Part I homework 2

Elisa Battista, Simone Cuonzo, Lorenzo Di Giannantonio

2025-01-10

The paper “Adaptive Conformal Inference Under Distribution Shift” by Gibbs and Candès addresses a critical challenge in modern machine learning: constructing prediction sets that maintain a desired coverage frequency when the underlying data distribution evolves over time. The authors focus on an online learning scenario, where covariate-response pairs arrive sequentially. Unlike traditional conformal inference methods that rely on the assumption of exchangeability, ACI adapts dynamically to non-stationary environments, ensuring robust performance under distributional shifts.

ACI provides a powerful and general methodology for transforming the outputs of any black-box prediction algorithm into prediction sets with valid coverage guarantees. Crucially, it is compatible with a wide range of machine learning models capable of producing either point predictions or quantile estimates, thereby enabling its application across diverse domains.

The objective is to construct a sequence of prediction sets such that for a target coverage level , the true response lies within at least % of the time. This guarantees reliable uncertainty quantification in dynamic and unpredictable environments, which is crucial in high-stakes applications such as finance.

This paper introduces ACI as a method for forming prediction sets that are inherently robust to changes in the marginal distribution of the data. The approach is both conceptually simple and practically effective. It requires tracking only a single parameter that dynamically models distributional shifts, making it computationally efficient and easy to implement. Moreover, ACI’s generality allows it to seamlessly integrate with any modern machine learning algorithm capable of providing either point predictions or estimated quantiles.

Over long time intervals, ACI achieves the desired target coverage frequency without making strong assumptions about the underlying data-generating process. Additionally, when the distribution shift is relatively small and the prediction algorithm satisfies certain regularity conditions, ACI achieves approximate marginal coverage at most individual time steps. This dual capability—ensuring long-term coverage while adapting to immediate changes—positions ACI as a crucial tool for reliable prediction in evolving data environments.

In the context of regression modeling, **Conformal Inference** provides a powerful framework for constructing prediction sets that guarantee coverage for new observations with a specified confidence level. Given a fitted regression model that predicts the value of Y from the features X, Conformal Inference aims to determine whether a candidate value y for Y_t is a reasonable estimate.

To assess the reasonableness of y, we define a **conformity score** $S(X, Y)$, which measures how well the candidate value y conforms to the model’s predictions. If the regression model provides point predictions $\hat{\mu}(X)$, the conformity score is the absolute distance between the predicted value and the candidate:

$$S(X_t, y) = |\hat{\mu}(X_t) - y|$$

Alternatively, for quantile regression models that predict quantiles of the conditional distribution of Y given X, a conformity score can be defined as the signed distance between y and the fitted quantiles:

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

where $\hat{q}(X_t; \alpha/2)$ and $\hat{q}(X_t; 1-\alpha/2)$ are the estimated lower and upper α-quantiles of Y_t, respectively.

The goal is to determine how small the conformity score $S(X_t, y)$ should be in order for y to be considered a reasonable prediction for Y_t. To this end, we employ a **calibration set** $D_{\text{cal}} \subseteq \{(X_r, Y_r)\}_{1 \le r \le t-1}$ that is distinct from the training data used to fit the model. Using this calibration set, we define the empirical quantiles of the conformity scores as:

$$\hat{Q}(p) := \inf \left\{ s : \frac{1}{|D_{\text{cal}}|} \sum_{(X_r, Y_r) \in D_{\text{cal}}} \mathbb{I}(S(X_r, Y_r) \le s) \ge p \right\}$$

where $\mathbb{I}$ denotes the indicator function, and $\hat{Q}(p)$ represents the p-th quantile of the conformity scores in the calibration set. A value y is considered a reasonable prediction for Y_t if its conformity score satisfies:

$$S(X_t, y) \le \hat{Q}(1-\alpha)$$

The key observation is that if the data $D_{\text{cal}} \cup \{(X_t, Y_t)\}$ are exchangeable, the rank of $S(X_t, Y_t)$ among the conformity scores $\{S(X_r, Y_r)\}_{(X_r, Y_r) \in D_{\text{cal}} \cup \{(X_t, Y_t)\}}$ will follow a uniform distribution. Therefore, the probability that $S(X_t, Y_t)$ is less than or equal to $\hat{Q}(1-\alpha)$ is given by:

$$P(S(X_t, Y_t) \le \hat{Q}(1-\alpha)) = \frac{\lceil |D_{\text{cal}}|(1-\alpha) \rceil}{|D_{\text{cal}}| + 1}$$

Thus, the prediction set C_t can be defined as:

$$C_t := \{y : S(X_t, y) \le \hat{Q}(1-\alpha)\}$$

which guarantees a marginal coverage probability of:

$$P(Y_t \in C_t) = P(S(X_t, Y_t) \le \hat{Q}(1-\alpha)) = \frac{\lceil |D_{\text{cal}}|(1-\alpha) \rceil}{|D_{\text{cal}}| + 1}$$

This procedure is known as **split conformal inference**, as it relies on splitting the data into a training set and a calibration set. The method can be further adjusted to work without data splitting, known as **full conformal inference**, although this requires greater computational resources.

In the previous section, we assumed a stationary distribution for the data, which allowed us to estimate a single score function $S(\cdot)$ and quantile function $\hat{Q}(\cdot)$. However, in real-world scenarios, the distribution of data may change over time. As a result, both the score function and the quantile function need to be updated regularly to align with the most recent observations.

At each time t, we assume that a fitted score function $S_t(\cdot)$ and corresponding quantile function $\hat{Q}_t(\cdot)$ are available. The **miscoverage rate** for a prediction set is the probability that the true value of the response variable Y_t does not lie within the prediction set $\hat{C}_t(\alpha)$ at time t.

At each time t, we assume that a fitted score function $S_t(\cdot)$ and corresponding quantile function $\hat{Q}_t(\cdot)$ are available. The **realized miscoverage rate** of the prediction set $\hat{C}_t(\alpha) := \{y : S_t(X_t, y) \le \hat{Q}_t(1-\alpha)\}$ is defined as:

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

where the probability is taken over the test point (X_t, Y_t) as well as the data used to fit $S_t(\cdot)$ and $\hat{Q}_t(\cdot)$.

Since the data distribution is non-stationary, we do not expect $M_t(\alpha)$ to be equal, or even close to, α. Nonetheless, we hypothesize that if the conformity scores used to fit $\hat{Q}_t(\cdot)$ cover the bulk of the distribution of $S_t(X_t, Y_t)$, then there may exist an alternative value $\alpha_t^* \in [0,1]$ such that $M_t(\alpha_t^*) \approx \alpha$.

To formalize this, we assume that, with probability one, $\hat{Q}_t(\cdot)$ is continuous and non-decreasing, with $\hat{Q}_t(0) = -1$ and $\hat{Q}_t(1) = 1$. This is not the case for the split conformal quantile functions defined previously, but in the absence of ties among the conformity scores, we can adjust our definition to ensure continuity by smoothing over any jump discontinuities in $\hat{Q}_t(\cdot)$. Under these conditions, $M_t(\alpha)$ will be non-decreasing on $[0,1]$ with $M_t(0) = 0$ and $M_t(1) = 1$, allowing us to define:

$$\alpha_t^* := \sup\{\alpha \in [0,1] : M_t(\alpha) \leq \alpha\}$$

The function $\alpha_t^* := \sup\{\alpha \in [0,1] : M_t(\alpha) \leq \alpha\}$ defines the optimal threshold α_t^* for the miscoverage rate $M_t(\alpha)$, ensuring that the miscoverage rate is as close as possible to the desired level α.

Moreover, if we assume that:

$$P(S_t(X_t, Y_t) = \hat{Q}_t(1 - \alpha_t^*)) = 0$$

This assumption is important because it ensures that the conformity score $S_t(X_t, Y_t)$ does not exactly equal the value of the conformity quantile $\hat{Q}_t(1 - \alpha_t^*)$ with positive probability.

In other words, this assumption implies that there are no "ties" (or situations of indeterminacy) where the conformity score is equal to the quantile value. If such a situation were to occur with positive probability, it would not be possible to properly calibrate the threshold for the marginal coverage.

When this assumption holds, the value α_t^* can be used to achieve exact or approximately exact marginal coverage equal to α. In practice, without this assumption, there could be uncertainty in determining the value of α_t^*, and the coverage guarantee might not be properly maintained.

To calibrate α_t^*, we use a simple **online update**. This update procedure examines the empirical miscoverage frequency of previous prediction sets and adjusts our estimate of α_t^* accordingly, by increasing or decreasing it. Specifically, let α_1 denote our initial estimate (in our experiments, we choose $\alpha_1 = \alpha$). We recursively define the sequence of miscoverage events as:

$$\mathrm{err}_t := \begin{cases} 1 & \text{if } Y_t \notin \hat{C}_t(\alpha_t) \\ 0 & \text{otherwise} \end{cases}$$

where $\hat{C}_t(\alpha_t) := \{y : S_t(X_t, y) \leq \hat{Q}_t(1 - \alpha_t)\}$. Given a step size parameter $\delta > 0$, we consider the update:

$$\alpha_{t+1} := \alpha_t + \gamma(\alpha - \mathrm{err}_t)$$

This formula defines the updating rule for α_t^*, to improve the accuracy of the coverage over time.

We refer to this as **adaptive conformal inference**, where err_t represents our estimate of the historical miscoverage frequency.

An alternative update rule can be defined as:

$$\alpha_{t+1} = \alpha_t + \gamma\left(\alpha - \sum_{s=1}^{t} w_s \cdot \mathrm{err}_s\right)$$

The weights w_s are chosen so that the sum $\sum_{s=1}^{t} w_s \cdot \mathrm{err}_s$ gives more importance to the most recent errors, reflecting the belief that errors closer to time t are more relevant for updating the threshold α_t.

A possible scheme for the weights could be:

$$w_s = \frac{0.95^{t-s}}{\sum_{s=1}^{t} 0.95^{t-s}}$$

where the weight decays exponentially as s becomes smaller, meaning that more recent errors are given higher weights.

Using this formulation for the weights, the authors have noticed that the results obtained for α_t, calculated using the two proposed update methods are very similar, the main difference is that the trajectories obtained with the second update method are smoother, with less local variation in α_t.

For simplicity, in this article, we will focus on the first update rule.

We now focus on the choice of γ, which is crucial for balancing adaptability and stability in the update process for α_t.

The value of γ represents a tradeoff: increasing γ improves the method's adaptability to observed distributional shifts but also introduces greater volatility in α_t. Excessive fluctuations in α_t can be problematic, as they may cause the method to oscillate between overly conservative and overly anti-conservative prediction sets, leading to undesirable instability.

Such instability is undesirable in practice, as it undermines the reliability of the adaptive conformal inference procedure. Therefore, selecting an appropriate value for γ is essential to maintain stable yet responsive updates to α_t.

There is evidence supporting the intuition that in environments with greater distributional shifts, the algorithm needs to be more adaptable, and thus γ should be chosen to be larger. We will consider $\gamma = 0.005$, which has been found to provide relatively stable trajectories for α_t, while still being sufficiently large to allow α_t to adapt to observed shifts. Additionally, larger values of α effectively protect against distribution shifts. On the other hand, setting α too small causes adaptive conformal inference to behave similarly to non-adaptive methods, where α_t is held constant over time.

```r
library(quantmod)
```

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

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

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

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

```
## Caricamento del pacchetto richiesto: TTR
## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
```

```
library(rugarch)
```

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

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

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

```
library(quantreg)
```

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

```
library(FinTS)
library(dplyr)
```

```
##
## ######################### 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(forecast)
```

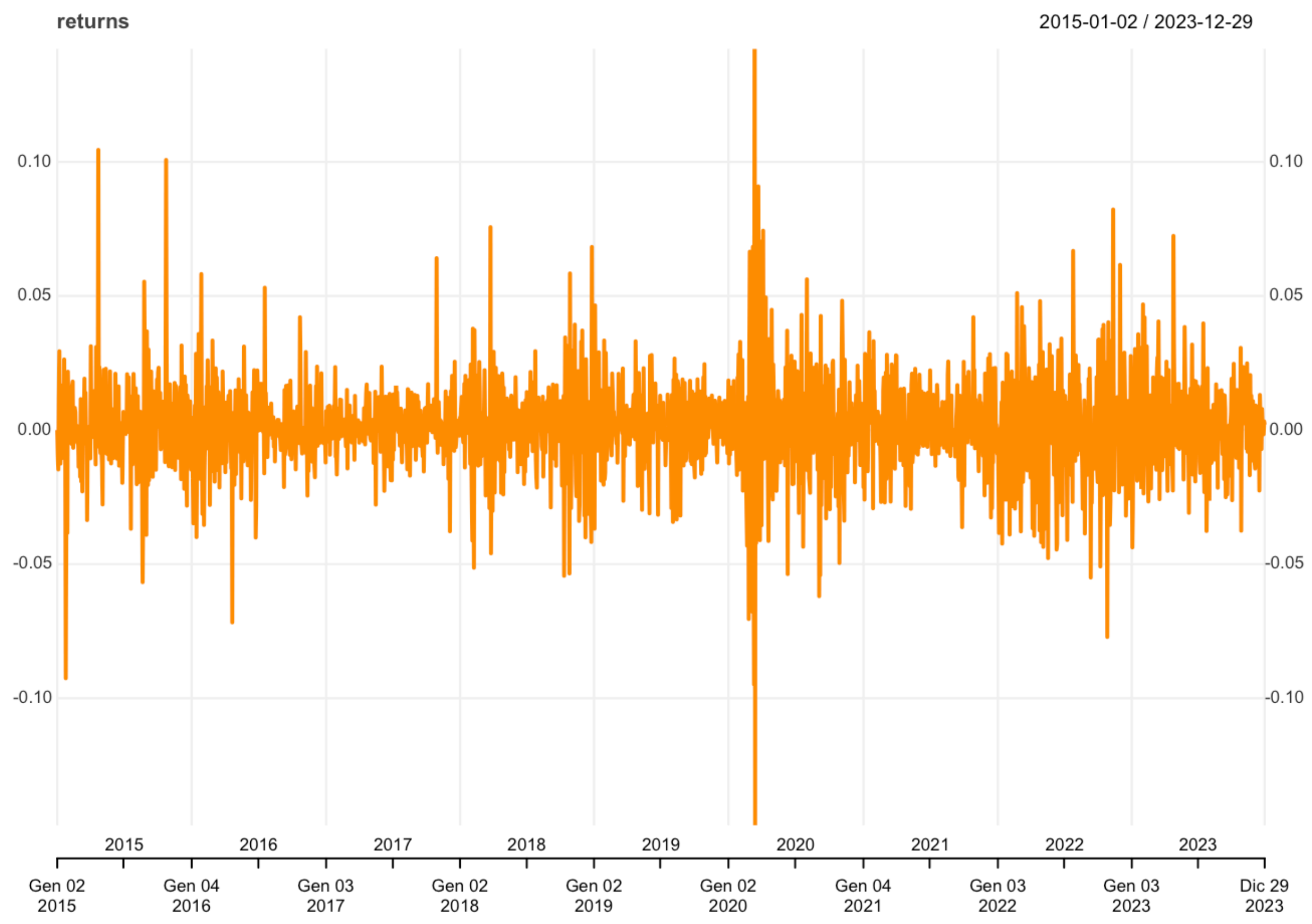
```
##
## Caricamento pacchetto: 'forecast'
```

```
## Il seguente oggetto è mascherato da 'package:FinTS':
##
##     Acf
```

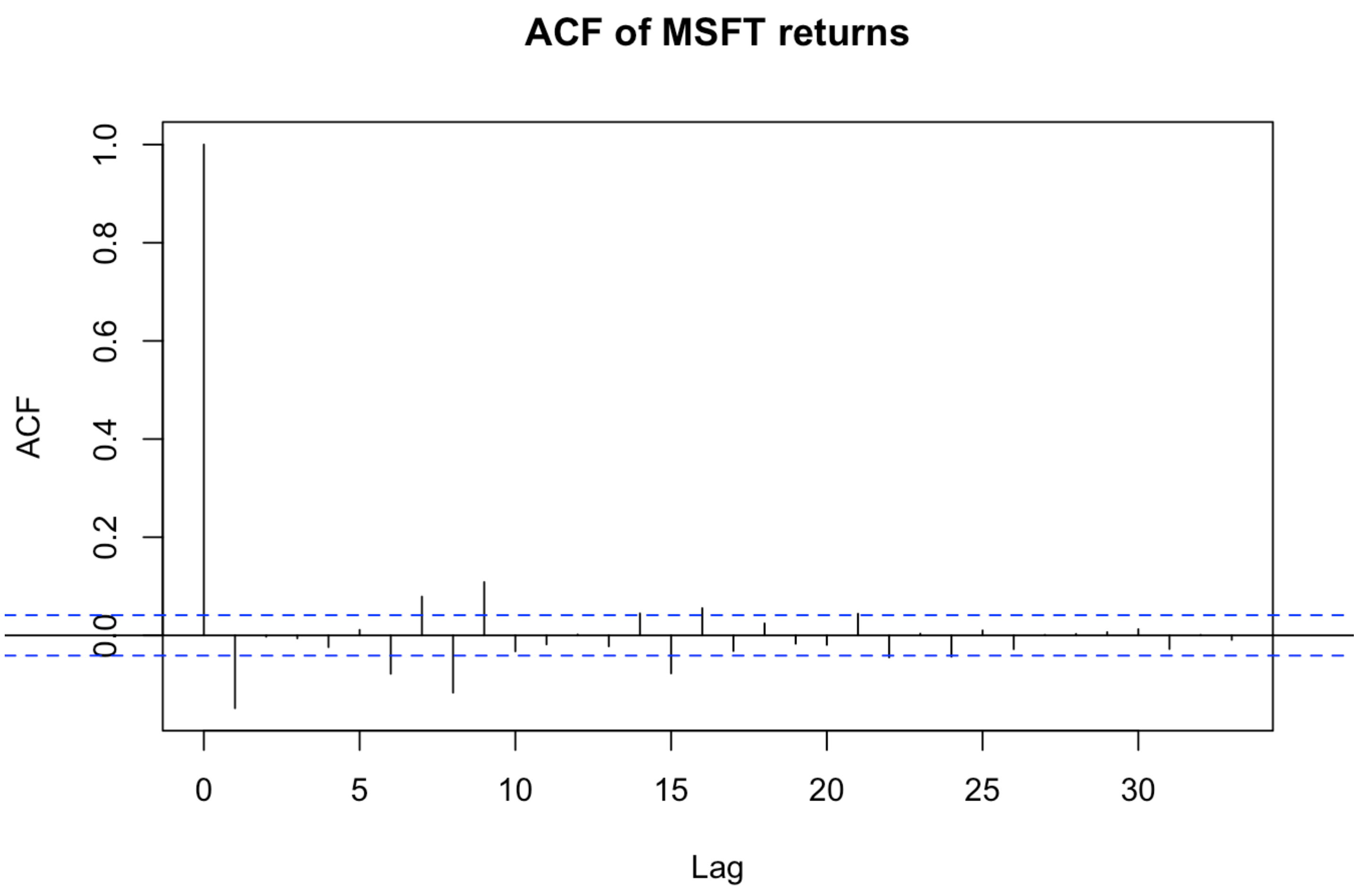
```
getSymbols("MSFT", from = "2015-01-01", to = "2023-12-31")
```

```
## [1] "MSFT"
```

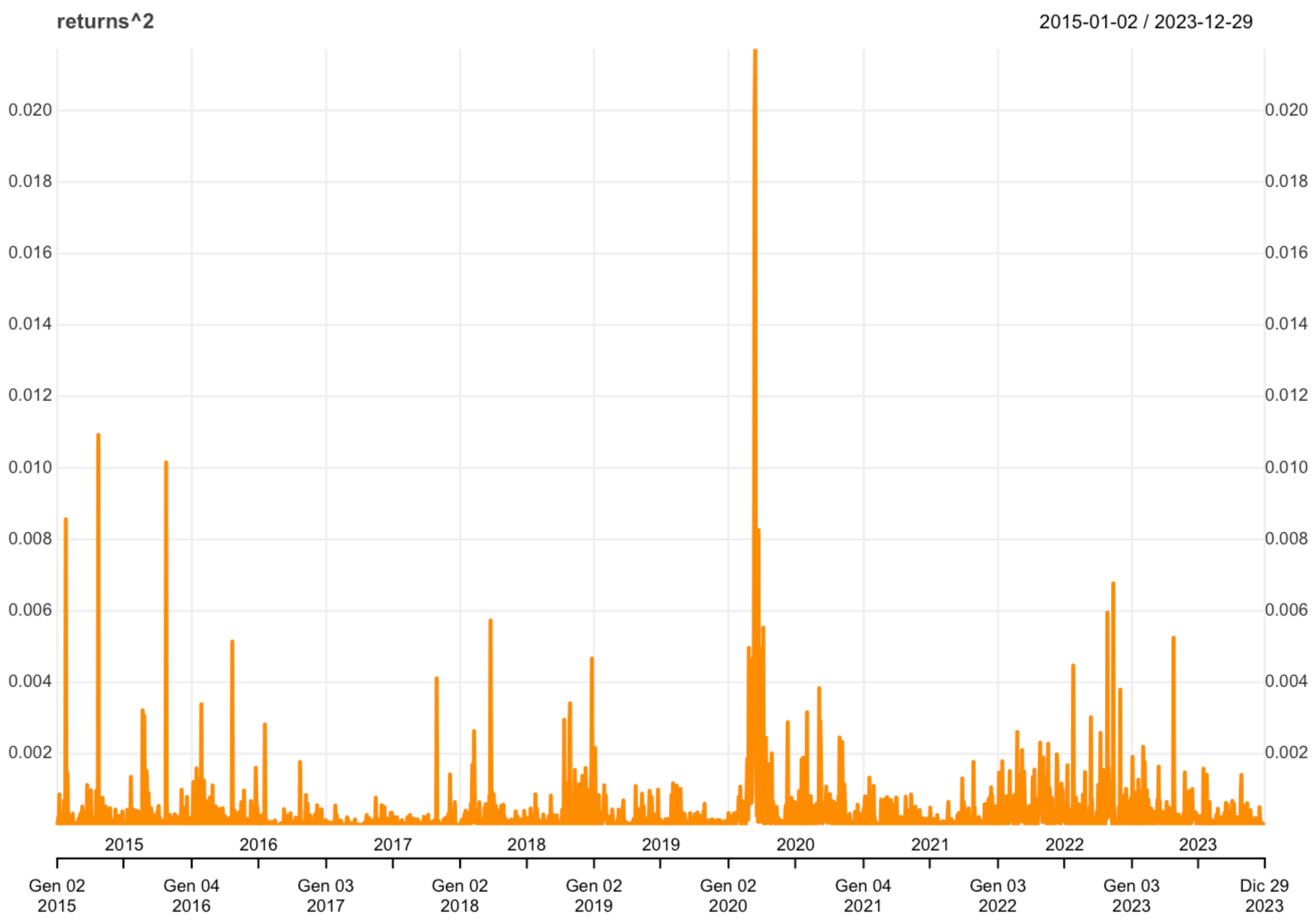
```
msft <- Cl(MSFT)     # extracting closing prices

returns <- dailyReturn(msft)
returns <- na.omit(returns)

chart_Series(returns, main = "MSFT returns")
```



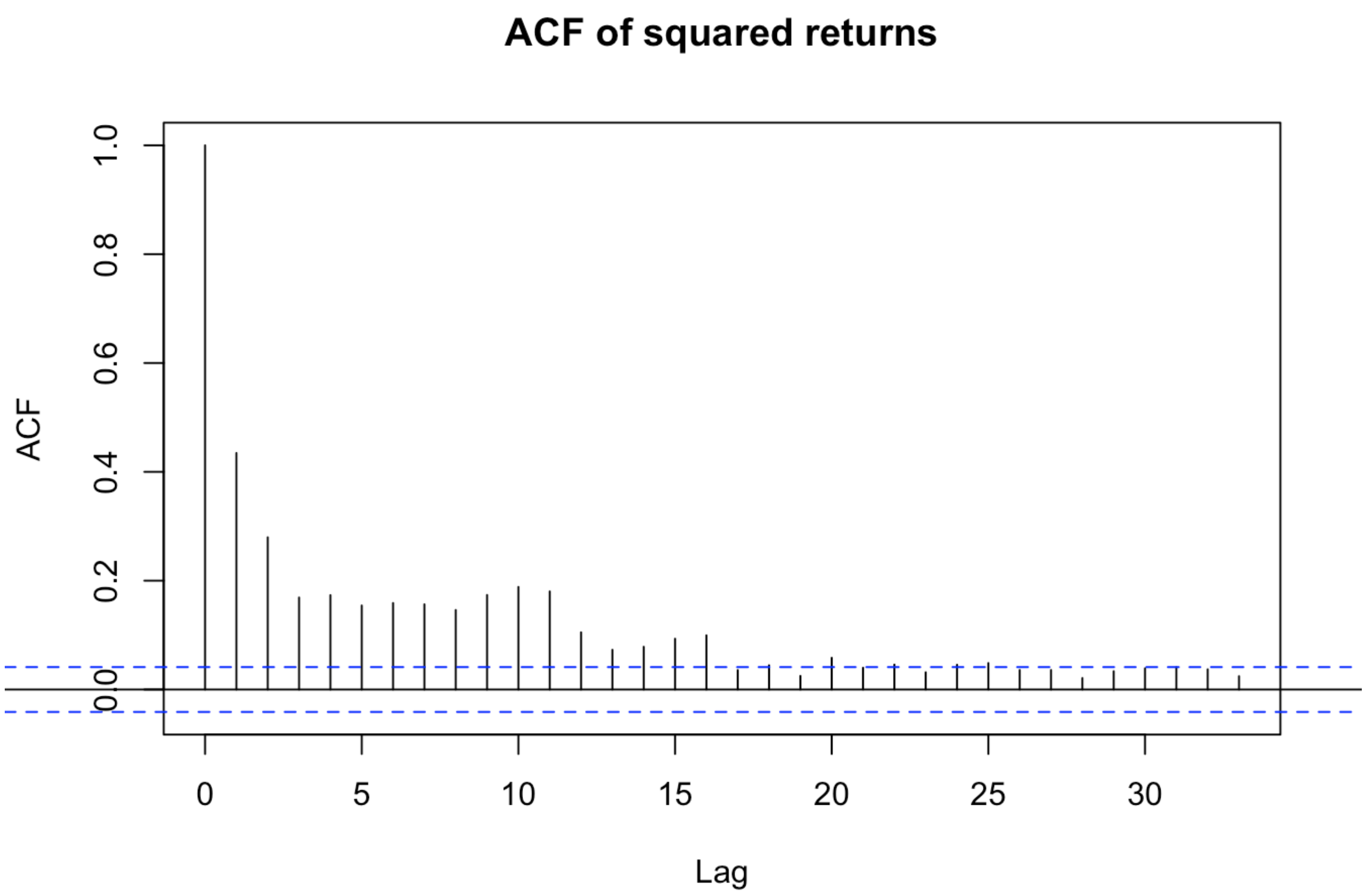
```r
acf(returns, main = "ACF of MSFT returns")
```



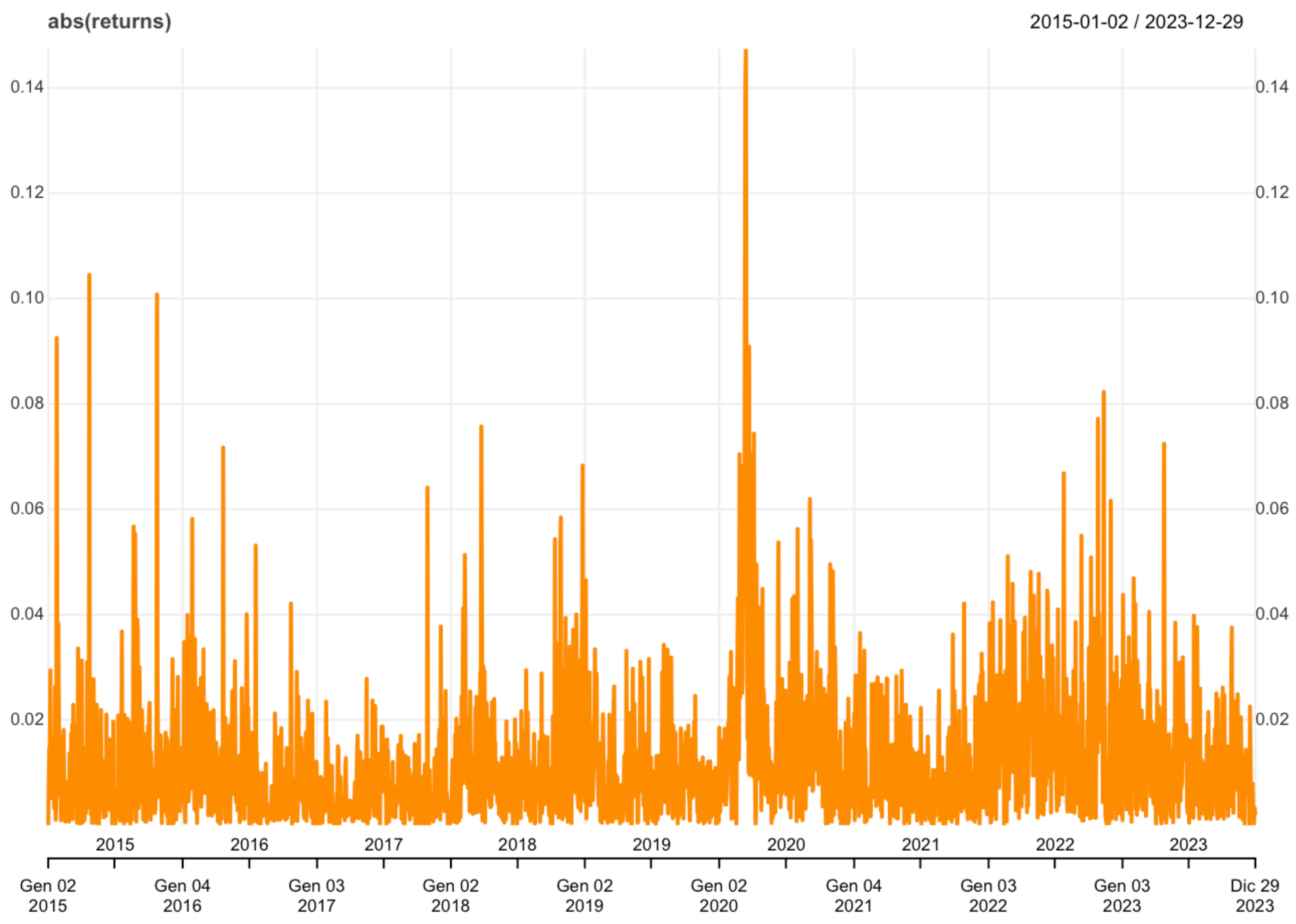
```
chart_Series(returns^2, main = "squared returns of MSFT")
```



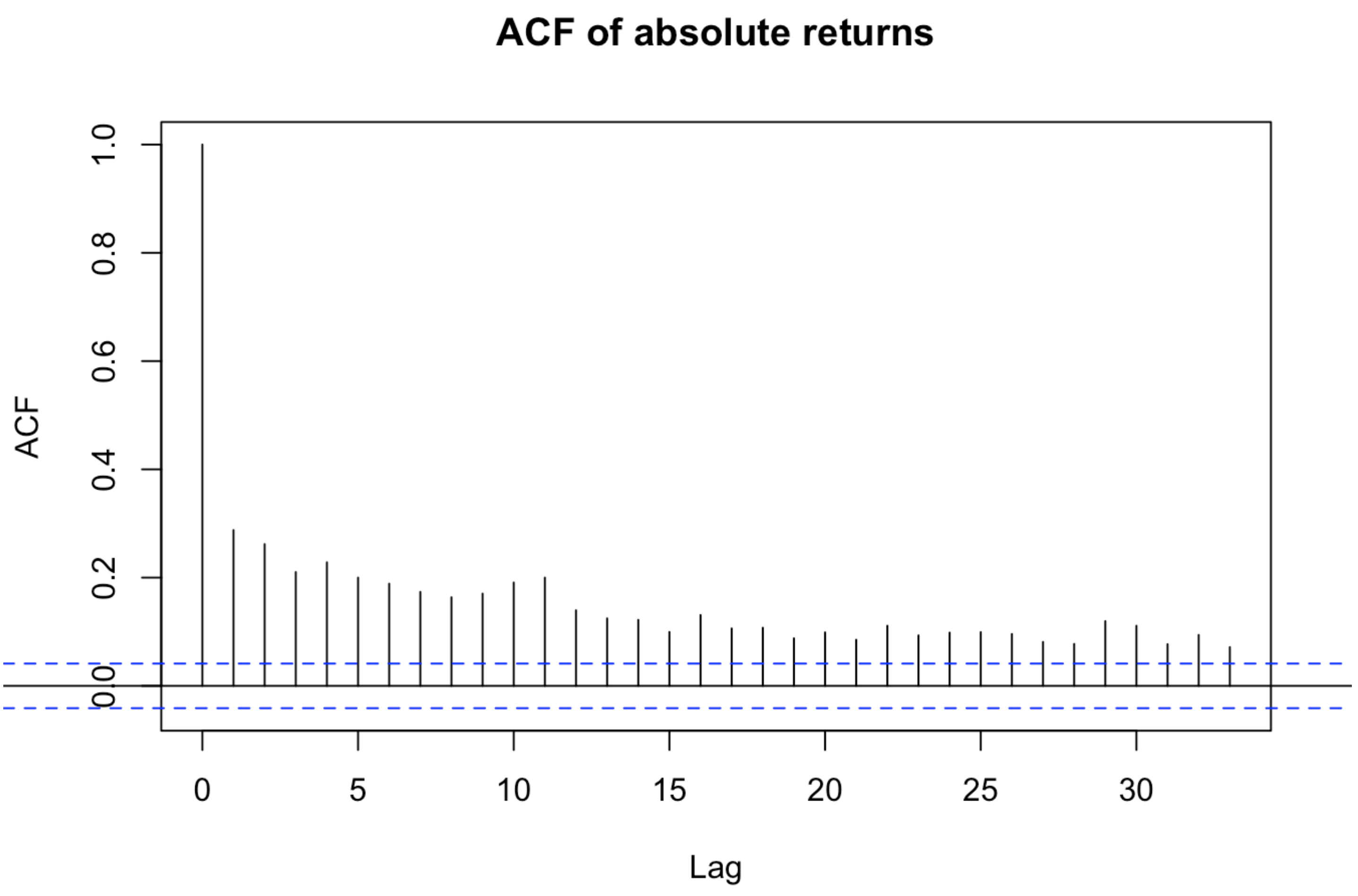
```
acf(returns^2, main = "ACF of squared returns")
```



```
chart_Series(abs(returns), main = "MSFT absolute returns")
```



```
acf(abs(returns), main = "ACF of absolute returns")
```



```
# we notice serial correlation in squared and abs returns, which indicates arch ef
fect!

Box.test(returns, type = "Ljung-Box", lag = 12)
```

```
##
##  Box-Ljung test
##
## data:  returns
## X-squared = 140.48, df = 12, p-value < 2.2e-16
```

```
Box.test(returns^2, type = "Ljung-Box", lag = 12)
```

```
##
##  Box-Ljung test
##
## data:  returns^2
## X-squared = 1203.1, df = 12, p-value < 2.2e-16
```

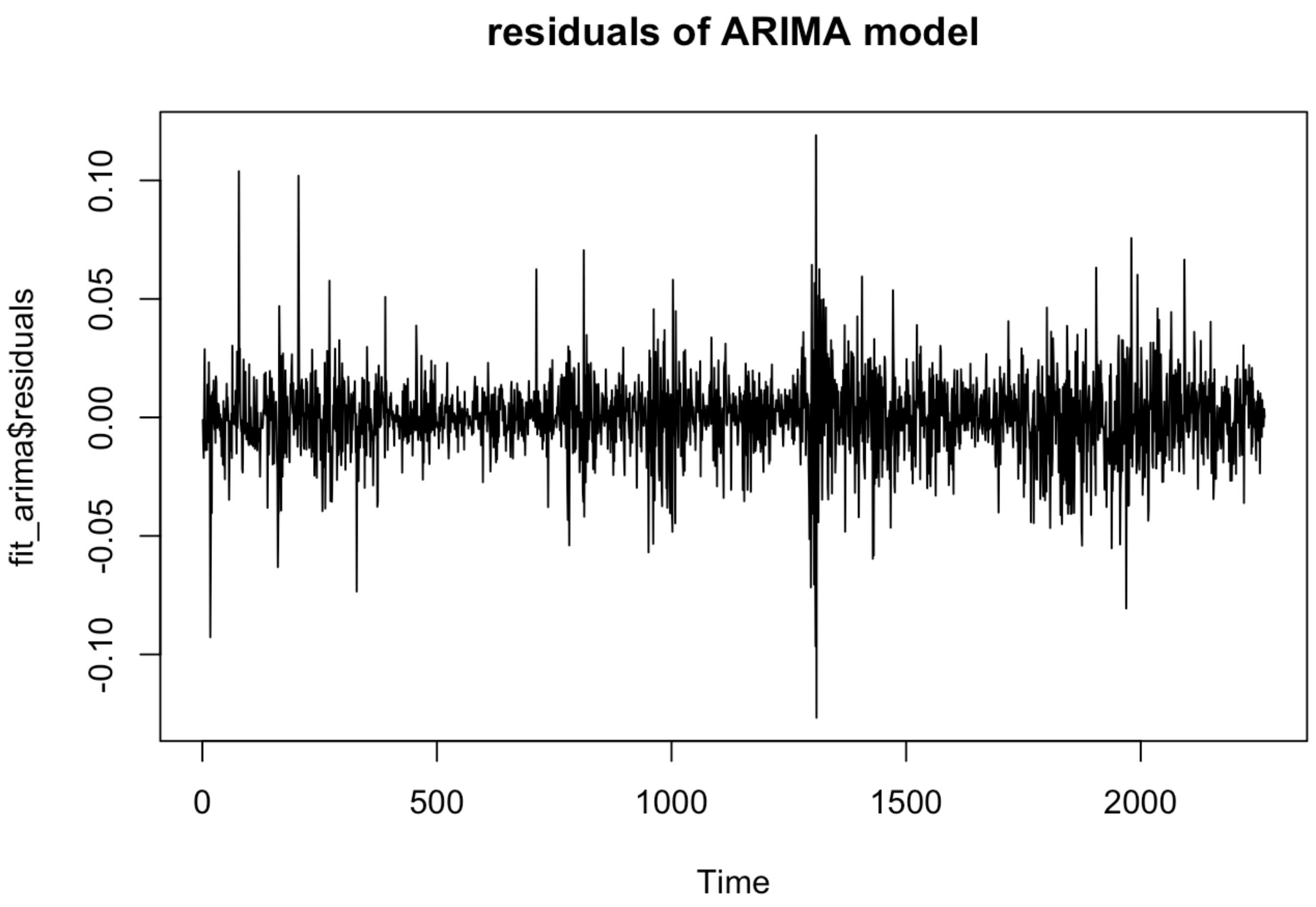
```
Box.test(abs(returns), type = "Ljung-Box", lag = 12)
```

```
##
##  Box-Ljung test
##
## data:  abs(returns)
## X-squared = 1147.6, df = 12, p-value < 2.2e-16
```

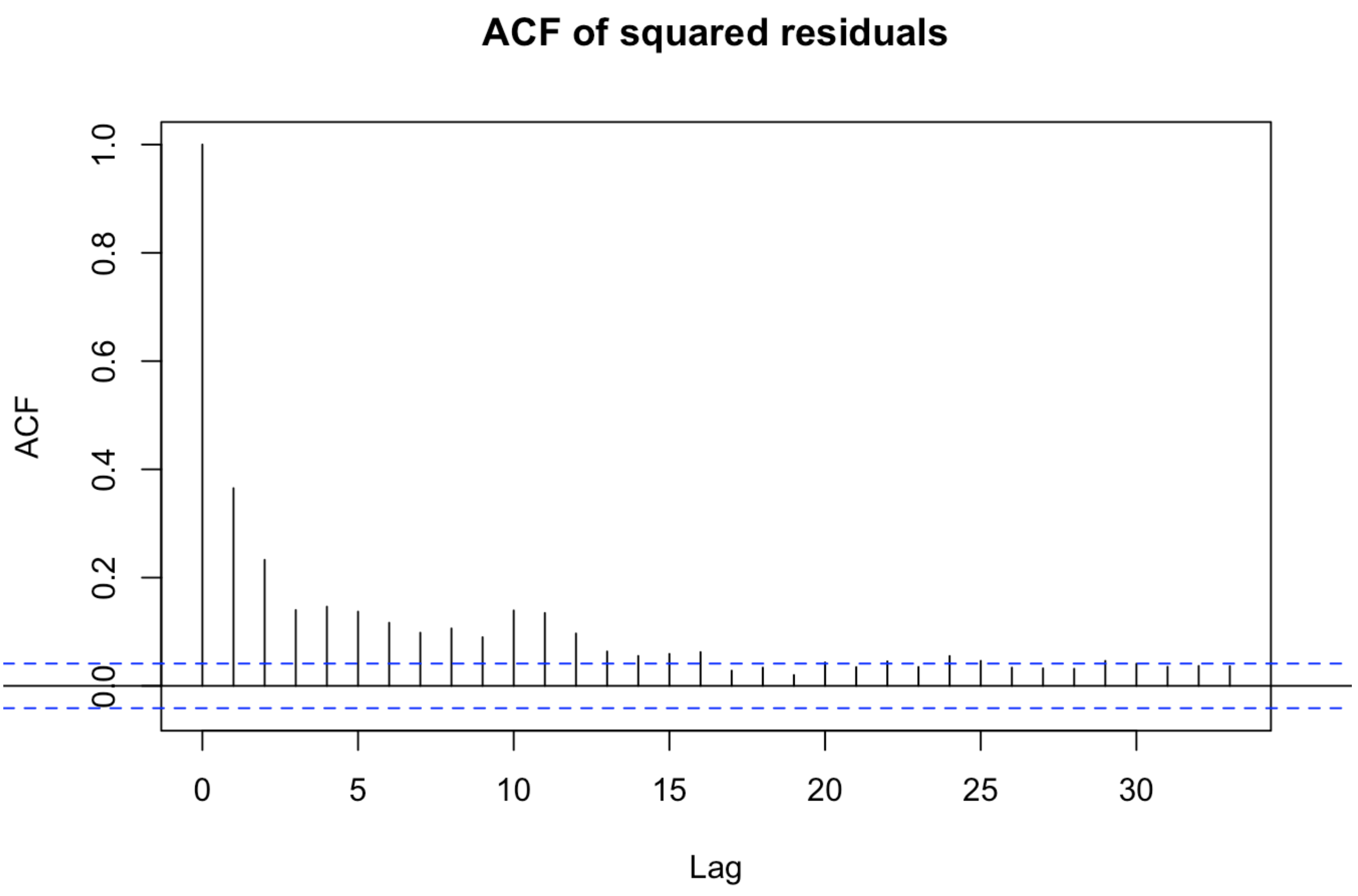
```
# if the p value is low (<0.05) we reject the null of "no serial correlation".
# a significative result in squared returns (or abs returns) supports the necessit
y of a GARCH model.

fit_arima <- auto.arima(returns)
summary(fit_arima)
```

```
## Series: returns
## ARIMA(3,0,5) with non-zero mean
##
## Coefficients:
##           ar1     ar2     ar3     ma1      ma2      ma3     ma4     ma5    mean
##       -0.7859  0.7743  0.8402  0.6554  -0.9050  -0.7865  0.0815  0.0555  0.0011
## s.e.   0.0540  0.0606  0.0407  0.0567   0.0619   0.0445  0.0292  0.0281  0.0002
##
## sigma^2 = 0.0002939:  log likelihood = 5997.47
## AIC=-11974.94   AICc=-11974.84   BIC=-11917.69
##
## Training set error measures:
##                         ME       RMSE        MAE MPE MAPE      MASE
## Training set -1.878507e-05 0.01711066 0.01192631 NaN  Inf 0.6558746
##                      ACF1
## Training set -0.002029582
```

```
plot(fit_arima$residuals, main = "residuals of ARIMA model")
```



```
acf(fit_arima$residuals^2, main = "ACF of squared residuals")
```



```
# we can notice volatility clusters in the residuals
# suggesting an ARCH/GARCH model, rather than ARIMA

ArchTest(returns, lags = 12, demean = FALSE)
```

```
##
##  ARCH LM-test; Null hypothesis: no ARCH effects
##
## data:  returns
## Chi-squared = 507.72, df = 12, p-value < 2.2e-16
```

```
# p < 0.05: we reject the null of no ARCH effects, so ARCH exists
```

In the initial part of the code we start by selecting the data we need, do the manipulations to get the returns and start with the analysis that allow us to understand and prove that we have ARCH effects: first we simply plot the returns (and their square) and the ACF just to have a look at the series and then we perform some statistical test. The result is that data show some volatility clustering and this tells us that it is correct to use an ARCH/GARCH model.

```
models <- list(
  list(variance = "sGARCH", dist = "norm"),
  list(variance = "sGARCH", dist = "std"),
  list(variance = "sGARCH", dist = "ged"),
  list(variance = "eGARCH", dist = "norm"),
  list(variance = "eGARCH", dist = "std"),
  list(variance = "eGARCH", dist = "ged"),
  list(variance = "apARCH", dist = "norm"),
  list(variance = "apARCH", dist = "std"),
  list(variance = "apARCH", dist = "ged")
)

model_results <- list()

for (i in seq_along(models)) {
  model_spec <- ugarchspec(
    variance.model = list(model = models[[i]]$variance, garchOrder = c(1, 1)),
    mean.model = list(armaOrder = c(0, 0), include.mean = FALSE),
    distribution.model = models[[i]]$dist
  )

  # model estimation
  model_fit <- ugarchfit(spec = model_spec, data = returns)

  # saving the results
  model_results[[i]] <- list(
    spec = models[[i]],
    fit = model_fit,
    aic = infocriteria(model_fit)["Akaike", 1],
    bic = infocriteria(model_fit)["Bayes", 1],
    loglik = likelihood(model_fit),
    residuals = residuals(model_fit, standardize = TRUE)
  )
}
 #data frame for the comparison
comparison <- data.frame(
  Model = sapply(model_results, function(x) paste0(x$spec$variance, "-", x$spec$di
st)),
  AIC = sapply(model_results, function(x) x$aic),
  BIC = sapply(model_results, function(x) x$bic),
  LogLikelihood = sapply(model_results, function(x) x$loglik)
)

#sorting models by AIC
comparison <- comparison[order(comparison$AIC), ]
print("Confronto tra i modelli:")
```

```
## [1] "Confronto tra i modelli:"
```

```
print(comparison)
```

```
##         Model       AIC       BIC LogLikelihood
## 8  apARCH-std -5.607967 -5.592795      6354.218
## 5  eGARCH-std -5.605899 -5.593256      6350.878
## 9  apARCH-ged -5.593242 -5.578070      6337.550
## 6  eGARCH-ged -5.591888 -5.579245      6335.017
## 2  sGARCH-std -5.576115 -5.566001      6316.163
## 3  sGARCH-ged -5.569971 -5.559856      6309.207
## 4 eGARCH-norm -5.477374 -5.467259      6204.387
## 7 apARCH-norm -5.467125 -5.454482      6193.786
## 1 sGARCH-norm -5.457556 -5.449970      6180.954
```

```
# best model: apARCH-std con
#AIC       BIC            LogLikelihood
#-5.607967 -5.592795      6354.218

#best model index after sorting
best_index <- rownames(comparison)[1]
print(paste("best model index:", best_index))
```

```
## [1] "best model index: 8"
```

```
best_index <- as.numeric(best_index)

#selects the best model (lowest AIC)
best_model <- model_results[[best_index]]
residuals_best <- best_model$residuals

#coefficients of the best model
print("Coefficients of the best model (lowest AIC):")
```

```
## [1] "Coefficients of the best model (lowest AIC):"
```

```
print(best_model$spec)
```

```
## $variance
## [1] "apARCH"
##
## $dist
## [1] "std"
```

```
print(coef(best_model$fit))
```

```
##       omega      alpha1       beta1      gamma1       delta       shape
## 0.001170423 0.129590170 0.871214237 0.685794655 0.872873445 4.792555308
```

```
# 2 comments on this output:
#1. The parameter $\gamma_1 = 0.68582677$ indicates the presence of leverage effec
t, i.e. negative shocks have a greater impact on volatility than positive shocks o
f equal magnitude.
#2. The value $\beta_1 = 0.87121796$ suggests a high persistence of volatility, i.
e. past shocks continue to influence the level of volatility for a long time.

#test on squared residuals
ljung_box <- Box.test(residuals_best^2, lag = 10, type = "Ljung-Box")
print("Ljung-Box Test on squared residuals:")
```

```
## [1] "Ljung-Box Test on squared residuals:"
```

```
print(ljung_box)
```

```
##
##  Box-Ljung test
##
## data:  residuals_best^2
## X-squared = 4.1977, df = 10, p-value = 0.938
```

```
#p-value= 0.938, we cannot reject the null of "no serial correlation".
```

What we did, after understanding the behaviour of our data, is to check which one among several different GARCH models would fit better. We considered three GARCH models: the standard, the asymmetric power (apARCH) and the exponential (eGARCH) (the last two allow us to tackle the asymmetries linked to the leverage effect). Furthermore for each of these models we considered three different distributions of the noises which are the normal distribution, the Student's t and the GED. After fitting the models with our data we used the AIC and BIC criteria to select the best model, we saved the results in a data frame and choose the model with the lowest values. We ended up with the apARCH and student's t distribution for errors being the best model to describe our data and R but since it required (we realized this going on with the conformal forecasting analysis) a very high computational time to run, we decided to use the second best model which is the eGARCH: it showed very similar values for AIC and BIC but it took less time to run the algorithm.

formally the exponential GARCH (eGARCH) can be expressed as:

$$\log(\sigma^2_{t|t-1}) = \omega + \sum_{i=1}^{p} \left(\alpha_i \varepsilon_{t-i} + \gamma_i \left(|\varepsilon_{t-i}| - \mathbb{E}(\varepsilon_{t-i})\right)\right) + \sum_{j=1}^{q} \beta_j \log(\sigma^2_{t-j})$$

```
orders <- expand.grid(p = 0:3, q = 0:3)

results <- data.frame(Order = character(), AIC = numeric(), BIC = numeric(), strin
gsAsFactors = FALSE)

for (i in 1:nrow(orders)) {
  tryCatch({
    spec <- ugarchspec(
      variance.model = list(model = "eGARCH", garchOrder = c(1, 1)),  # fixed vola
tility order
      mean.model = list(armaOrder = c(orders$p[i], orders$q[i]), include.mean = TR
UE),  # ARMA(p, q)
      distribution.model = "std"  # Heavy-tailed t-distribution
    )

    fit <- ugarchfit(spec = spec, data = returns, solver = "hybrid")

    results <- rbind(results, data.frame(
      Order = paste0("(", orders$p[i], ",", orders$q[i], ")"),
      AIC = infocriteria(fit)["Akaike", 1],
      BIC = infocriteria(fit)["Bayes", 1]
    ))
  }, error = function(e) {
    cat("Error for order (", orders$p[i], ",", orders$q[i], "): ", e$message, "\
n")
  })
}
```

```
## Warning in arima(data, order = c(modelinc[2], 0, modelinc[3]), include.mean =
## modelinc[1], : possibile errore di convergenza: optim ha restituito codice = 1
## Warning in arima(data, order = c(modelinc[2], 0, modelinc[3]), include.mean =
## modelinc[1], : possibile errore di convergenza: optim ha restituito codice = 1
```

```
results <- results[order(results$AIC), ]

cat("AIC, BIC for different orders (p, q):\n")
```

```
## AIC, BIC for different orders (p, q):
```

```
print(results)
```

```
##    Order       AIC       BIC
## 15 (2,3) -5.620114 -5.592299
## 16 (3,3) -5.619886 -5.589542
## 6  (1,1) -5.617378 -5.597149
## 10 (1,2) -5.617325 -5.594567
## 7  (2,1) -5.617321 -5.594563
## 8  (3,1) -5.616591 -5.591305
## 11 (2,2) -5.616539 -5.591252
## 14 (1,3) -5.616480 -5.591193
## 12 (3,2) -5.615709 -5.587893
## 9  (0,2) -5.615100 -5.594870
## 13 (0,3) -5.615027 -5.592269
## 5  (0,1) -5.615017 -5.597317
## 3  (2,0) -5.614833 -5.594604
## 4  (3,0) -5.614795 -5.592037
## 2  (1,0) -5.614741 -5.597040
## 1  (0,0) -5.611062 -5.595890
```

```r
library(rugarch)
library(tseries)
best_order <- gsub("[()]", "", results$Order[1])
p <- as.numeric(strsplit(best_order, ",")[[1]][1])
q <- as.numeric(strsplit(best_order, ",")[[1]][2])

cat("p:", p, "q:", q, "\n")
```

```
## p: 2 q: 3
```

```r
#for parsimony, we consider:p=1, q=1
spec <- ugarchspec(
  variance.model = list(model = "eGARCH", garchOrder = c(1, 1)),
  mean.model = list(armaOrder = c(p, q), include.mean = TRUE),
  distribution.model = "std"
)

fit <- ugarchfit(spec = spec, data = returns, solver = "hybrid")

residuals <- residuals(fit, standardize = TRUE)


cat("Ljung-Box test on residuals:\n")
```

```
## Ljung-Box test on residuals:
```

```r
print(Box.test(residuals, lag = 10, type = "Ljung-Box"))
```

```
##
##  Box-Ljung test
##
## data:  residuals
## X-squared = 8.1859, df = 10, p-value = 0.6107
```

```
#p value is significantly higher than 0.05.
#this suggests that there's no evidence of significant autocorrelation in the resi
duals.
# it is a desirable result: the residuals behaves as a white nose so the model cap
tured very well the serial dynamics in the data.

cat("Ljung-Box test on squared residuals:\n")
```

```
## Ljung-Box test on squared residuals:
```

```
print(Box.test(residuals^2, lag = 10, type = "Ljung-Box"))
```

```
##
##  Box-Ljung test
##
## data:  residuals^2
## X-squared = 3.8581, df = 10, p-value = 0.9535
```

```
#also in this case p-value is very high.
# this suggests that there's no evidence of residual heteroskedasticity in the dat
a, the GARCH model seems to capture correctly the conditional volatility.

print(mean(return))
```

```
## Warning in mean.default(return): l'argomento non è numerico o logico: si
## restituisce NA
```

```
## [1] NA
```

Now we check which is the best order for the eGARCH model

```r
orders <- expand.grid(p = 1:3, q = 1:3)
results <- data.frame(Order = character(), AIC = numeric(), BIC = numeric())

for (i in 1:nrow(orders)) {
  spec <- ugarchspec(
    variance.model = list(model = "eGARCH", garchOrder = c(orders$p[i], orders$q[
i])),
    mean.model = list(armaOrder = c(1, 1), include.mean = FALSE),
    distribution.model = "std"
  )
  fit <- ugarchfit(spec = spec, data = returns, solver = "hybrid")

  results <- rbind(results, data.frame(
    Order = paste0("(", orders$p[i], ",", orders$q[i], ")"),
    AIC = infocriteria(fit)["Akaike", 1],
    BIC = infocriteria(fit)["Bayes", 1]
  ))
}

results <- results[order(results$AIC), ]
print("AIC, BIC results for different orders (p, q):")
```

```
## [1] "AIC, BIC results for different orders (p, q):"
```

```r
print(results)
```

```
##   Order       AIC       BIC
## 5 (2,2) -5.607228 -5.581942
## 1 (1,1) -5.606901 -5.589200
## 7 (1,3) -5.606588 -5.583830
## 4 (1,2) -5.606437 -5.586208
## 3 (3,1) -5.606099 -5.578284
## 2 (2,1) -5.605902 -5.583144
## 8 (2,3) -5.605743 -5.577928
## 9 (3,3) -5.605726 -5.572853
## 6 (3,2) -5.605154 -5.574810
```

the result is that the best order for the eGARCH model is (1, 1) and that’s a good thing since we need the lowest parameters possible.

```
lookback=1250

results <- garchConformalForcasting(returns = returns, alpha = 0.05, gamma = 0.003
,lookback=lookback,garchP=1,garchQ=1,startUp = 100,
                                    verbose=FALSE,updateMethod="Simple", garchType
= "sGARCH",noise="norm",stud=1,armap=0,armaq=0)

results2 <- garchConformalForcasting(returns = returns, alpha = 0.05, gamma = 0.00
3,lookback=lookback,garchP=1,garchQ=1,startUp = 100,
                                     verbose=FALSE,updateMethod="Momentum", garchT
ype = "sGARCH",noise="norm",stud=1,armap=0,armaq=0)

results3 <- garchConformalForcasting(returns = returns, alpha = 0.05, gamma = 0.00
3,lookback=lookback,garchP=1,garchQ=1,startUp = 100,
                                     verbose=FALSE,updateMethod="Simple", garchTyp
e = "sGARCH",noise="norm",stud=0,armap=0,armaq=0)

results4 <- garchConformalForcasting(returns = returns, alpha = 0.05, gamma = 0.00
3,lookback=lookback,garchP=1,garchQ=1,startUp = 100,
                                     verbose=FALSE,updateMethod="Momentum", garchT
ype = "sGARCH",noise="norm",stud=0,armap=0,armaq=0)

date <- index(msft)[lookback:length(index(msft))]

errSeqOC <- results[[2]]
mean(errSeqOC)
```

```
## [1] 0.04827586
```

```
errSeqNC <- results[[3]]
mean(errSeqNC)
```

```
## [1] 0.05320197
```

```
errSeqOC2 <- results2[[2]]
mean(errSeqOC2)
```

```
## [1] 0.0453202
```

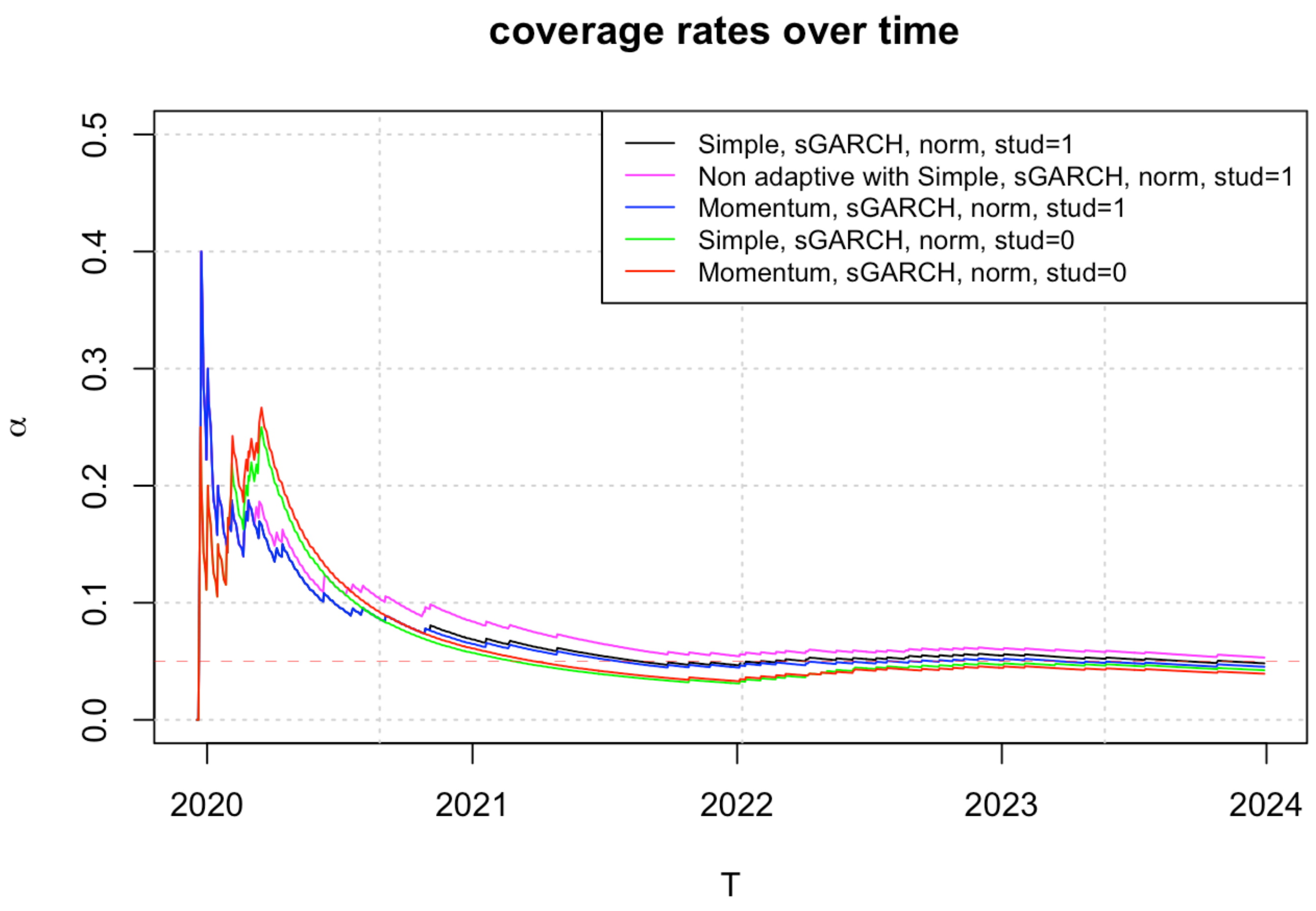
```
errSeqOC3 <- results3[[2]]
mean(errSeqOC3)
```

```
## [1] 0.04236453
```

```
errSeqOC4 <- results4[[2]]
mean(errSeqOC4)
```

```
## [1] 0.03940887
```

```
plot(date, cummean(errSeqOC), ylim = c(0, 0.5), type= "l" , col= "black",
     main = "coverage rates over time ", xlab = "T", ylab = expression(alpha));gri
d()
points(date, cummean(errSeqNC), type= "l", col="magenta")
points(date, cummean(errSeqOC2), type= "l", col="blue")
points(date, cummean(errSeqOC3), type= "l", col="green")
points(date, cummean(errSeqOC4), type= "l", col="red")
abline(h = 0.05, col = "red", lwd = 0.3, lty = 2)

legend("topright", legend = c(
  "Simple, sGARCH, norm, stud=1",
  "Non adaptive with Simple, sGARCH, norm, stud=1",
  "Momentum, sGARCH, norm, stud=1",
  "Simple, sGARCH, norm, stud=0",
  "Momentum, sGARCH, norm, stud=0"
), col = c("black", "magenta", "blue", "green", "red"), lty = 1, cex = 0.8)
```



The non-normalized score amplifies the influence of extreme events and distributional shifts, resulting in frequent oscillations in err_t. These oscillations destabilize the algorithm and cause significant fluctuations in the coverage rate. From the graph, it is evident that the absence of normalization leads to greater instability in the coverage rate, characterized by large and erratic variations.

As noted by the authors of the paper, adopting the “momentum” strategy instead of the classical approach does not produce notable differences in the results.

Finally, we observe that the non-adaptive strategy initially outperforms the adaptive approach when using a non-normalized score. However, as t increases, its performance declines and ultimately falls behind all other methods.

```
a_results <- garchConformalForcasting(returns = returns, alpha = 0.05, gamma = 0.0
03,lookback=lookback,garchP=1,garchQ=1,startUp = 100,
                                      verbose=FALSE,updateMethod="Simple", garchTy
pe = "eGARCH",noise="std",stud=1,armap=1,armaq=1)

a_results2 <- garchConformalForcasting(returns = returns, alpha = 0.05, gamma = 0.
003,lookback=lookback,garchP=1,garchQ=1,startUp = 100,
                                       verbose=FALSE,updateMethod="Momentum", garc
hType = "eGARCH",noise="std",stud=1,armap=1,armaq=1)

a_results3 <- garchConformalForcasting(returns = returns, alpha = 0.05, gamma = 0.
003,lookback=lookback,garchP=1,garchQ=1,startUp = 100,
                                       verbose=FALSE,updateMethod="Simple", garchT
ype = "eGARCH",noise="std",stud=0,armap=1,armaq=1)

a_results4 <- garchConformalForcasting(returns = returns, alpha = 0.05, gamma = 0.
003,lookback=lookback,garchP=1,garchQ=1,startUp = 100,
                                       verbose=FALSE,updateMethod="Momentum", garc
hType = "eGARCH",noise="std",stud=0,armap=1,armaq=1)

a_errSeqOC <- a_results[[2]]
mean(a_errSeqOC)
```

```
## [1] 0.05024631
```

```
a_errSeqNC <- a_results[[3]]
mean(a_errSeqNC)
```

```
## [1] 0.05418719
```

```
a_errSeqOC2 <- a_results2[[2]]
mean(a_errSeqOC2)
```

```
## [1] 0.04827586
```

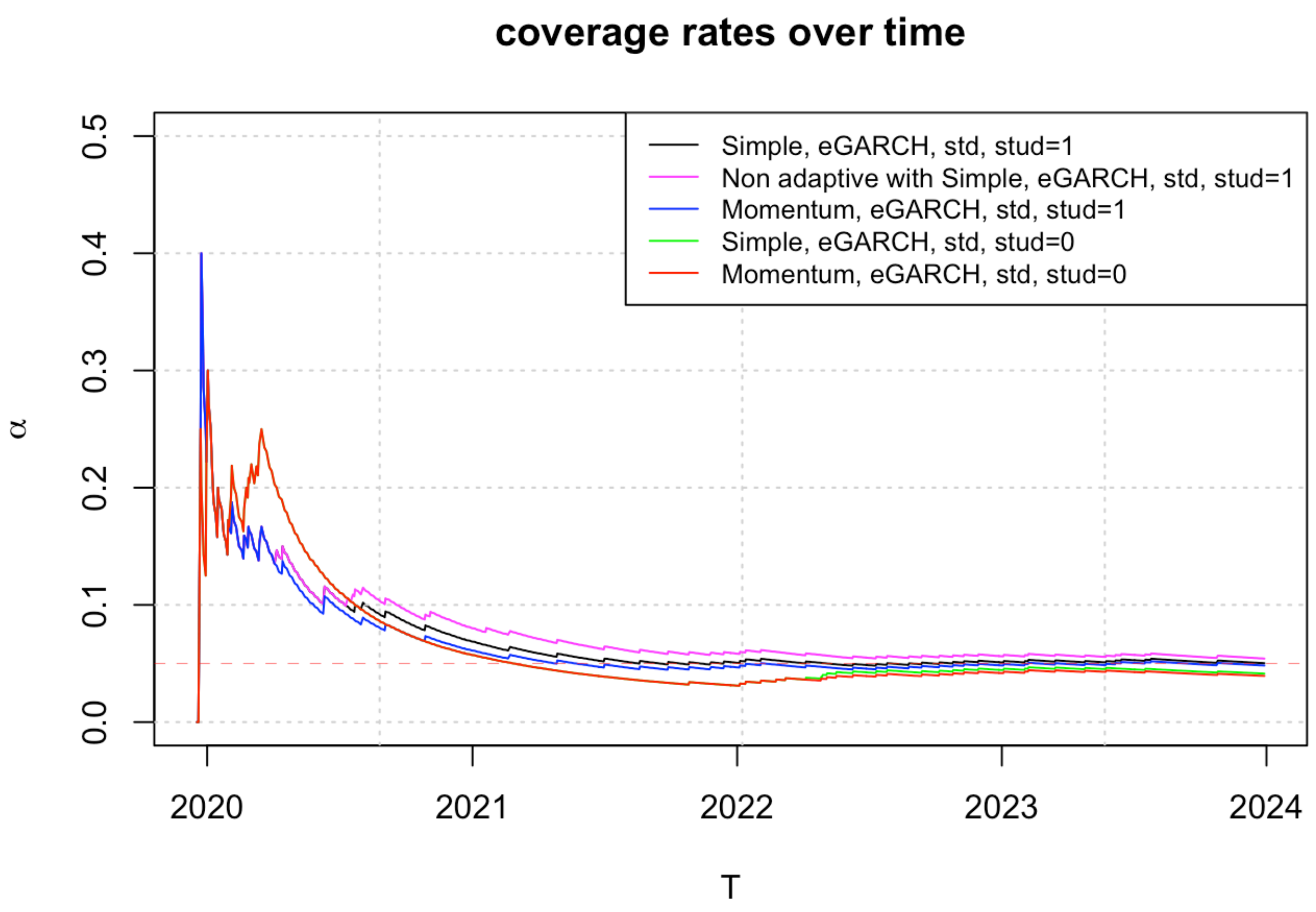
```
a_errSeqOC3 <- a_results3[[2]]
mean(a_errSeqOC3)
```

```
## [1] 0.04137931
```

```
a_errSeqOC4 <- a_results4[[2]]
mean(a_errSeqOC4)
```

```
## [1] 0.03940887
```

```
plot(date, cummean(a_errSeqOC), ylim = c(0, 0.5), type= "l" , col= "black",
     main = "coverage rates over time", xlab = "T", ylab = expression(alpha));gri
d()
points(date, cummean(a_errSeqNC), type= "l", col="magenta")
points(date, cummean(a_errSeqOC2), type= "l", col="blue")
points(date, cummean(a_errSeqOC3), type= "l", col="green")
points(date, cummean(a_errSeqOC4), type= "l", col="red")
abline(h = 0.05, col = "red", lwd = 0.3, lty = 2)

legend("topright", legend = c(
  "Simple, eGARCH, std, stud=1",
  "Non adaptive with Simple, eGARCH, std, stud=1",
  "Momentum, eGARCH, std, stud=1",
  "Simple, eGARCH, std, stud=0",
  "Momentum, eGARCH, std, stud=0"
), col = c("black", "magenta", "blue", "green", "red"), lty = 1, cex = 0.8)
```

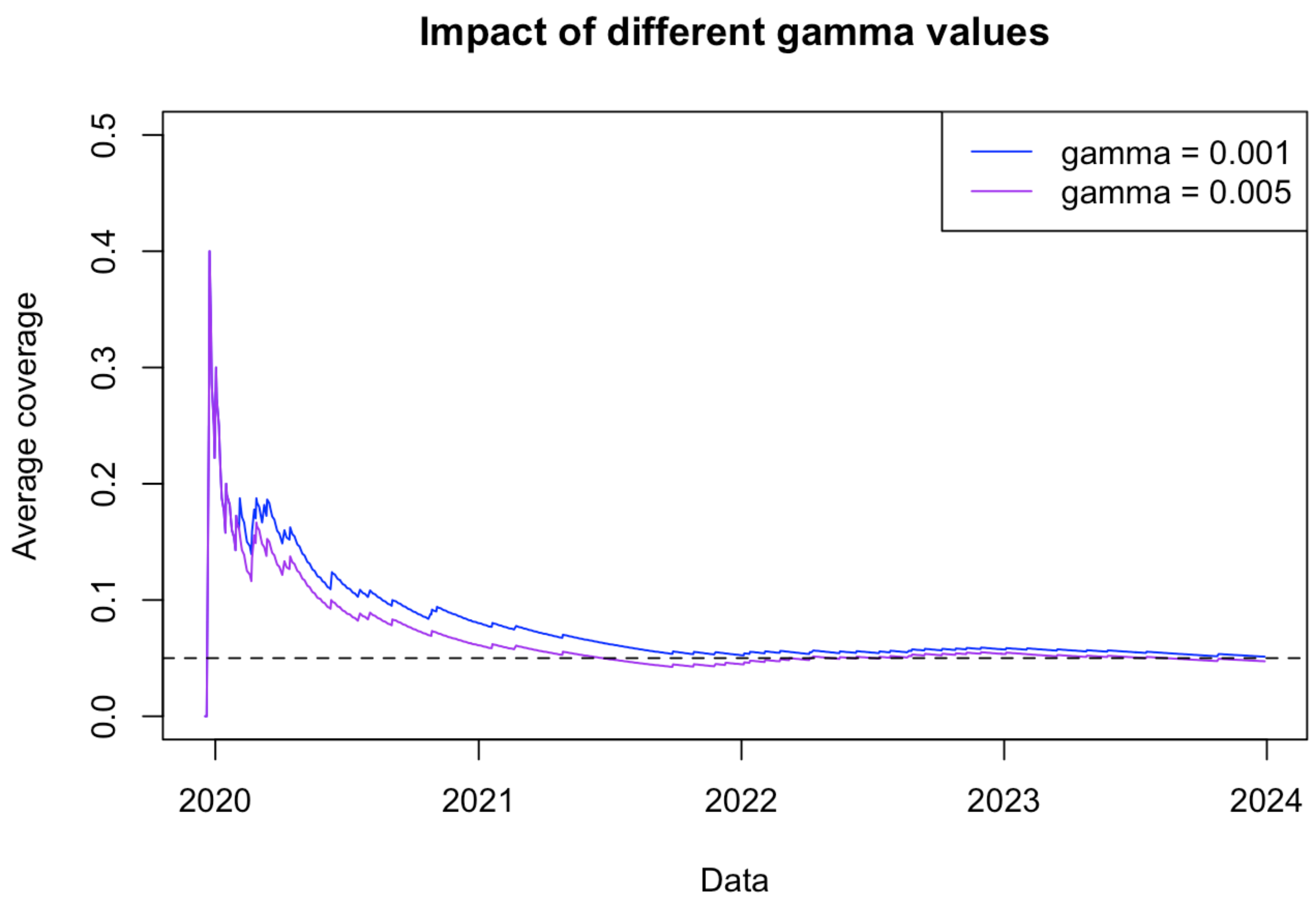


The improved results can be attributed to the use of the eGARCH model with t-student noise, which more effectively captures the underlying data distribution and volatility. This improved modeling leads to smaller conformity scores, thereby reducing the number of errors ($err_t = 1$) and enabling α_t to converge more rapidly toward the target value of $\alpha = 0.005$. Selecting an appropriate model, such as the eGARCH with t-student noise, is therefore critical for enhancing the stability and performance of the adaptive algorithm.

```
gamma_values <- c(0.001, 0.005)

results_gamma <- list()
for (gamma in gamma_values) {
  print(paste("I'm using gamma =", gamma))
  results_gamma[[as.character(gamma)]] <- garchConformalForcasting(
    returns = returns, alpha = 0.05, gamma = gamma,
    lookback = 1250, garchP = 1, garchQ = 1,
    startUp = 100, updateMethod="Simple", garchType = "sGARCH",noise="norm",stud=
1,armap=0,armaq=0)

}
```

```
## [1] "I'm using gamma = 0.001"
## [1] "I'm using gamma = 0.005"
```

```
plot(date, cummean(results_gamma[["0.001"]][[2]]), type = "l", col = "blue", ylim
= c(0, 0.5),
     main = "Impact of different gamma values", xlab = "Data", ylab = "Average cov
erage")
lines(date, cummean(results_gamma[["0.005"]][[2]]), col = "purple")
abline(h = 0.05, col = "black", lty = 2)
legend("topright", legend = c("gamma = 0.001", "gamma = 0.005"),
       col = c("blue", "purple"), lty = 1)
```



We observe that as γ increases, the coverage improves since the method becomes more responsive and better adapts to observed distribution shifts.

However, for values of $\gamma = 0.01$ or larger, the following error may occur:

```
Error in quantile.default(recentScores, 1 - alphat): 'probs' outside [0,1]
```

This error indicates that the value calculated for $1 - \alpha_t$ in the quantile computation falls outside the valid range $[0, 1]$. This happens because α_t becomes negative during the dynamic update. The error could also occur if α_t exceeds 1 during the update, although this is rare since α is small.

The issue arises from the high value of γ, which causes the update of α_t to be too sensitive, pushing it outside the valid range $[0, 1]$.