ECON90033 – QUANTITATIVE ANALYSIS OF FINANCE I

TUTORIAL 8

Download the t8e2 Excel data file from the subject website and save it to your computer or USB flash drive. Read this handout and complete the tutorial exercises before your tutorial class so that you can ask for help during the tutorial if necessary.

ARCH and *GARCH* Processes

As you learnt in the week 6 lectures, an autoregressive conditional heteroskedasticity process of order q, *ARCH*(q), for variable Y is defined by the conditional mean of y_t, the conditional distribution of the error term ε_t, and the conditional variance of y_t, where the condition is the available information in time t, Ω_t. set Namely,

Conditional mean: $y_t = \mu_t + \varepsilon_t$

Conditional distribution: $\varepsilon_t : idN(0, h_t)$, i.e., independently and normally distributed with zero mean and h_t variance.[1]

Conditional variance: $h_t = \alpha_0 + \sum_{i=1}^{q} \alpha_i \varepsilon_{t-i}^2 \ , \ \alpha_i \underset{i=0}{\overset{q}{\geq}} 0 \ , \ \sum_{i=1}^{q} \alpha_i < 1$

Introducing $\eta_t \equiv \varepsilon_t^2 - h_t$ and manipulating the conditional variance equation, we obtain $\varepsilon_t^2 = \alpha_0 + \sum_{i=1}^{q} \alpha_i \varepsilon_{t-i}^2 + \eta_t$, which is an *AR*($q$) process in ε_t^2.

A generalization of this to an *ARMA*(q,p) process in ε_t^2, i.e., $\varepsilon_t^2 = \alpha_0 + \sum_{i=1}^{q} \alpha_i \varepsilon_{t-i}^2 + \sum_{j=1}^{p} \beta_j h_{t-j} + \eta_t$, leads to the generalized autoregressive conditional heteroskedasticity process of orders p and q, *GARCH*(p,q). In this case the equations for the conditional mean and distribution are the same as above, while the third equation becomes:

Conditional variance: $h_t = \alpha_0 + \sum_{i=1}^{q} \alpha_i \varepsilon_{t-i}^2 + \sum_{j=1}^{p} \beta_j h_{t-j} \ , \ \alpha_i \underset{i=0}{\overset{q}{\geq}} 0 \, , \beta_j \underset{j=1}{\overset{p}{\geq}} 0 \, , \sum_{i=1}^{q} \alpha_i + \sum_{j=1}^{p} \beta_j < 1$

[1] Note that this is the conditional distribution of ε_t, conditional on Ω_t. Its unconditional distribution is supposed to be *idN*(0, σ^2).

These processes can be illustrated with computer simulation, i.e., with artificial data generated from some *ARCH* and *GARCH* processes whose parameters are known. In order to facilitate simulation, it is useful to rewrite the second equation for the conditional distribution as

$$\varepsilon_t = \nu_t \sqrt{h_t}$$

where $\{\nu_t\}$ is a sequence of independently distributed standard normal random variables.[2]

Exercise 1

The purpose of this exercise is to simulate *ARCH* and *GARCH* processes, just like in Ex 2 and Ex 3 of the week 6 lectures.

Launch *RStudio*, create a new *RStudio* project and script, and name both *t8e1*.

a) Draw a random sample of 200 from the standard normal distribution using the *rnorm() R* function. Call this series *nu* (Greek letter ν).

Random numbers in computer simulations are actually pseudorandom numbers[3] generated with some algorithm. R has several functions for this purpose, one of them is

rnorm(n = n0, mean = mean0, sd = sd0)

It returns a sequence of n0 normally distributed random numbers with mean0 expected value and sd0 standard deviation. This sequence is not truly random, it is completely determined by an initial value and the algorithm of rnorm(), but for the sake of statistical tests it can be treated as random.

The initial value is called seed. If we do not specify it, R uses the clock of the system to establish one. In simulation exercises, however, it is important to ensure that the results are reproducible, so we set the seed ourselves with the

set.seed(k0)

function of R, where k0 is an arbitrary integer.

To make sure that we generate the same set of pseudorandom numbers, let this integer be 654321.

[2] This specification is equivalent to the original one. To see this, just recall the basic properties of the expected value and the variance of a random variable, namely that for a random variable X and constant k, $E(kX) = kE(X)$ and $Var(kX) = k^2 Var(X)$. Moreover, kX has the same type of probability distribution, e.g. normal, as X.

[3] A pseudorandom sequence of numbers is generated by a completely deterministic and repeatable algorithm. Yet, it appears unpredictable and satisfies standard tests for statistical randomness.

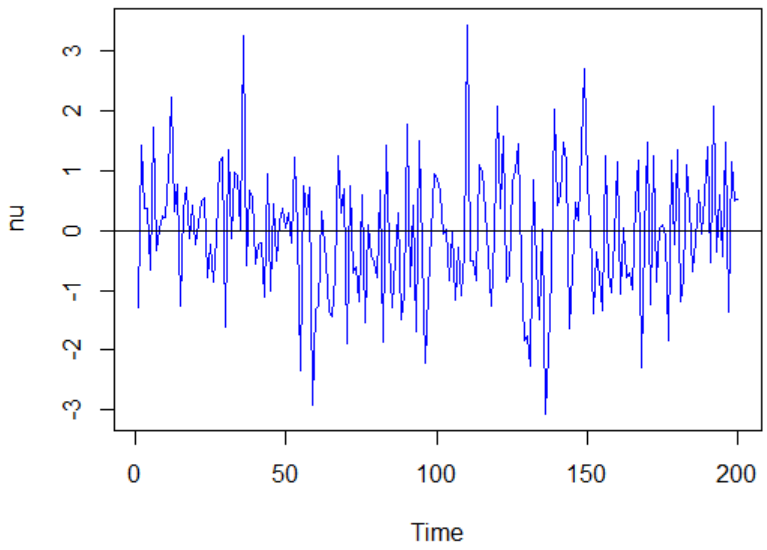
Execute the following *R* commands:

```
set.seed(654321)
nu = ts(rnorm(200, mean = 0, sd = 1), start = 1)
```

Note that to make this exercise more realistic, we defined the *nu* sequence as a time series object.[4] Plot it to see that in indeed it looks random.

```
plot.ts(nu, col = "blue")
abline(h=0)
```

return the following time series plot:



b) Using *nu*, simulate an *ARCH*(1) error series for *t* = 1,…, 200 as

$$\varepsilon_t = \nu_t \sqrt{1 + 0.5\varepsilon_{t-1}^2} \quad , \quad \nu_t \equiv nu \, , \, \varepsilon_t \equiv eps$$

assuming that $\varepsilon_1 = 0$, so $\varepsilon_2 = \nu_2$.

This task can be done in three steps. First, we create the *eps* time series object for *t* = 1, …, 200:

```
eps = ts(start = 1, end = 200)
```

At this stage *eps* is an 'empty' time series, i.e., each of its elements is missing, denoted as *NA* in *R*. You can verify this by executing

[4] This is not necessary, we could treat the sequence as an atomic vector.

print(eps)

Next, we set the first *eps* value equal to zero,

eps[1] = 0

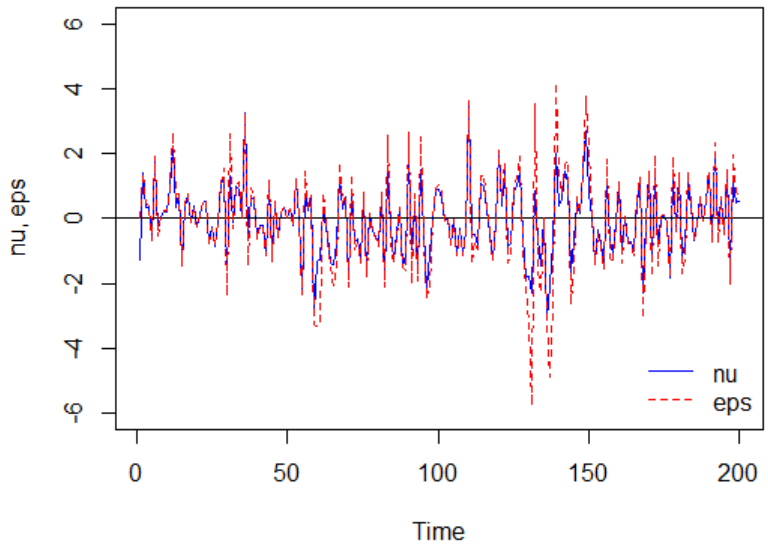
Finally, we use a loop and calculate *eps* iteratively for *t* = 2,…, 200:

```
for (t in 2:200)
  {eps[t] = ts(nu[t] * sqrt(1 + 0.5*eps[t-1]^2))}
```

The first part of this command, *for (t in 2:200)*, specifies that *t* is increased by 1 from 2 to 200, and in each step or iteration the expression within the *{ }* brackets in the second line of the command is evaluated.

Plot $\{\nu_t\}$ and $\{\varepsilon_t\}$ together.

```
plot.ts(nu, col = "blue", ylab = "nu, eps", ylim = c(-6, 6), lty = 1)
abline(h=0)
lines(eps, col = "red", lty = 2)
legend("bottomright", bty = "n", legend = c("nu", "eps"), col = c("blue", "red"), lty = 1:2)
```



As you can see, $\{\nu_t\}$ and $\{\varepsilon_t\}$ seem to fluctuate around zero and $\{\varepsilon_t\}$ mimics $\{\nu_t\}$, but with more pronounced deviations from zero.

c) Using *eps*, simulate the following *AR*(1) process

$$y_t = 0.9y_{t-1} + \varepsilon_t$$

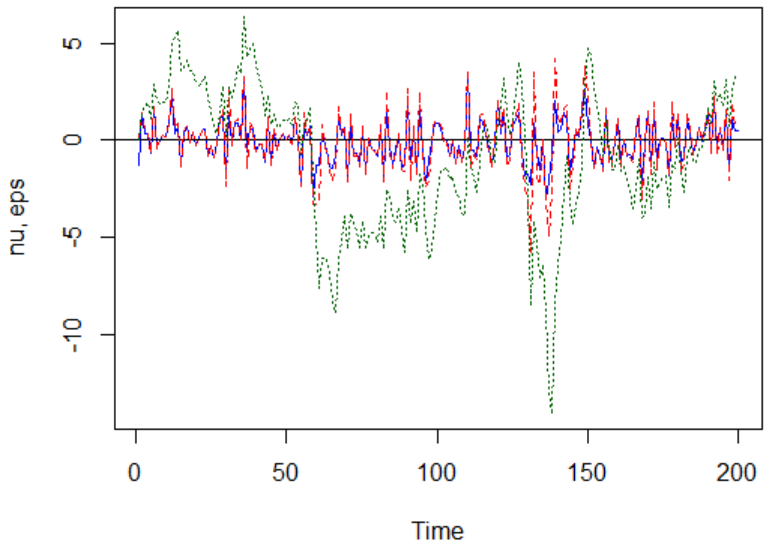
with zero initial value.

We can simulate $\{y_t\}$ in three steps, similarly to $\{\varepsilon_t\}$:

```
y = ts(start = 1, end = 200)
y[1] = 0
for (t in 2:200)
  {y[t] = 0.9*y[t-1] + eps[t]}
```

Add the $\{y_t\}$ series to the previous time series plot. What do you observe?

```
plot.ts(nu, col = "blue", ylab = "nu, eps", ylim = c(-14, 6), lty = 1)
abline(h=0)
lines(eps, col = "red", lty = 2)
lines(y, col = "darkgreen", lty = 3)
legend("bottomright", bty = "n", legend = c("nu", "eps", "y"),
       col = c("blue", "red", "darkgreen"), lty = c(1,2,3))
```



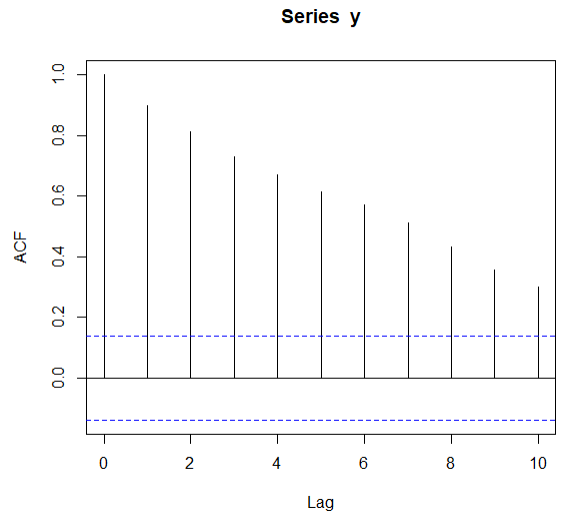
Reviewing the formulas for ε_t and y_t in parts (b) and (c), you can see that the data generating process for Y is an $AR(1)$-$ARCH(1)$ process, i.e., it has an $AR(1)$ conditional mean and an $ARCH(1)$ error. Accordingly, as the plot above illustrates, each unusually large (in absolute value) shock in $\{\nu_t\}$ is associated with a persistently large variation in ε_t and y_t, and $\{y_t\}$ tends to remain away from its unconditional mean, zero.

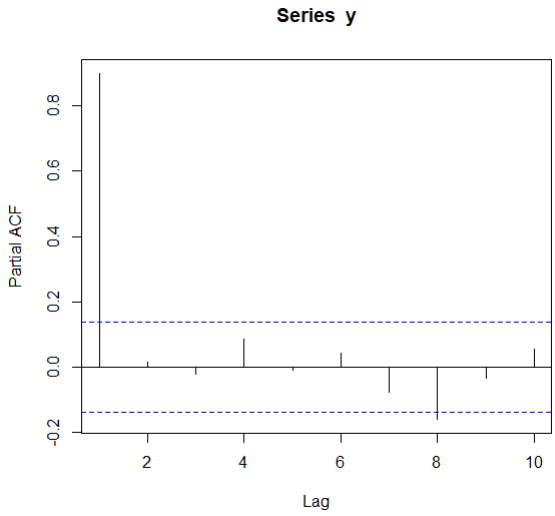
d) Suppose now that we do not know how $\{y_t\}$ has been generated but treat it as some observed data and develop its sample autocorrelation and partial autocorrelation functions.

```
ka = min(10, length(y)/5)
```

acf(y, lag.max = ka, plot = TRUE)
pacf(y, lag.max = ka, plot = TRUE)

return the following correlograms:





The first correlogram illustrates the *SACF*. The sample autocorrelation coefficients decline gradually and, at least up until lag 6, exponentially. The second correlogram illustrates the *SPACF*. The first sample partial autocorrelation coefficient is significant. Apart from the first, only eights 8th sample partial autocorrelation coefficient appears significant (at the 5% level), but it is probably just a statistical fluke, a type I error.

The patterns of these correlograms suggest that $\{y_t\}$ can be modelled as an *AR*(1) process.

e) Fit an *AR*(1) model to $\{y_t\}$ and check the correlograms of the residuals for lags 1,…., 10.

The

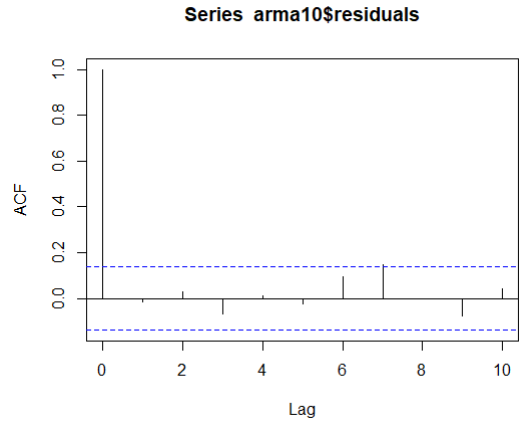
library(forecast)
arma10 = Arima(y, c(1,0,0))
summary(arma10)
library(lmtest)
coeftest(arma10, df = arma10$nobs - length(arma10$coef))

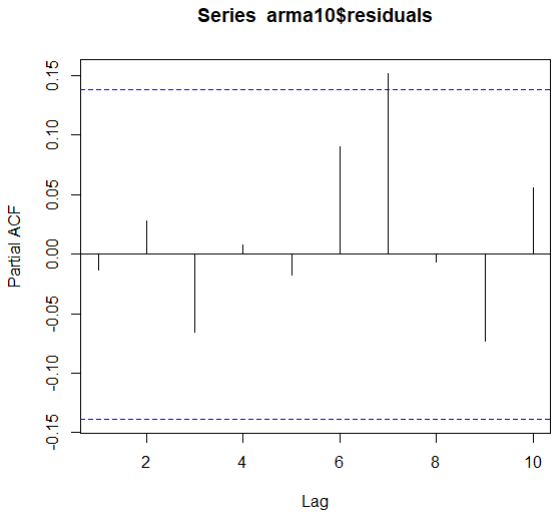
commands return the following printouts:

```
Series: y
ARIMA(1,0,0) with non-zero mean

Coefficients:
         ar1      mean
      0.9021   -0.7194
s.e.  0.0298    1.0630

sigma^2 = 2.378:  log likelihood = -370.24
AIC=746.49   AICc=746.61   BIC=756.38


t test of coefficients:

           Estimate Std. Error t value Pr(>|t|)
ar1        0.902125   0.029761 30.3125   <2e-16 ***
intercept -0.719385   1.063041 -0.6767   0.4994
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The intercept (mean) is insignificant, but the slope (*ar1* coefficient) is certainly significant.

acf(arma10$residuals, lag.max = ka, plot = TRUE)
pacf(arma10$residuals, lag.max = ka, plot = TRUE)





These correlograms do not have any pattern and only the coefficients at lag 7 appear significant.

k = length(arma10$coef)
Box.test(arma10$residuals, type = "Ljung-Box", lag = 10, fitdf = k)

```
        Box-Ljung test

data:  arma10$residuals
X-squared = 9.0429, df = 8, p-value = 0.3387
```

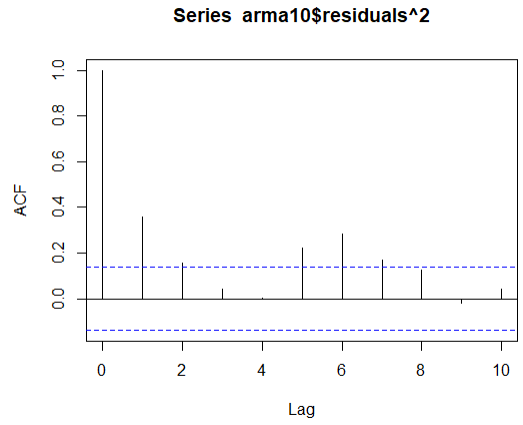
The *LB* test maintains the null hypothesis of no autocorrelation of orders 1,…., 10.

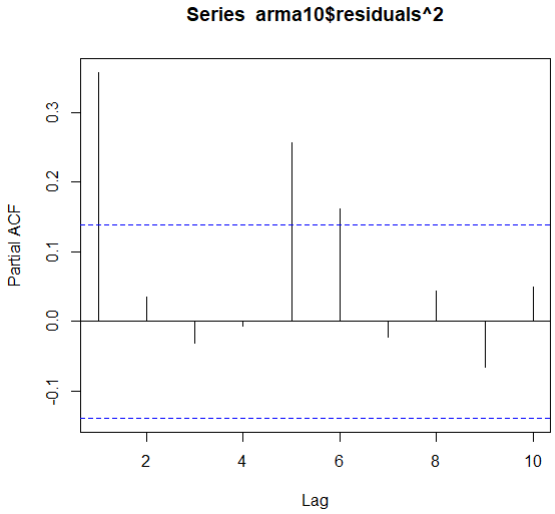
All in all, the residuals behave as a white noise, so we can tentatively accept the *AR*(1) model for the conditional mean.

Next, we need to study the squared residuals to see whether ε_t might be conditionally heteroskedastic, in particular, whether the conditional variance of ε_t might be generated by an *AR* or *ARMA* process. If yes, this should show up in the squared residuals, e_t^2.

f) Check the squared residuals for autocorrelation of orders 1,…., 10. What can you say about the possibility of *ARCH* errors?

acf(arma10$residuals^2, lag.max = 10, plot = TRUE)
pacf(arma10$residuals^2, lag.max = 10, plot = TRUE)





Box.test(arma10$residuals^2, type = "Ljung-Box", lag = 10, fitdf = k)

```
        Box-Ljung test

data:  arma10$residuals^2
X-squared = 67.846, df = 7, p-value = 4.021e-12
```

This time there are several significant spikes on the correlograms, and the *LB* test rejects the null hypothesis of no autocorrelation of orders 1,…., 10. so an *ARCH* or *GARCH* model seems to be warranted.

g) Perform an *ARCH LM* test with lags 1, 5 and 10. What do these tests suggest to you?

These tests can be performed with the *ArchTest()* function of the *FinTS* package. If you do not have this package on your computer, install it and then execute the following commands:

```
library(FinTS)
ArchTest(arma10$residuals, lags = 1)
ArchTest(arma10$residuals, lags = 5)
ArchTest(arma10$residuals, lags = 10)
```

You should get the following printouts

```
        ARCH LM-test; Null hypothesis: no ARCH effects

data:  arma10$residuals
Chi-squared = 25.43, df = 1, p-value = 4.588e-07

        ARCH LM-test; Null hypothesis: no ARCH effects

data:  arma10$residuals
Chi-squared = 36.398, df = 5, p-value = 7.908e-07

        ARCH LM-test; Null hypothesis: no ARCH effects

data:  arma10$residuals
Chi-squared = 40.984, df = 10, p-value = 1.136e-05
```

The *p*-values are practically zero, so we reject the null hypothesis of no *ARCH* effect of order 1, orders 1-5 and orders 1-10.[5]

h) The *ARCH LM* tests suggest that an *ARCH* or *GARCH* model might be appropriate to model $\{y_t\}$. They do not provide real guidance though for model specification. Since in general it is a good idea to start with some relatively simple specification, estimate again an *AR*(1) model of $\{y_t\}$, but assume this time that the errors are generated by an *ARCH*(1) process.

[5] We performed the *LM* test three times to see how robust it is to the lag length. This kind of robustness check is very useful in practice. Keep in mind though, that portmanteau tests like this *LM* test might have very small power when the composite null hypothesis is the combination of a large number of simple null hypothesis, so it is better to keep the lag length relatively (compared to the sample size) low.

Like almost every task, the estimation of a (G)ARCH model can be done in several different ways. We are going to rely on the rugarch R package, in particular on the two essential functions of this package, ugarchspec() and ugarchfit().

Function ugarchspec() is used to specify the model:

```
spec0 = ugarchspec(mean.model = list(armaOrder = c(p10, q10),
                   include.mean = TRUE/FALSE),
                   variance.model = list(model = "sGARCH",
                                         garchOrder = c(p20, q20)),
                   distribution.model = "norm")
```

where include.mean refers to the intercept in the mean equation and distribution.model to the conditional distribution of the error term in the mean equation, which can be normal ("norm"), student-t ("std") etc., and "sGARCH" to the simple (i.e., original) GARCH model.

Function ugarchfit() estimates the specified model:

```
estimate0 = ugarchfit(spec = spec0, data = data0, solver = "solnp")
```

where solnp is the default numerical algorithm used to maximize the log-likelihood function.[6]

Returning to the exercise, recall that in part (e) the intercept (mean) of the *AR*(1) model turned out to be insignificant. Yet, for the sake of comparison, it is better to keep it, so in the *ugarchspec()* function we use *include.mean = TRUE*. As for the conditional distribution of $\{\varepsilon_t\}$, assume it is normal.

Execute the following commands:

```
library(rugarch)
spec1 = ugarchspec(mean.model = list(armaOrder = c(1,0), include.mean = TRUE),
                   variance.model = list(model = "sGARCH", garchOrder = c(1,0)),
                   distribution.model = "norm")
estimate1 = ugarchfit(spec = spec1, data = y, solver = "solnp")
estimate1
```

The printout is fairly long printout, so it is best considered bit-by-bit. It starts with the estimated model:

[6] There are other solver options as well. If *solnp* does not converge, it is worth to try the *hybrid* strategy solver which first tries *solnp* and if it fails, it systematically experiments with three other solvers.

```
*----------------------------------*
*          GARCH Model Fit         *
*----------------------------------*

Conditional Variance Dynamics
-----------------------------------
GARCH Model      : sGARCH(1,0)
Mean Model       : ARFIMA(1,0,0)
Distribution     : norm

Optimal Parameters
------------------------------------
        Estimate  Std. Error  t value Pr(>|t|)
mu     -0.002384    0.827721 -0.00288 0.997702
ar1     0.901207    0.026301 34.26511 0.000000
omega   1.439309    0.212409  6.77611 0.000000
alpha1  0.387720    0.122662  3.16089 0.001573

Robust Standard Errors:
        Estimate  Std. Error   t value Pr(>|t|)
mu     -0.002384    0.709079 -0.003362 0.997318
ar1     0.901207    0.026128 34.492228 0.000000
omega   1.439309    0.216703  6.641858 0.000000
alpha1  0.387720    0.110650  3.504019 0.000458

LogLikelihood : -357.746
```

On this printout *ARFIMA* stands for fractionally integrated *ARIMA* model, which is a generalized version of *ARIMA*(*p*,*d*,*q*) models for fractional *d* values. We do not discuss the details because we'll always use integer values only, mainly 0 and 1. The other probably unfamiliar term, *omega*, stands for the intercept in the variance equation (i.e. it is α_0 in the relevant formulas of the lecture notes).

R displays two sets of results corresponding to two different estimates of the standard errors. The first are regular standard errors based on the Maximum Likelihood (*ML*) and normal distribution, while the second are based on the Quasi Maximum Likelihood (*QML*) and they are robust against violations of the normality assumption.

In this case, qualitatively there is no difference between the results based on the regular and on the robust standard errors. Either way, in the mean equation the intercept (*mu*) is insignificant but the slope of the *ar1* term is significant, just like in part (e), and in the variance equation both the intercept (*omega*) and the slope of the *ARCH*(1) term (*alpha1*) are significant.

Next, there are four information criteria on the printout: the already familiar *AIC* (*Akaike*) and *BIC* (*Bayes*), and two new ones, the *Shibata* and *Hannan-Quinn* criteria. We do not discuss the details of the latter two criteria, but remember that the same decision rule

applies to all four, namely, the smaller the better.[7]

```
Information Criteria
---------------------

Akaike       3.6175
Bayes        3.6834
Shibata      3.6167
Hannan-Quinn 3.6442
```

As usual, the actual values of these criteria are irrelevant, they are only informative in comparison to other values computed for the same (dependent) variable and from the same sample, so we can ignore them at this stage.

The third part of the printout shows the results of weighted Ljung-Box tests on the standardized residuals and on the standardized squared residuals.[8]

```
Weighted Ljung-Box Test on Standardized Residuals
------------------------------------
                        statistic p-value
Lag[1]                    0.08282  0.7735
Lag[2*(p+q)+(p+q)-1][2]   0.35602  0.9891
Lag[4*(p+q)+(p+q)-1][5]   1.25652  0.9014
d.o.f=1
H0 : No serial correlation

Weighted Ljung-Box Test on Standardized Squared Residuals
------------------------------------
                        statistic p-value
Lag[1]                     0.1350  0.7133
Lag[2*(p+q)+(p+q)-1][2]    0.1423  0.8909
Lag[4*(p+q)+(p+q)-1][5]    0.7020  0.9225
d.o.f=1
```

Each *p*-value is fairly large, so the null hypotheses of no autocorrelation of (i) oder 1, (ii) orders 1-2, and (iii) orders 1-5, are maintained for the standardized residuals and the standardized squared residuals alike. This means that the *AR*(1)-*ARCH*(1) model is adequate.

---

[7] This is the decision rule in *R* and EViews, for example, but not necessarily in other software packages. Also note that each of these criteria has alternative formulas, so do not compare them across software packages without checking the formulas.

[8] The weighted *LB* test statistic is $Q_{WLB} = T(T+2)\sum_{k=1}^{s} \frac{s-k+1}{s} \frac{r_k^2}{T-k}$, while the original, i.e. unweighted, *LB* test statistic is $Q_{LB} = T(T+2)\sum_{k=1}^{s} \frac{r_k^2}{T-k}$.

The same conclusion can be drawn from the next part of the printout, which shows the results of weighted *LM* tests for *ARCH* effects remaining in the standardized residuals:

```
Weighted ARCH LM Tests
------------------------------------
             Statistic Shape Scale P-Value
ARCH Lag[2]    0.01434 0.500 2.000  0.9047
ARCH Lag[4]    0.17803 1.397 1.611  0.9651
ARCH Lag[6]    2.41899 2.222 1.500  0.5882
```

Again, each *p*-value is fairly large, so there are not *ARCH* effects left in the residuals.

The last three parts of the printout display the results of three groups of tests. We do not discuss the details of those tests, but it is important to understand their purpose and the conclusions they imply.

The first group consists of joint and individual Nyblom stability tests for parameter stability, i.e., for structural change in the data generating process:

```
Nyblom stability test
------------------------------------
Joint Statistic:  0.7736
Individual Statistics:
mu     0.1729
ar1    0.1755
omega  0.2491
alpha1 0.2032

Asymptotic Critical Values (10% 5% 1%)
Joint Statistic:          1.07 1.24 1.6
Individual Statistic:     0.35 0.47 0.75
```

R performed a joint test for all four parameters and four individual tests. The decision rule for each of these tests is the same, reject the null if the observed test statistic value is larger than the critical value. Comparing the test statistics to the relevant asymptotic critical values, you can see that there is no sign of parameter instability.

The second group consists of sign bias tests for leverage effects, i.e., that negative and positive returns have different influence on future volatility:

```
Sign Bias Test
------------------------------------
                   t-value    prob sig
Sign Bias            1.376 0.17032
Negative Sign Bias   1.344 0.18037
Positive Sign Bias   1.653 0.09993   *
Joint Effect         4.727 0.19293
```

Each *p*-value is above 0.099, so these tests provide further support to the *AR*(1)-*ARCH*(1) model.

Finally, the third group consists of adjusted Pearson tests for goodness of fit, which are chi-squared goodness of fit tests for the comparison of the empirical distribution of the standardized residuals and the chosen conditional distribution of ε_t, which is normal in this example.

```
Adjusted Pearson Goodness-of-Fit Test:
------------------------------------
  group statistic p-value(g-1)
1    20        10       0.9529
2    30        25       0.6782
3    40        30       0.8492
4    50        38       0.8725
```

In this table *group* is the number of equidistant class intervals and no matter whether we consider 20, 30, 40 or even 50 class intervals, the null hypothesis of normal distribution is maintained.

(i) Finally, let's consider one more convenient feature of the *rugarch* package. After having estimated a *GARCH* model, 12 informative plots can be drawn either all at once or one-by-one interactively.

Execute the

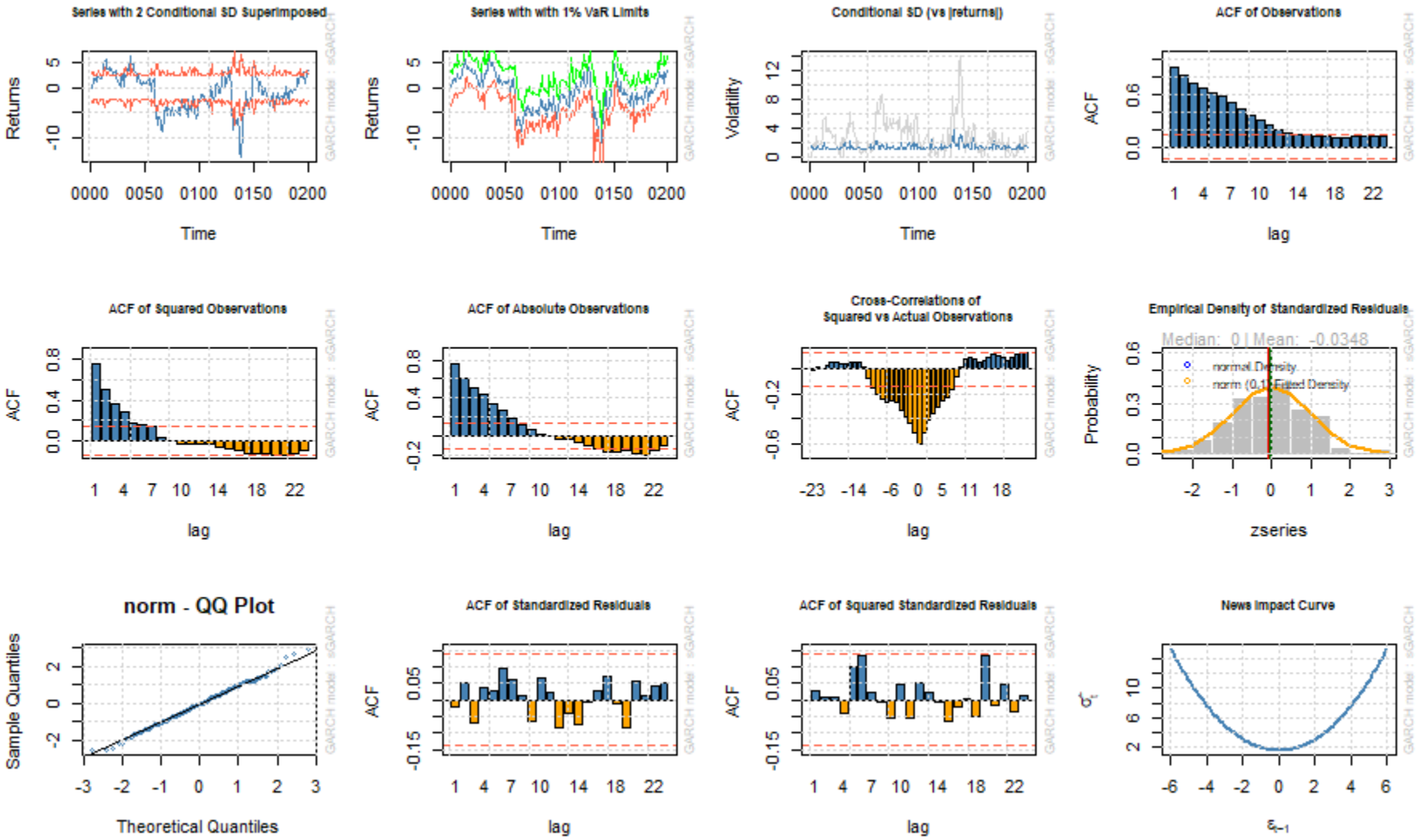
plot(estimate1, which = 'all')

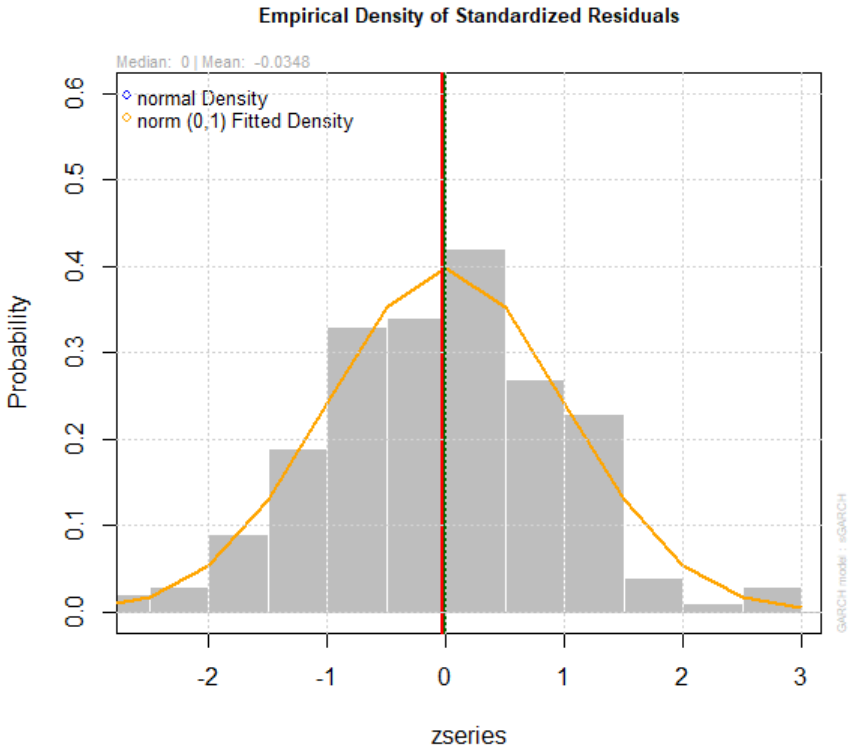
command to get the 12 plots on the next page. To get only one of those 12 plots, replace 'all' with the number of the required plot.[9] For example, to visualize the empirical distribution (density) of the standardized residuals, execute

plot(estimate1, which = 8)

It returns a histogram of the standardized residuals with a superimposed standard normal curve, as displayed on page 16. This plot also suggests that the conditional distribution of ε_t is indeed normal.

[9] The plots are numbered by row starting in cell (1,1).


Series with 2 Conditional SD Superimposed
Returns
Time
Series with with 1% VaR Limits
Returns
Time
Conditional SD (vs |returns|)
Volatility
Time
ACF of Observations
ACF
lag
ACF of Squared Observations
ACF
lag
ACF of Absolute Observations
ACF
lag
Cross-Correlations of
Squared vs Actual Observations
ACF
lag
Empirical Density of Standardized Residuals
Median: 0 | Mean: -0.0348
normal Density
norm (0,1) Fitted Density
Probability
zseries
norm - QQ Plot
Sample Quantiles
Theoretical Quantiles
ACF of Standardized Residuals
ACF
lag
ACF of Squared Standardized Residuals
ACF
lag
News Impact Curve


Empirical Density of Standardized Residuals
Median: 0 | Mean: -0.0348
normal Density
norm (0,1) Fitted Density
Probability
0.0
0.1
0.2
0.3
0.4
0.5
0.6
-2
-1
0
1
2
3
zseries

Case Study for *ARCH* / *GARCH* Modelling of Conditional Variance

This case study is based on Perlin, M.S., Mastella, M., Vancin, D.F., and Ramos, H.P(2021): A GARCH Tutorial with R, *Revista de Administração Contemporânea*, 25(1), e200088. https://doi.org/10.1590/1982-7849rac2021200088.

We are going to follow the same steps as in Exercise 1, without repeating all the explanations. If something is unclear, please go back and review Exercise 1.

Exercise 2

The objective of this case study is to illustrate the application of *GARCH* modelling on some real data, namely on the Bovespa Index (also known as Ibovespa), which is the benchmark index of the major stocks traded on the B3 Brazilian equity market (Brasil Bolsa Balcão).

a) Open the *t8e2.xlsx Excel* file in Excel. As you can see, Sheet 1 has two columns. The first column, *Date*, contains the dates from 4 January 2000 to 15 June 2020, excluding weekends and public holidays when B3 was closed. The second column, *Price*, contains the corresponding daily closing values of the Bovespa index.

   Launch *RStudio*, create a new *RStudio* project and script, and name both *t8e2*. Import the data from *t8e2.xlsx* file and attach it to your project.

b) Prepare a time series plot of *Price*.

   In order to do so, first you need to create a *ts* object by assigning the relevant *Date* value to each *Price* observation. When some time series data are regular in the sense that the observations are evenly spaced, and the frequency is annual, quarterly, monthly or weekly, this can be done by plugging the frequency of the data and the times of the first and last observations as *yyyy*, *c(yyyy, q)*, *c(yyyy, mm)* or *c(yyyy, ww)* into the *ts()* function.

   In this case, however, there are two pitfalls. The first is that the data frequency is daily, and the start and end times must be provided in the *c(yyyy, ddd)* format, where *ddd* is the day in the given year. For example, 4 January 2000, is the 4th day in year 2000, so *ddd* = 4. For days in January this is straightforward, but it is less so for days in the other months. To this end, we can rely on the following date conversion function:

   as.Date(x, …)

where x is a calendar date to be converted to character, or vice versa, a character to be converted to calendar date.

In this example, the first and last dates are 4 January 2000 and 15 June 2020, and to see what days they were in year 2000 and 2020, respectively, execute

```
first = as.Date("2000-01-04") - as.Date("2000-01-01") + 1
last = as.Date("2020-06-15") - as.Date("2020-01-01") + 1
```

The

```
show(first)
show(last)
```

commands should return

```
Time difference of 4 days
```

and

```
Time difference of 167 days
```

The second pitfall is that the data are irregular and cannot be illustrated with a time series plot without filling in the gaps with the *NA* missing value indicator. This can be achieved in three steps. First, we create a date variable that covers every day between 4 January 2000 and 15 June 2020:

```
df1 = data.frame(Date = seq(as.Date("2000-01-04"),
                            as.Date("2020-06-15"), by = 1))
```

Second, we set up another data frame for the Date and Price variables in the active t7e2 data set:

```
df2 = data.frame(Date, Price)
```

Third, we merge the two data frames by adding column *Price* from *df2* to *df1* based on *Date*, with the *join* function(s) of the *dplyr* package. After having installed the *dplyr* library, execute:

```
library(dplyr)
df = left_join(df1, df2, by = "Date")
```

To see the new data frame, execute

```
print(df)
```

You should have something like this in your console:

```
          Date Price
1   2000-01-04 15851
2   2000-01-05 16245
3   2000-01-06 16107
4   2000-01-07 16309
5   2000-01-08    NA
6   2000-01-09    NA
7   2000-01-10 17022
8   2000-01-11 16573
9   2000-01-12 16617
10  2000-01-13 17298
11  2000-01-14 17658
12  2000-01-15    NA
13  2000-01-16    NA
```

Etc.

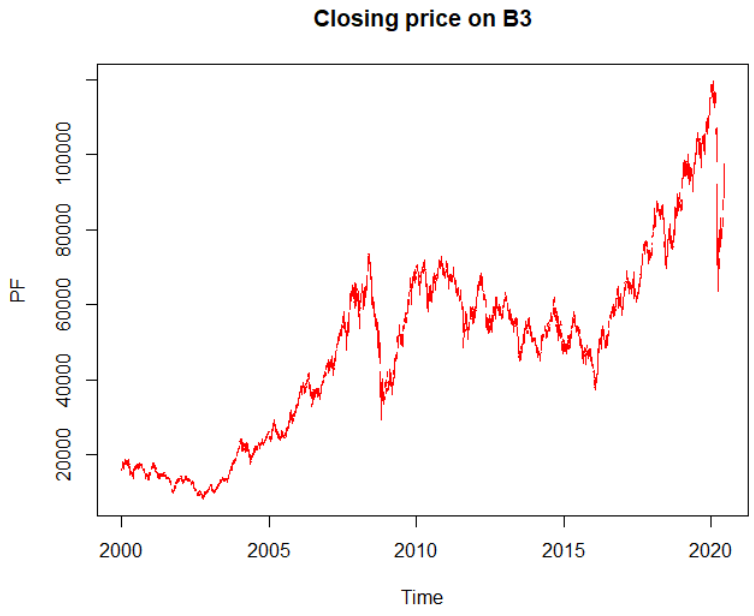
Now we can create a *ts* object for *Price*,

PF = ts(df[,-1], start = c(2000, first), end = c(2020, last), frequency = 365)

where *df[,-1]* refers to the *df* data frame without the first column (i.e., to the *Price* column), and generate the required time series plot in the usual way. For example,

plot.ts(PF, main = "Closing price on B3", col = "red")

returns:



c) Calculate and plot the daily log returns:

$$R_t = \ln\left(\frac{Price_t}{Price_{t-1}}\right)$$

If you perform this task using the *PF* time series object, you end up with *NA* for each day *t* where PF_t or/and PF_{t-1} is/are missing. If, for example, B3 operated on day $t - 1$, was closed on day *t* due to some public holiday, and reopened on day $t + 1$, you would have $PF_t = NA$ and $R_t = R_{t+1} = NA$ for both day *t* and day $t + 1$. This would be, however, a mistake because $PF_t = R_t = NA$ due to the fact that B3 was closed on day *t*. Instead, day *t* should be ignored and R_{t+1} should be the logarithm of PF_{t+1} / PF_{t-1}.

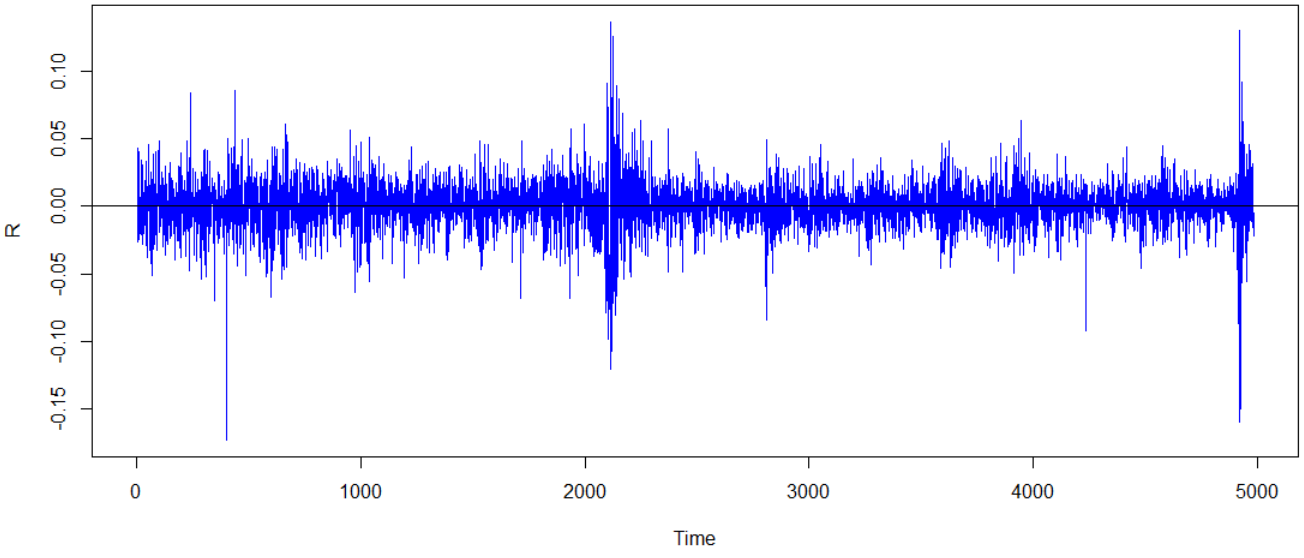
For this reason, instead of using *PF*, the daily log returns are calculated from the original *Price* observations:

```
R = ts(log(Price / lag(Price, 1)))
```

The

```
plot.ts(R, col = "blue")
abline(h=0)
```

commands return the following figure:

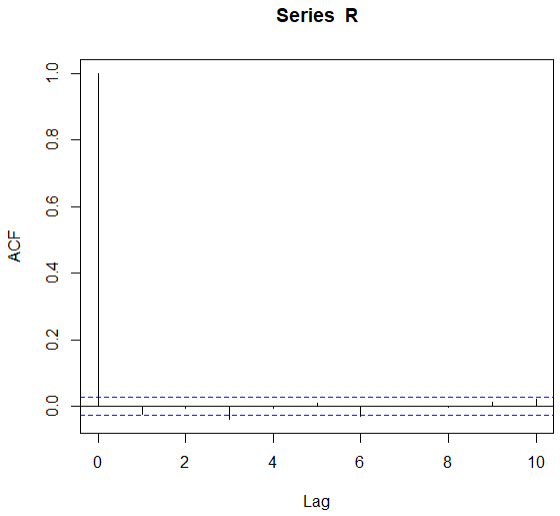


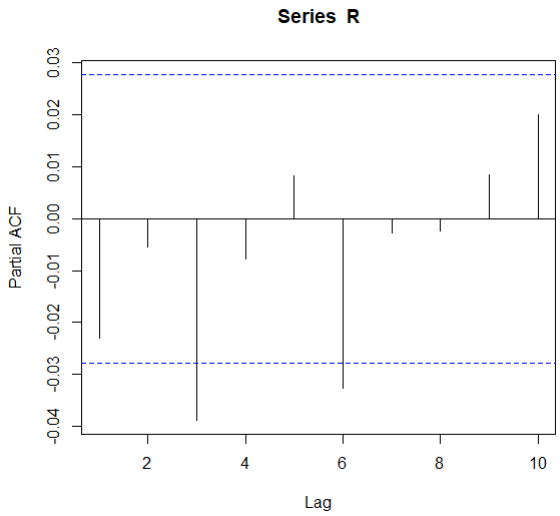
Returns seem to be centered somewhere around zero. In addition, we can observe that occasionally there are sequences of unusually large absolute values, indicating 'volatility clustering'.

d) Develop the correlograms for the log returns. Consider lags 1-10.

```
R = na.remove(R)
acf(R, lag.max = 10, plot = TRUE)
pacf(R, lag.max = 10, plot = TRUE)
```

return[10]





The sample autocorrelation and partial autocorrelation coefficients are fairly small in absolute value.[11] Still, on both correlograms there are significant spikes at lags 3 and 6. There is not, however, any obvious pattern on the correlograms, so one might start modelling the mean of $\{R_t\}$ with a constant only, i.e., with an *ARIMA*(0,0,0) model.

e) Yet, to see whether the *ARIMA*(0,0,0) model fits indeed to the data best, run

```
library(forecast)
best.arima = auto.arima(R, seasonal = FALSE, ic = "aicc",
                        approximation = FALSE, stepwise= FALSE, trace = TRUE)
```

You should get the following printout:

[10] The first element of the *R* object is *NA*, it has to be removed before calling the *acf()*and *pacf()* functions.
[11] The only exception is r_1, but it is always equal to one by default.

```
ARIMA(0,0,0) with zero mean     : -25613.3
ARIMA(0,0,0) with non-zero mean : -25613.12
ARIMA(0,0,1) with zero mean     : -25613.88
ARIMA(0,0,1) with non-zero mean : -25613.78
ARIMA(0,0,2) with zero mean     : -25612.07
ARIMA(0,0,2) with non-zero mean : -25612
ARIMA(0,0,3) with zero mean     : -25617.9
ARIMA(0,0,3) with non-zero mean : -25618
ARIMA(0,0,4) with zero mean     : -25616.04
ARIMA(0,0,4) with non-zero mean : -25616.16
ARIMA(0,0,5) with zero mean     : -25614.35
ARIMA(0,0,5) with non-zero mean : -25614.44
ARIMA(1,0,0) with zero mean     : -25613.85
ARIMA(1,0,0) with non-zero mean : -25613.75
ARIMA(1,0,1) with zero mean     : Inf
ARIMA(1,0,1) with non-zero mean : Inf
ARIMA(1,0,2) with zero mean     : Inf
ARIMA(1,0,2) with non-zero mean : Inf
ARIMA(1,0,3) with zero mean     : -25615.87
ARIMA(1,0,3) with non-zero mean : -25615.97
ARIMA(1,0,4) with zero mean     : Inf
ARIMA(1,0,4) with non-zero mean : Inf
ARIMA(2,0,0) with zero mean     : -25611.97
ARIMA(2,0,0) with non-zero mean : -25611.89
```

```
ARIMA(2,0,0) with non-zero mean : -25611.89
ARIMA(2,0,1) with zero mean     : Inf
ARIMA(2,0,1) with non-zero mean : Inf
ARIMA(2,0,2) with zero mean     : -25611.7
ARIMA(2,0,2) with non-zero mean : -25611.79
ARIMA(2,0,3) with zero mean     : Inf
ARIMA(2,0,3) with non-zero mean : Inf
ARIMA(3,0,0) with zero mean     : -25617.33
ARIMA(3,0,0) with non-zero mean : -25617.4
ARIMA(3,0,1) with zero mean     : -25615.49
ARIMA(3,0,1) with non-zero mean : -25615.63
ARIMA(3,0,2) with zero mean     : Inf
ARIMA(3,0,2) with non-zero mean : Inf
ARIMA(4,0,0) with zero mean     : -25615.59
ARIMA(4,0,0) with non-zero mean : -25615.69
ARIMA(4,0,1) with zero mean     : Inf
ARIMA(4,0,1) with non-zero mean : Inf
ARIMA(5,0,0) with zero mean     : -25613.96
ARIMA(5,0,0) with non-zero mean : -25614.03



Best model: ARIMA(0,0,3) with non-zero mean
```

ARIMA(0,0,3) with non-zero mean produced the smallest *AICc* statistic, -25618. It is certainly smaller than *AICc* of *ARIMA*(0,0,0), which is -25613.3, though the improvement is rather small.

To see the details of the *ARIMA*(0,0,3) model, execute

summary(best.arima)
library(lmtest)
coeftest(best.arima, df = best.arima$nobs - length(best.arima$coef))

They return the following printouts:

```
Series: R
ARIMA(0,0,3) with non-zero mean

Coefficients:
          ma1      ma2      ma3   mean
      -0.0239  -0.0052  -0.0414  4e-04
s.e.   0.0142   0.0142   0.0146  2e-04

sigma^2 = 0.0003425:  log likelihood = 12814.01
AIC=-25618.01   AICc=-25618   BIC=-25585.44
```

and

```
t test of coefficients:

             Estimate  Std. Error t value Pr(>|t|)
ma1       -0.02389211  0.01415073 -1.6884 0.091397 .
ma2       -0.00518926  0.01424635 -0.3643 0.715686
ma3       -0.04138197  0.01460151 -2.8341 0.004614 **
intercept  0.00035366  0.00024401  1.4494 0.147291
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

There is only one significant coefficient at the 5% level, but it belongs to *ma3*, supporting the *ARIMA*(0,0,3) specification.

It is also important to check whether the residuals from *ARIMA*(0,0,3) are serially uncorrelated. The

```
k = length(best.arima$coef)
Box.test(best.arima$residuals, type = "Ljung-Box", lag = 10, fitdf = k)
```

commands return

```
        Box-Ljung test

data:  best.arima$residuals
X-squared = 8.1279, df = 5, p-value = 0.1493
```

There is no sign of autocorrelation of orders 1 to 10, not even at the 10% significance level. Hence, we can accept the *ARIMA*(0,0,3) model for the mean of $\{R_t\}$.

f) To see whether an *ARCH* variance equation is warranted by the data, perform the *ARCH LM* test for lags up to 5.

It is possible to perform these tests by running *ArchTest()* function of the *FinTS R* package five times for lags 1, 2, 3, 4, 5, but it is nicer to combine *ARCHTest()* with a *for* loop:

```
library(FinTS)
for (h in 1:5) {
  print(ArchTest(best.arima$residuals, lags = h))
}
```

You should get the following printouts:

```
        ARCH LM-test; Null hypothesis: no ARCH effects

data:  best.arima$residuals
Chi-squared = 404.25, df = 1, p-value < 2.2e-16


        ARCH LM-test; Null hypothesis: no ARCH effects

data:  best.arima$residuals
Chi-squared = 877.75, df = 2, p-value < 2.2e-16


        ARCH LM-test; Null hypothesis: no ARCH effects

data:  best.arima$residuals
Chi-squared = 919, df = 3, p-value < 2.2e-16


        ARCH LM-test; Null hypothesis: no ARCH effects

data:  best.arima$residuals
Chi-squared = 936.06, df = 4, p-value < 2.2e-16


        ARCH LM-test; Null hypothesis: no ARCH effects

data:  best.arima$residuals
Chi-squared = 967.86, df = 5, p-value < 2.2e-16
```

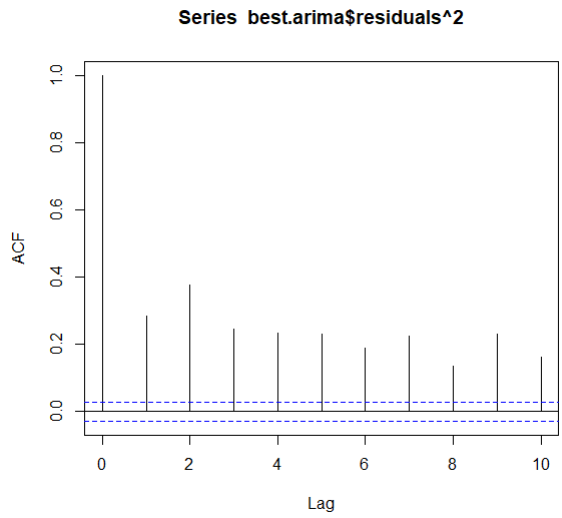
On these printouts *df* is equal to the lag length, i.e., the first printout is for *ARCH* effect of order 1, the second is for *ARCH* effects of orders 1 and 2, and so on.

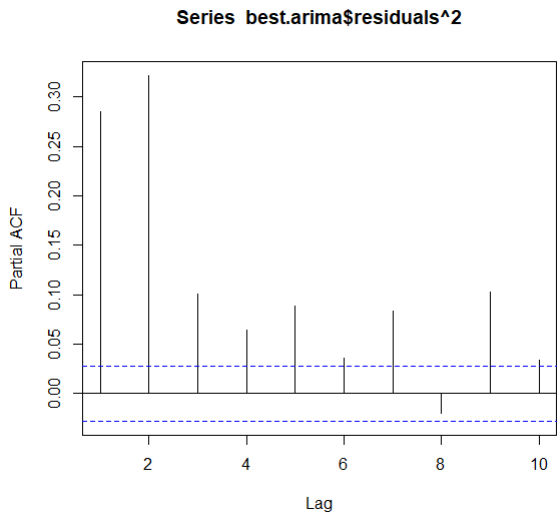
The *p*-values are practically zero, so each test rejects the null hypothesis of no *ARCH* errors.

g) To get some idea about the variance equation, develop the correlograms for the squared residuals from the *ARIMA*(0,0,3) model. Consider lags 1-10.

acf(best.arima$residuals^2, lag.max = 10, plot = TRUE)
pacf(best.arima$residuals^2, lag.max = 10, plot = TRUE)

produce the correlograms shown on the next page.





Every coefficient is significant at 5% level. In this case it is best to experiment with a parsimonious *GARCH*(1,1) variance equation.

h) Fit an *ARMA*(0,3)-*GARCH*(1,1) model to $\{R_t\}$. Assume that error term in the mean equation is normally distributed.

Execute the following commands:

```
library(rugarch)
spec_v1 = ugarchspec(mean.model = list(armaOrder = c(0,3), include.mean = TRUE),
                     variance.model = list(model = "sGARCH", garchOrder = c(1,1)),
                     distribution.model = "norm")
estimate_v1 = ugarchfit(spec = spec_v1, data = R)
print(estimate_v1)
```

The first part of the printout is at the top of the next page. It shows that no matter whether we rely on the usual or on the robust standard errors, in the mean equation *mu* and *ma3* are significant, just like in the *ARIMA*(0,0,3) model in part (e), while in the variance equation *omega*, *alpha1* and *beta1* are all significant.

The weighted *LB* tests at the bottom of the next page maintain the null hypotheses of no autocorrelation of (i) order 1, (ii) orders 1-5, and (iii) orders 1-9, for the standardized residuals and the standardized squared residuals alike, supporting the model specification.

```
*----------------------------------*
*          GARCH Model Fit         *
*----------------------------------*

Conditional Variance Dynamics
-----------------------------------
GARCH Model      : sGARCH(1,1)
Mean Model       : ARFIMA(0,0,3)
Distribution     : norm

Optimal Parameters
------------------------------------
        Estimate  Std. Error    t value Pr(>|t|)
mu      0.000760    0.000207    3.67009 0.000242
ma1    -0.003222    0.015117   -0.21313 0.831226
ma2    -0.008968    0.015137   -0.59245 0.553547
ma3    -0.035220    0.015079   -2.33570 0.019507
omega   0.000008    0.000001   10.24968 0.000000
alpha1  0.075500    0.003914   19.28718 0.000000
beta1   0.900178    0.005847  153.96155 0.000000

Robust Standard Errors:
        Estimate  Std. Error    t value Pr(>|t|)
mu      0.000760    0.000238    3.19427 0.001402
ma1    -0.003222    0.012635   -0.25499 0.798732
ma2    -0.008968    0.015499   -0.57860 0.562861
ma3    -0.035220    0.014646   -2.40467 0.016187
omega   0.000008    0.000001    5.31534 0.000000
alpha1  0.075500    0.005928   12.73597 0.000000
beta1   0.900178    0.007837  114.86400 0.000000



Weighted Ljung-Box Test on Standardized Residuals
------------------------------------
                        statistic p-value
Lag[1]                     0.5563  0.4558
Lag[2*(p+q)+(p+q)-1][8]    0.9192  1.0000
Lag[4*(p+q)+(p+q)-1][14]   2.5099  0.9994
d.o.f=3
H0 : No serial correlation

Weighted Ljung-Box Test on Standardized Squared Residuals
------------------------------------
                        statistic p-value
Lag[1]                     0.1069  0.7438
Lag[2*(p+q)+(p+q)-1][5]    0.8145  0.9000
Lag[4*(p+q)+(p+q)-1][9]    2.1932  0.8807
d.o.f=2
```

In addition, the weighted *ARCH LM* tests also maintain the null hypothesis of no (additional) *ARCH* effects.

```
Weighted ARCH LM Tests
------------------------------------
            Statistic Shape Scale P-Value
ARCH Lag[3]  0.001582 0.500 2.000  0.9683
ARCH Lag[5]  0.172356 1.440 1.667  0.9719
ARCH Lag[7]  1.375309 2.315 1.543  0.8460
```

So far so good. The remining tests, however, cast some doubt on the model.

First, the joint Nyblom stability test rejects the null hypothesis that the parameter values are constant, just like the individual tests on *omega* and *beta1*, and at the 5% significance level on *alpha1* too.

```
Nyblom stability test
------------------------------------
Joint Statistic:  12.1874
Individual Statistics:
mu     0.20916
ma1    0.18000
ma2    0.02544
ma3    0.19198
omega  2.90766
alpha1 0.64102
beta1  1.11752

Asymptotic Critical Values (10% 5% 1%)
Joint Statistic:         1.69 1.9 2.35
Individual Statistic:    0.35 0.47 0.75
```

Second, the test for positive sign bias rejects the null hypothesis that large and small positive shocks have the same impact on volatility, and the sign test for the joint (i.e., negative and positive) effect also rejects the null hypothesis that large and small, in absolute value, shocks have the same impact on volatility.

```
Sign Bias Test
------------------------------------
                   t-value      prob sig
Sign Bias           0.2191 0.8266007
Negative Sign Bias  1.4846 0.1377102
Positive Sign Bias  2.3232 0.0202111  **
Joint Effect       17.2206 0.0006366 ***
```

Hence, this model does not account for the difference between large and small shocks.

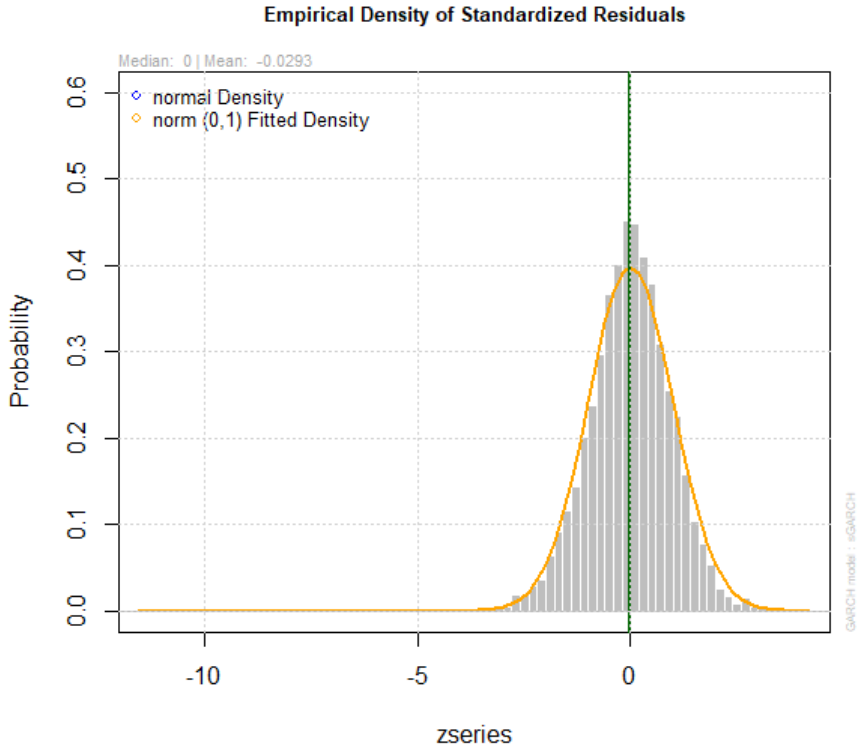
Third, the adjusted chi-squared goodness of fit tests reject the null suggesting that the conditional distribution of ε_t is not normal, as assumed originally.

```
Adjusted Pearson Goodness-of-Fit Test:
------------------------------------
  group statistic p-value(g-1)
1    20     69.34    1.184e-07
2    30     65.93    1.069e-04
3    40     87.01    1.604e-05
4    50     88.63    4.531e-04
```

i) Finally, to get further insights, obtain two plots by executing

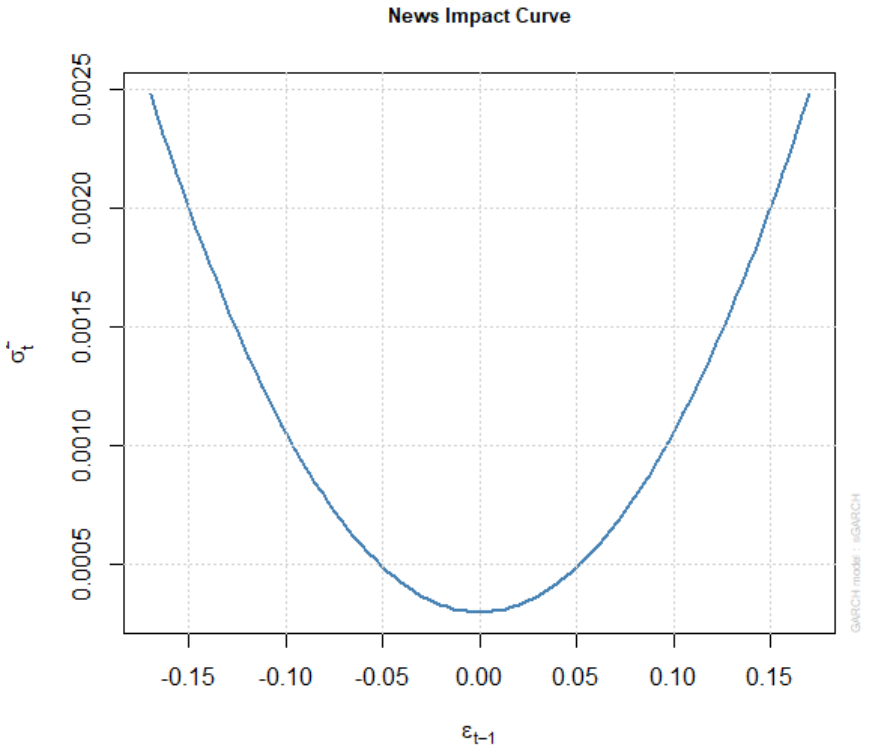
plot(estimate_v1, which = 8)
plot(estimate_v1, which = 12)

The first plot,



shows that there is indeed some discrepancy between the empirical distribution and the hypothesized (standard normal) distribution. This discrepancy, however, does not appear to be large at all, and the histogram is pretty much symmetric.

The second plot,



returns the news impact curve, a helpful tool to visualize the response of the variance to the negative and positive shocks. Interestingly, this curve looks pretty much symmetric, providing no support to the sign bias tests.