HW2

Matteo Toschi & Elie Ghazzoul

12-01-2025

Ex 1: Part 1

” The first extension we will evaluate is the Adaptive Conformal Inference (ACI) framework we covered in class: Gibbs and Candes (2021).”

Part A

Overview

Adaptive Conformal Inference (ACI) is a method for constructing prediction sets in an online setting where the data-generating distribution can vary over time. The goal is to maintain valid prediction set coverage, ensuring that the actual label is contained within the predicted set with a specified probability. ACI extends traditional conformal inference methods by adapting to non-exchangeable data and addressing distribution shifts.

Conformal Inference Framework

Conformal inference provides a flexible framework for creating prediction sets based on conformity scores derived from any predictive model.

- A conformity score $S(X, Y)$ measures how well a candidate prediction aligns with the model's output.
- A separate calibration set D_{cal} , that is different from the data that was used to fit the regression model(Training data), is used to compute quantiles of conformity scores. These quantiles define thresholds for prediction set inclusion. Given a target coverage $1 - \alpha$, the threshold is determined by:

$$\hat{Q}(p) = \inf \left\{ s : \frac{1}{|D_{cal}|} \sum_{(X,Y) \in D_{cal}} \mathbb{I}[S(X, Y) \leq s] \geq p \right\}.$$

and we say that y is a reasonable prediction for Y_t if:

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

Conformal inference guarantees coverage under the assumption of exchangeable data:

$$P(Y_t \in \hat{C}_t) = 1 - \alpha.$$

Adaptive Conformal Inference

ACI generalizes conformal inference to handle time-varying distributions. The key components are:

1. **Dynamic Quantile and Score Estimation**

At each time step t, new conformity scores $S_t(\cdot)$ and quantiles $\hat{Q}_t(\cdot)$ are computed using recent data. This adapts the prediction sets to changes in the data distribution.

- A **time-specific** score function $S_t(\cdot)$ that may be **updated** to reflect more recent data.
- A **time-specific** quantile $\hat{Q}_t(\cdot)$ estimated from the current or recent score distribution.
    - As the environment changes (e.g., new market conditions), old calibration data may become less relevant, and the model or the score distribution can be recalibrated to the **latest** conditions.

2. **Adjusting Coverage via Online Updates**

The miscoverage rate at time t is:

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

which may deviate from the target α due to distributional shifts. ACI defines a time-varying coverage parameter α_t, adjusted recursively:

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

where $\mathrm{err}_t = \mathbb{I}[Y_t \notin \hat{C}_t]$ and $\gamma > 0$ controls the update step size.

This ensures the method gradually corrects for under- or over-coverage.

- If err_t =1 too often (too many misses), α_t **increases** so that the method **expands** intervals (to cover more future points).
- If misses are **rare**, α_t drifts back down, **narrowing** intervals to stay near the target coverage.

3. **Robustness and Coverage Guarantees**

Over long time intervals, ACI achieves the target coverage:

$$\lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} \mathrm{err}_t = \alpha,$$

independent of the true data-generating process.

4. **Trade-offs in Adaptivity**

The step size γ balances adaptivity and stability:

- Larger γ: Faster response to distribution shifts but higher variability in α_t.
- Smaller γ: More stable α_t, less responsive to shifts.

Summary

Adaptive Conformal Inference builds on the strengths of conformal methods, allowing prediction set validity under dynamic and non-stationary environments. By tracking a single parameter α_t and leveraging an online update mechanism, ACI achieves robust coverage guarantees **without assuming exchangeability.** It operates as a

wrapper that can integrate with any predictive model, enhancing its reliability in real-world applications with distributional variability. Maintains coverage near the target level by tracking a single parameter α_t (or equivalently, adjusting the threshold) and correcting for missed predictions. Finally, guarantees long-term average coverage even when data distributions shift significantly.

Part B

```
# Load necessary libraries
library(quantmod)
```

```
## Warning: package 'quantmod' was built under R version 4.3.3
```

```
## Loading required package: xts
```

```
## Warning: package 'xts' was built under R version 4.3.3
```

```
## Loading required package: zoo
```

```
## Warning: package 'zoo' was built under R version 4.3.3
```

```
##
## Attaching package: 'zoo'
```

```
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```
## Loading required package: TTR
```

```
## Warning: package 'TTR' was built under R version 4.3.3
```

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

```
library(quantreg)
```

```
## Warning: package 'quantreg' was built under R version 4.3.3
```

```
## Loading required package: SparseM
```

```
## Warning: package 'SparseM' was built under R version 4.3.3
```

```
##
## Attaching package: 'SparseM'
```

```
## The following object is masked from 'package:base':
##
##     backsolve
```

```
## Warning in .recacheSubclasses(def@className, def, env): undefined subclass
## "ndiMatrix" of class "replValueSp"; definition not updated
```

```
library(rugarch)
```

```
## Warning: package 'rugarch' was built under R version 4.3.3
```

```
## Loading required package: parallel
```

```
##
## Attaching package: 'rugarch'
```

```
## The following object is masked from 'package:stats':
##
##     sigma
```

```
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 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.  #
## #                                                                              #
## ################################################################################
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:xts':
##
##     first, last
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(zoo)


garchConformalForcasting <- function(returns, alpha = 0.05, gamma = 0.01, lookback = 1250,
                                     garchP = 1, garchQ = 1, startUp = 100, verbose = FALSE,
                                     updateMethod, momentumBW = 0.8, scoreType, garchType = "sGA
RCH") {
  myT <- length(returns)
  T0 <- max(startUp, lookback)

  if (!garchType %in% c("sGARCH", "eGARCH")) {
    stop("Invalid garchType. Choose between 'sGARCH' and 'eGARCH'.")
  }

  garchSpec <- ugarchspec(mean.model = list(armaOrder = c(0, 0), include.mean = FALSE),
                          variance.model = list(model = garchType, garchOrder = c(garchP, garch
Q)),
                          distribution.model = "norm")
  alphat <- alpha
  errSeqOC <- rep(0, myT - T0 + 1)
  errSeqNC <- rep(0, myT - T0 + 1)
  alphaSequence <- rep(alpha, myT - T0 + 1)
  scores <- rep(0, myT - T0 + 1)

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

    garchFit <- ugarchfit(garchSpec, returns[(t - lookback + 1):(t - 1)], solver = "hybrid")
    sigmaNext <- sigma(ugarchforecast(garchFit, n.ahead = 1))
    if (scoreType == "Simple") {
      scores[t - T0 + 1] <- abs(returns[t]^2 - sigmaNext^2)
    } else {
      scores[t - T0 + 1] <- abs(returns[t]^2 - sigmaNext^2) / sigmaNext^2
    }

    recentScores <- scores[max(t - T0 + 1 - lookback + 1, 1):(t - T0)]
    if (any(is.na(recentScores)) || length(recentScores) == 0) {
      stop("Invalid recentScores: Check scores computation or lookback period.")
    }

    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))

    alphaSequence[t - T0 + 1] <- alphat
    if (updateMethod == "Simple") {
      alphat <- alphat + gamma * (alpha - errSeqOC[t - T0 + 1])
      alphat <- max(0, min(1, alphat))
    } 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))
      alphat <- max(0, min(1, alphat))
    }
    if (t %% 100 == 0) {
      print(sprintf("Done %g steps", t))
    }
  }

  return(list(alphaSequence = alphaSequence, errSeqOC = errSeqOC, errSeqNC = errSeqNC))
}


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

```
## [1] "JPM"
```

```
Jpm_prices<- Cl(JPM)
returns <- dailyReturn(Jpm_prices)
returns <- na.omit(returns)

results_Studentized <- garchConformalForcasting(returns = returns, alpha = 0.05, gamma = 0.01,lo
okback=1250,garchP=1,garchQ=1,startUp = 100,verbose=FALSE,updateMethod="Simple", scoreType ="Stu
dentized",garchType = "sGARCH")
```

```
## [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 2100 steps"
## [1] "Done 2200 steps"
```

```
date <- index(Jpm_prices)[1250:length(index(Jpm_prices))]
alphaSequence_St <- results_Studentized[[1]]
errSeqOC_St <- results_Studentized[[2]]; mean(errSeqOC_St)
```

```
## [1] 0.04630542
```

```
errSeqNC_St <- results_Studentized[[3]]; mean(errSeqNC_St)
```

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

```
#A lower cumulative error rate indicates better performance.
#In most cases, Studentized scores perform better in real-world applications due to their robust
ness.
#Compare coverage rates and error rates for sGARCH and eGARCH.

results_Simple <- garchConformalForcasting(returns = returns, alpha = 0.05, gamma = 0.01,lookbac
k=1250,garchP=1,garchQ=1,startUp = 100,verbose=FALSE,updateMethod="Simple", scoreType ="Simple",
garchType = "sGARCH")
```

```
## [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 2100 steps"
## [1] "Done 2200 steps"
```

```
alphaSequence_Si <- results_Simple[[1]]
errSeqOC_Si <- results_Simple[[2]]; mean(errSeqOC_Si)
```

```
## [1] 0.04039409
```

```
errSeqNC_Si <- results_Simple[[3]]; mean(errSeqNC_Si)
```

```
## [1] 0.02758621
```

```
results_StudentizedM <- garchConformalForcasting(returns = returns, alpha = 0.05, gamma = 0.01,l
ookback=1250,garchP=1,garchQ=1,startUp = 100,verbose=FALSE,updateMethod="Momentum",scoreType ="S
tudentized",garchType = "sGARCH")
```

```
## [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 2100 steps"
## [1] "Done 2200 steps"
```

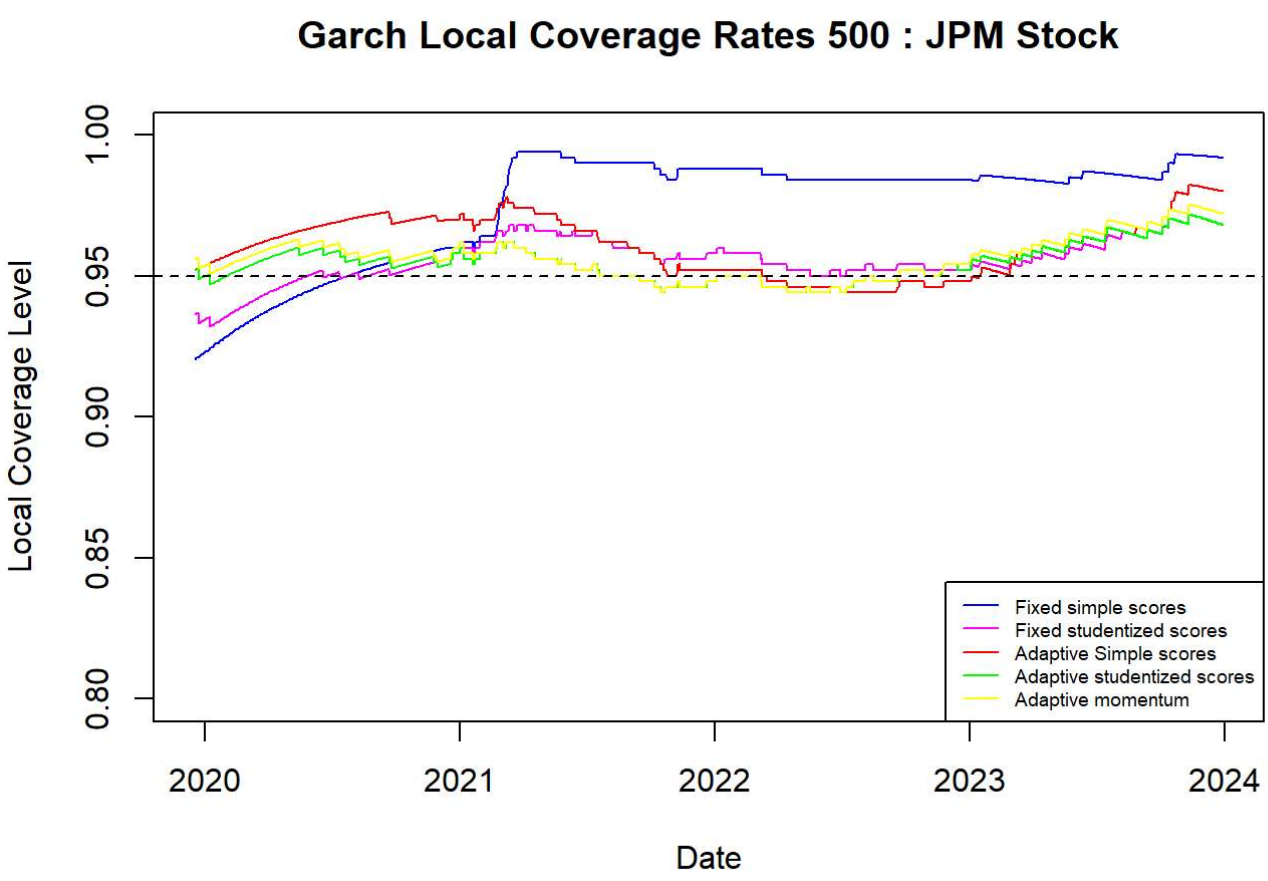
```
alphaSequence_StM <- results_StudentizedM[[1]]
errSeqOC_StM <- results_StudentizedM[[2]]; mean(errSeqOC_StM)
```

```
## [1] 0.04433498
```

```
errSeqNC_StM <- results_StudentizedM[[3]]; mean(errSeqNC_StM)
```

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

```
compute_local_coverage <- function(errors, window = 250) {
  1 - rollmean(errors, k = window, fill = NA)
  n <- length(errors)
  localCov <- rep(NA, n)

  for (t in 1:n) {
    startIdx <- max(1, t - floor(window / 2))
    endIdx <- min(n, t + floor(window / 2))

    localCov[t] <- 1 - mean(errors[startIdx:endIdx], na.rm = TRUE)
  }

  return(localCov)
}
window_size <- 500


#GARCH(1,1)
local_coverage_fixed <- compute_local_coverage(errSeqNC_St, window = window_size)
local_coverage_adaptive_simple <- compute_local_coverage(errSeqOC_St, window = window_size)
local_coverage_adaptive_momentum <- compute_local_coverage(errSeqOC_StM, window = window_size)
local_coverage_fixedSi <- compute_local_coverage(errSeqNC_Si,window = window_size)
local_coverage_adaptive_simpleSi <- compute_local_coverage(errSeqOC_Si,window = window_size)

plot(date, local_coverage_fixedSi, type = "l", col = "blue", ylim = c(0.8, 1),
     main = "Garch Local Coverage Rates 500 : JPM Stock", xlab = "Date", ylab = "Local Coverage
Level")
lines(date, local_coverage_fixed, col = "magenta")
lines(date, local_coverage_adaptive_simpleSi, col = "red")
lines(date, local_coverage_adaptive_simple, col = "green", type="s")
lines(date,local_coverage_adaptive_momentum,col="yellow")
abline(h = 0.95, col = "black", lty = 2)  # Target coverage level
legend("bottomright", legend = c("Fixed simple scores","Fixed studentized scores", "Adaptive Sim
ple scores","Adaptive studentized scores","Adaptive momentum"),
       col = c("blue","magenta", "red","green","yellow"), lty = 1,cex=0.6)
```




```
#eGARCH(1,1)

results_EStudentized <- garchConformalForcasting(returns = returns, alpha = 0.05, gamma = 0.01,l
ookback=1250,garchP=1,garchQ=1,startUp = 100,verbose=FALSE,updateMethod="Simple", scoreType ="St
udentized",garchType = "eGARCH")
```

```
## [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 2100 steps"
## [1] "Done 2200 steps"
```

```
alphaSequence_ESt <- results_EStudentized[[1]]
errSeqOC_ESt <- results_EStudentized[[2]]; mean(errSeqOC_ESt)
```

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

```
errSeqNC_ESt <- results_EStudentized[[3]]; mean(errSeqNC_ESt)
```

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

```
results_ESimple <- garchConformalForcasting(returns = returns, alpha = 0.05, gamma = 0.01,lookba
ck=1250,garchP=1,garchQ=1,startUp = 100,verbose=FALSE,updateMethod="Simple", scoreType ="Simpl
e",garchType = "eGARCH")
```

```
## [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 2100 steps"
## [1] "Done 2200 steps"
```

```
alphaSequence_ESi <- results_ESimple[[1]]
errSeqOC_ESi <- results_ESimple[[2]]; mean(errSeqOC_ESi)
```

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

```
errSeqNC_ESi <- results_ESimple[[3]]; mean(errSeqNC_ESi)
```

```
## [1] 0.02955665
```

```
results_EStudentizedM <- garchConformalForcasting(returns = returns, alpha = 0.05, gamma = 0.01,
lookback=1250,garchP=1,garchQ=1,startUp = 100,verbose=FALSE,updateMethod="Momentum",scoreType
="Studentized",garchType = "eGARCH")
```

```
## [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 2100 steps"
## [1] "Done 2200 steps"
```

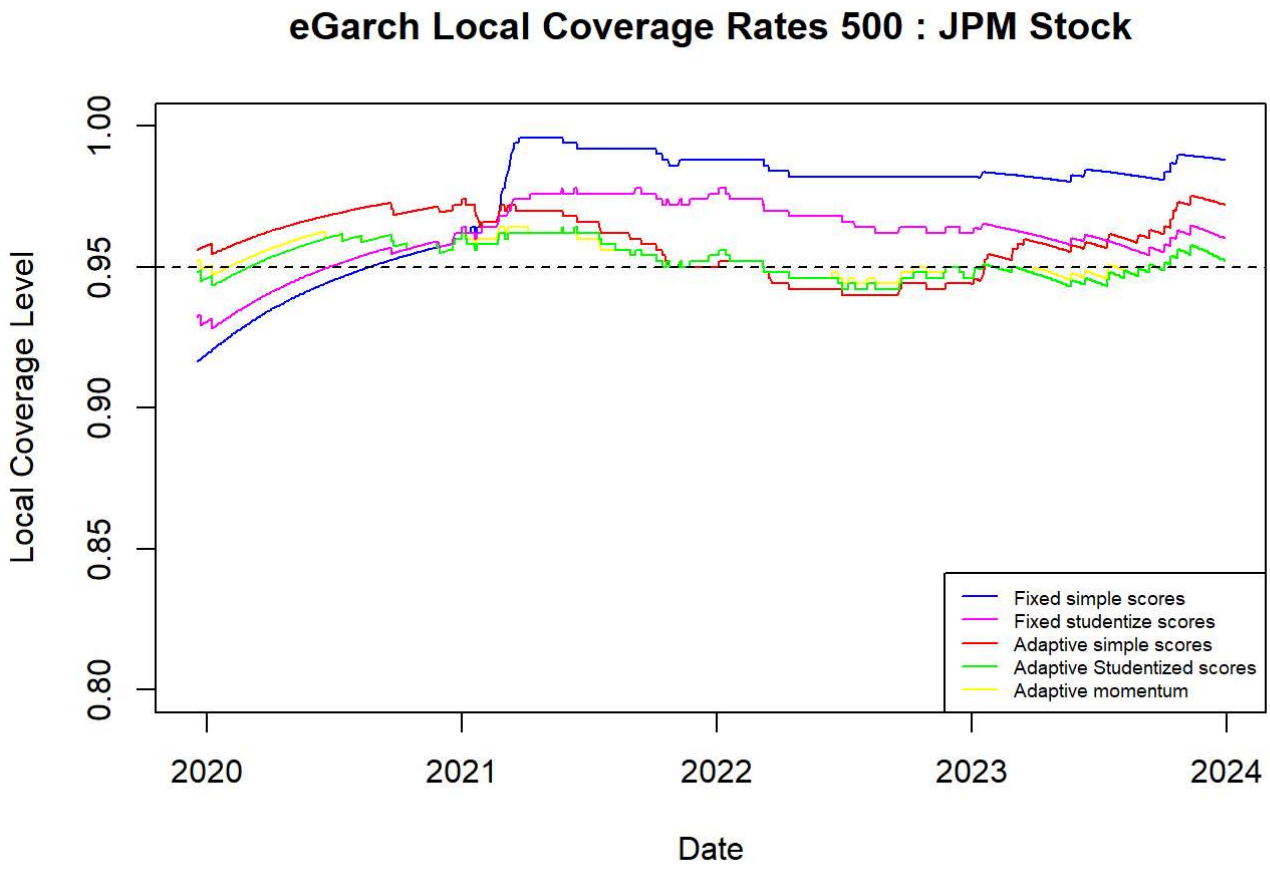
```
alphaSequence_EStM <- results_EStudentizedM[[1]]
errSeqOC_EStM <- results_EStudentizedM[[2]]; mean(errSeqOC_EStM)
```

```
## [1] 0.04729064
```

```
errSeqNC_EStM <- results_EStudentizedM[[3]]; mean(errSeqNC_EStM)
```

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

```
local_coverage_Efixed_St <- compute_local_coverage(errSeqNC_ESt, window = window_size)
local_coverage_Eadaptive_simple_St <- compute_local_coverage(errSeqOC_ESt, window = window_size)
local_coverage_adaptive_Emomentum <- compute_local_coverage(errSeqOC_EStM, window = window_size)
local_coverage_EfixedSi <- compute_local_coverage(errSeqNC_ESi,window = window_size)
local_coverage_Eadaptive_simpleSi <- compute_local_coverage(errSeqOC_ESi,window = window_size)

plot(date, local_coverage_EfixedSi, type = "l", col = "blue", ylim = c(0.8, 1),
     main = "eGarch Local Coverage Rates 500 : JPM Stock", xlab = "Date", ylab = "Local Coverage
Level")
lines(date, local_coverage_adaptive_Emomentum, col="yellow")
lines(date, local_coverage_Eadaptive_simpleSi, col = "red")
lines(date, local_coverage_Eadaptive_simple_St, col = "green", type="s")
lines(date, local_coverage_Efixed_St, col = "magenta")
abline(h = 0.95, col = "black", lty = 2)  # Target coverage level
legend("bottomright", legend = c("Fixed simple scores ","Fixed studentize scores","Adaptive simp
le scores", "Adaptive Studentized scores","Adaptive momentum"),
       col = c("blue","magenta", "red","green","yellow"), lty = 1,cex=0.6)
```



```r
#Compare  all of our strategies
compute_metrics <- function(errSeq, alpha_target = 0.05) {
  mean_error_rate <- mean(errSeq, na.rm = TRUE)  # Mean error rate
  coverage_rate <- 1 - mean_error_rate           # Coverage rate

  return(c(Coverage_Rate = coverage_rate, Error_Rate = mean_error_rate))
}

# Compute metrics for all strategies
strategies <- list(
  "Fixed Alpha sGARCH - Studentized" = compute_metrics(errSeqNC_St),
  "Adaptive Simple sGARCH - Studentized" = compute_metrics(errSeqOC_St),
  "Adaptive Momentum sGARCH- Studentized" = compute_metrics(errSeqOC_StM),
  "Fixed Alpha sGARCH - Simple" = compute_metrics(errSeqNC_Si),
  "Adaptive Simple sGARCH - Simple" = compute_metrics(errSeqOC_Si),
  "Fixed Alpha - eGARCH (Studentized)" = compute_metrics(errSeqNC_ESt),
  "Adaptive Simple - eGARCH (Studentized)" = compute_metrics(errSeqOC_ESt),
  "Adaptive Momentum - eGARCH (Studentized)" = compute_metrics(errSeqOC_EStM),
  "Fixed Alpha - eGARCH (Simple)" = compute_metrics(errSeqNC_ESi),
  "Adaptive Simple - eGARCH (Simple)" = compute_metrics(errSeqOC_ESi)
)

# Create a data frame for the results
results_df <- as.data.frame(do.call(rbind, strategies))
results_df$Strategy <- rownames(results_df)
rownames(results_df) <- NULL

# Reorder columns for clarity
results_df <- results_df %>%
  select(Strategy, Coverage_Rate, Error_Rate)

# Print the results table
print(results_df)
```

```
##                                    Strategy Coverage_Rate Error_Rate
## 1           Fixed Alpha sGARCH - Studentized     0.9546798 0.04532020
## 2       Adaptive Simple sGARCH - Studentized     0.9536946 0.04630542
## 3      Adaptive Momentum sGARCH- Studentized     0.9556650 0.04433498
## 4                 Fixed Alpha sGARCH - Simple     0.9724138 0.02758621
## 5            Adaptive Simple sGARCH - Simple     0.9596059 0.04039409
## 6          Fixed Alpha - eGARCH (Studentized)     0.9605911 0.03940887
## 7     Adaptive Simple - eGARCH (Studentized)     0.9517241 0.04827586
## 8   Adaptive Momentum - eGARCH (Studentized)     0.9527094 0.04729064
## 9               Fixed Alpha - eGARCH (Simple)     0.9704433 0.02955665
## 10         Adaptive Simple - eGARCH (Simple)     0.9576355 0.04236453
```

The eGARCH model achieves a smoother and more consistent convergence to the target coverage level (0.95). This is particularly evident in the **adaptive momentum** and **adaptive studentized** strategies, where the local coverage level remains more stable compared to sGARCH. Stability in convergence suggests that eGARCH better captures the volatility dynamics of the JPM stock, as it accounts for asymmetric effects (e.g., different impacts of positive vs. negative returns). In the eGARCH plots, the adaptive momentum strategy demonstrates a more robust

adjustment to changes in volatility over time. The smoother progression of the coverage level highlights that the momentum-weighted update method benefits from the enhanced modeling capabilities of eGARCH. While both models perform reasonably well in achieving coverage, the eGARCH model demonstrates greater consistency across all strategies.

eGARCH handles the dynamics of financial volatility better, especially for strategies that adapt to varying error rates. The ability to capture asymmetries and adjust more effectively to sudden changes in volatility underpins eGARCH's superior performance in these comparisons. This behavior is crucial for accurately predicting uncertainty in volatile assets like JPM stock.

The decision to choose **the adaptive studentized strategy for eGARCH** is based on its ability to account for the variability in the data's heteroscedasticity, which is particularly prominent in financial time series like stock returns. By normalizing the conformity scores with the predicted variance, the studentized approach ensures that the method adapts more effectively to changes in volatility. Additionally, the adaptive mechanism allows the coverage level to dynamically adjust based on recent errors, improving both the efficiency and reliability of the prediction intervals. Combined with the eGARCH model, which better captures asymmetries and extreme events in financial data, this strategy provides a more robust framework for handling the complexities of real-world market conditions while maintaining accurate and close coverage rates.

Gamma choice

The parameter **gamma** plays a crucial role in the adaptive mechanism of conformal forecasting. It controls the rate at which the confidence level (**alphat**) adjusts in response to deviations between observed outcomes and the model's predictions. Essentially, **gamma** determines how aggressively the model reacts to errors in conformity scores, impacting both the responsiveness and stability of the forecasting method.

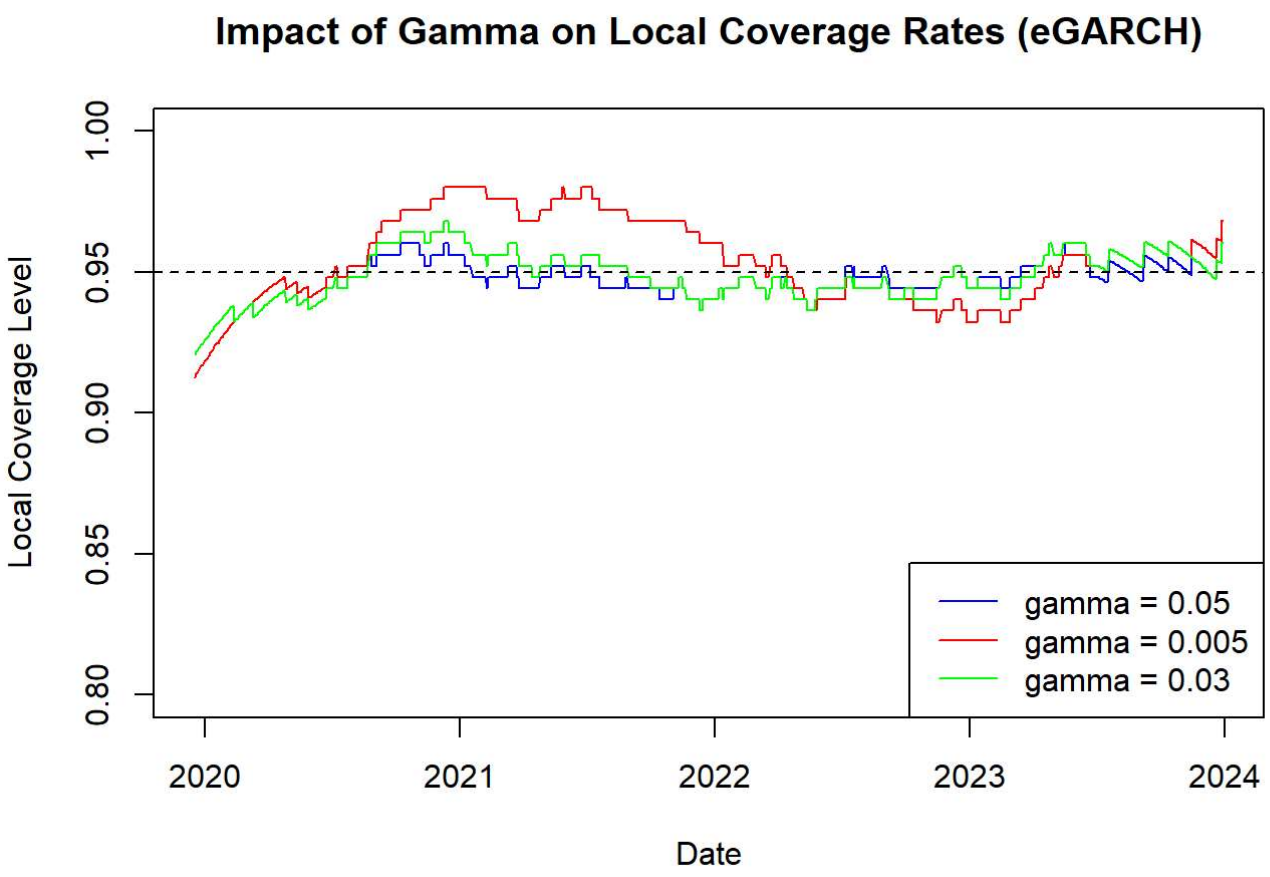
When **gamma** is increased, the model becomes more reactive to short-term fluctuations. This means that the confidence level (**alphat**) is updated more quickly in response to recent errors. Such responsiveness can be advantageous in highly volatile environments, where rapid changes in market conditions necessitate immediate adjustments. However, this increased sensitivity may also lead to over-adjustment, resulting in oscillations or instability in the confidence level. These oscillations can undermine the reliability of the forecast by introducing noise into the adjustment process.

In summary, the choice of **gamma** reflects a balance between responsiveness and stability. Higher values are suited to dynamic scenarios requiring quick adaptation, while lower values ensure smooth and consistent adjustments in more stable contexts. Selecting an appropriate **gamma** depends on the nature of the data and the specific objectives of the forecasting task, with careful tuning required to achieve optimal performance.

```r
# Gamma 3 different values
# Define gamma values to evaluate
gamma_values <- c(0.05, 0.005, 0.03)

# Create a list to store results
results_gamma <- list()

# Run the chosen strategy (Adaptive Studentized eGARCH) for each gamma value
for (gamma in gamma_values) {
  results <- garchConformalForcasting(
    returns = returns,
    alpha = 0.05,
    gamma = gamma,
    lookback = 1250,
    garchP = 1,
    garchQ = 1,
    startUp = 100,
    verbose = FALSE,
    updateMethod = "Simple",
    scoreType = "Studentized",
    garchType = "eGARCH"
  )

  # Store results for comparison
  results_gamma[[paste0("gamma_", gamma)]] <- results
}
```

```
## [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 2100 steps"
## [1] "Done 2200 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 2100 steps"
## [1] "Done 2200 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 2100 steps"
## [1] "Done 2200 steps"
```

```
# Extract error sequences for local coverage computation
local_coverage <- lapply(results_gamma, function(res) {
  compute_local_coverage(res$errSeqOC, window = 250)
})

# Plot local coverage rates for different gamma values
plot(date, local_coverage[[1]], type = "l", col = "blue", ylim = c(0.8, 1),
     main = "Impact of Gamma on Local Coverage Rates (eGARCH)", xlab = "Date", ylab = "Local Cov
erage Level")
lines(date, local_coverage[[2]], col = "red")
lines(date, local_coverage[[3]], col = "green")
abline(h = 0.95, col = "black", lty = 2)  # Target coverage level
legend("bottomright", legend = c("gamma = 0.05", "gamma = 0.005", "gamma = 0.03"),
       col = c("blue", "red", "green"), lty = 1,)
```




```
# Compute mean coverage rate and error rate for each gamma
gamma_metrics <- sapply(results_gamma, function(res) {
  error_rate <- mean(res$errSeqOC, na.rm = TRUE)
  coverage_rate <- 1 - error_rate
  c(Coverage_Rate = coverage_rate, Error_Rate = error_rate)
})

# Convert metrics into a data frame
gamma_metrics_df <- as.data.frame(t(gamma_metrics))
gamma_metrics_df$Gamma <- rownames(gamma_metrics_df)
rownames(gamma_metrics_df) <- NULL

# Print the metrics table
print(gamma_metrics_df)
```

```
##   Coverage_Rate Error_Rate       Gamma
## 1     0.9458128 0.05418719  gamma_0.05
## 2     0.9536946 0.04630542 gamma_0.005
## 3     0.9477833 0.05221675  gamma_0.03
```

gamma =0.05 Shows faster adaptation initially (reaching close to the target 0.95 quickly), but it exhibits larger fluctuations and slightly undershoots the target coverage over time. The rapid adjustment might lead to less stable long-term coverage.

gamma =0.03 Displays a more stable progression and smoother behavior around the 0.95 target. While it adapts slightly slower than the blue line, it achieves a more consistent coverage level in the long run.

The results reveal that the choice of the adaptation rate (γ) significantly impacts the stability and accuracy of the local coverage rates. From the plot, the green line (γ=0.03) demonstrates a smoother progression and maintains consistent alignment with the target coverage level of 0.95 over time. In contrast, the blue line (γ=0.05) adapts more rapidly in the initial periods but exhibits larger fluctuations and occasionally undershoots the desired coverage level. This suggests that while higher adaptation rates allow for quicker responses to changes in volatility, they may introduce instability in long-term coverage performance. The green line strikes a balance between adaptation speed and stability, making γ=0.03 a more robust choice for maintaining accurate and reliable prediction intervals.

***Gamma=0.03 is the chosen one*.**

Ex 2 : part 2

Online Expert Aggregation on ACI (AgACI)

Introduction

The second section of the paper of the Zaffran et al. (2022) talks about the Adaptive Conformal Inference (ACI). The ACI is an innovative approach because is very useful to extend Conformal Prediction (PC) to time series data, especially in the presence of changes in the distribution over time. We see how this is possible. The Conformal Prediction and the Adaptive Conformal Inference have the same goal: provide predictive intervals that contain the future value of the response variable with a given level of confidence, but they work in two different ways. The CP use a fixed α instead, in ACI, α_t change during the time learning over the errors of the past predictions. In fact, analyzing the past errors, the predict interval will be increased or decreased. α_{t+1} for the ACI is estimated by:

$$\alpha_{t+1} = \alpha_t + \gamma \cdot (\alpha - 1\{y_t \notin \mathcal{C}_{\alpha_t}(x_t)\})$$

where:

- $\mathcal{C}_{\alpha_t}(x_t)$ is the predicted interval and it assigns at y value 1 if y doesn't belong at the predict interval $\mathcal{C}_{\alpha_t}(x_t)$ instead if y is in the interval its value will be equal to 0
- γ is the Update factor

Can happen that in the time series are characterized by trend, seasonality or dependence by past values but for the ACI this is not a problem because thanks to the Update factor (γ) ACI can adapt to these characteristics. when happen this, the random split of the data is no longer functional, so we move on to a sequential split where the order of the data is respected.

As said, ACI is very useful to implement the Conformal Prediction to understand if the future value of a time series are or not in the predict interval, and to do this it uses the value of the past error with the update factor. Can happen that the value of the past error is negative; in this situation with updating the confidence levels/error (α_t), infinite predictive intervals can be generated. This happens because, specially with negative values of α, there is a tendency to make the length of the interval become indefinite ($Q_{1-\alpha_t} = +\infty$).
To resolve this problem, the paper introduces two different ways: the first one involves using a maximum value for the length which can be either Twice the maximum observed residual or the maximum known quantile; this

approach allows to calculate an approximate average length, but introduces a simplification that may not reflect the real performance of the method. Instead, the second way involves using the median instead of the mean; this approach is less influence from extreme values (the infinite) and it gives a better and stable length of the intervals.

To implement even more the ACI it's possible to modify the γ, making it an adaptive parameter, during the time; this is what happens in the **Online Expert Aggregation on ACI** (AgACI). This because choosing an optimal value of the predicted parameter is essential because it can influence the method. The AgACI works with the "**experts**"; for 1..k different value of the γ there are k different experts.

At each time t, with the ACI, we estimated the confidence interval for each value of the expert(1…K) and giving as result an interval of time t, for each expert, with 2 bound composed by an upper bound (u) and lower bound(l) $[b_k^{(l)}(x_t), b_k^{(u)}(x_t)]$ and continue with a measure of the efficiency and validity of each expert, the method **pinball loss** where for each expert is given a weight that is updated over time,it penalizes experts who produce long intervals and those who do not include the true value y_t.

Pinball Loss for Interval Bounds

1. **Quantile View**:
   - The **lower bound** $b_{t,k}^{(\ell)}(x)$ is treated like the $\beta^{(\ell)} = \frac{\alpha}{2}$ quantile.
   - The **upper bound** $b_{t,k}^{(u)}(x)$ is treated like the $\beta^{(u)} = 1 - \frac{\alpha}{2}$ quantile.
2. **Pinball Loss Definition**:
   For a quantile level β, the pinball (check) loss $\rho_\beta(y, \hat{q})$ is defined as

$$\rho_\beta(y, \hat{q}) \;=\; \begin{cases} \beta\,(y - \hat{q}), & \text{if } y \geq \hat{q}, \\ (\beta - 1)\,(y - \hat{q}), & \text{if } y < \hat{q}. \end{cases}$$

   - Apply $\rho_{\beta^{(\ell)}}$ to the lower bound, and $\rho_{\beta^{(u)}}$ to the upper bound.

Online Expert Aggregation rule

At each time t, after experts produce their bounds and we observe y_t, we:

1. **Compute Losses**:
   - For each expert k, evaluate the **cumulative** pinball loss over times $s = 1, \ldots, t$:

$$L_{t,k}^{(\ell)} \;=\; \sum_{s=1}^{t} \rho_{\beta^{(\ell)}}\Big(y_s,\; b_{s,k}^{(\ell)}(x_s)\Big),$$

$$L_{t,k}^{(u)} \;=\; \sum_{s=1}^{t} \rho_{\beta^{(u)}}\Big(y_s,\; b_{s,k}^{(u)}(x_s)\Big).$$

2. **Update Weights**:
   - Each expert k has weights $\omega_{t,k}^{(\ell)}$ and $\omega_{t,k}^{(u)}$, starting (for example) at 1:

$$\omega_{1,k}^{(\ell)} = \omega_{1,k}^{(u)} = 1, \quad \forall k.$$

   - An **online aggregator rule** Φ then updates the weights, typically as a function that decreases with the cumulative loss:

$$\omega_{t,k}^{(\ell)} \;=\; \Phi\Big(L_{t,k}^{(\ell)}\Big), \quad \omega_{t,k}^{(u)} \;=\; \Phi\Big(L_{t,k}^{(u)}\Big).$$

The paper uses the BOA method to update the weight over time. The final result will be an interval with a new lower and upper bound with the aggregation of all the expert with the updated weights:

$$b_{t+1}^{(l)}(x) = \frac{\sum_{k=1}^{K} w_k^{t+1} \cdot b_k^{(l)}(x)}{\sum_{k=1}^{K} w_k^{t+1}}, b_{t+1}^{(u)}(x) = \frac{\sum_{k=1}^{K} w_k^{t+1} \cdot b_k^{(u)}(x)}{\sum_{k=1}^{K} w_k^{t+1}}.$$

Conclusion

Online Expert Aggregation on ACI (AgACI) is a method that:

1. **Runs multiple ACI procedures** (each with a different γ_k) in parallel,
2. **Scores** each one using **pinball loss** for upper/lower bounds,
3. **Aggregates** these bounds via an **online weighting** scheme emphasizing better-performing experts,
4. Yields a **single** final interval at each time step with **adaptive** coverage properties and improved robustness over a single ACI approach.

Part B

eGARCH focus on estimating the **variance** of a time series and then using this variance to adjust prediction interval using γ.
AgACI: Directly estimates **quantiles** (bounds) for prediction intervals at each time step, rather than focusing on variance + The aggregation step weights these bounds based on their historical performance (via pinball loss), creating a **data-driven ensemble approach** to interval prediction.

The loss function focuses on the quality of interval bounds, not variance or volatility. It treats interval estimation as a **quantile regression problem**, where the goal is to accurately predict the lower and upper bounds of a response variable.By combining multiple experts through a weighted aggregation, AgACI improves robustness and adaptiveness without relying on assumptions about the volatility of the data.

We chose eGARCH due to its ability to capture volatility clustering and asymmetry, key characteristics of financial time series.

Although quantile regression could be a potential alternative for constructing prediction intervals, the choice of GARCH in this context is justified by its ability to model the time-dependent volatility dynamics inherent in financial data. GARCH models excel in capturing volatility clustering, a hallmark of financial time series, where periods of high volatility are followed by relative calm. This capability enables GARCH to forecast conditional volatility (σ), which is used to normalize returns (via studentized scores) and form the basis for adaptive prediction intervals. In contrast, quantile regression directly estimates conditional quantiles (e.g., 5th and 95th percentiles) without explicitly modeling volatility. While effective for static distributions, quantile regression lacks the built-in mechanism to account for temporal volatility patterns, making it less suited for data with dynamic variance.

To integrate GARCH into the AgACI framework, we adapt the algorithm to replace the quantile-based conformity score with a GARCH-based approach that leverages conditional volatility forecasts. Each expert corresponds to a distinct gamma γ value, controlling how the prediction intervals adapt over time. The GARCH model is employed to estimate conditional variance σ_t^2, capturing volatility clustering and time-varying variance dynamics. Using these forecasts, we compute studentized conformity scores, defined as the deviation of actual returns from predicted variance, normalized by the variance itself. These scores guide the dynamic adjustment of the miscoverage level

α_t for each expert, ensuring that intervals remain adaptive to evolving data patterns. Finally, an aggregation step combines the prediction intervals from all experts using a rule . This GARCH-based AgACI approach combines the strengths of volatility modeling with the robustness of ensemble prediction, effectively adapting to the dynamic characteristics of financial time series.

Basic Description of the BOA Rule

The Bayes Optimal Aggregation (BOA) rule is a probabilistic method that combines predictions from multiple experts to minimize expected loss. By assigning weights to each expert based on Bayesian principles, BOA ensures that the aggregation reflects the likelihood of each expert's accuracy.

How BOA Works

Weight Updates

- BOA begins with **prior probabilities** for each expert, representing initial beliefs about their reliability.
- As new data is observed, these probabilities are updated using Bayes' theorem, considering the likelihood of the observed outcomes given each expert's predictions.
- Experts with higher posterior probabilities are assigned greater weights in the aggregation process.

Formula

The weight w_i for expert i is calculated as:

$$w_i = \frac{P(D \mid h_i) \cdot P(h_i)}{\sum_j P(D \mid h_j) \cdot P(h_j)}$$

where:

- $P(D \mid h_i)$ is the likelihood of the data given expert i's predictions.
- $P(h_i)$ is the prior probability of expert i.
- The denominator ensures that the weights sum to 1.

BOA aims to minimize the expected cumulative loss **by weighting experts according to their posterior probabilities,** leading to decisions that are coherent with the current data and prior beliefs.
Through Bayesian updating, BOA dynamically adjusts expert weights as new data becomes available, allowing it to respond to changes in expert performance over time.

By combining Bayesian principles with expert aggregation, BOA provides a robust and theoretically sound method for adapting to varying levels of expert reliability over time.

This code integrates **eGARCH modeling** with the **Adaptive Conformal Inference (ACI)** framework to dynamically construct prediction intervals for financial time series data. The goal is to forecast conditional volatility using an eGARCH(1,1) model and create adaptive prediction intervals that reflect both historical volatility patterns and recent prediction accuracy. To achieve this, multiple "experts" are created, each corresponding to a different learning rate γ_t for adjusting confidence levels α_t dynamically over time. The final prediction interval is derived by aggregating the intervals from all experts using the **Bernstein Online Aggregation (BOA)** rule.

```r
library(quantmod)
library(rugarch)
library(opera)
```

```
## Warning: package 'opera' was built under R version 4.3.3
```

```r
agaciVolatilityForecasting <- function(returns, alpha = 0.05,
                                       gammaGrid = seq(0.001, 0.01, by = 0.002),
                                       lookback = 1250, garchP = 1, garchQ = 1,
                                       startUp = 100, verbose = FALSE,
                                       updateMethod = "Simple", momentumBW = 0.95) {
  myT <- length(returns)
  T0 <- max(startUp, lookback)

  # Define GARCH specification
  garchSpec <- ugarchspec(
    mean.model = list(armaOrder = c(0, 0), include.mean = FALSE),
    variance.model = list(model = "eGARCH", garchOrder = c(garchP, garchQ)),
    distribution.model = "norm"
  )

  # Initialize storage variables
  K <- length(gammaGrid)
  Tlen <- myT - T0

  # Storage for forecasts, errors, and intervals
  garchForecastVec <- rep(NA, Tlen)
  scores <- array(NA, dim = Tlen)
  alphaSeq <- matrix(alpha, nrow = K, ncol = Tlen)
  alphaSeq2 <- rep(alpha, Tlen)  # Base alpha for BOA
  errSeqOC <- matrix(NA, nrow = K, ncol = Tlen)
  aggregatedalpha <- rep(NA, Tlen)
  err <- rep(NA, Tlen)
  lower_limit <- 0.001
  upper_limit <- 0.999

  # Storage for expert intervals
  experts_low <- matrix(NA, nrow = K, ncol = Tlen)
  experts_high <- matrix(NA, nrow = K, ncol = Tlen)

  # === 1) Loop over test set ===
  for (t in T0:(myT - 1)) {
    # Fit GARCH model
    garchFit <- ugarchfit(garchSpec, returns[(t - lookback + 1):(t - 1)], solver = "hybrid")
    sigmaNext <- sigma(ugarchforecast(garchFit, n.ahead = 1))
    garchForecast <- sigmaNext^2  # Variance forecast

    colIdx <- t - T0 + 1  # Column index for storage

    # Save GARCH forecast
    garchForecastVec[colIdx] <- garchForecast

    # Compute conformity score
    scores[colIdx] <- abs(returns[t]^2 - sigmaNext^2) / sigmaNext^2
    historical <- scores[max(1, colIdx - lookback + 1):(colIdx - 1)]

    # Update alpha and compute expert intervals
    for (g in seq_along(gammaGrid)) {
```

```
      gamma <- gammaGrid[g]

      if (t == T0) {
        thr <- quantile(historical, 1 - alpha)
        errSeqOC[g, colIdx] <- as.numeric(scores[colIdx] > thr)
      } else {
        thr <- quantile(historical, 1 - alphaSeq[g, colIdx - 1])
        errSeqOC[g, colIdx] <- as.numeric(scores[colIdx] > thr)

        # Update alphaSeq
        if (updateMethod == "Simple") {
          alphaSeq[g, colIdx] <- alphaSeq[g, colIdx - 1] + gamma * (alpha - errSeqOC[g, colIdx])
        } else if (updateMethod == "Momentum") {
          w <- rev(momentumBW^(1:(colIdx - 1)))
          w <- w / sum(w)
          alphaSeq[g, colIdx] <- alphaSeq[g, colIdx - 1] + gamma * (alpha - sum(errSeqOC[g, colI
dx] * w))
        }

        # Clipping
        alphaSeq[g, colIdx] <- min(upper_limit, max(lower_limit, alphaSeq[g, colIdx]))
      }

      # Compute expert prediction intervals
      scale <- sqrt(garchForecast)
      experts_low[g, colIdx] <- -scale * qnorm(1 - alphaSeq[g, colIdx] / 2)
      experts_high[g, colIdx] <- scale * qnorm(1 - alphaSeq[g, colIdx] / 2)
    }
  }

  # 2) BOA Aggregation
  mlpol_grad_low <- mixture(
    Y = rep(0, Tlen),  # Placeholder for aggregation (not using Y values here)
    experts = t(experts_low),
    model = "BOA",
    loss.gradient = TRUE,
    loss.type = list(name = "pinball", tau = alpha / 2)
  )

  mlpol_grad_high <- mixture(
    Y = rep(0, Tlen),  # Placeholder for aggregation
    experts = t(experts_high),
    model = "BOA",
    loss.gradient = TRUE,
    loss.type = list(name = "pinball", tau = 1 - alpha / 2)
  )

  aggregated_low <- mlpol_grad_low$prediction
  aggregated_high <- mlpol_grad_high$prediction

  # 3) Return Results
  return(list(
```

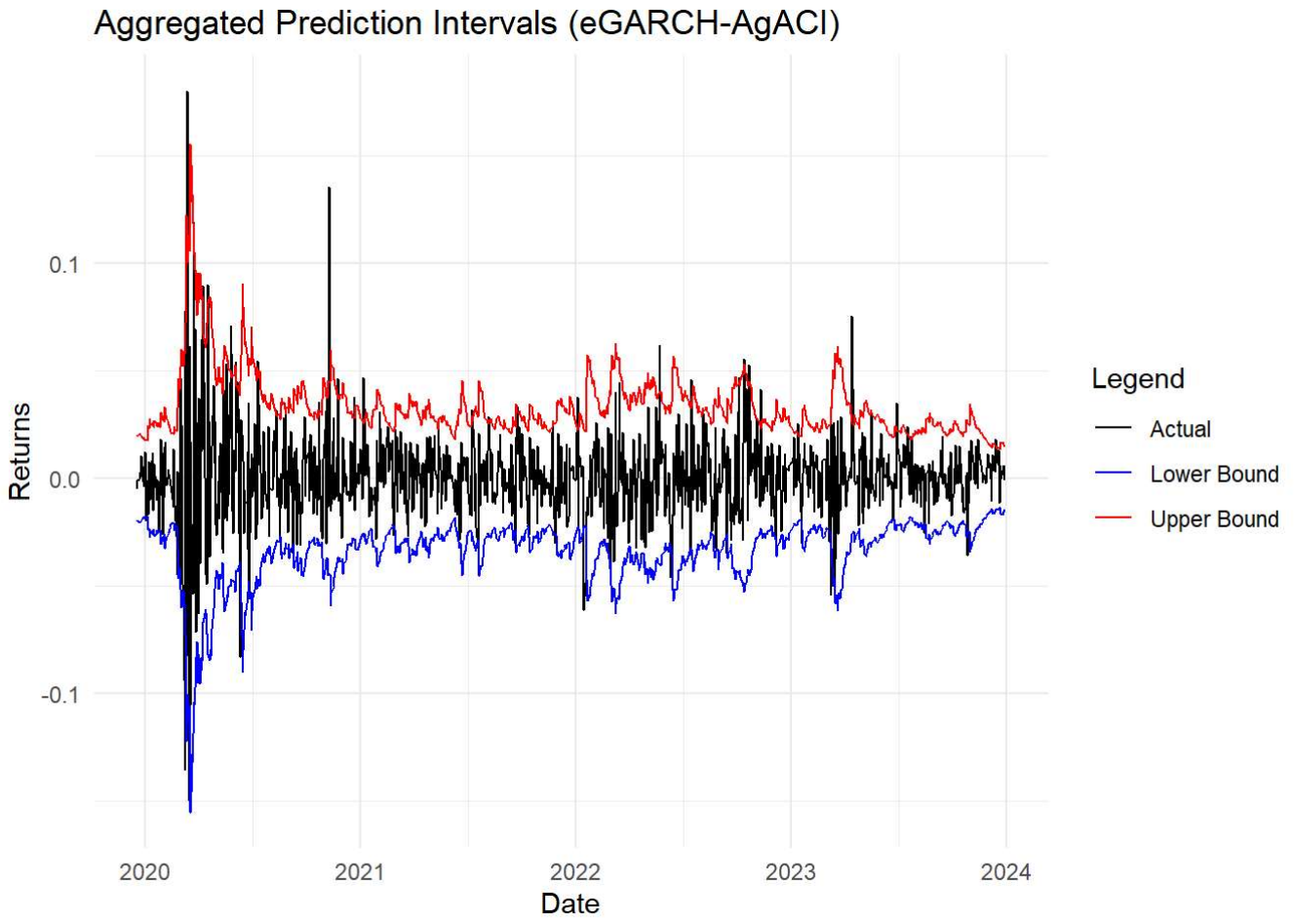
```
    Y_low = aggregated_low,
    Y_high = aggregated_high,
    experts_low = experts_low,
    experts_high = experts_high,
    alphaSeq = alphaSeq,
    errSeqOC = errSeqOC,
    garchForecastVec = garchForecastVec
  ))
}


gammaGrid = seq(0.001, 0.01, by = 0.002)
results <- agaciVolatilityForecasting(
  returns = returns,
  alpha = 0.05,
  gammaGrid = gammaGrid,
  lookback = 1250,
  garchP = 1,
  garchQ = 1,
  startUp = 100,
  verbose = TRUE,
  updateMethod = "Simple",
  momentumBW = 0.95
)
```

The code works by fitting an eGARCH model to recent historical returns to forecast conditional variance σ_t^2 for the next time step. Using these forecasts, prediction intervals are constructed for each expert based on their dynamically updated confidence levels. A conformity score, which measures how well the predicted intervals align with actual returns, is used to adjust α_t iteratively for each expert. BOA is then applied to combine the individual intervals into a single robust interval by assigning weights to experts based on their historical performance. This approach ensures the intervals are both adaptive to changing market conditions and robust against individual model inaccuracies.

```
# Plot aggregated prediction intervals
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 4.3.3
```

```
plot_df <- data.frame(
  Date = index(returns)[(length(returns) - length(results$Y_low) + 1):length(returns)],
  Lower = results$Y_low,
  Upper = results$Y_high,
  Actual = as.numeric(returns[(length(returns) - length(results$Y_low) + 1):length(returns)])
)

ggplot(plot_df, aes(x = Date)) +
  geom_line(aes(y = Actual, color = "Actual")) +
  geom_line(aes(y = Lower, color = "Lower Bound")) +
  geom_line(aes(y = Upper, color = "Upper Bound")) +
  labs(
    title = "Aggregated Prediction Intervals (eGARCH-AgACI)",
    y = "Returns",
    color = "Legend"
  ) +
  theme_minimal() +
  scale_color_manual(values = c("Actual" = "black", "Lower Bound" = "blue", "Upper Bound" = "red"))
```

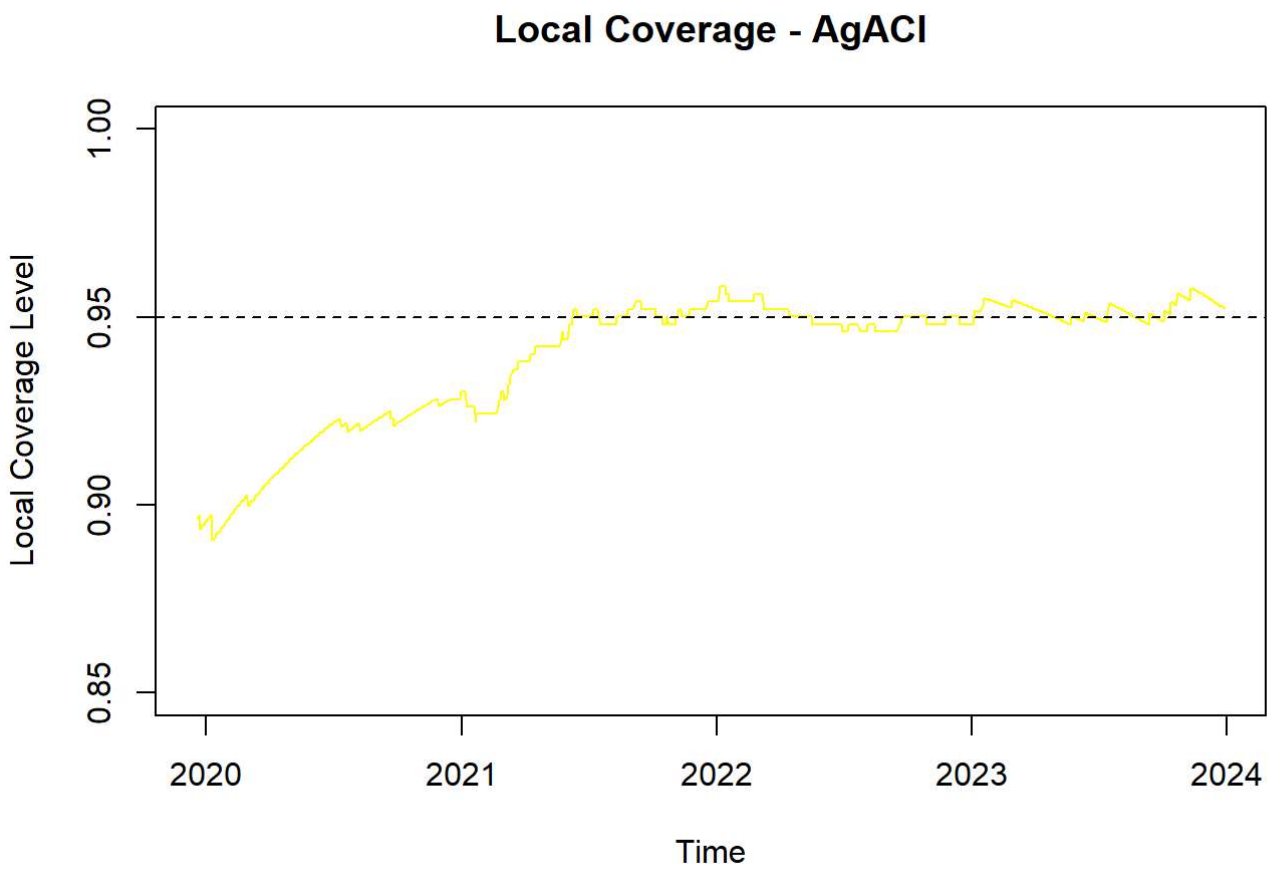


The plot shows the **aggregated prediction intervals** generated by the **eGARCH-AgACI framework** for a financial time series. The black line represents the actual returns, while the red and blue lines are the upper and lower bounds of the prediction intervals, respectively.

The intervals dynamically adjust to changing volatility, widening during high-volatility periods (e.g., early 2020) and narrowing during stable periods. The actual returns mostly fall within the bounds, indicating the method achieves the desired coverage level (e.g., 95%). Occasional deviations beyond the bounds reflect the inherent uncertainty of financial data.

Overall, the framework effectively adapts to market conditions, leveraging eGARCH to capture volatility clustering and BOA to combine expert predictions for robust and reliable intervals.

```
actual_returns <- as.numeric(returns[(length(returns) - length(results$Y_low) + 1):length(return
s)])
calculateLocalCov <- function(actual, lower, upper, windowSize = 500) {
  n <- length(actual)
  localCov <- rep(NA, n)

  for (t in 1:n) {
    startIdx <- max(1, t - floor(windowSize / 2))
    endIdx <- min(n, t + floor(windowSize / 2))
    covered <- (actual[startIdx:endIdx] >= lower[startIdx:endIdx]) &
      (actual[startIdx:endIdx] <= upper[startIdx:endIdx])
    localCov[t] <- mean(covered, na.rm = TRUE)
  }

  return(localCov)
}
localCov_agACI <- calculateLocalCov(
  actual = actual_returns,
  lower = results$Y_low,
  upper = results$Y_high,
  windowSize = 500
)

plot(index(returns)[(length(returns) - length(results$Y_low) + 1):length(returns)],
     localCov_agACI, type = "l", col = "yellow", ylim = c(0.85, 1),
     main = "Local Coverage - AgACI",
     xlab = "Time", ylab = "Local Coverage Level")
abline(h = 0.95, col = "black", lty = 2)  # Livello target 1 - alpha
```



Simple Studentized eGARCH (Green Line: gamma =0.03):

- This approach uses a fixed gamma value and adjusts prediction intervals based solely on conformity scores from the studentized returns.
- While the model generally aligns with the target coverage level of 0.95, it shows greater variability during certain periods, reflecting its sensitivity to individual model limitations and market volatility.

AgACI Strategy:

- The aggregated coverage rates adapt dynamically by leveraging multiple gamma values and the BOA aggregation rule.
- It balances the contributions of multiple experts, leading to a robust and stable coverage rate
- The local coverage rates are consistently close to the desired level of 0.95, with fewer fluctuations over time, particularly during periods of market stability.

I would choose the **AgACI strategy** over the simple studentized eGARCH due to its superior stability, adaptability, and reliability. AgACI provides smoother and more consistent coverage rates by aggregating predictions from multiple experts with varying learning rates, minimizing the impact of individual model inaccuracies. This dynamic combination allows AgACI to better adapt to changing market conditions, particularly during periods of high volatility, where the fixed approach of the studentized eGARCH may underperform. While the simple studentized eGARCH performs adequately in static conditions, AgACI's robustness makes it the better choice for financial time series with dynamic volatility patterns.