GARCH Model in stock returns and predictive Ability in the Change of Interest Rate

Alexander Dautel

Thorsten Disser

Binhui Hu

Nicolas Yiannakou

Berlin, 14.08.2016

Contents

1 Introduction, Theory and Design

1.1 Introduction

When taking a look at the stock market, DAX variation attracted our attention. From the data published in EU Finance, we can easily conclude the DAX varied much between the years 2001 and 2005. In the meantime, the key interest rates were changed by the European Central Bank many times. This phenomenon attracted our attention. In the twentieth century, Andrew A. ChristieIn (1982), Mark J. Flannery (1984), Christopher M. James(1984) and many other scientists explored the relation between theses two factors by applying many statistical and financial models. In our report, we extract the data from the DAX index and the interest rate levels and changes during the years 2001 and 2005, and try to discover the relationship between the stock returns and the change of the interest rate based on time series models.

1.2 Theory and Design

1.2.1 Theory Background

In order to implement our motivations, we propose the following time series models: In our report, we mainly focus on applying statistical tests on log returns of the DAX data. To be more precise,we pre-process the the DAX data into the log returns in the next section in order to prepare for modelbuilding in the later process, such as ARMA, ARCH and GARCH models, or splitting the data set at specific breakpoints or into homogenous intervals. Then we compare the time point of breakpoints in log returns and change of the interest rate, so that in the finial section we can conclude to what degree the returns can successfully predict a change of the interest rate.

1.2.2 Software support

1. Programming in R, using Rstudio to for the model building
2. Presentation slides and report based on Latex

2 Data extraction and Pre-processing

2.1 Data Extraction

We are aiming at exploring the relationship between stock returns and changes in the interest rate during 2001 and 2005. To meet this goal, our completed raw data consisted of these two parts. The DAX index data comes from EU finance and the interest rate resources from the European Central Bank (`https://www.ecb.europa.eu`). In the end of this part, we take a look at the raw data. Firstly, we focus on the DAX index. To achieve this aim, we install the *zoo* package so that data is properly ordered according to a timeline with format *zoo*, we convert our time series through the following programming:

```
1 library(zoo)
2 SPL.zoo = zoo(SPL[,-1], order.by =
3           as.Date(strptime(as.character(SPL[,  1]), "%Y-%m-%d")))
```

Now we generated a plot via Rstudio based on the following programming:

```
1 par(mfrow = c(1,  1), mar = c(4,  2,  1,  1))
2 prices  = SPL.zoo$Adj.Close
3 plot(prices,  type = "l",  col = "blue3",
4           xlab = "Time",  ylab = "Returns",  xlim = NULL,  lwd = 1)
5 title(main = "Demeaned Price",  cex.main = 1)
```

Looking at the plot of the DAX data, there exists a fluctuated trend. Between the year 2001 and the middle of the year 2003, the DAX data declined from the peak at around 6000 to 2000, later a slowly increasing trend was shown with fluctuation to around 5500 by the end of the year 2005. However, the expectation and variance changed over time, and thus, the data is obviously non-stationary. Secondly, the interest rate changed seven times in the five years, here we list the time points and the corresponding rates after official announcement:

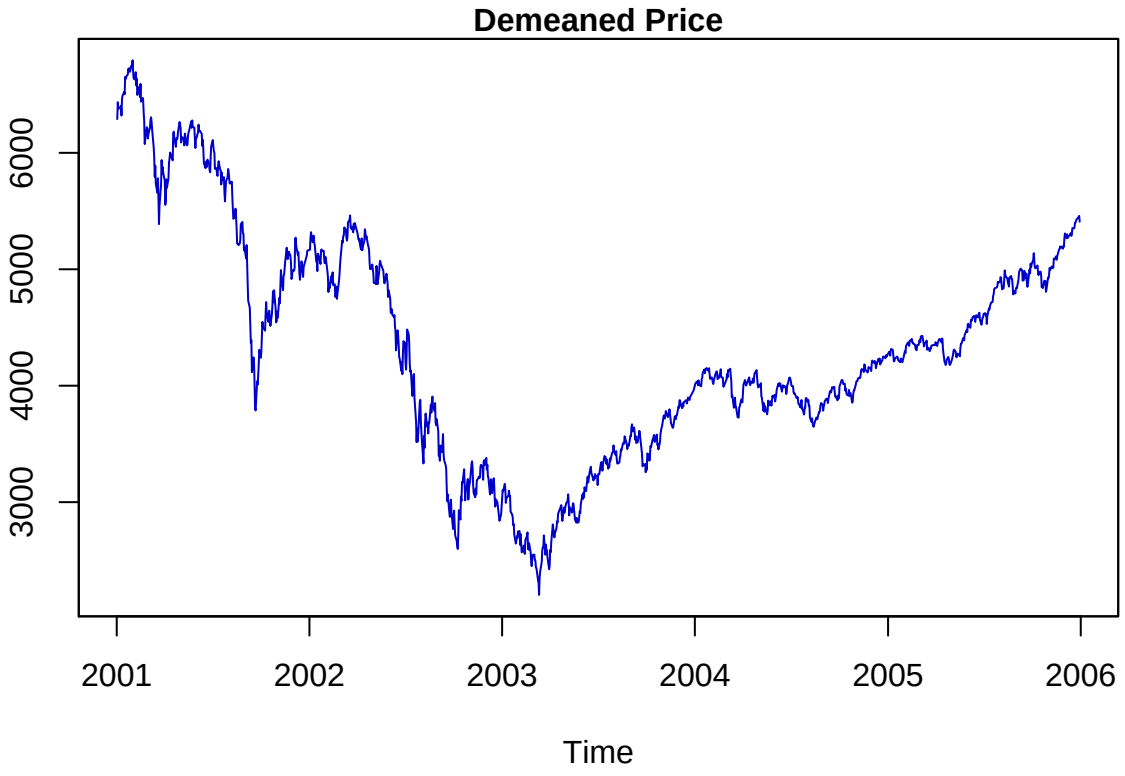

Figure 1: Demeaned Price

Time point	Percentage
2005-12-06	3.25
2003-06-06	3.00
2003-03-07	3.50
2002-12-06	3.75
2001-11-09	4.25
2001-09-18	4.75
2001-08-31	5.25
2001-05-11	5.50
2001-01-02	5.75

Table 1: Interest Rate Changes

This table indicates a slightly increasing trend from 3.00% to 5.75% over the years and the adjusting of interest rate is more frequent in the year 2001.

2.2 Pre-processing

In order to obtain the data with more statistical characteristics. Therefore, we intend to calculate the log returns of the time series. We firstly generate the log returns with `returns = diff(log(prices))` where prices yield the data with *zoo* package.
In order to obtain the overall fluctuating trend and impression, we subtract the mean of log returns, and draw the plot:

```
1 par(mfrow = c(1, 1), mar = c(4, 2, 1, 1))
2 plot(returns.dm, type = "l", col = "blue3",
3          xlab = "Time", ylab = "Returns", xlim = NULL, lwd = 1)
4 title(main = "Demeaned Returns", cex.main = 1)
```

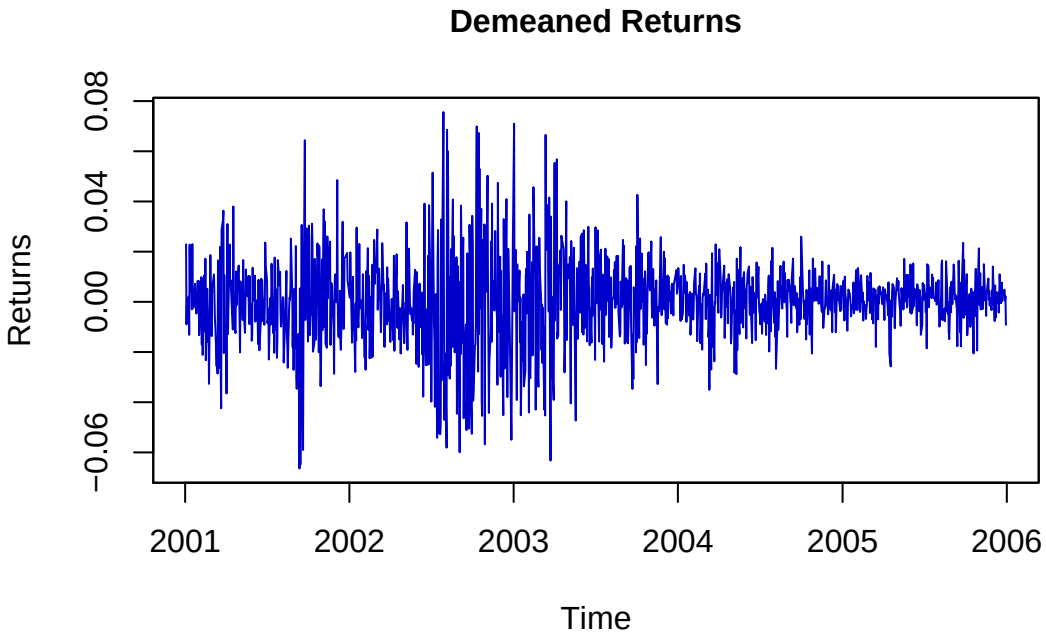

Figure 2: Demeaned Returns

From the descriptive figure, we can conclude the following characteristics : Firstly, the log return is not homogeneous. For example, the variances between the month in January of 2002 and in the January of 2003, are significantly different compared with the range of the data in the figure. Secondly, high frequency returns of most financial assets usually exhibit volatility clustering. When we take a look at the above figure, the volatility clustering is shown, where the volatility clustering in finance refers to the series of observations, as noted as Mandelbrot (1963), that "large changes tend to be followed by large changes, of either sign, and small changes tend to be followed by small changes." In our case, e.g. from June of 2002 till May of 2003 the comparatively larger variances clustered together, which means large changes tend to be followed by large changes while on the contrary, variances are smaller in the period between June of 2001 and May of 2002.

Now we could also summarize the data according to the command `summary(returns.dm)`:

	Index	returns.dm
Min.	2001-01-03	-0.0664035
1st Qu	2002-04-08	-0.0088866
Median	2003-07-09	0.0005585
Mean	2003-07-06	0.0000000
3rd Qu.:	2004-10-05	0.0086571
Max.	2005-12-30	0.0086571

Table 2: Returns

From this table, we can conclude the beginning and the end yield the minimum and maximum of the data.

Moreover , with the command `n = length(prices)`, we could easily see the total number of observations is 1272, which is large enough to implement time series models afterwards.

In the rest of this section, we apply the Augmented Dickey-Fuller (ADF) test (via package *tseries*)for testing if our time serie of log return is stationary.

```
1 adf.test(returns.dm)
2 Augmented Dickey-Fuller Test
3 data: returns.dm
4 Dickey-Fuller = -11.123, Lag order = 10, p-value = 0.01
5 alternative hypothesis: stationary
```

We assume our significant level is 5%, our test p-value is $0.01 < 0.05$. Thus, we reject our null hypothesis (non-stationary) and choose the alternative hypothesis (stationary).

3 ARMA Process

3.1 Theory

We need to identify an appropriate model in order to make predictions about returns. Plotting returns we can clearly observe some volatility clustering. In other words, during some time periods the data seem to be more sensitive than others. In particular we see that during the second half of 2002 and the first half of 2003 returns seem to be fluctuate more intense than other periods of the investigated timespan. Those differences in mean deviation imply that regression error terms are not independent. Therefore, a GARCH (General Autoregressive Conditional Heteroskedasticity) model is appropriate for predicting DAX returns.
However, we need to specify an ARMA (Autoregressive Moving Average) model to generate the residuals needed for the GARCH process. It is useful to view the realization of a random variable as a combination of signal and noise, where signal can be a pattern of the random variables previous values and noise, deviations from the mean that are not explained by the model i.e the error terms. An ARMA model tries to separate this signal from the noise, and use this signal to obtain forecasts for future values. These types of models are composed by two polynomials, one associated with the autoregressive (AR) model and another one associated with the moving average (MA) model. The AR part of the model describes how and up to what extent the explanatory variable depends on its own past values, also mentioned as lags, while the MA part defines the way our output variable relies on past values of stochastic errors. The length of each polynomial is called the order and gives the number of lags for each part that have a significant direct impact on the value of the variable. AR and MA orders are notated with p and q respectively. The main application of an ARMA model is to be used as a predictive model. However, in order to do so the weakly stationary assumption has to be fulfilled. A series is called stationary when it has no trend, i.e. the mean is remains unchanged throughout the series, or autocorrelations, correlations with its own previous values, remain constant over time. In addition to this, variations around the mean has to be of the same latitude, therefore variance must be constant. In the case where the time series is not stationary an ARIMA model can be used to make a process stationary by differentiating the data one or several times. The main difference between an ARMA and an ARIMA model is the additional component d, which describes the number of differences needed for stationarity. In our model an ARIMA model with

differencing degree of one on the logarithmized DAX prices would yield theoretically the same results as an ARMA model of the same AR and MA order on returns, since one can observe returns by differentiating the logarithm of prices once.

3.2 Implementation

Selecting the appropriate model is a nontrivial task. A misspecification of the model can lead to misleading results. There are several ways to determine the order or the number of AR and MA terms. The most straightforward is to plot the Autocorrelation and Partial Autocorrelation functions. To do so we used the commands *acf* and *pacf* from the *stats* package as shown below:

```
1 acf(coredata(returns.dm), lag.max = NULL, plot = TRUE)
2 pacf(coredata(returns.dm), lag.max = NULL, plot = TRUE)
```

We had to use the *zoo* function to ignore the time index of our observations since they are of category *zoo* . The first one gives the correlation coefficients between the series and different lags of its self and it helps us to determine the order of MA part. The necessary number of MA terms is given by the point where the ACF cuts-off to insignificant levels. If there is no such a point and the ACF does not decreases smoothly then an MA term is not need and the series follows an AR series. In the case where ACF decreases smoothly differentiation of the series is needed. Similarly, the order of the AR part is appointed by the cut-off in the PACF. The intuition behind this is because partial autocorrelation is the correlation between a variable and a lag, which is explained exclusively by this lag.

As the ACF plot below shows, autocorrelation drops immediately to insignificant levels after the first lag indicating that the value of p should be zero. Similarly, the value of q should also be zero since the PACF plot indicates no significant values after the first lag. Thus, the suggested model is an ARMA (0,0) meaning that time series is white noise.

However, when the decision for a model is made based on ACF and PACF the results can be unclear and opinions about the optimality of the resulted model are subjective. Thus, we introduce an additional step to our model selection procedure to confirm our decision namely, by minimising the information criteria. When deciding about the order of a model one faces a trade-off between the risk of misspecification and the number of parameters of the model. An increase in the number of parameters leads

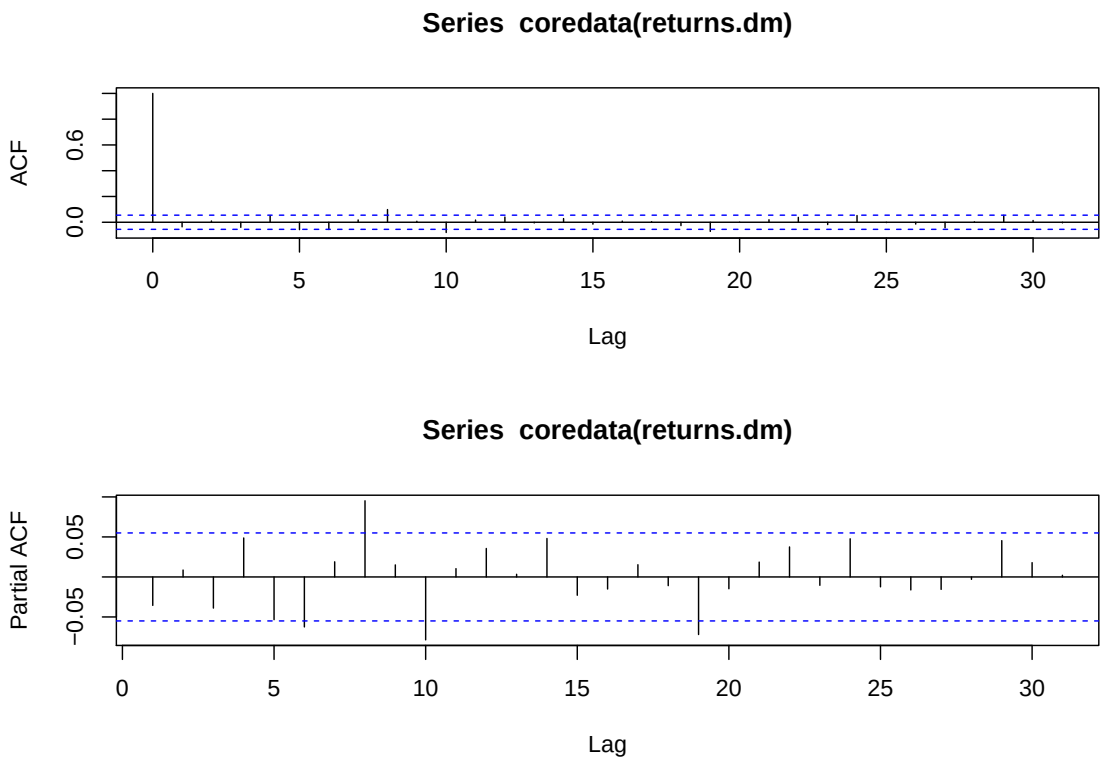

Figure 3: Autocorrelation and Partial Autocorrelation Functions

automatically to an improvement in the models ability to explain our observations yet it makes the model more complex and more difficult to estimate. The risk of overfitting also arises with increasing parameters where small deviations from the mean have a significant impact on the model leading to information loss about the real underlying pattern. Information criteria face this problem by adding a penalty factor for additional coefficients to the goodness of fit. There exist a variety of information criteria depending on the way additional coefficients are punished. We chose to use the Akaike Information Criterion (AIC) over the Bayesian Information Criterion (BIC) to compare different models because unlike BIC it gives the model closer to the truth. AIC estimates the Kullback- Leibler divergence, the information lost due to the deviation of the model from reality, based on the Maximum- Likelihood Estimation. As a result, the efficiency of model increases with lower AIC values because it means it is closer to the truth. On the other hand, BIC estimates the function of the posterior probability of a model being the real one, conditional on the observed data. To make the comparison process automatic we constructed an empty matrix with the number of columns and rows corresponding to the upper limit of MA and AR lags, respectively. We then created a loop, which calculates the AIC values for each model we want to compare, using the function we constructed at the beginning of the code, and fills in the empty matrix. The matrix position containing the lowest AIC value is printed using the "which" command designating the most appropriate model. The whole procedure is described in the upcoming lines:

```
1  ltt.arima = 2
2  aic.arima.matrix = matrix(NA, ncol = ltt.arima + 1,
3                                          nrow = ltt.arima + 1)
4  colnames(aic.arima.matrix) = 0:2 #order of MA model (q)
5  rownames(aic.arima.matrix) = 0:2 #order of AR model (p)
6
7  for (i in 0 : ltt.arima) {
8    for (j in 0 : ltt.arima) {
9      aic.arima.matrix[i + 1, j + 1] = AIC(arima(returns.dm,
10                                         order = c(i, 0, j)))
11   }
12 }
13
```

```
aic.arim.min = min(aic.arima.matrix)
which(aic.arima.matrix == aic.arim.min, arr.ind = T)
```

We determined the upper limit of lags equal to two for both parts considering the literature, which supports that for this family of models the summation of coefficients should not exceed three lags and the maximum of lags for each part should be equal to two. The idea behind this is twofold. Firstly, a model with many parameters becomes vary complex and is difficult to estimate. Secondly, the risk for overfitting emerges with all the negatives discussed above.

	0	1	2
0	-6697.13	-6698.79	-6696.89
1	-6698.74	-6697.32	-6695.29
2	-6696.77	-6695.31	-6697.20

Table 3: ARMA AIC

As the table above shows, the smallest AIC value is located at the first row, second column, which corresponds to ARMA(0,1) or equivalently to MA(1). Hence, it is more efficient to fit our data into a moving- average process in order to extract the residuals needed for the GARCH model. However, we decided to restrict the candidate models to those containing both AR and MA coefficients since the difference in AIC is relatively small. The model with the smallest AIC satisfying this restriction is ARMA(1,1). Nevertheless, we run the GARCH process with both residuals from MA(1) and ARMA(1,1) yielding identical results.
After deciding about the length of the model we run the *arima* command from the *stats* package as seen in the line following:

```
arima101 = arima(returns.dm, order = c(1,0,1))
```

This command yields estimates for the coefficients and variance minimising an objective function based on the conditional sum of squares and by calculating the initial values for AR and MA coefficients. As discussed above, applying ARMA(1,1) and ARIMA(1,0,1) would yield the same results. However, in practise this is not the case and comparing both methods one would obtain slightly different results. Applying the *arma* command of the *tseries* package on the same data we obtain slightly different estimates. This happens due to the different fitting approaches of both methods. In contrast to *arima*,

arma minimises only the conditional sum of squares and sets the initial values of AR and MA coefficients to zero. Minor differentiated estimates will also occur if we choose to fit log prices into an ARIMA(1,1,1) and let the model do the differencing since, calculations for initial values will be different and constant is omitted. Nonetheless, those differences are minimal and do not affect the quality of estimates or in our case the residuals. In the lines following we create a string containing the residuals generated from the model and transformed it into *zoo* object in order to be consistent with the rest of the data.

4 ARCH and GARCH Process

4.1 Theory

4.1.1 ARCH/GARCH

Autoregressive conditional heteroscedasticity (ARCH) and generalized autoregressive conditional heteroscedasticity (GARCH) models have become the standard tools to model the volatility of time series. They are especially useful in financial applications, where a time series is not homoscedastic and volatility forecasting plays can improve the accuracy of interval forecasts.

4.1.2 Information Criteria

Information criteria are tools to compare different statistical models on a given data set. The criteria we choose to compare our models, namely the Akaike Information Criterion (AIC), the Bayesian Information Criterion (BIC), and the Hannan-Quinn Information Criterion (HQIC) are frameworks that consider the trade-off between the Log-Likelihood of model and its dimensions (usually Log-Likelihood L can be maximized by very complex models, but the number of parameters k is penalized in all three criteria). Two of those information criteria further take the number of data points n as an argument.
The three information criteria are defined as follows:

$$AIC = -2L + 2k$$
$$BIC = -2L + k \times ln(n)$$
$$HQIC = -2L + 2k \times ln(ln(n))$$

4.2 Implementation

4.2.1 Programming based on Theory

4.2.1.1 Grid Search

Our next goal was to determine the ARCH or GARCH model that best fits the data. To do this, we used a grid search approach for the lags p of the ARCH model or the combination of ARCH terms p and GARCH terms q for the GARCH model.

```
1  ltt.garch = 10
2  hqic.both = matrix(rep(0, ltt.garch ^ 2), nrow = ltt.garch)
3
4  for(i in 1 : ltt.garch) {
5    for(j in 1 : ltt.garch) {
6      hqic.both[i, j] = HQIC(garch(res.arima101,
7      order = c(i, j))$n.likeli,
8                                i + j + 1, n)
9    }
10 }
11
12 hqic.both
13 # row number: GARCH component, col number: ARCH component
14 which(hqic.both == min(hqic.both), arr.ind = TRUE)
15 garch.min.h = min(hqic.both)
16 arch.min.h  > garch.min.h
17 # result:
18 # GARCH(1,1) has lowest HQIC
19 # also lower than ARCH(1,1)
```

In line 1, an integer variable called ltt.garch is defined. This is the number of lags or GARCH parameters that is going to be tested, and it defines the size of the grid. Then, a variable that will in the end contain the information criterion values for each grid point is defined. In the case of ARCH model testing (where only one parameter is to be determined), this is a vector that is defined as `NA`, in the case of GARCH model testing, a matrix that is filled with zeroes. We call this the *goal vector* or *goal matrix*. With these two variables set up, one (or two) simple for-loop(s) search through the grid and evaluate the ARCH or GARCH model with parameters according to the respective grid points. The information criterion value is then written into the goal vector or matrix at the respective place. Since this goal vector/matrix has been defined in the global environment, those values will be available outside of the for loop(s) for later computations. After the grid search is done and the goal vector or matrix is filled, the only thing left to do is find the parameter (or combination of parameters) that performed best with respect to the information criterion that was tested. This is done

in lines 14: The command `which(aic.both == min(aic.both), arr.ind = TRUE)` returns the row and column indices for the best model according to our criterion. This procedure is performed with all three different information criteria (AIC, BIC, and HQIC) as the value to be minimized first for potential ARCH models, and then for potential GARCH models. When these comparisons yield different recommendations (i.e. when the information criteria disagree in their assessment of the best models), a decision is made by human reasoning.

4.2.1.2 ARCH/GARCH models

Both ARCH and GARCH modeling were implemented using the garch function from the package *tseries*.

```
garch.11 = garch(res.arima101, order = c(1, 1))
```

The vital arguments to this function are the time series that is to be fitted, in this case the residuals from our ARMA model, and the order of the ARCH/GARCH model. Note that the order argument is vector of length 2 containing the parameters p and q (in that order). Since an ARCH model is a specialization of a GARCH model with $q = 0$, we can use the garch command `garch(x, order = c(0, q))` to fit any ARCH(q) model.

4.2.1.3 Information Criteria

Each model on those grids was then evaluated using three different information criteria:

1. The Akaike Information Criterion (AIC)
2. The Bayesian Information Criterion (BIC)
3. The Hannan-Quinn Information Criterion (HQIC)

While a function to calculate the AIC for a fitted model object, like an ARCH or GARCH model, is already implemented in R via the *stats* package, we wrote our own simple functions for the BIC and HQIC.

```
{AIC(object, ..., k = 2)}
```

The `AIC` function from the *stats* package in R takes as arguments one or more objects (like a fitted ARCH or GARCH model) and a penalty k for the number of free parameters in the model. The penalty parameter k is set to 2 by default. This parameter can be changed to accommodate the use of the `AIC` function for other information criteria. In this respect, we note that the function calls `AIC(object, ..., k = log(n))` and `AIC(object, ..., k = 2 * log(log(n)))` would calculate the same values as our own functions `BIC` and `HQIC`. However, we rather wrote the functions ourselves to practice function definitions and to avoid any confusion due to the function name `AIC` (instead of using the `AIC` function to calculate, for example, the BIC, we can now use an appropriately named function `BIC`).

```
1  BIC = function(n.like, k, n) {
2    # Calculates and returns the Bayesian Information Criterion
3    #
4    # Args:
5    #  n.like: negative log-likelihood, as for example produced
6    #          by the 'garch' command from the 'tseries' package
7    #  k:      number of free parameters to be estimated
8    #  n:      number of data points
9    2 * n.like + k * log(n)
10 }
```

Our own function `BIC` works slightly different than the `AIC` function in that it does not take the whole object as an argument, but rather the parameters from the model fitting, namely the negative log-likelihood $n.like$, the number of free parameters to be estimated k, and the number of observations n. Then, a simple computation according to the definition of the BIC is performed and returned.

```
1  HQIC = function(n.like, k, n) {
2    # Calculates and returns the Hannan-Quinn IC
3    #
4    # Args:
5    #  n.like: negative log-likelihood, as for example produced
6    #          by the 'garch' command from the 'tseries' package
7    #  k:      number of free parameters to be estimated
```

```
8   #  n:       number of data points
9   2 * n.like + 2 * k * log(log(n))
10 }
```

The function `HQIC` works takes the same parameters as the `BIC` function (the negative log-likelihood $n.like$, the number of free parameters to be estimated k, and the number of observations n) and works in a very similar way. Only the computation of the return value varies, according to the definition of the HIQC.

4.2.2 Empirical Study with ARCH/GARCH model

4.2.2.1 Grid Search

With knowledge from theory and practical experience, we can assume that both parameters p and q to be equal to or smaller than 10. So, in all cases in this project, we defined the lags to test variables with a value of 10. That means, the one-dimensional grid spans over all integers from and including 1 to 10, and the two-dimensional grid over all tuples of integer combinations from and including 1 to 10.

4.2.2.2 ARCH

The grid search resulted in the suggestion of an ARCH(7) model by all three information criteria as the best ARCH model specification. This was the model we fitted to be compared with the best GARCH models.

4.2.2.3 GARCH

For the GARCH modeling part, a GARCH$(1, 2)$ model seemed to be the best model specification according to the AIC. However, both BIC and HQIC suggested a GARCH$(1, 1)$ model as the best alternative. As previously stated, in case of disagreement in the model assessment of different information criteria, a human decision has to be made. Due to our from experience in other projects we knew that GARCH$(1, 1)$ models are the most common GARCH-specifications; in addition to that, with all else equal, a smaller number of parameters is always to be favoured in model selection, so we chose to proceed with a GARCH$(1, 1)$ model. Both the GARCH$(1, 1)$ and the GARCH$(1, 2)$ model outperformed the best ARCH(7) model with respect to all three information criteria values,

so the GARCH$(1,1)$ model was our standard volatility model of choice to be compared with alternative modeling approaches.

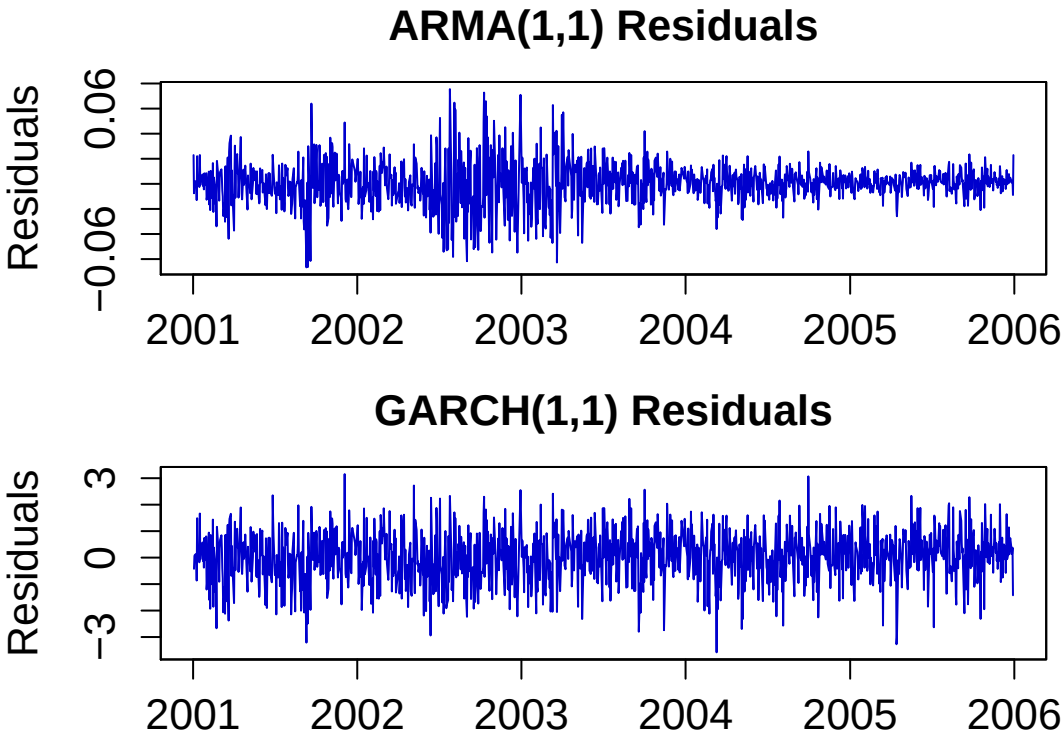

Figure 4: ARMA vs. GARCH Residuals

As can be seen in Figure 4, the GARCH modeling has removed the large volatility clusters that were present in the ARMA residuals. In the following section, we describe alternative ways to do this.

5 Breakpoints Estimation and Homogenous Intervals

After modeling the GARCH process in the previous chapter we also want to use a second approach to compare the interest rate change effects with. For this part we decided to use an algorithm to determine the most likely breakpoints in the series of the ARMA residuals. Our idea is in a first step to compare the resulting breakpoint dates with the dates of the interest rate changes by the European Central Bank to determine if they are near each other. After this first descriptive approach we use the estimated breakpoints and interest rate changes to construct homogenous intervals for each method by standardizing each period. In the aftermath we compare the resulting series and the GARCH series to evaluate if the interest changes are a good predictor for changes in the return volatility and if they are useful to model the residuals.
Before we decide on a specific algorithm we have to choose the kind of breakpoint test we want to use. Given that the returns fluctuate around a constant mean we can dismiss methods for breakpoints in the mean. The two options we consider are either a CUMSUM based test or a direct test of the variance of the residuals. For the CUSUM based test one would cumulate the squared residuals in a series and test for breakpoints in the slope of this series. Alternatively, one can test directly for changes in the variance of the residuals of the ARMA process and estimate the breakpoints. For our project we decided to use the latter method for several reasons. First of all, this approach is the more forward and easier to interpret and also incorporates the direction of the residuals (negative or positive). Also it gives more clear results in the first tries we conducted regarding the consistency of the estimated change points. In specific we choose to implement the PECT algorithm included in the ?breakpoint?-package for R by Rebecca Killick to estimate multiple breakpoints exactly. This algorithm is used to calculate the optimal positioning and number of breakpoints in a given series. In our case based on the variance.

```
1 breakpoints = cpt.var(returns, penalty="MBIC", pen.value=0.05,
2                       know.mean = FALSE, method = "PELT")
```

For the implementation of the algorithm we determine several options for the penalty term, the significance level of the type I error, the treatment of the mean and the type of algorithm used. We tried several combinations of options to get an impression of

the consistency of the estimates, especially for the significance level and the penalty term. Finally, we choose the options as seen in [figure xx]. As information criterion we use MBIC which is a modified Bayesian Information Criterion, although there was not much variance in the results under different information criteria. For the penalty value we stuck to the standard 0.05 level. Setting a higher significance level did not change the number of breakpoints detected in the series. For the mean we decided to not take it as known but used the option "know.mean = FALSE", which let the algorithm estimate the mean by maximum likelihood. As method we take as mentioned above the PECT algorithm, because it can detect multiple change points and gives an exact date of change, which is useful to model the homogeneous intervals later. The results are not exactly the same every time one applies the algorithm though. This is due to the fact that a numeric method is used to determine the breakpoints. But the number of detected points stayed the same in all tries we conducted and the dates only changed marginal. The results can therefore be seen as robust. We estimate a total of six breakpoints (from which one is always the last date of the series and can be neglected). The estimated breakpoints are then modified as a zoo variable vector we can use for the further procedure.

5.1 Homogenous Intervals

In this section we show the construction of the homogenous intervals for the breakpoint intervals as well as the interest rate intervals. Our motivation for the construction of homogenous intervals is to see it as an alternative model for the residuals and evaluate it in comparison with each other and the GARCH model.

5.2 Breakpoint Test Based Homogenous Intervals

For the breakpoint based homogenous intervals we divide the series of ARMA residuals into several intervals according to the breakpoint test described in the previous part. In total we have five breakpoints (the last breakpoint is the last date and can be excluded) resulting in six separate series with different length. Now we take each of these series and standardize them by using the scale function which demeans it and divides it by its own individual standard deviation. In conclusion all standardized intervals have a mean of zero and a variance of one. In the next step we put the standardized intervals back together to get a homogenous series with the same length as the original residuals

series.

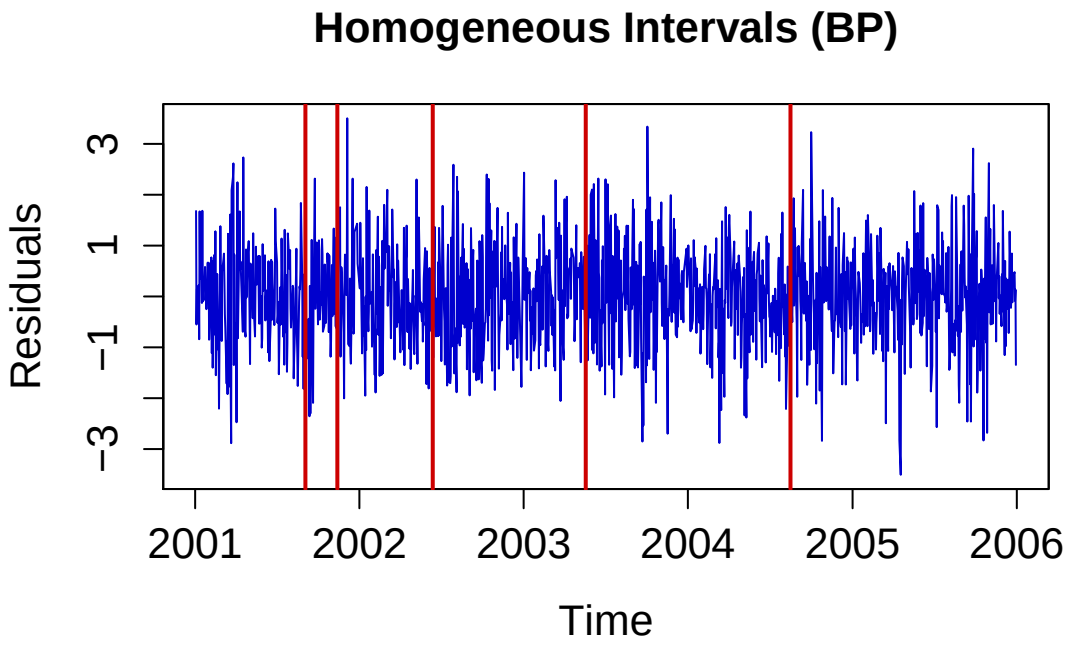

Figure 5: Homogenous Intervals (BP)

The above graph shows the homogenized series based on the breakpoint test. The red lines show the positions of the estimated breakpoints in the series. Compared with the original ARMA residual this looks much more smooth. But there are still parts that show some easy to recognize heterogeneity, especially in the longer intervals. Also one cannot reject the null hypothesis of the Box-Ljung test, indicating autocorrelation in the process. We expected that result somehow, because under the assumption of autocorrelation in the whole series dividing it in only six intervals is not much given the number of observations. Also the GARCH residuals plot looks much more smooth in comparison.

5.3 Interest Rate Change Based Homogenous Intervals

For the interest rate change based homogenous intervals we apply the same procedure as with the breakpoint based version in the chapter before. But instead of the estimated breakpoints we take the interest rate change dates to split the series into several

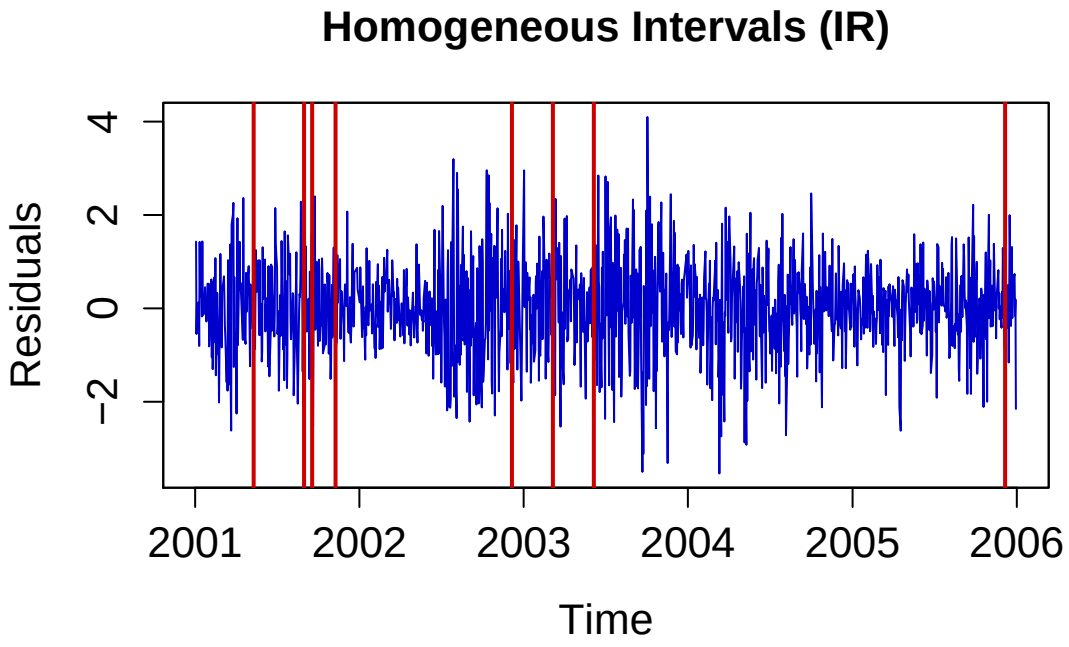

Figure 6: Homogenous Intervals (IR)

intervals. In our reviewed time period we have a total of eight interest changes by the European Central Bank, resulting in nine separate intervals. As before we standardize each interval with the scale function and put the standardized parts back together to get a new "homogenous" series of the residuals.
In the graph above the red lines show the dates of the interest rate changes and thereby represent the edges of the standardized intervals. The length of the intervals differs a lot and so especially the two longest intervals do not seem to have a homogenous variance. Also the Box-Ljung test indicates autocorrelation in the series.

5.4 Results

Comparing the interest rate change dates with the estimated breakpoint dates we can see two dates close to each other. The first is the interest rate change at the 31st of August 2001 which coincides with a breakpoint at 3rd September 2001 (date may vary based on the numeric procedure of the algorithm, but will stay roughly the same). The second is at the 9th November 2001 and a breakpoint at 13th November the same year.

The rest of the dates do not seem to coincidence. From this point of view we can only see a weak over all connection between the interest rate changes and the breakpoints. But based on the two dates in 2001 just mentioned we still can suspect the actions of the ECB can have some effect on the volatility. In the next steps we first compare the interest rate change based series with the estimated GARCH residuals and then with the breakpoint test based homogenous interval series.

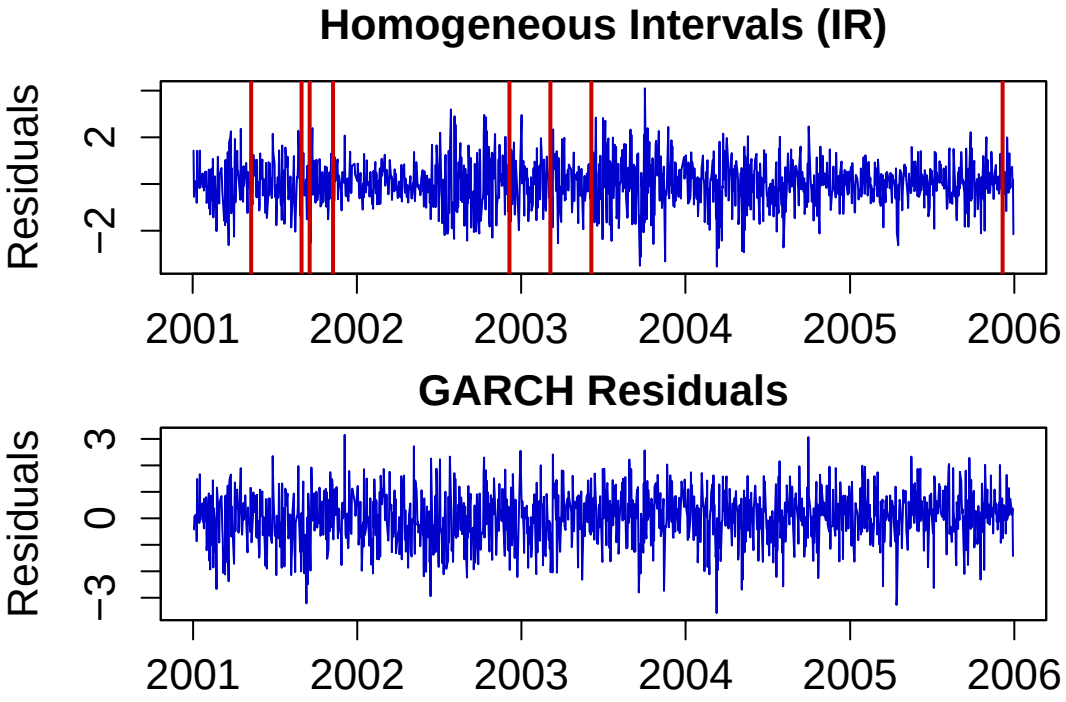

Figure 7: Homogenous Intervals (IR), GARCH res.

Compared with the GARCH model (bottom) the interest rate based series (top) looks much more uneven over time with parts of very low volatility and parts of high volatility, especially in the long intervals. Based on this graphs we can conclude that GARCH model clearly is the better model and that the interest rate changes are not sufficient the model the DAX residuals. But this is also due to the fact that the GARCH model is more complex than the homogenous intervals approach. If we compare the two interval models the graphs look more similar.

Here we see the interest based series in the upper part and the breakpoint based series in the lower part. The two graphs look more similar that the two above, but here also

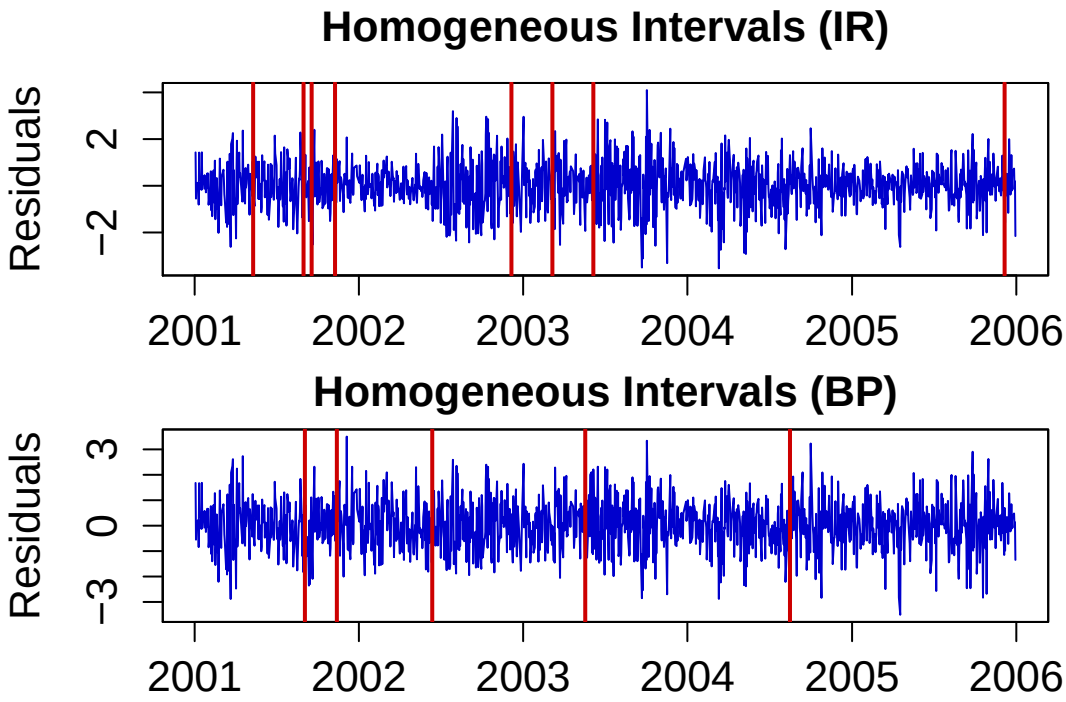

Figure 8: Homogenous Intervals (IR), Homogenous Intervals (BP)

the interest rate based model looks more heterogeneous. The breakpoint based series has a more even interval length which makes the overall series look more smooth. Also it takes care of the obvious breakpoints one can for example see in the middle of 2002 in the upper plot. To summarize our results, we can say that interest changes are no good indicator for changes in the market volatility, at least taken alone. But it still can be seen as one of many factors influencing it. Some interest change points are close to the estimated breakpoints of the algorithm and homogenous intervals look for some parts much more smooth than the ARMA errors.

6 Conclusion

At the end we can conclude that based on our research the interest rates set by the Central Bank can probably be seen as one factor influencing the market volatility. But we can also say that it is not enough to make satisfying predictions for volatility changes. At some time points the breakpoints coincident with interest rate changes, but at other points the interest change did not seem to have a significant impact. Also there are some obvious change points in the volatility independent of changes of the interest rate in the Euro zone. For this reason, it could be interesting to research if other macro impulses caused the change of volatility structure.
For our programming the construction of the GARCH model took the biggest part. Based on the ACF and PACF, as well as the information criteria, the choice of time series model for the returns was not clear-cut and shows several viable options. We decide in this cases to go with the simpler model, which is also recommend by the literature. For the modelling of the error terms the picture is similar. Here we also choose a simpler model. We are able to construct a good fitting model we can use as a benchmark. The homogenous interval method does not fit the data as well, neither for the interest rate based nor for the breakpoint based. But this is also due to the simplicity of the model. In a next step it would be interesting to see if an interval specific GARCH-modeling could improve the fit of our benchmark GARCH model. Also a variation of the parameters would be useful to make a clearer conclusion of the possible effect. For this one could change the market index used, the observed time interval or use the interest rate changes of another currency zone.

7 Appendix: Full Source Code

```
1 ##################################
2 ##  Authors: Alexander Dautel, Thorsten Disser, Binhui Hu,
      Nicolas Yiannakou
3 ##  Date: 24.06.2016
4 ##  Project: SPL project
5 ##  Script: Interest_Rate_Changes_And_Breakpoints_Dax_
      2001-2005.R
```

```
6  ##  Input Data: Dax Data Comma.csv (Daily Dax Data from
       2001-2005)
7  ################################
8
9  # Workspace and Working Directory
       ------------------------------------------------
10
11 rm(list = ls())
12 getwd()
13
14 folder.wd = "~"
15 setwd(folder.wd)
16 getwd()
17
18 # Function Definitions
       ---------------------------------------------------------
19
20 IPak = function(pkg){
21   # Function will install new packages and require already
         installed ones
22   #
23   # Args:
24   #  pkg: A character string containing the packages to be
         used in the
25   #       project.
26   #
27   # Returns:
28   #  None, apart from the function calls for install.packages
         () and/or
29   #  require().
30   new.pkg = pkg[!(pkg %in% installed.packages()[, "Package"])]
31   if (length(new.pkg))
32     install.packages(new.pkg, dependencies = TRUE)
33   sapply(pkg, require, character.only = TRUE)
```

```
34 }
35
36 BIC = function(n.like, k, n) {
37   # Calculates and returns the Bayesian Information Criterion
38   #
39   # Args:
40   #  n.like: negative log-likelihood, as for example produced
41   #          by the 'garch' command from the 'tseries' package
42   #  k:      number of free parameters to be estimated
43   #  n:      number of data points
44   2 * n.like + k * log(n)
45 }
46
47 HQIC = function(n.like, k, n) {
48   # Calculates and returns the Hannan-Quinn Information
       Criterion
49   #
50   # Args:
51   #  n.like: negative log-likelihood, as for example produced
52   #          by the 'garch' command from the 'tseries' package
53   #  k:      number of free parameters to be estimated
54   #  n:      number of data points
55   2 * n.like + 2 * k * log(log(n))
56 }
57
58 # Install and Load Packages
     ---------------------------------------------------
59
60 packages = c("tseries", "nlme", "zoo", "ecp", "bfast",
61              "stockPortfolio", "changepoint", "lint")
62 IPak(packages)
63
64 # Read data
     ---------------------------------------------------
```

```
65
66 SPL       = read.csv("DAX Data Comma.csv")
67 SPL.zoo   = zoo(SPL[, -1],
68                 order.by = as.Date(strptime(as.character(SPL[,
                       1]), "%Y-%m-%d")))
69 prices    = SPL.zoo$Adj.Close
70 n         = length(prices)
71 returns   = diff(log(prices))
72
73 # Descriptives
   ---------------------------------------------------------------------
74
75 mean        = mean(returns)
76 returns.dm = returns - mean
77 summary(returns.dm)
78
79 ##plot of the raw data(begin,add in 13th,August by BH)
80 par(mfrow = c(1, 1), mar = c(4, 2, 1, 1))
81 prices  = SPL.zoo$Adj.Close
82 plot(prices, type = "l", col = "blue3",
83       xlab = "Time", ylab = "Returns", xlim = NULL, lwd = 1)
84 title(main = "Demeaned Price", cex.main = 1)
85 ##plot of the raw data(end)
86
87 par(mfrow = c(2, 1))
88 plot(returns, type = "l")
89 plot(returns.dm, type = "l")
90
91 sq.ret          = returns.dm ^ 2
92 sq.ret.cum      = cumsum(sq.ret)
93 sq.ret.cum.df = diff(sq.ret.cum)
94
95 plot(sq.ret.cum, type = "l")
```

```
96  plot(sq.ret.cum.df, type = "l")
97
98  acf(coredata(returns.dm), lag.max = NULL, plot = TRUE)
99  pacf(coredata(returns.dm), lag.max = NULL, plot = TRUE)
100
101 Box.test(returns.dm ^ 2, lag = 10, type = c("Ljung-Box"),
        fitdf = 0)
102
103 # Breakpoint Testing
        ----------------------------------------------------------
104
105 breakpoints      = cpt.var(returns, penalty = "MBIC", pen.value
        = 0.05,
106                            know.mean = FALSE, method = "PELT")
107 breakpoints.vec = breakpoints@cpts
108 breakpoints.zoo = index(returns[breakpoints.vec])
109
110 par(mfrow = c(1, 1), mar = c(6, 5, 4, 2))
111 plot(returns.dm, type = "l", col = "blue3",
112      xlab = "Time", ylab = "Returns", xlim = NULL, lwd = 1)
113 abline(a = NULL, b = NULL, h = NULL,
114        v = breakpoints.zoo[-6], col = "red3", reg = NULL,
115        coef = NULL, untf = FALSE, lwd = 2)
116 title(main = "Demeaned␣Returns␣and␣Breakpoints", cex.main = 1)
117
118 # ARIMA
        ----------------------------------------------------------
119
120 #AIC for returns#
121
122 ltt.arima = 2
123 aic.arima.matrix = matrix(NA, ncol = ltt.arima + 1, nrow = ltt
        .arima + 1)
```

```
124 colnames(aic.arima.matrix) = 0:2 #order of MA model (q)
125 rownames(aic.arima.matrix) = 0:2 #order of AR model (p)
126
127 for (i in 0 : ltt.arima) {
128   for (j in 0 : ltt.arima) {
129     aic.arima.matrix[i + 1, j + 1] = AIC(arima(returns.dm,
          order = c(i, 0, j)))
130   }
131 }
132
133 aic.arim.min = min(aic.arima.matrix)
134 which(aic.arima.matrix == aic.arim.min, arr.ind = T)
135
136 #--> arima(1,0,1) has the smallest AIC --> Best model for
      returns ARMA(1,1)
137
138 # --> ARMA (1, 1)
139
140 arima101 = arima(returns.dm, order = c(1, 0, 1)) #-6697.83
141
142 Box.test(returns.dm, lag = 5, type = "Ljung-Box", fitdf = 0)
143
144 # Generate squared residuals
145 res.arima101          = arima101$residuals[is.na(arima101$
      residuals) == FALSE]
146 rev.date              = rev(as.Date(strptime(as.character(SPL[,
      1]), "%Y-%m-%d")))
147 res.arima101.zoo      = zoo(res.arima101, order.by = rev.date)
148 sq.res.arima101.zoo = res.arima101.zoo ^ 2
149 plot(res.arima101.zoo, main = "Residuals", type = "l")
150 plot(sq.res.arima101.zoo, main = "Squared Residuals", type = "
      l")
151 # from the plots we can observe some obvious volatility
      clusters
```

```
152
153 acf(coredata(sq.res.arima101.zoo),
154       main = "ACF Squared Residuals",   ylim = c(-0.5, 1))
155 pacf(coredata(sq.res.arima101.zoo),
156       main = "PACF Squared Residuals", ylim = c(-0.5, 1))
157 # residuals clearly not independent
158
159 # ARCH AIC
    ------------------------------------------------------------------------
160
161 # ARCH
162
163 # use AIC to test
164 ltt.arch = 10  # ltt: lags to test (compare models up to this
      lag)
165 aic.arch = NA
166
167 for(i in 1 : ltt.arch) {
168   aic.arch[i] = AIC(garch(res.arima101, order = c(0, i)))
169 }
170
171 arch.min.a = min(aic.arch)
172 aic.arch
173 which(aic.arch == min(aic.arch), arr.ind = TRUE)
174 # result: q = 7 (ARCH 7)
175
176 # ARCH BIC
    ------------------------------------------------------------------------
177
178 # ARCH
179
180 # use bic to test
```

```
ltt.arch = 10  # ltt: lags to test (compare models up to this lag)
bic.arch = NA

for(i in 1 : ltt.arch) {
  bic.arch[i] = BIC(garch(res.arima101, order = c(0, i))$n.likeli, i + 1, n)
}

arch.min = min(bic.arch)
bic.arch
which(bic.arch == min(bic.arch), arr.ind = TRUE)
# result: q = 7 (ARCH 7)
# ARCH HQIC
```

```


# ARCH

# use hqic to test
ltt.arch = 10  # ltt: lags to test (compare models up to this lag)
hqic.arch = NA

for(i in 1 : ltt.arch) {
  hqic.arch[i] = HQIC(garch(res.arima101, order = c(0, i))$n.likeli, i + 1, n)
}

arch.min.h = min(hqic.arch)
hqic.arch
which(hqic.arch == min(hqic.arch), arr.ind = TRUE)
# result: q = 7 (ARCH 7)

```

```
# GARCH AIC
-----------------------------------------------------------------------------


# quick comparison
garch.11 = garch(res.arima101, order = c(1, 1))
AIC(garch.11) < min(aic.arch)
# GARCH(1, 1) better than best model with only ARCH component

ltt.garch = 10  # ltt: lags to test (compare models up to this
    lag)
aic.both = matrix(rep(0, ltt.garch ^ 2), nrow = ltt.garch)

for(p in 1 : ltt.garch) {
  for(q in 1 : ltt.garch) {
    aic.both[p, q] = AIC(garch(res.arima101, order = c(p, q)))
  }
}

aic.both  # row number: GARCH component, col number: ARCH
    component
which(aic.both == min(aic.both), arr.ind = TRUE)
garch.min.a = min(aic.both)
arch.min.a  > garch.min.a
# result: GARCH(1, 2) has lowest AIC
# if ONLY ARCH, then ARCH(8) seems best,
# but when we include a GARCH part, we settle on
# GARCH part = 1 and ARCH part = 2 for the GARCH(1, 2) model
# GARCH BIC
-----------------------------------------------------------------------------


ltt.garch = 10  # ltt: lags to test (compare models up to this
    lag)
```

```
236 bic.both = matrix(rep(0, ltt.garch ^ 2), nrow = ltt.garch)
237
238 for(p in 1 : ltt.garch) {
239   for(q in 1 : ltt.garch) {
240     bic.both[p, q] = BIC(garch(res.arima101, order = c(p, q))$
          n.likeli,
241                          p + q + 1, n)
242   }
243 }
244
245 bic.both  # row number: GARCH component, col number: ARCH
    component
246 which(bic.both == min(bic.both), arr.ind = TRUE)
247 garch.min = min(bic.both)
248 arch.min > garch.min
249 # result: GARCH(1, 1) has lowest bic
250 # if ONLY ARCH, then ARCH(7) seems best,
251 # but when we include a GARCH part,
252 # we settle on GARCH part = 1 and ARCH part = 2 for the GARCH
    (1, 1) model
253 # GARCH HQIC
    ------------------------------------------------------------
254
255 ltt.garch = 10  # ltt: lags to test (compare models up to this
    lag)
256 hqic.both = matrix(rep(0, ltt.garch ^ 2), nrow = ltt.garch)
257
258 for(i in 1 : ltt.garch) {
259   for(j in 1 : ltt.garch) {
260     hqic.both[i, j] = HQIC(garch(res.arima101, order = c(i, j)
          )$n.likeli,
261                            i + j + 1, n)
262   }
```

```
263 }
264
265 hqic.both  # row number: GARCH component, col number: ARCH
      component
266 which(hqic.both == min(hqic.both), arr.ind = TRUE)
267 garch.min.h = min(hqic.both)
268 arch.min.h  > garch.min.h
269 # result: GARCH(1, 1) has lowest hqic
270 # if ONLY ARCH, then ARCH(7) seems best,
271 # but when we include a GARCH part,
272 # we settle on GARCH part = 1 and ARCH part = 1
273 # for the GARCH(1, 1) model
274
275 # Model Decision
      ---------------------------------------------------------------
276
277 # Use a GARCH(1, 1) model.
278
279 garch.11 = garch(res.arima101, order = c(1, 1))
280 # plot(res.arima101, type = "l")
281 plot(res.arima101.zoo, type = "l")
282 # plot(garch.11, which = "3", type = "l")
283 print(garch.11)
284 garch.11.res         = residuals(garch.11)
285 garch.11.res.zoo     = zoo(garch.11.res, order.by = rev.date)
286
287 Box.test(garch.11.res ^ 2, lag = 10, type = c("Ljung-Box"),
      fitdf = 0)
288
289 # for comparison: ARMA vs. GARCH residuals
290
291 # ARMA residuals
292 par(mfrow = c(1, 1), mar = c(6, 5, 4, 2))
293 plot(res.arima101.zoo, type = "l", col = "blue3",
```

```
294       xlab = "Time", ylab = "Residuals", lwd = 1)
295 title(main = "ARMA(1,1) Residuals", cex.main = 1)
296
297 # GARCH residuals
298 par(mfrow = c(1, 1), mar = c(6, 5, 4, 2))
299 plot(garch.11.res.zoo, type = "l", col = "blue3",
300       xlab = "Time", ylab = "Residuals", lwd = 1)
301 title(main = "GARCH(1,1) Residuals", cex.main = 1)
302
303 # both in one plot
304 par(mfrow = c(2, 1), mar = c(2, 4, 3, 3))
305 plot(res.arima101.zoo, type = "l", col = "blue3",
306       xlab = "Time", ylab = "Residuals", lwd = 1)
307 title(main = "ARMA(1,1) Residuals", cex.main = 1)
308 plot(garch.11.res.zoo, type = "l", col = "blue3",
309       xlab = "Time", ylab = "Residuals", lwd = 1)
310 title(main = "GARCH(1,1) Residuals", cex.main = 1)
311
312 # Get interest rate changes
    ------------------------------------------------------------
313
314 int.rate       = read.csv("int_rate.csv", header = T, sep = ","
    )
315 int.rate.zoo   = zoo(int.rate[, -1],
316                      order.by = as.Date(strptime(as.character(
                           int.rate[, 1]),
317                                                  "%Y-%m-%d")))
318 int.rate.diff         = diff(int.rate.zoo)
319 int.rate.changes.zoo = index(int.rate.diff)
320 int.rate.changes.vec = which(index(returns) %in% int.rate.
    changes.zoo)
321 print(int.rate.changes.zoo)
322
```

```
323 # Merge DAX and ECB data
    ------------------------------------------------------------
324
325 dax.int = merge(returns.dm, sq.ret, sq.ret.cum,
326                 sq.ret.cum.df, int.rate.diff, fill = 0)
327 head(dax.int)
328
329 data.class(dax.int)
330
331 # dax.int: zoo object with time index and 5 vectors for
        returns.dm, sq.ret,
332 # sq.ret.cum, sq.ret.cum.df, int.rate.diff
333
334 # ALT BP/IR
    ------------------------------------------------------------
335
336 # need
337 # list where returns divided by breakpoints
338
339 split(returns, f = breakpoints.zoo)
340 aggregate(returns, by = vec)
341
342 bp.plus1 = c(1, breakpoints.vec[-6])
343 Inter = as.list(rep(NA, 6))
344
345 for(i in bp.plus1) {
346   Inter[i] = scale(returns[bp.plus1[i] : breakpoints.vec[i]])
347 }
348
349 a = 1:100
350 b = c(rep(1, 10), rep(2, 50), rep(3, 40))
351 split(a, f = b)
352
```

```
353 # Breakpoint Part
    ------------------------------------------------------------------
354
355 newdf1 = returns[1 : breakpoints.vec[1]]
356 plot(newdf1, type = "l")
357 var(newdf1)
358 mean(newdf1)
359
360 Inter1 = scale(newdf1)
361 plot(Inter1, type = "l")
362
363 newdf2 = returns[(breakpoints.vec[1] + 1) : breakpoints.vec
        [2]]
364 plot(newdf2, type = "l")
365 var(newdf2)
366
367 Inter2 = scale(newdf2)
368 plot(Inter2, type = "l")
369
370 newdf3 = returns[(breakpoints.vec[2] + 1) : breakpoints.vec
        [3]]
371 plot(newdf3, type = "l")
372 var(newdf3)
373
374 Inter3 = scale(newdf3)
375 plot(Inter3, type = "l")
376
377 newdf4 = returns[(breakpoints.vec[3] + 1) : breakpoints.vec
        [4]]
378 plot(newdf4, type = "l")
379 var(newdf4)
380
381 Inter4 = scale(newdf4)
382 plot(Inter4, type = "l")
```

```
383
384 newdf5 = returns[(breakpoints.vec[4] + 1) : breakpoints.vec
    [5]]
385 plot(newdf5, type = "l")
386 var(newdf5)
387
388 Inter5 = scale(newdf5)
389 plot(Inter5, type = "l")
390
391 newdf6 = returns[(breakpoints.vec[5] + 1) : breakpoints.vec
    [6]]
392 plot(newdf6, type = "l")
393 var(newdf6)
394
395 Inter6 = scale(newdf6)
396 plot(Inter6, type = "l")
397
398 BPInter = c(Inter1, Inter2, Inter3, Inter4, Inter5, Inter6)
399 par(mfrow = c(1, 1), mar = c(6, 5, 4, 2))
400 plot(BPInter, type = "l", col = "blue3",
401       xlab = "Time", ylab = "Residuals", lwd = 1)
402 abline(a = NULL, b = NULL, h = NULL,
403         v = breakpoints.zoo[-6], col = "red3", reg = NULL,
404         coef = NULL, untf = FALSE, lwd = 2)
405 title(main = "Homogeneous Intervals (BP)", cex.main = 1)
406
407 Box.test(BPInter ^ 2, lag = 10, type = c("Ljung-Box"), fitdf =
    0)
408
409 # IR Change rate
    ------------------------------------------------------------
410
411
412 ir1 = returns[1 : int.rate.changes.vec[1]]
```

```
413 plot(ir1, type = "l")
414 var(ir1)
415
416 irs1 = scale(ir1)
417 plot(irs1, type = "l")
418
419 ir2 = returns[(int.rate.changes.vec[1] + 1) : int.rate.
      changes.vec[2]]
420 plot(ir2, type = "l")
421 var(ir2)
422
423 irs2 = scale(ir2)
424 plot(irs2, type = "l")
425
426 ir3 = returns[(int.rate.changes.vec[2] + 1) : int.rate.
      changes.vec[3]]
427 plot(ir3, type = "l")
428 var(ir3)
429
430 irs3 = scale(ir3)
431 plot(irs3, type = "l")
432
433 ir4 = returns[(int.rate.changes.vec[3] + 1) : int.rate.
      changes.vec[4]]
434 plot(ir4, type = "l")
435 var(ir4)
436
437 irs4 = scale(ir4)
438 plot(irs4, type = "l")
439
440 ir5 = returns[(int.rate.changes.vec[4] + 1) : int.rate.
      changes.vec[5]]
441 plot(ir5, type = "l")
442 var(ir5)
```

```
443
444 irs5 = scale(ir5)
445 plot(irs5, type = "l")
446
447 ir6 = returns[(int.rate.changes.vec[5] + 1) : int.rate.
      changes.vec[6]]
448 plot(ir6, type = "l")
449 var(ir6)
450
451 irs6 = scale(ir6)
452 plot(irs6, type = "l")
453
454 ir7 = returns[(int.rate.changes.vec[6] + 1) : int.rate.
      changes.vec[7]]
455 plot(ir7, type = "l")
456 var(ir7)
457
458 irs7 = scale(ir7)
459 plot(irs7, type = "l")
460
461 ir8 = returns[(int.rate.changes.vec[7] + 1) : int.rate.
      changes.vec[8]]
462 plot(ir8, type = "l")
463 var(ir8)
464
465 irs8 = scale(ir8)
466 plot(irs8, type = "l")
467
468 ir9 = returns[(int.rate.changes.vec[8] + 1) : 1271]
469 plot(ir9, type = "l")
470 var(ir9)
471
472 irs9 = scale(ir9)
473 plot(irs9, type = "l")
```

```
474
475 IRInter = c(irs1, irs2, irs3, irs4, irs5, irs6, irs7, irs8,
      irs9)
476
477 par(mfrow = c(1, 1), mar = c(6, 5, 4, 2))
478 plot(IRInter, type = "l", col = "blue3",
479       xlab = "Time", ylab = "Residuals", lwd = 1)
480 abline(a = NULL, b = NULL, h = NULL,
481         v = int.rate.changes.zoo , col = "red3", reg = NULL,
482         coef = NULL, untf = FALSE, lwd = 2)
483 title(main = "Homogeneous Intervals (IR)", cex.main = 1)
484
485 Box.test(IRInter ^ 2, lag = 10, type = c("Ljung-Box"), fitdf =
      0)
486
487
488 # Plots
    ---------------------------------------------------------------------
489
490
491 par(mfrow = c(1, 1), mar = c(6, 5, 4, 2))
492 plot(returns.dm, type = "l", col = "blue3",
493       xlab = "Time", ylab = "Returns", xlim = NULL, lwd = 1)
494 title(main = "Demeaned Returns", cex.main = 1)
495
496 par(mfrow = c(1, 1), mar = c(6, 5, 4, 2))
497 plot(IRInter, type = "l", col = "blue3",
498       xlab = "Time", ylab = "Residuals", lwd = 1)
499 abline(a = NULL, b = NULL, h = NULL,
500         v = int.rate.changes.zoo , col = "red3", reg = NULL,
501         coef = NULL, untf = FALSE, lwd = 2)
502 title(main = "Homogeneous Intervals (IR)", cex.main = 1)
503
```

```
504 par(mfrow = c(1, 1), mar = c(6, 5, 4, 2))
505 plot(BPInter, type = "l", col = "blue3",
506      xlab = "Time", ylab = "Residuals", lwd = 1)
507 abline(a = NULL, b = NULL, h = NULL,
508        v = breakpoints.zoo[-6], col = "red3", reg = NULL,
509        coef = NULL, untf = FALSE, lwd = 2)
510 title(main = "Homogeneous Intervals (BP)", cex.main = 1)
511
512 par(mfrow = c(1, 1), mar = c(6, 5, 4, 2))
513 plot(garch.11.res.zoo, type = "l", col = "blue3",
514      xlab = "Time", ylab = "Residuals", lwd = 1)
515 title(main = "GARCH Residuals", cex.main = 1)
516
517
518 par(mfrow = c(2, 1), mar = c(2, 4, 3, 3))
519 plot(IRInter, type = "l", col = "blue3",
520      xlab = "Time", ylab = "Residuals", lwd = 1)
521 abline(a = NULL, b = NULL, h = NULL,
522        v = int.rate.changes.zoo, col = "red3", reg = NULL,
523        coef = NULL, untf = FALSE, lwd = 2)
524 title(main = "Homogeneous Intervals (IR)", cex.main = 1)
525
526
527 par(mar = c(3, 4, 2, 3))
528 plot(BPInter, type = "l", col = "blue3",
529      xlab = "Time", ylab = "Residuals", lwd = 1)
530 abline(a = NULL, b = NULL, h = NULL,
531        v = breakpoints.zoo[-6], col = "red3", reg = NULL,
532        coef = NULL, untf = FALSE, lwd = 2)
533 title(main = "Homogeneous Intervals (BP)", cex.main = 1)
534
535 par(mfrow = c(2, 1), mar = c(2, 4, 3, 3))
536 plot(IRInter, type = "l", col = "blue3",
537      xlab = "Time", ylab = "Residuals", lwd = 1)
```

```
538 abline(a = NULL, b = NULL, h = NULL,
539         v = int.rate.changes.zoo, col = "red3", reg = NULL,
540         coef = NULL, untf = FALSE, lwd = 2)
541 title(main = "Homogeneous Intervals (IR)", cex.main = 1)
542
543 par(mar = c(3, 4, 2, 3))
544 plot(garch.11.res.zoo, type = "l", col = "blue3",
545       xlab = "Time", ylab = "Residuals", lwd = 1)
546 title(main = "GARCH Residuals", cex.main = 1)
```