

S&P 500: ARMA-GARCH model

Daniel Herrera

2023-04-23

```
library(fGarch)
library(xts)
library(TSA)
library(rugarch)
```

Motivation

As you may have noticed, we switched the workflow a bit and are coding in R. Part of data science is knowing the advantages and disadvantages of languages and packages. R is generally highly regarded for statistical analysis; however, here it is very evident as we will be trying to model the S&P 500 using an ARMA(p,q) + GARCH(\tilde{p}, \tilde{q}) process. The package ‘rugarch’ allows for a very important implementation detail, that the ARMA and GARCH models are fit simultaneously. This is done using quasi-MLE because it allows us to not make a distribution about the distribution of the true error distribution, which is generally made out to be normally distributed.

Model Statement

$$X_t = \mu + \sum_{j=1}^p \phi_j X_{t-j} + \sum_{j=0}^q \theta_j W_{t-j}$$

$$W_t = \sigma_t \varepsilon_t$$

$$\sigma_t^2 = b_0 + \sum_{j=1}^{p_0} b_j W_{t-j}^2 + \sum_{j=1}^{q_0} a_j \sigma_{t-j}^2$$

We fit an ARMA(p,q) model onto X_t and then use quasi-MLE to estimate the ARMA and GARCH parameters simultaneously as mentioned previously.

We see when fitting an AR(1) + GARCH(1,1) that the arch and garch effects are significant p-value < 0.05 for α_1 and β_1 . As well, the distribution estimated is the student’s t-distribution with 4.9 df, essentially 5.

```
# load data
df <- read.csv('spy_returns.csv')

X = diff(log(df$Close))
time_series <- xts(x = X, order.by = as.Date(df$Date[2:length(df$Date)]))

# fit model
fit = garchFit(~arma(1,0) + garch(1,1), cond.dist = "std", data = X, trace = F)
summary(fit)
```

```

##
## Title:
## GARCH Modelling
##
## Call:
## garchFit(formula = ~arma(1, 0) + garch(1, 1), data = X, cond.dist = "std",
##          trace = F)
##
## Mean and Variance Equation:
## data ~ arma(1, 0) + garch(1, 1)
## <environment: 0x7f7c02427e28>
## [data = X]
##
## Conditional Distribution:
## std
##
## Coefficient(s):
##          mu          ar1          omega          alpha1          beta1          shape
## 1.0172e-03 -5.4067e-02  2.5875e-06  1.7955e-01  8.0900e-01  4.8524e+00
##
## Std. Errors:
## based on Hessian
##
## Error Analysis:
##      Estimate Std. Error t value Pr(>|t|)
## mu      1.017e-03  1.230e-04   8.272 2.22e-16 ***
## ar1     -5.407e-02  2.039e-02  -2.652  0.008 **
## omega    2.587e-06  5.918e-07   4.372 1.23e-05 ***
## alpha1   1.796e-01  2.405e-02   7.465 8.33e-14 ***
## beta1    8.090e-01  2.134e-02  37.903 < 2e-16 ***
## shape    4.852e+00  5.024e-01   9.659 < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Log Likelihood:
## 8753.595      normalized:  3.451733
##
## Description:
## Tue Apr 25 15:20:30 2023 by user:
##
##
## Standardised Residuals Tests:
##
##          Statistic p-Value
## Jarque-Bera Test  R      Chi^2 659.9502 0
## Shapiro-Wilk Test R      W      0.9712919 0
## Ljung-Box Test    R      Q(10) 11.94904 0.2884818
## Ljung-Box Test    R      Q(15) 20.72318 0.1459095
## Ljung-Box Test    R      Q(20) 28.97101 0.08833168
## Ljung-Box Test    R^2 Q(10) 6.474836 0.773918
## Ljung-Box Test    R^2 Q(15) 15.18708 0.4380273
## Ljung-Box Test    R^2 Q(20) 17.59089 0.6143384
## LM Arch Test      R      TR^2 9.174317 0.6879737
##
## Information Criterion Statistics:

```

```
##          AIC          BIC          SIC          HQIC
## -6.898734 -6.884921 -6.898745 -6.893723
```

The resulting model is as follows:

$$\begin{aligned}X_t &= \mu + \phi_1 X_{t-1} + W_t \\W_t &= \sigma_t \varepsilon_t \\\sigma_t^2 &= \omega + \alpha_1 W_{t-1}^2 + \beta_1 \sigma_{t-1}^2\end{aligned}$$

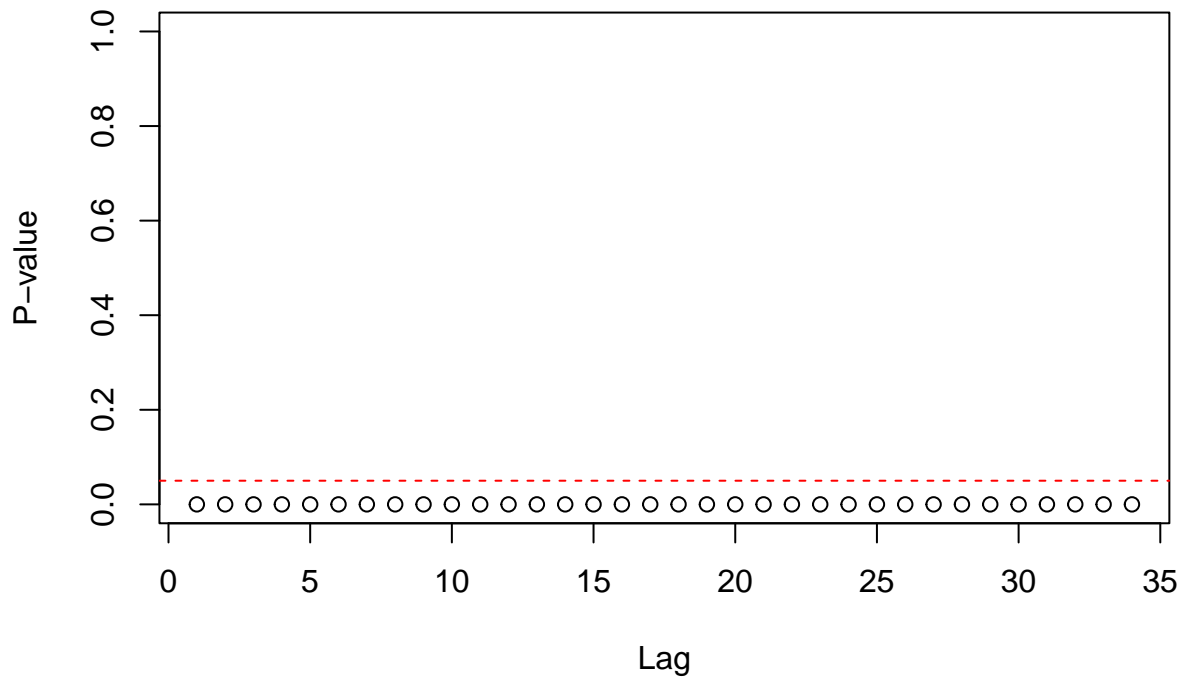
$$\begin{aligned}X_t &= 1.017 \times 10^{-3} - 5.407 \times 10^{-2} X_{t-1} + W_t \\W_t &= \sigma_t \varepsilon_t \\\sigma_t^2 &= 2.587 \times 10^{-6} + .1796 W_{t-1}^2 + .809 \sigma_{t-1}^2\end{aligned}$$

McLeod Li Test

We revisit the McLeod-Li test to quickly check for arch effects to justify our model choice on the time series **after** fitting the AR(1) process.

Based on the McLeod-Li test, we recall that we would reject the null hypothesis that there are no ARCH effects in the time series and conclude that there are ARCH effects.

```
McLeod.Li.test(arima(X, order = c(1,0,0)))
```



Predictions

Below we notice that we are still predicting 0 for our mean model. This occurs due to the nature of our forecasting process, predicting n steps ahead where n is large and our assumption of a constant mean. Thus, as n increases, our model approaches the μ as estimated by our fit, which is 0.00099 as shown in the coefficients below.

```
# test - train split
train_p= 0.8

create_split <- function(ts, train_p) {
  train_len = round(length(ts) * train_p)
  train_ts = ts[1:train_len]
  test_ts = ts[c((train_len + 1):length(ts))]

  return(list(train = train_ts, test = test_ts))
}

data_spl = create_split(time_series, 0.8)
train = data_spl$train
test = data_spl$test

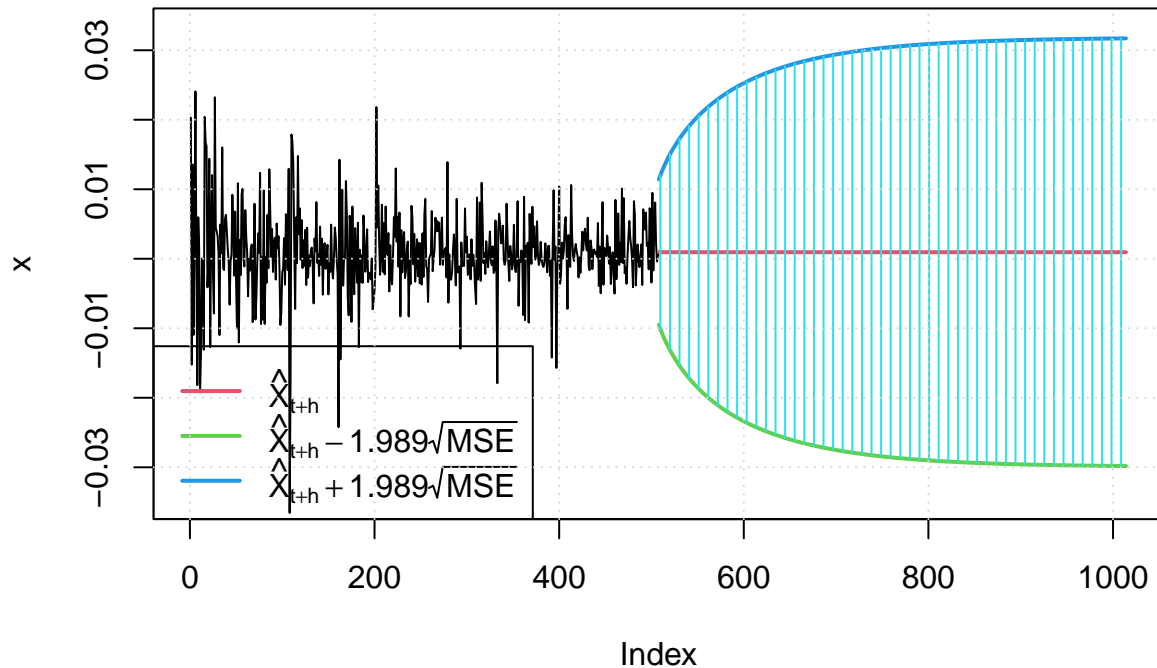
# fit the model with training data
ar1g_fit = garchFit(~arma(1,0) + garch(1,1), cond.dist = "std", data = train, trace = F)
print(summary(ar1g_fit))
```

```
##
## Title:
## GARCH Modelling
##
## Call:
## garchFit(formula = ~arma(1, 0) + garch(1, 1), data = train, cond.dist = "std",
## trace = F)
##
## Mean and Variance Equation:
## data ~ arma(1, 0) + garch(1, 1)
## <environment: 0x7f7c22374490>
## [data = train]
##
## Conditional Distribution:
## std
##
## Coefficient(s):
##          mu          ar1          omega          alpha1          beta1          shape
## 9.9789e-04 -5.9579e-02  2.1985e-06  1.6753e-01  8.2333e-01  4.8456e+00
##
## Std. Errors:
## based on Hessian
##
## Error Analysis:
##          Estimate Std. Error t value Pr(>|t|)
## mu          9.979e-04  1.364e-04   7.314 2.59e-13 ***
## ar1         -5.958e-02  2.266e-02  -2.629 0.008569 **
## omega        2.199e-06  6.307e-07   3.486 0.000491 ***
## alpha1       1.675e-01  2.666e-02   6.284 3.30e-10 ***
## beta1        8.233e-01  2.427e-02  33.924 < 2e-16 ***
```

```
## shape 4.846e+00 5.659e-01 8.563 < 2e-16 ***
## ---
## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Log Likelihood:
## 7020.609 normalized: 3.460132
##
## Description:
## Tue Apr 25 15:20:30 2023 by user:
##
##
## Standardised Residuals Tests:
##
##               Statistic p-Value
## Jarque-Bera Test  R    Chi^2 436.264 0
## Shapiro-Wilk Test R    W    0.9740287 0
## Ljung-Box Test   R    Q(10) 10.48056 0.3993899
## Ljung-Box Test   R    Q(15) 16.78574 0.3318337
## Ljung-Box Test   R    Q(20) 23.01495 0.288061
## Ljung-Box Test   R^2  Q(10) 6.01504 0.8139981
## Ljung-Box Test   R^2  Q(15) 14.55819 0.483684
## Ljung-Box Test   R^2  Q(20) 17.80389 0.600325
## LM Arch Test     R    TR^2 7.569294 0.8178186
##
## Information Criterion Statistics:
##      AIC      BIC      SIC      HQIC
## -6.914350 -6.897745 -6.914368 -6.908258
##
## NULL
```

```
# make future fcs
fcst <- predict(ar1g_fit, n.ahead = length(test), plot = T)
```

Prediction with confidence intervals



Rolling Window (One-Step ahead)

We switch to a more practical approach of using a one-step ahead forecast for $\hat{\sigma}_t$. In essence, we are fitting a series of ARMA + GARCH models to create our forecasts. We first fit our model on the train set, then given the new observed value at time $t+1$, we fit another model using all previous values until time $t+1$ to predict time $t+2$. This seems like the most practical implementation in practice if we wanted to predict these volatilities day to day to create trading strategies.

The plot below shows the predicted volatility of the test set using the one-step ahead forecast approach and the true returns of the S&P500. We can see how the volatility peaks align closely with the swings in returns of the market and additionally give insight into quieter market periods.

```
rolling_forecast <- function(y_train, y_test, p_ar, q_ma, p_garch, q_garch) {
  n_test <- length(y_test)
  rolling_preds <- c()

  # change back to 1:n_test
  for (i in 1:n_test) {
    # Make special case for initial fcst
    if (i == 1) {
      train <- y_train
    }
    else{
      train <- c(y_train, y_test[1:(i - 1)])
    }
  }
}
```

```

# Define the ARMA-GARCH model
# s is standard garch model
# distribution is students t dist
spec <- ugarchspec(variance.model = list(model = "sGARCH", garchOrder = c(p_garch, q_garch)),
                    mean.model = list(armaOrder = c(p_ar, q_ma), include.mean = TRUE),
                    distribution.model = "std")

# Fit the model
model_fit <- ugarchfit(spec, train, solver = "hybrid")

# One step ahead forecast
pred <- ugarchforecast(model_fit, n.ahead = 1)
rolling_preds[i] <- sigma(pred)[1]
# rolling_preds[i] <- sqrt(as.numeric(pred@forecast$sigmaFor[1]))

}

# # Convert the predictions to a time-series object with the same index as y_test
# rolling_preds_ts <- ts(rolling_preds, start = start(y_test), end = end(y_test), frequency = frequen

return(rolling_preds)
}

# Assuming y_train and y_test are your training and testing time-series data
p_ar <- 1
q_ma <- 0
p_garch <- 1
q_garch <- 1

rolling_preds <- rolling_forecast(train, test, p_ar, q_ma, p_garch, q_garch)

# need to convert rolling preds to ts
time_index <- time(test)

## Plot the test set
plot(time_index, rolling_preds, type = "l", col = "red", main = "Test Set vs. Rolling Predictions", xlab = "Time",
      ylim = c(-0.04, 0.04))

# Add the rolling predictions to the existing plot
lines(time(test), test[,1], col = "black")

# Add a legend to the plot
legend("topleft", legend = c("Test Set", "Rolling Predictions"), col = c("black", "red"), lty = 1, bty = "n")

```

Test Set vs. Rolling Predictions

