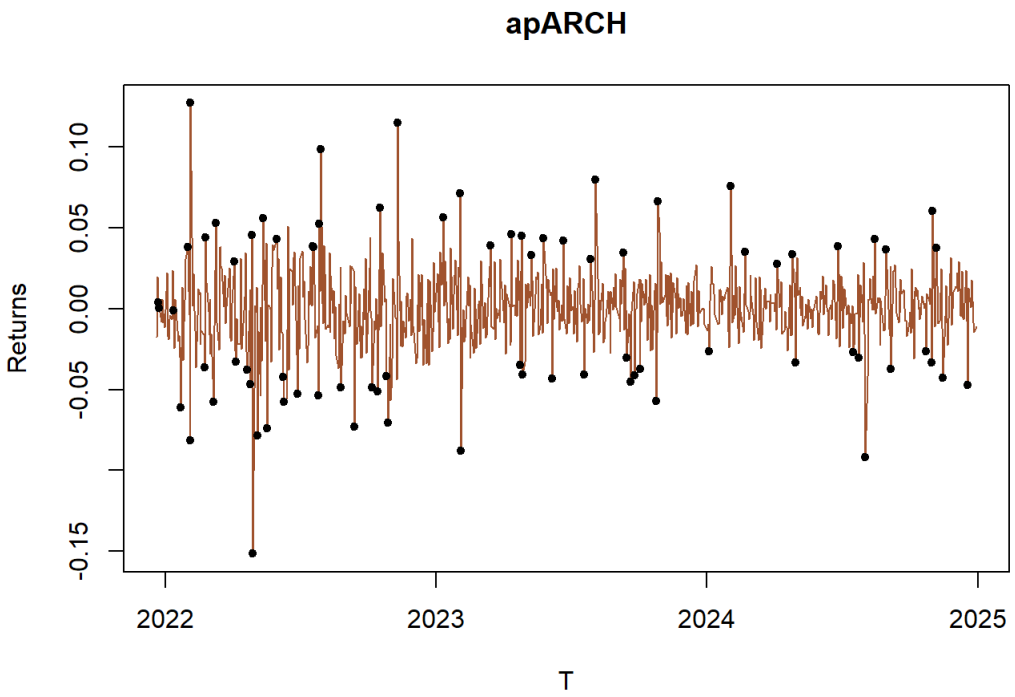
```
localCovt_first_gamma <- calculateLocalCoverage(results[["gamma_0.01"]][["errSeqOC"]], windowSize = 250)
 localCovt_second_gamma <- calculateLocalCoverage(results[["gamma_0.005"]][["errSeqOC"]], windowSize = 250)
 localCovt_third_gamma <- calculateLocalCoverage(results[["gamma_7e-05"]][["errSeqOC"]], windowSize = 250)

 target_value= 1-0.10

 # Plot the results (optional)
plot(localCovt_first_gamma, col = "brown", type = "l",
     main = "Local Coverage Frequency Over Time",
     xlab = "Time", ylab = "Local Coverage Frequency",
     lwd = 2, xlim = c(260, length(localCovt_first_gamma)))
lines(localCovt_second_gamma, col = "lightblue", lwd = 2)
lines(localCovt_third_gamma, col = "magenta", lwd = 2)
lines(target_value, col = "black", lty = 2)  # Target value line
legend("bottomright",
       legend = c("Gamma 0.01", "Gamma 0.005", "Gamma 7e-05"),
       col = c("brown", "lightblue", "magenta"),
       lwd = c(2, 2, 2), lty = c(1, 1, 1))
```



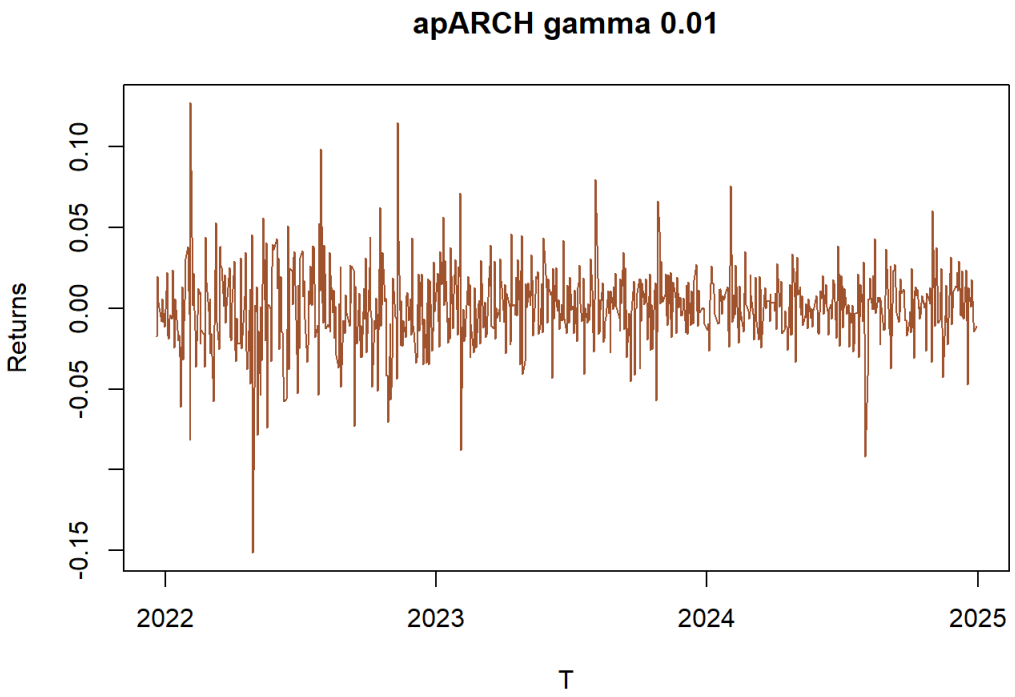
As we can see, the one that has performed better is gamma 0.01 because it remains constant over the time, having just few drops and peaks with the respect the gamma 0.005. From this plot we can understand that lower is the adaptation rate, the lower is the performance. AMZN has got abrupt changes through time and gamma is not able to capture all of them if it is low. Furthermore, when we decrease the value, we are adapting

less our intervals to the changes through time, but as we can see our time series changes a lot and that's why a high gamma is a good compromise. Just for completeness, we have plotted apARCH norm with the first gamma value and apARCH norm with the best gamma value.

```
plot(date, returns[1250:length(index(returns))], type= "l", col= "sienna",
    main = "apARCH", xlab = "T", ylab = "Returns")
mycol = ifelse(aparch_norm$errSeqOC==0, "lightblue", "black")
mycex = ifelse(aparch_norm$errSeqOC==0, 0, 1)
points(date, pch = 20, cex = mycex, as.numeric(returns[1250:length(index(returns))]), col=mycol)
```




```
plot(date, returns[1250:length(index(returns))], type= "l", col= "sienna",
    main = "apARCH gamma 0.01", xlab = "T", ylab = "Returns")
mycol = ifelse(results[["gamma_0.01"]][["errSeqOc"]]==0, "lightblue", "black")
mycex = ifelse(results[["gamma_0.01"]][["errSeqOc"]]==0, 0, 1)
points(date, pch = 20, cex = mycex, as.numeric(returns[1250:length(index(returns))]), col=mycol)
```



There is a slight difference among the two plots. Again the ACI is not able at all to cover the extreme events, but with a good adaptation rate it performs quite well.