Financial Econometrics Practical

Practical 6: Univariate Volatility Modelling

NF Katzke

Table of Contents

1 Introduction

In this session we see how to build volatility models (specifically from the widely used GARCH family of models) using R. We will be using several packages that are designed to be great at doing volatility analyses on financial data.

Also see this blog for some cool ideas on applications of volatility modelling in R and its application.

These models have many applications, and the study of volatility is particularly important in financial modelling. As a key input to derivative pricing (think Black-Scholes formula where we have to input our estimate of implied volatility), financial engineers need a simple and robust means of using the past as effectively as possible to provide an estimate of this elusive measure.

The GARCH family is incredibly broad (Bollerslev 2008 identified over 180 types of GARCH models) and there are many packages in R that allow its calculation. Another particularly powerful platform is that of G@RCH in Oxmetrics, designed by Laurent. We will be using **Rugarch** and **rmgarch** designed by Alexios Ghalanos, while maybe also looking briefly at other packages (such as **MTS** and **fgarch**).

My recommended literature for this session includes the very good GARCH modeling summary paper by Hentschel (1995). Also see general summary papers by Bauwens, Laurent, and Rombouts (2006) & Silvennoinen and Teräsvirta (2009). Ofcourse also look at the Tsay (2014) and Tsay (2012) textbooks, as well as Ruppert (2011).

In terms of using **rugarch**, I suggest viewing the author of the package's blog on using **rugarch** here. It includes some great examples and research ideas that you can apply.

```
# Note: we will install two new packages for this tutorial: rugarch and forecast
library(rmsfuns)
load_pkg(c("devtools", "rugarch", "forecast", "tidyr", "Dplyr2Xts", "lubridate", "readr", "Perfor
dailydata <-
read_csv("https://raw.githubusercontent.com/Nicktz/FinMetrics/master/extdata/findata.csv",
col_types = cols(.default = "d", Date = "D") )
```

2 Data Import

Today we consider some of SA's largest financial institutions and banks (we consider the daily adjusted closing prices). The higher the frequency, the higher typically the volatility (noise often smoothen out as the frequency decreases). Let's load the data and go through the usual calculation of returns:

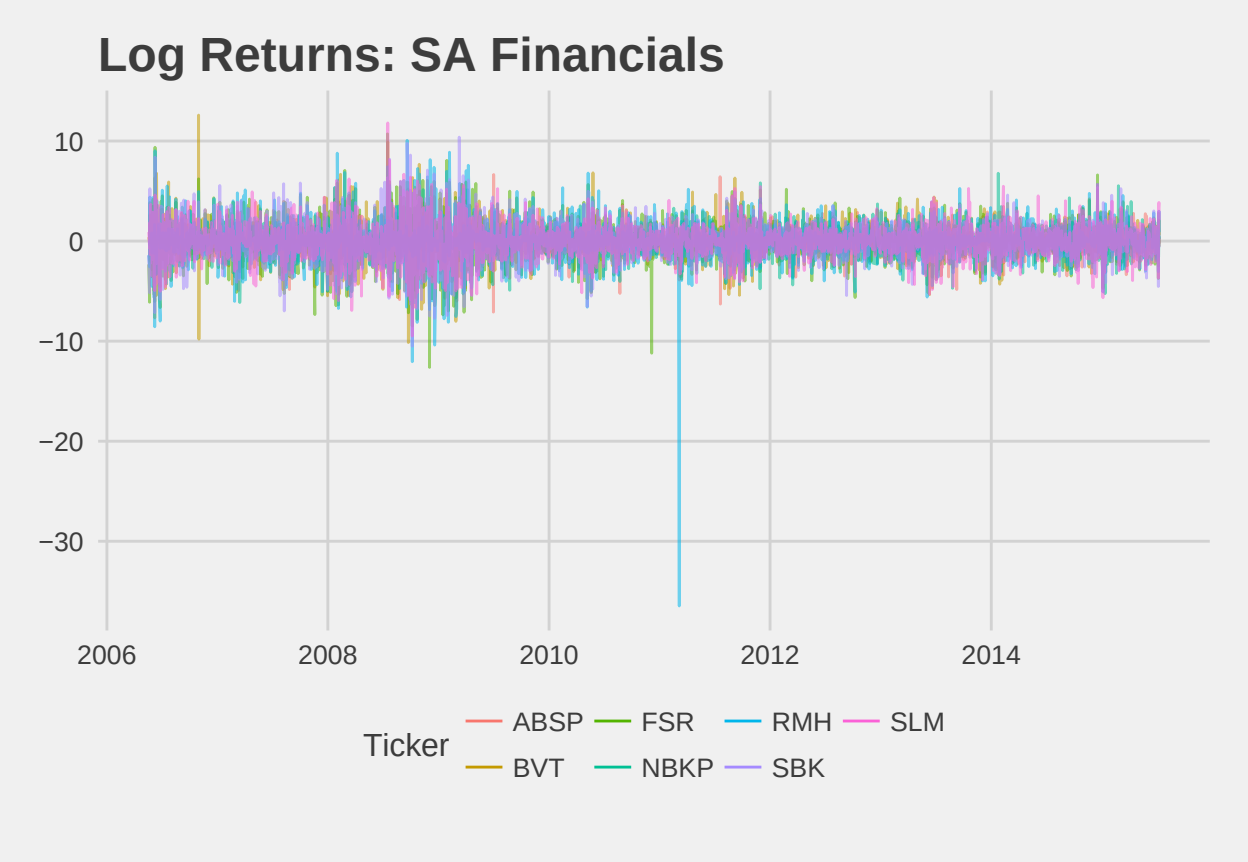
```
# dlog returns:
rtn <- (
  diff( log(dailydata %>% arrange(Date) %>% Dplyr2Xts()), lag=1)
        )*100

#drop the first observation and corresponding date:
rtn <- rtn[-1,]
# Center the data:
rtn <- scale(rtn,center=T,scale=F)
```

First, let's clean up the names. I know they are all listed on the JSE, and that they are closing prices...

```
  colnames(rtn) <-
  colnames(rtn) %>%  gsub("JSE.","",.) %>% gsub(".Close","",.)
```

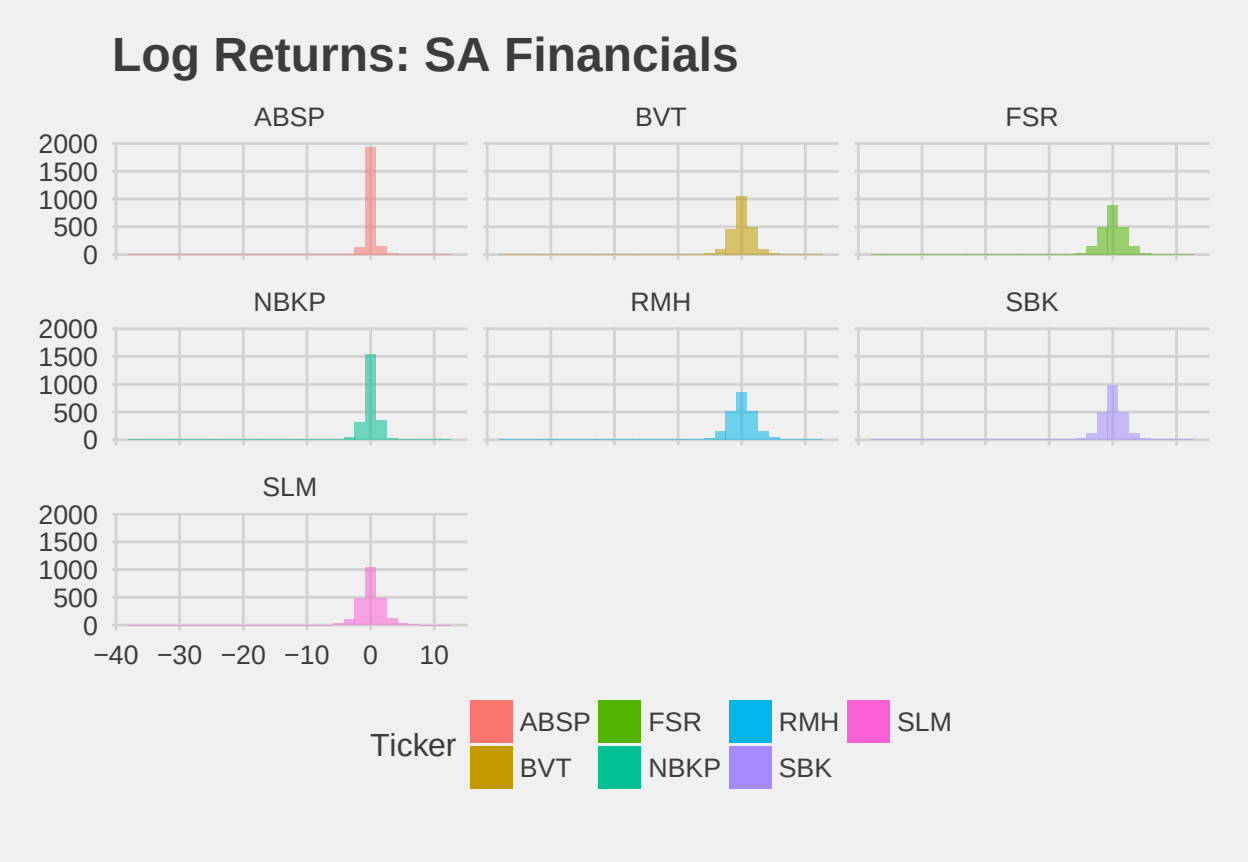
Next, let's plot the series to check for obvious outliers in the returns series:

```
Tidyrtn <-
rtn %>% Xts2Dplyr() %>%
  gather(Ticker, CP, -date)


ggplot(Tidyrtn) +
  geom_line(aes(x = date, y = CP, colour = Ticker, alpha = 0.5)) +
  ggtitle("Log Returns: SA Financials") +
  guides(alpha=FALSE) +
  theme_fivethirtyeight()
```




```
# Histograms: (Note the fat left tails...)
ggplot(Tidyrtn) +
  geom_histogram(aes(x = CP, fill = Ticker, alpha = 0.5)) +
  ggtitle("Log Returns: SA Financials") +
  facet_wrap(~Ticker) +
  guides(alpha=FALSE) +
  theme_fivethirtyeight()
```

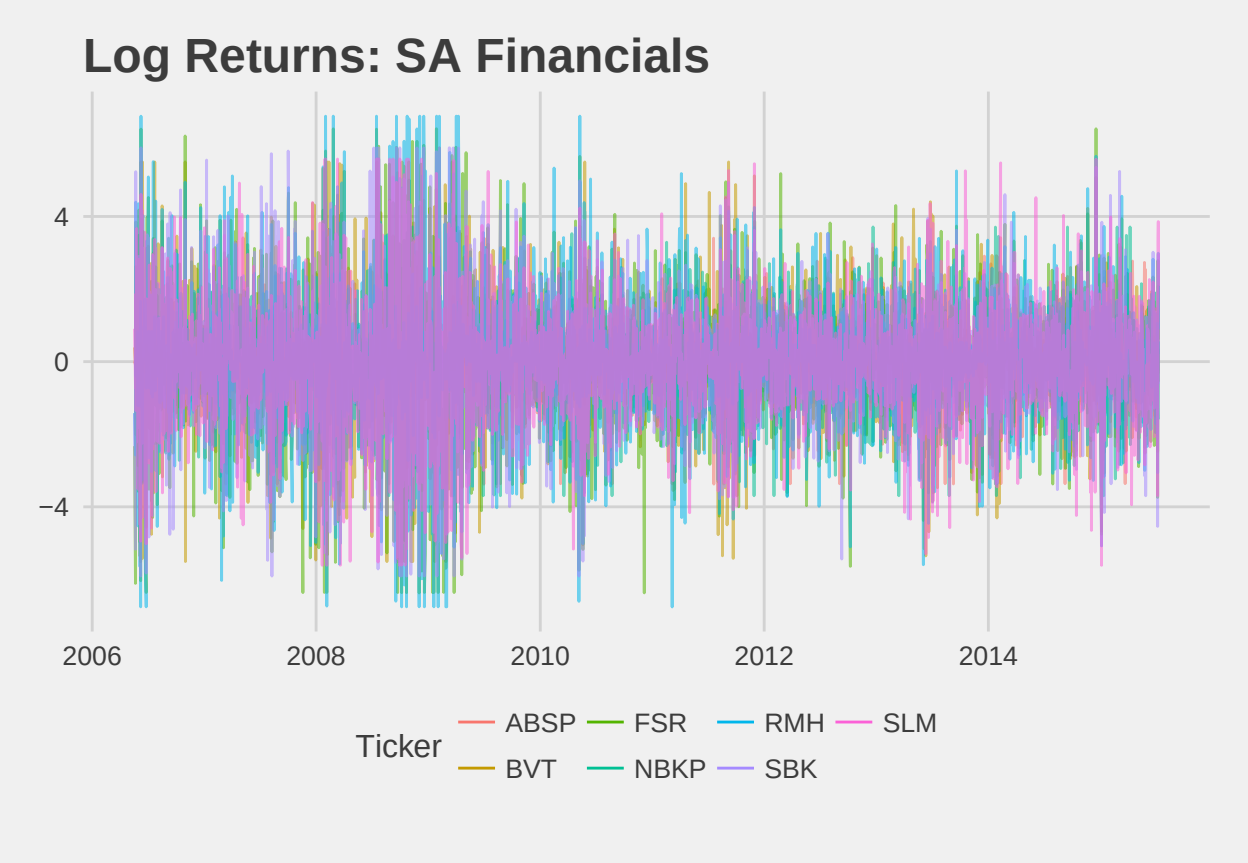
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



From the plots we see large left tails, but specifically a very large sharp sell off of RMH. Let's clean the data and replot it to see if this has been removed:

```
rtn <- Return.clean(rtn, method = c("none", "boudt", "geltner")[2], alpha = 0.01)


# Note that very large RMH outlier has now been removed...


Tidyrtn <-
rtn %>% Xts2Dplyr() %>%
  gather(Ticker, CP, -date)
```

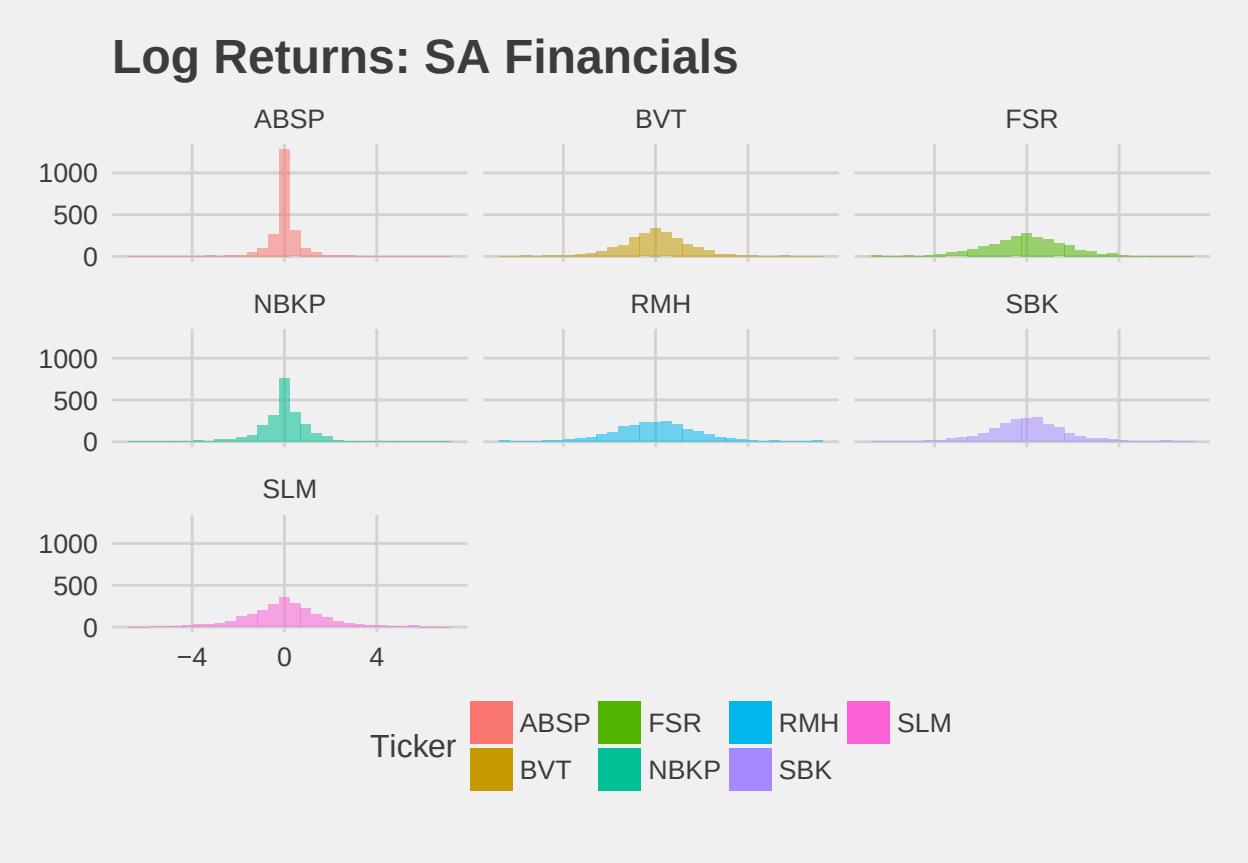
```
ggplot(Tidyrtn) +
  geom_line(aes(x = date, y = CP, colour = Ticker, alpha = 0.5)) +
  ggtitle("Log Returns: SA Financials") +
  guides(alpha=FALSE) +
  theme_fivethirtyeight()
```




```
ggplot(Tidyrtn) +
  geom_histogram(aes(x = CP, fill = Ticker, alpha = 0.5)) +
  ggtitle("Log Returns: SA Financials") +
  facet_wrap(~Ticker) +
  guides(alpha=FALSE) +
```

```
  theme_fivethirtyeight()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
# This removal of very large outliers are important for the convergence of our models later, so r
```

It seems that particularly ABSA has a large clumping together of data at zero - let's verify whether this is the case:

```
colSums(rtn == 0)
```

```
## ABSP  BVT  FSR NBKP  RMH  SBK  SLM
```

```
##    0    0    0    0    0    0    0
```

Right, so at least our fear of having many returns = 0 is unfound. It simply implies ABSA has many periods where returns are close, but not equal to, zero.

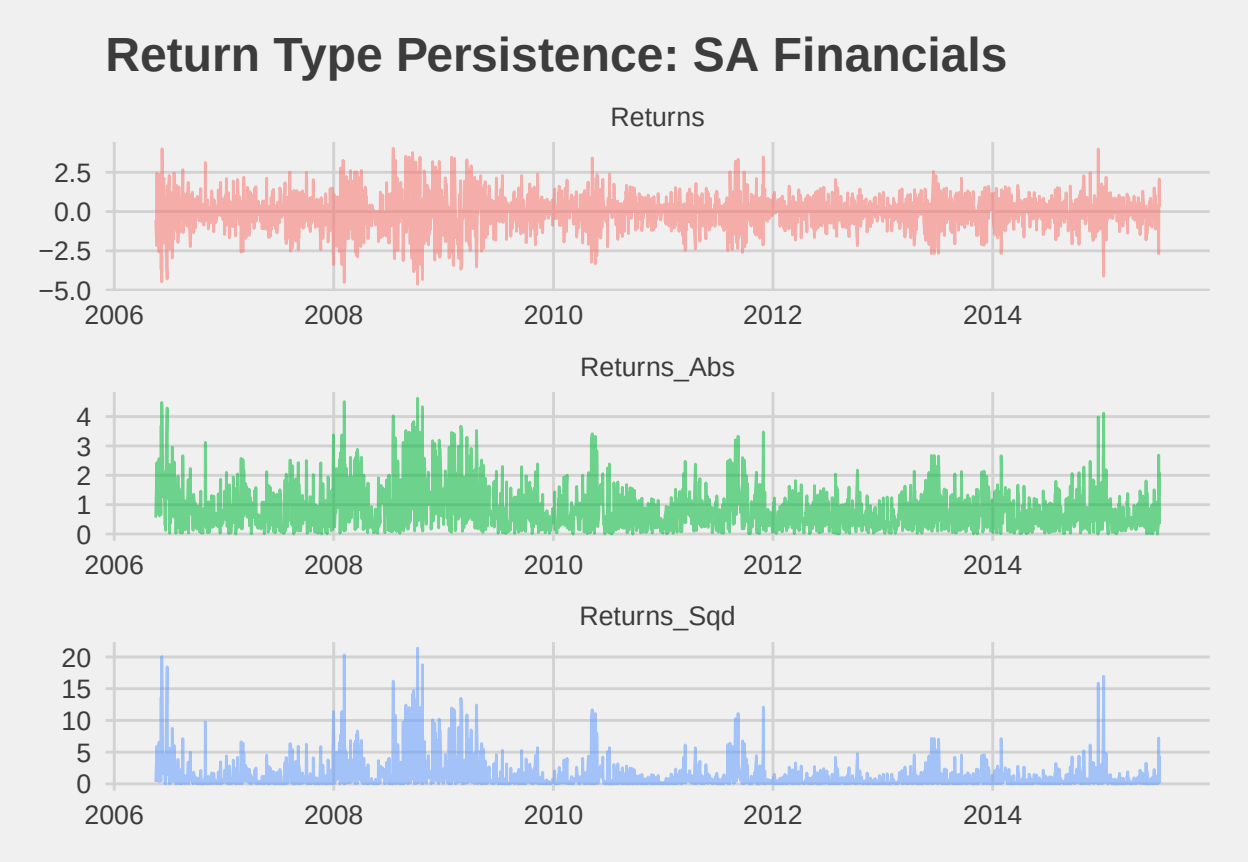
2.1 Auto-Persistence in Returns

As discussed in the class, ARCH effects are seen in the squared residuals of the mean equation, or as below when graphing the squared returns. Let's graph the returns, squared returns and the absolute returns for an equal weighted portfolio of these shares:

```
# Create the Simple returns:
Rtn <- log(1 + rtn/100)*100
# Remember - if you  don't divide rtn by 100 you might have log of negative values - producing NA
porteqw <- Return.portfolio(Rtn, weight = NULL, geometric = FALSE)
```

Now, let's plot the returns, sqd returns and |retuns|:

```
Plotdata = cbind(porteqw, porteqw^2, abs(porteqw))
colnames(Plotdata) = c("Returns", "Returns_Sqd", "Returns_Abs")

Plotdata <-
Plotdata %>% Xts2Dplyr() %>% gather(ReturnType, Returns, -date)

ggplot(Plotdata) +
  geom_line(aes(x = date, y = Returns, colour = ReturnType, alpha = 0.5)) +
  ggtitle("Return Type Persistence: SA Financials") +
  facet_wrap(~ReturnType, nrow = 3, ncol = 1, scales = "free") +
  guides(alpha=FALSE, colour = FALSE) +
  theme_fivethirtyeight()
```

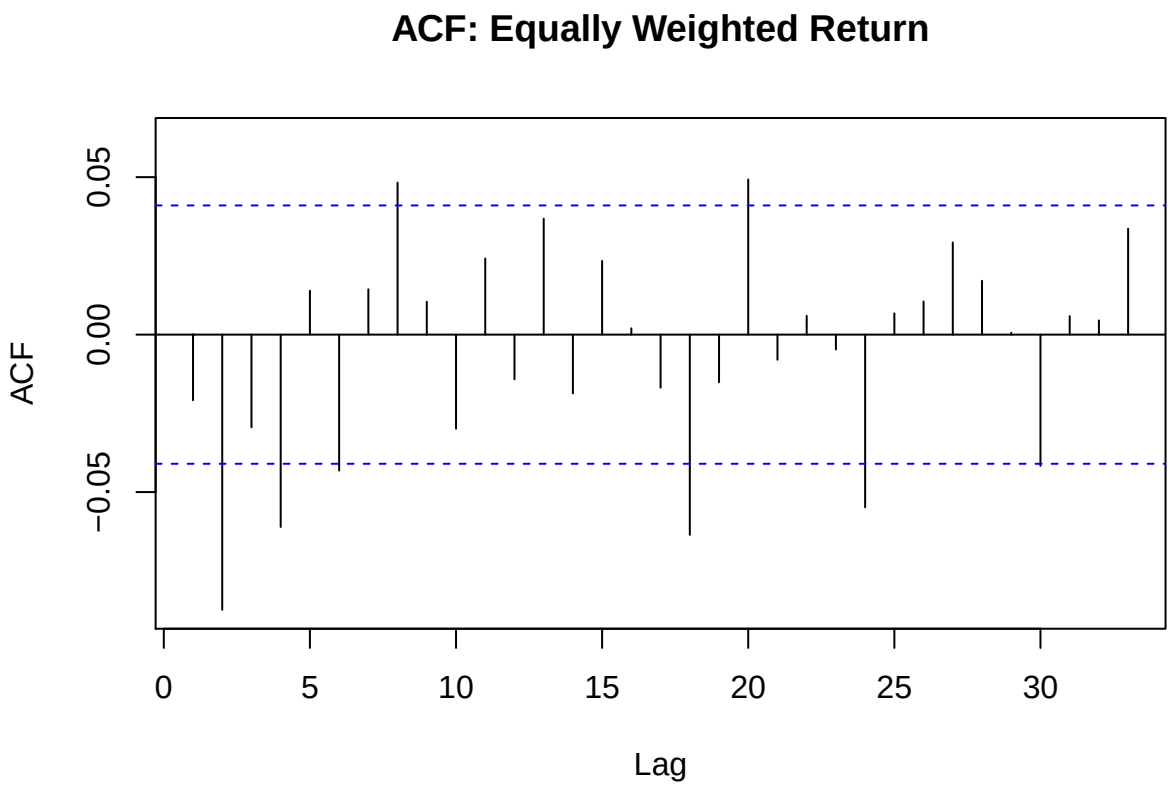


From the above figure it seems that:

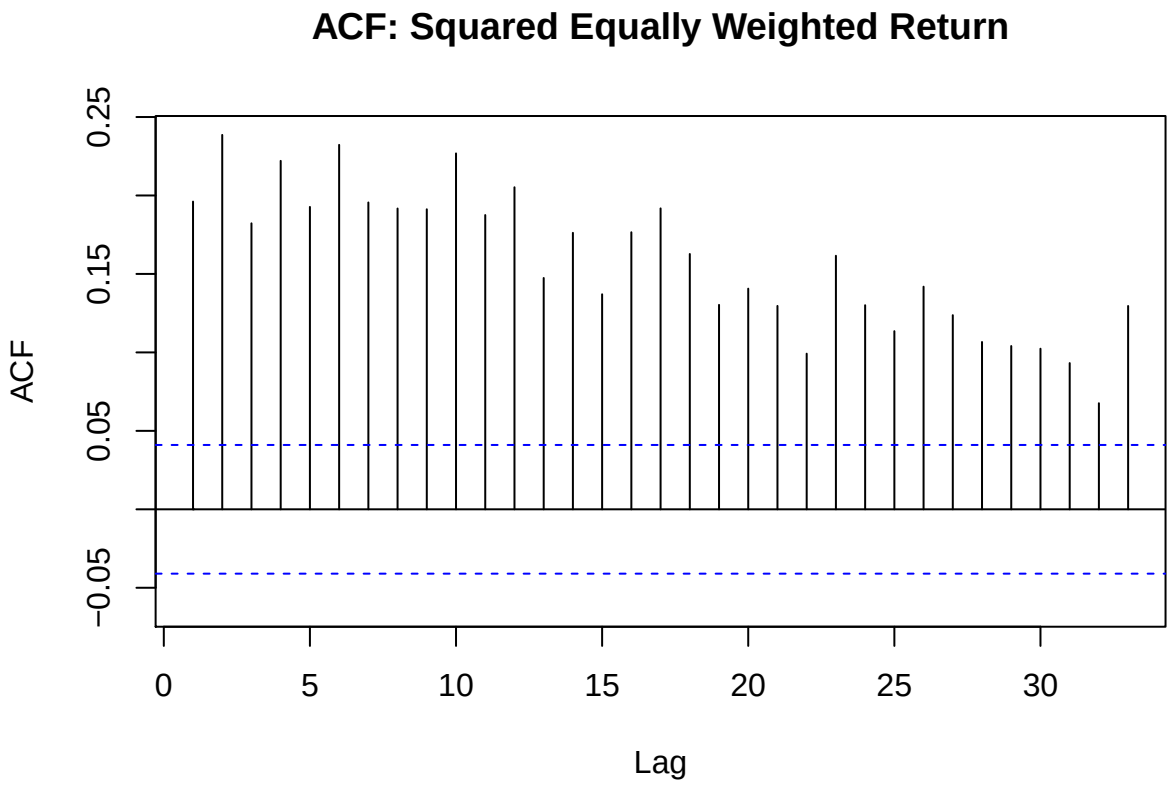
- There remains strong first order persistence in returns
- There is clearly periods of strong second order persistence
- There is clear evidence of long memory in the second order process.

To verify the above, let's plot the ACFs:

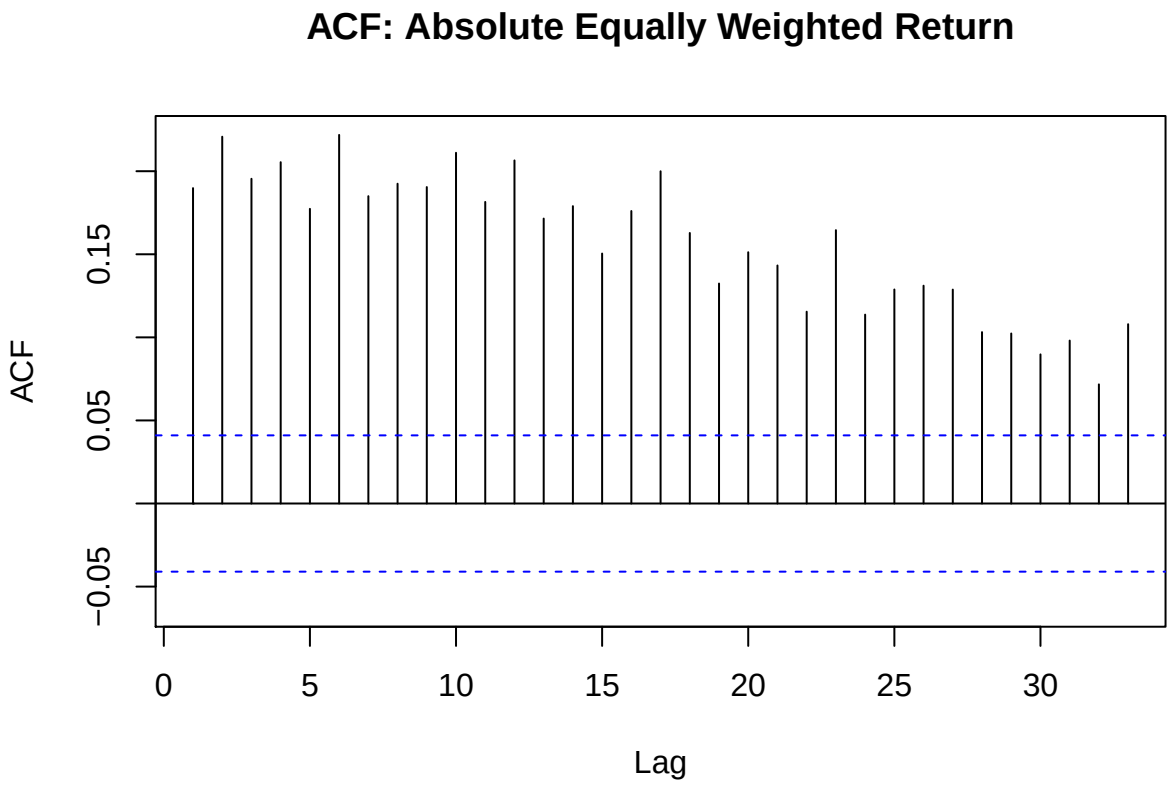
```
forecast::Acf(porteqw, main = "ACF: Equally Weighted Return")
```




```
forecast::Acf(porteqw^2, main = "ACF: Squared Equally Weighted Return")
```




```
forecast::Acf(abs(porteqw), main = "ACF: Absolute Equally Weighted Return")
```




```
# Note:I use the packagename and :: here - as other packages also (stupidly)  have the same name
```

The above proves what we expected - in particular very strong conditional heteroskedasticity, as well as long memory.

A formal test for ARCH effects: LBQ stats on squared returns:

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

```
##
##  Box-Ljung test
##
```

```
## data:  coredata(porteqw^2)
## X-squared = 1167.5, df = 12, p-value < 2.2e-16
```

Fin.Ts provides the ARCH-LM test for conditional heteroskedasticity in the returns:

```
load_pkg("FinTS")
ArchTest(porteqw)
```

```
##
##  ARCH LM-test; Null hypothesis: no ARCH effects
##
## data:  porteqw
## Chi-squared = 363.69, df = 12, p-value < 2.2e-16
```

Both tests reject the *nulls of no ARCH effects* - hence we need to control for the remaining conditional heteroskedasticity in the returns series,

3 Fitting GARCH using rugarch

Let's now fit the ARCH models using the package **rugarch**.

Remember, the standard garch(1,1) model is written as follows:

$$r_t = \mu + \varepsilon_t \tag{3.1}$$

$$\varepsilon_t = \sigma_t . z_t \tag{3.2}$$

$$\sigma_t^2 = \alpha + \beta_1 \varepsilon_{t-1}^2 + \beta_1 \sigma_{t-1}^2 \tag{3.3}$$

$$z_t \sim \mathcal{N}(0,1) \tag{3.4}$$

Fitting it, we first specify our garch setup, and then we fit it. We have to choose the mean model and the variance model. Let's choose the most appropriate mean model first, using the **autoarfima** function to test all combinations. note, that this estimation will take a long time if done directly (you can check this) - and as such we can make use of the parallel library which allows your machine to do similar parallel computations at the same time so as to save time (don't worry about the details, just use it):

```
# Enable parallel computing:
load_pkg("parallel")
cl = makePSOCKcluster(10)
#  I use the partial method below, otherwise it might take a long time to run.
AC = autoarfima(as.numeric(porteqw), ar.max = 2, ma.max = 2, criterion = 'AIC', method = 'partial
show(head(AC$rank.matrix))
```

```
##   AR MA Mean ARFIMA      AIC converged
## 1  2  2    0      0 3.024304         1
## 2  2  2    1      0 3.025027         1
## 3  1  2    0      0 3.030265         1
## 4  2  1    0      0 3.030314         1
## 5  1  2    1      0 3.030989         1
## 6  2  1    1      0 3.031037         1
```

So the function suggests we should use a ARIMA(2,2) model without a mean. For simplicity, let's suppose the above function suggested an AR(1) model to be used, let's now fit a GARCH(1,1) model on an AR(1) mean model:

```
# Note we specify the mean (mu) and variance (sigma) models separately:
garch11 <-
  ugarchspec(
    variance.model = list(model = c("sGARCH","gjrGARCH","eGARCH","fGARCH","apARCH")[1],
```

```
    garchOrder = c(1, 1)),
    mean.model = list(armaOrder = c(1, 0), include.mean = TRUE),
    distribution.model = c("norm", "snorm", "std", "sstd", "ged", "sged", "nig", "ghyp", "jsu")[1
# Now to fit, I use as.matrix and the data - this way the plot functions we will use later will u
garchfit1 = ugarchfit(spec = garch11,data=as.numeric(porteqw))

# Note it saved a S4 class object - having its own plots and functionalities:
class(garchfit1)
```

```
## [1] "uGARCHfit"
## attr(,"package")
## [1] "rugarch"
```

```
# Also see the output for the garchfit1:
garchfit1
```

```
##
## *---------------------------------*
## *          GARCH Model Fit        *
## *---------------------------------*
##
## Conditional Variance Dynamics
## -----------------------------------
## GARCH Model  : sGARCH(1,1)
## Mean Model   : ARFIMA(1,0,0)
## Distribution : norm
##
## Optimal Parameters
```

```
## ------------------------------------
##         Estimate  Std. Error  t value Pr(>|t|)
## mu      0.016065    0.018409  0.87266  0.38285
## ar1    -0.029527    0.021869 -1.35018  0.17696
## omega   0.022710    0.006311  3.59865  0.00032
## alpha1  0.082439    0.011158  7.38839  0.00000
## beta1   0.897921    0.013686 65.60926  0.00000
##
## Robust Standard Errors:
##         Estimate  Std. Error  t value Pr(>|t|)
## mu      0.016065    0.016889   0.9512 0.341504
## ar1    -0.029527    0.019700  -1.4989 0.133909
## omega   0.022710    0.006292   3.6095 0.000307
## alpha1  0.082439    0.011766   7.0065 0.000000
## beta1   0.897921    0.014694  61.1069 0.000000
##
## LogLikelihood : -3250.761
##
## Information Criteria
## ------------------------------------
##
## Akaike       2.8509
## Bayes        2.8635
## Shibata      2.8509
## Hannan-Quinn 2.8555
##
## Weighted Ljung-Box Test on Standardized Residuals
## ------------------------------------
```

```
##                         statistic   p-value
## Lag[1]                     0.0187 0.8912438
## Lag[2*(p+q)+(p+q)-1][2]    4.8792 0.0005823
## Lag[4*(p+q)+(p+q)-1][5]   10.0971 0.0009949
## d.o.f=1
## H0 : No serial correlation
##
## Weighted Ljung-Box Test on Standardized Squared Residuals
## ------------------------------------
##                         statistic p-value
## Lag[1]                     0.5055  0.4771
## Lag[2*(p+q)+(p+q)-1][5]    1.6581  0.7010
## Lag[4*(p+q)+(p+q)-1][9]    2.8901  0.7770
## d.o.f=2
##
## Weighted ARCH LM Tests
## ------------------------------------
##             Statistic Shape Scale P-Value
## ARCH Lag[3]     1.195 0.500 2.000  0.2742
## ARCH Lag[5]     1.738 1.440 1.667  0.5321
## ARCH Lag[7]     2.108 2.315 1.543  0.6944
##
## Nyblom stability test
## ------------------------------------
## Joint Statistic:  1.2618
## Individual Statistics:
## mu     0.05099
## ar1    0.74584
```

```
## omega  0.33059
## alpha1 0.46009
## beta1  0.39215
##
## Asymptotic Critical Values (10% 5% 1%)
## Joint Statistic:          1.28 1.47 1.88
## Individual Statistic:     0.35 0.47 0.75
##
## Sign Bias Test
## -----------------------------------
##                    t-value   prob sig
## Sign Bias           0.6640 0.5067
## Negative Sign Bias  0.4412 0.6591
## Positive Sign Bias  1.3962 0.1628
## Joint Effect        5.3851 0.1457
##
##
## Adjusted Pearson Goodness-of-Fit Test:
## -----------------------------------
##   group statistic p-value(g-1)
## 1    20     40.24     0.003045
## 2    30     49.49     0.010256
## 3    40     54.74     0.048461
## 4    50     72.17     0.017257
##
##
## Elapsed time : 0.2560151
```

```
# To save the GARCH(1,1) output easily to your current folder (assuming you saved a .Rproject fir
# sink("garch11.txt")
# garchfit1

# You will now have the output in txt format in your folder.
```

Let's look at what we can store from the fit:

```
slotNames(garchfit1)
```

[1] "fit" "model"

```
names(garchfit1@fit)
```

[1] "hessian" "cvar" "var"
[4] "sigma" "condH" "z"
[7] "LLH" "log.likelihoods" "residuals"
[10] "coef" "robust.cvar" "scores"
[13] "se.coef" "tval" "matcoef"
[16] "robust.se.coef" "robust.tval" "robust.matcoef" [19] "fitted.values" "convergence" "kappa"
[22] "persistence" "timer" "ipars"
[25] "solver"

```
names(garchfit1@model)
```

[1] "modelinc" "modeldesc" "modeldata" "pars" "start.pars" [6] "fixed.pars" "maxOrder" "pos.matrix" "fmodel" "pidx"
[11] "n.start"

```
# Use it now as follows:
garchfit1@fit$matcoef  # Model coefficients.
```

```
      Estimate  Std. Error    t value     Pr(>|t|)
```

mu 0.01606503 0.018409324 0.8726573 3.828499e-01 ar1 -0.02952743 0.021869263 -1.3501794 1.769584e-01 omega 0.02270981 0.006310642 3.5986526 3.198702e-04 alpha1 0.08243937 0.011157967 7.3883864 1.485478e-13 beta1 0.89792098 0.013685887 65.6092629 0.000000e+00

```
# But that looks terribly ugly...

#  Include it in your paper as follows:
load_pkg("xtable")
Table <- xtable(garchfit1@fit$matcoef)
print(Table, type = "latex", comment = FALSE)
```

	Estimate	Std. Error	t value	Pr(>|t|)
mu	0.02	0.02	0.87	0.38
ar1	-0.03	0.02	-1.35	0.18
omega	0.02	0.01	3.60	0.00
alpha1	0.08	0.01	7.39	0.00
beta1	0.90	0.01	65.61	0.00

```
# VERY IMPORTANT: add results = 'ais' at tthe top of the code chunk when adding latex commands...
```

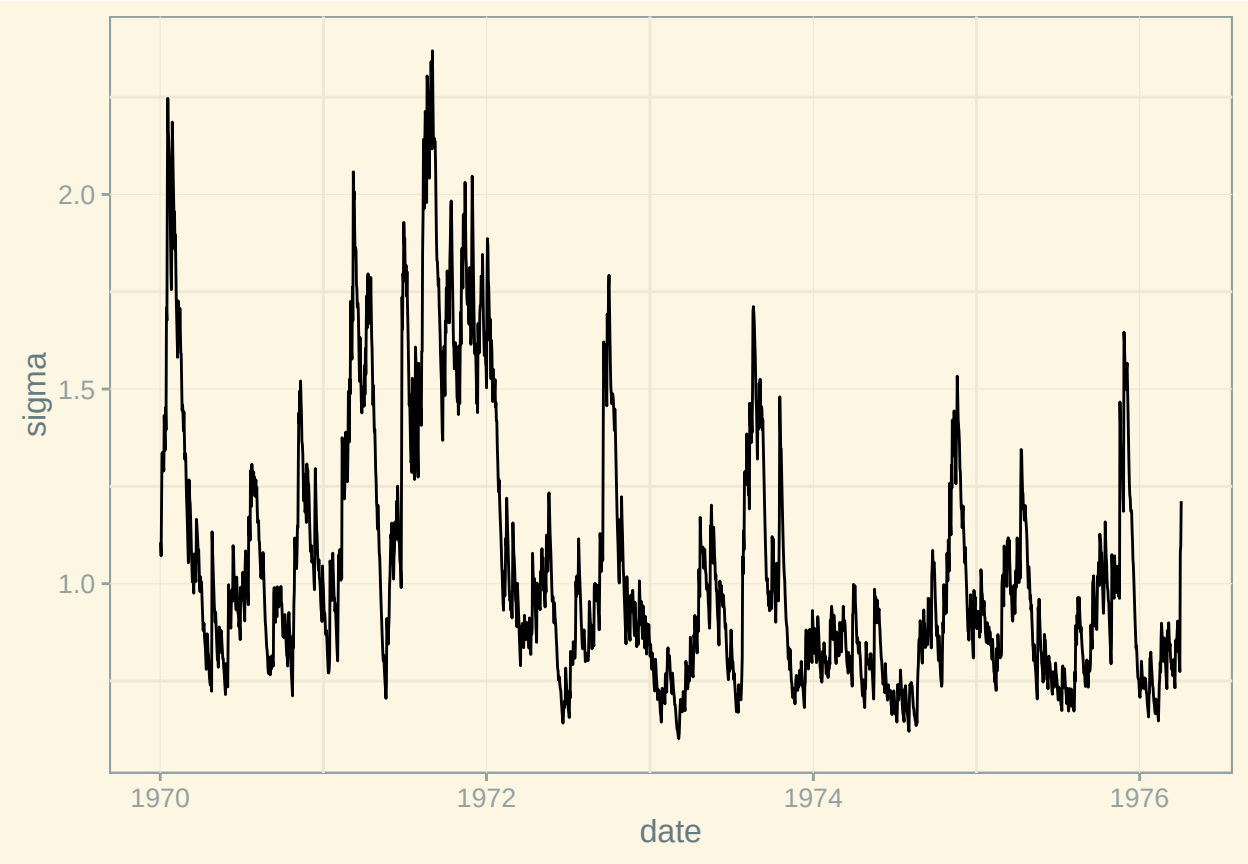
SO that looks pretty neat. But, as you can clearly see, there are very many other parameters and measures that we can call from our GARCH model. For a list of available commands, consult the Vignette.

3.0.1 Conditional variance Plot

```
persistence(garchfit1)
```

```
## [1] 0.9803603
```

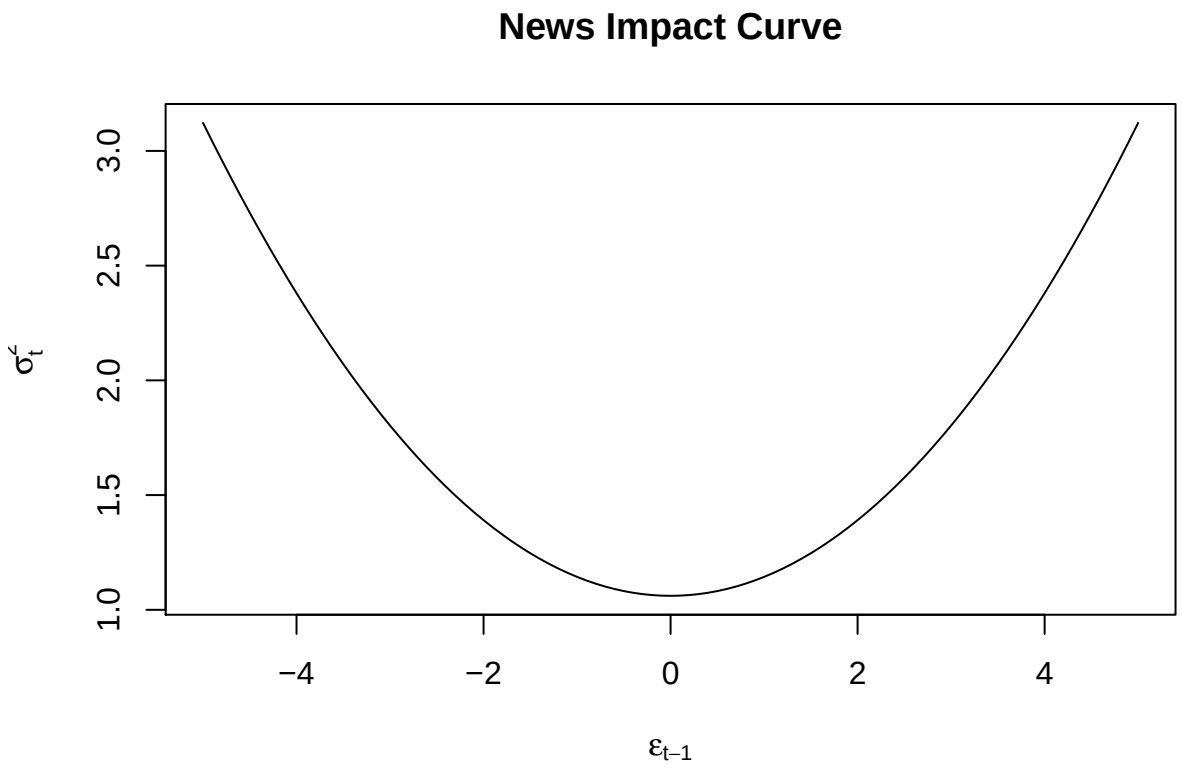
```
# Persistence is alpha + beta, and it is typically very high and close to 1


# To view the conditional variance plot, use:
sigma <- sigma(garchfit1) %>% Xts2Dplyr()
colnames(sigma) <- c("date", "sigma")
sigma <- sigma %>% mutate(date = as.Date(date))
ggplot(sigma) + geom_line(aes(x = date, y = sigma)) + theme_solarized()
```



3.0.2 News Impact Curve

To view the News-Impact curve, define and plot it as:

```
ni <- newsimpact(z = NULL, garchfit1)
plot(ni$zx, ni$zy, ylab=ni$yexpr, xlab=ni$xexpr, type="l", main = "News Impact Curve")
```




```
# Comparing this to other GARCH fits allow us insight into the model's dynamics...
# Test us other models yourself.
```

3.0.3 Volatility model fit

To check the fit of the model

```
infocriteria(garchfit1)
```

```
##
## Akaike       2.850929
```

```
## Bayes        2.863481
## Shibata      2.850920
## Hannan-Quinn 2.855507
```

```
# This can now be compared to other garch models to assess which provides the best fit to the und

# To get the model fitted values:
fitval <-fitted(garchfit1)
```

3.0.4 Residual types that are available:

```
sigma <- sigma(garchfit1)                    # Conditional resids
epsilon <-residuals(garchfit1)               # Ordinary resids
zt <- residuals(garchfit1,standardize=TRUE) # Standardized resids
# Check the class notes on the definitions of epsilon and zt:
ztbyhand = epsilon/sigma

# And they are exactly the same:
all.equal(zt, ztbyhand)
```

```
## [1] TRUE
```

```
# Check the class notes for the definition of these residuals and why they equate...
```

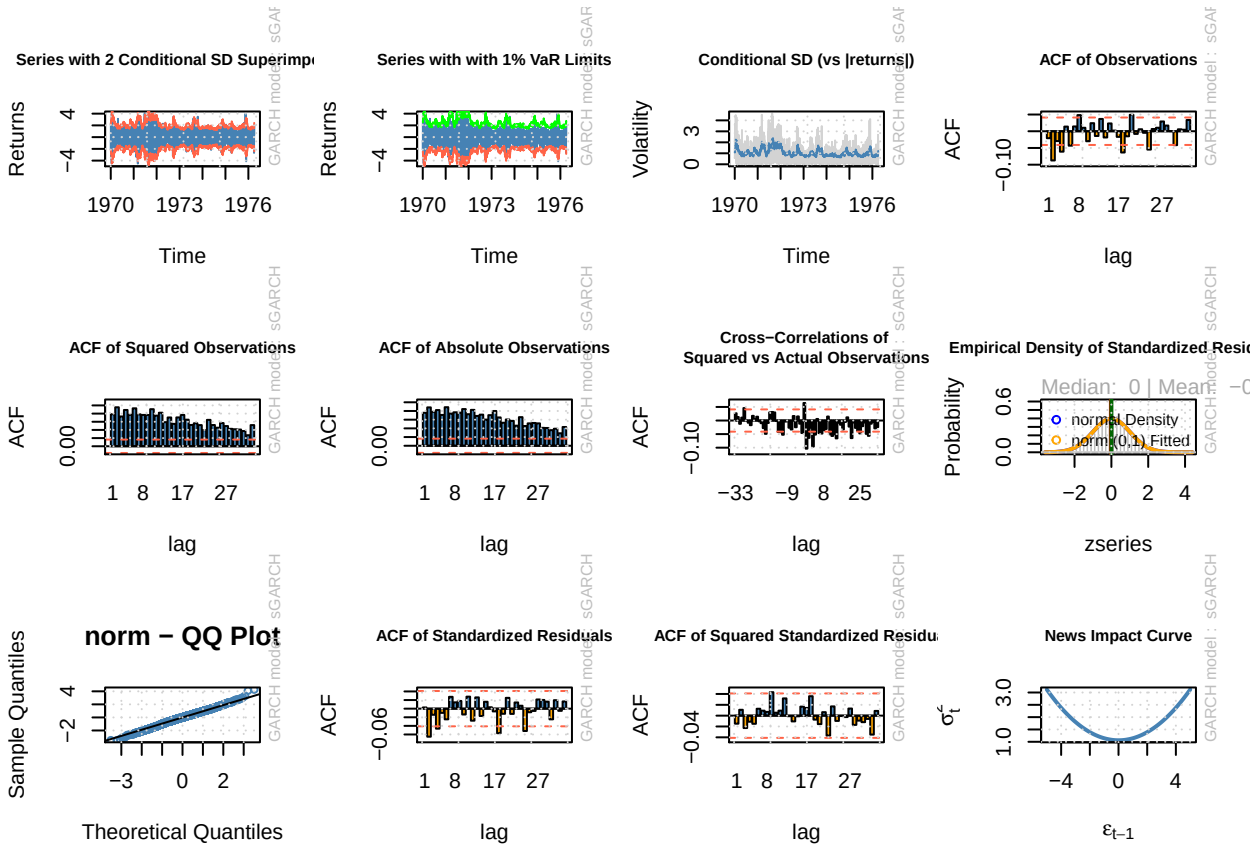
For a full list of commands, see the creator's blog for this summary:

4 Univariate GARCH plots

Rugarch offers a wide variety of plots that we can use to visually get a deeper look at the underlying structure of the second order model behaviour:

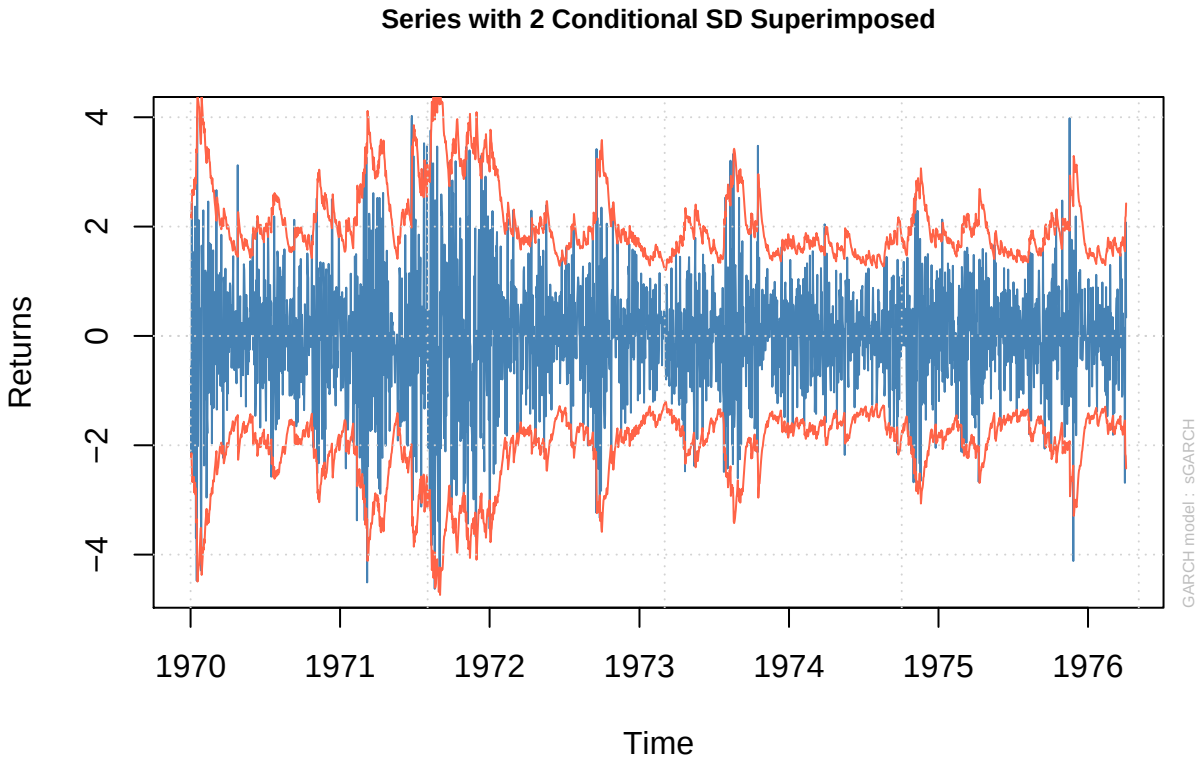
```
# To see all the options, run:
# plot(garchfit1)

# To see all the plots at once, use:
plot(garchfit1, which="all")
```

```
##
## please wait...calculating quantiles...
```

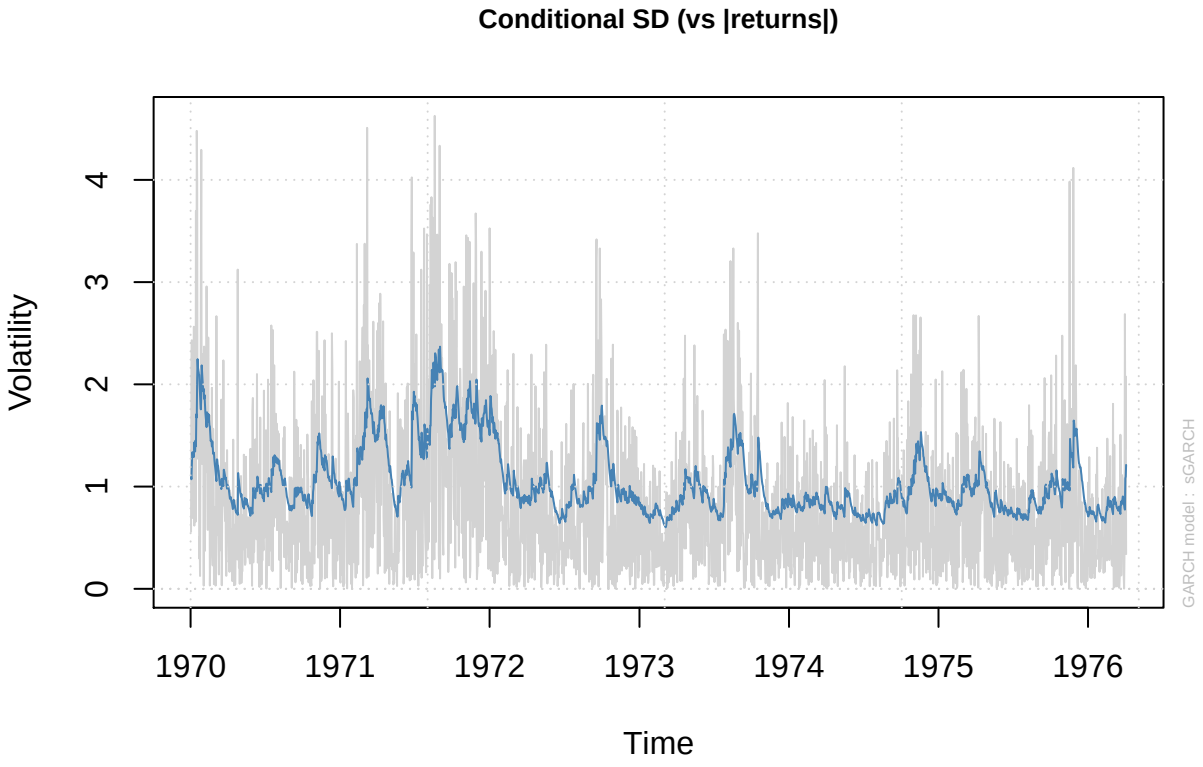


Let's consider a few of these more closely:

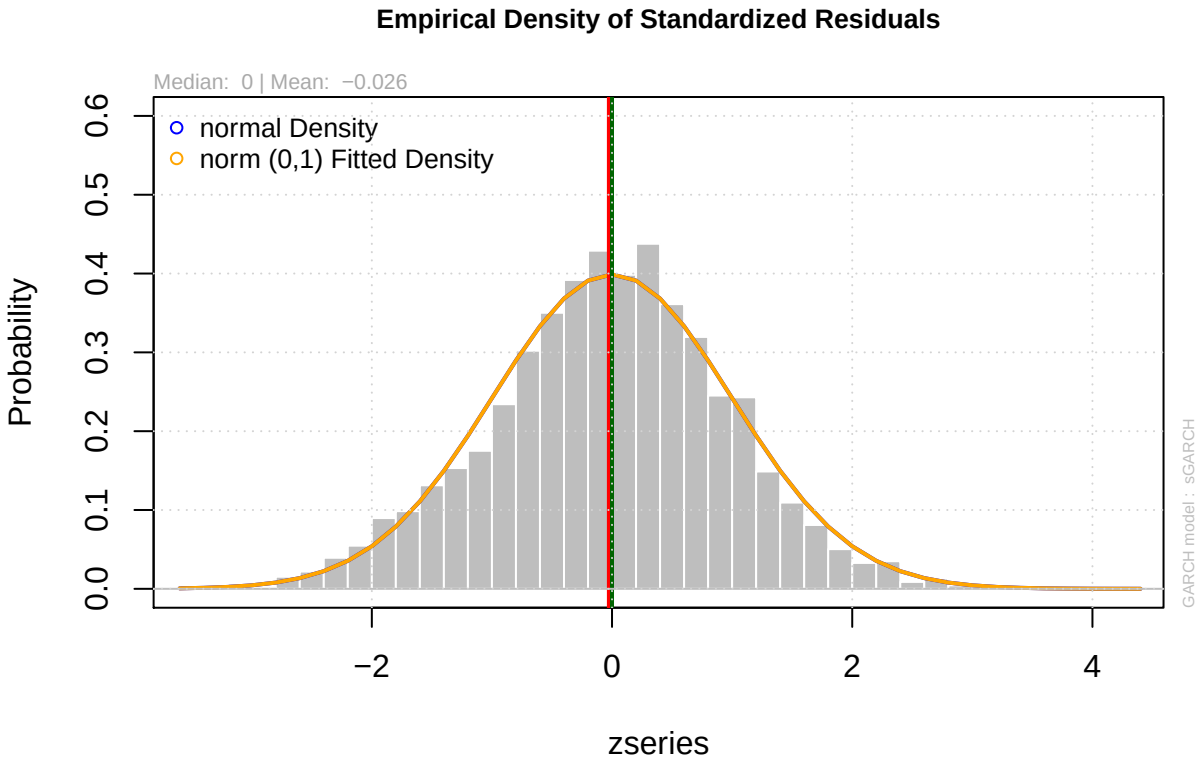
```
plot(garchfit1,which=1)
```



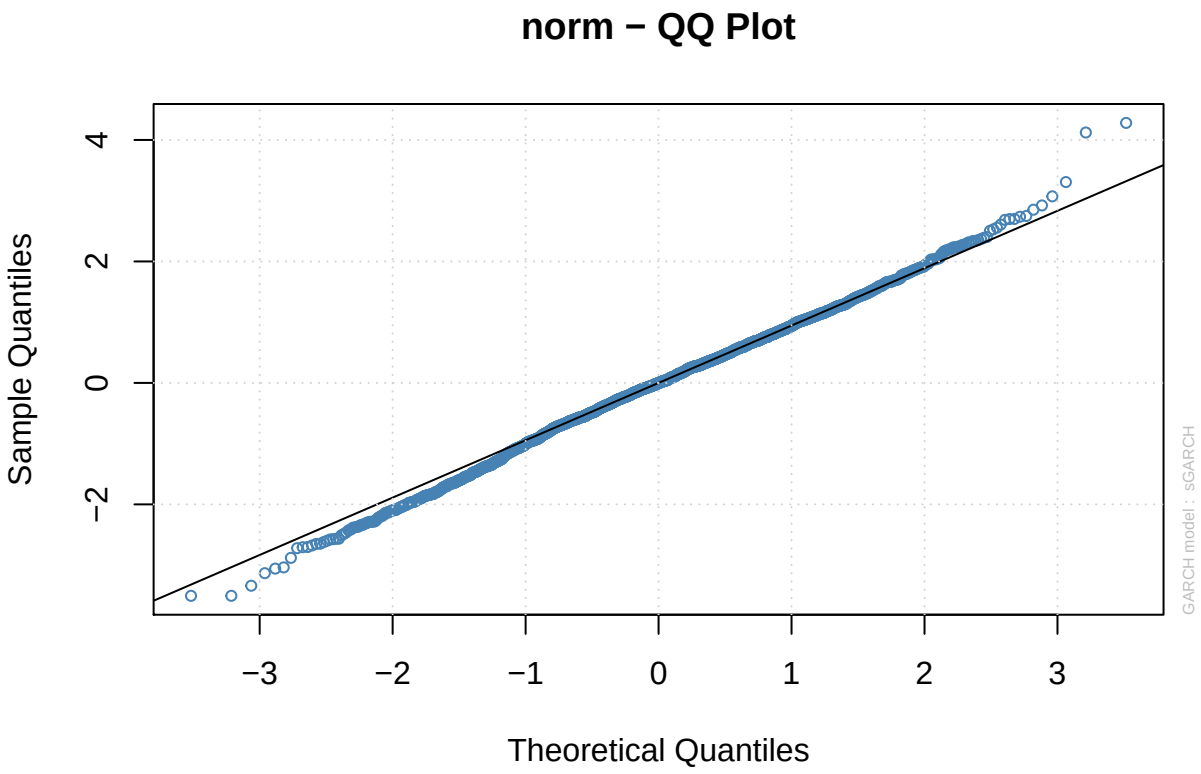

```
plot(garchfit1,which=3)
```



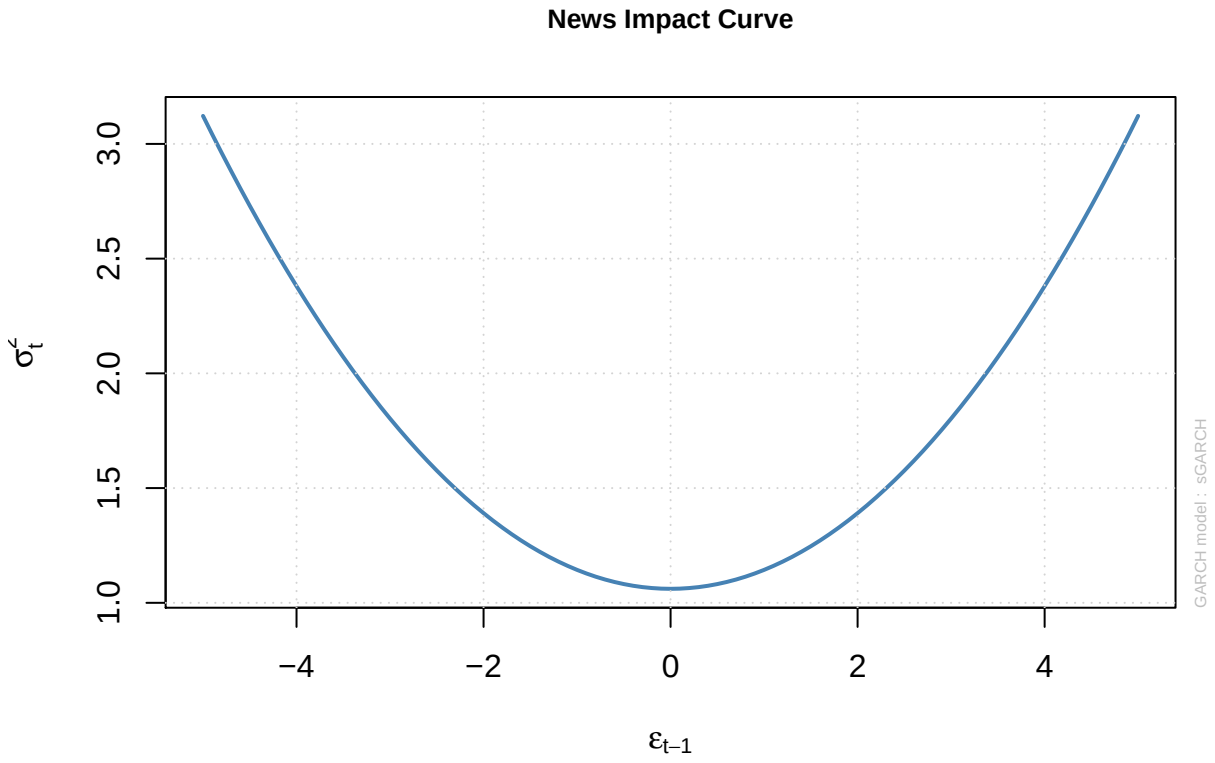

```
plot(garchfit1,which=8)
```




```
plot(garchfit1,which=9)
```




```
plot(garchfit1,which=12)
```



5 Other Univariate GARCH forms

For us to fit other models, or use other distributions, simply amend the code to reflect your choice.

Let's, e.g., fit a *GJR-GARCH* model using a student t distribution:

```
gjrgarch11=ugarchspec(variance.model = list(model = c("sGARCH","gjrGARCH","eGARCH","fGARCH","apAR
mean.model = list(armaOrder = c(1, 0), include.mean = TRUE),
distribution.model = c("norm", "snorm", "std", "sstd", "ged", "sged", "nig", "ghyp", "jsu")[3])
# Now to fit, I use as.matrix and the data - this way the plot functions we will use later will w
garchfit2 = ugarchfit(spec = gjrgarch11, data = as.matrix(porteqw))
```

```
garchfit2@fit$matcoef
```

```
##            Estimate  Std. Error    t value     Pr(>|t|)
## mu     -0.004830967 0.018701422 -0.2583209 7.961593e-01
## ar1    -0.034329844 0.021797912 -1.5749143 1.152762e-01
## omega   0.023854135 0.006189564  3.8539283 1.162376e-04
## alpha1  0.026918213 0.013363497  2.0143091 4.397709e-02
## beta1   0.903581322 0.013937557 64.8306824 0.000000e+00
## gamma1  0.094705390 0.021443905  4.4164246 1.003469e-05
## shape  21.298751732 8.393491104  2.5375319 1.116372e-02
```

Let's now compare the two models (garchfit1 and garchfit2) in terms of infocriteria, likelihood levels and also their NI curves.

First, we see that the sign bias test indicates that there is not a significant leverage effect - implying that negative shocks tend not to cause significantly more persistence than positive shocks. This is quite surprizing. Typically, the test suggests the need for a leverage model, e.g. the GJR or eGARCH. Let's assume it does, and compare the fits between the GARCH(1,1) and the GJR-GARCH(1,1):

```
signbias(garchfit1)
```

```
##                      t-value      prob sig
## Sign Bias          0.6640337 0.5067359
## Negative Sign Bias 0.4411855 0.6591205
## Positive Sign Bias 1.3962092 0.1627875
## Joint Effect       5.3850596 0.1456773
```

```
infocriteria(garchfit1)
```

```
##
## Akaike       2.850929
## Bayes        2.863481
## Shibata      2.850920
## Hannan-Quinn 2.855507
```

```
infocriteria(garchfit2)
```

```
##
## Akaike       2.840066
## Bayes        2.857639
## Shibata      2.840048
## Hannan-Quinn 2.846475
```

```
# AIC is smaller for GJR, indicating it is better at explaining the sample values.
```

Note that the shape parameter from the GJR-fit summary indicates it is significant and positive - implying negative shocks tend to cause higher persistence than positive shocks. The p-value also suggests this effect to be significant.

5.1 Comparing News Impact Curves

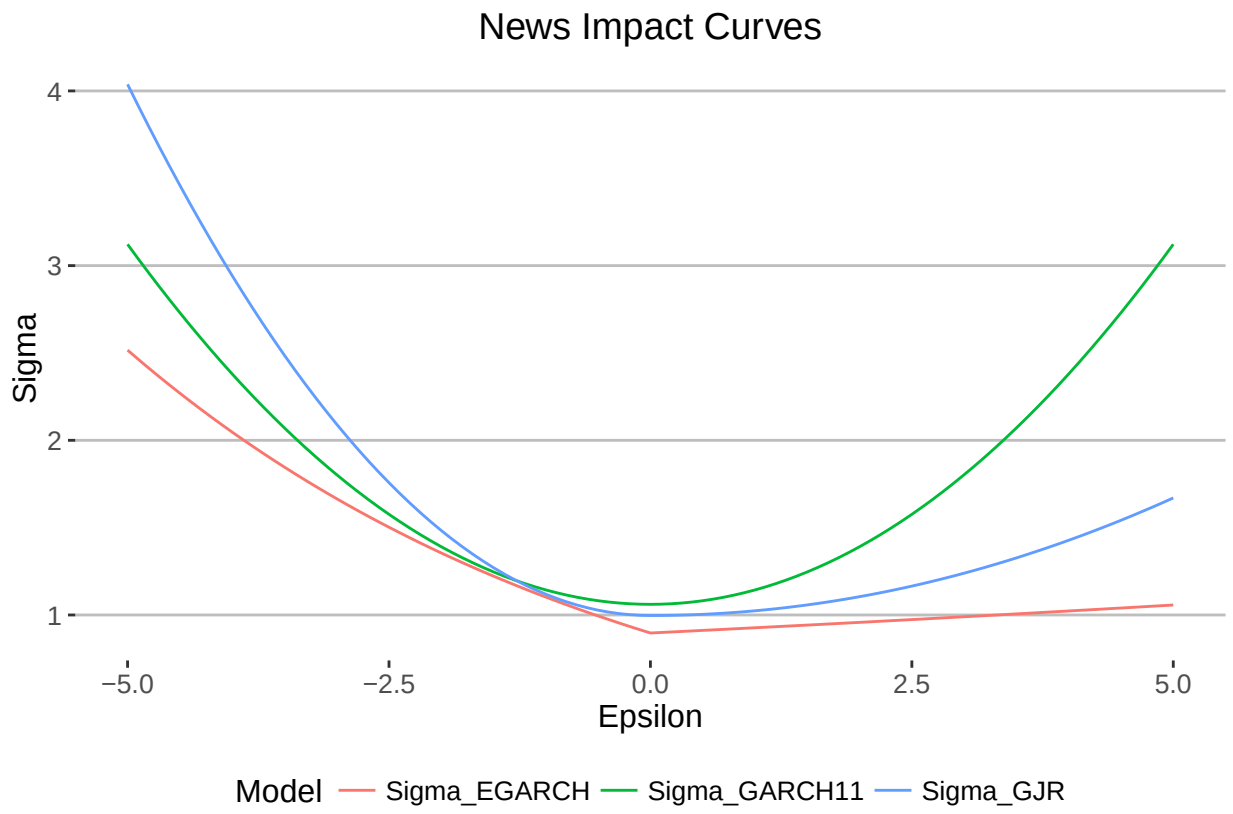
In order to get a visual indication of how residuals impact persistance of the second order GARCH models, we can use NI Curves. Let's fit another widely used model, **eGARCH**, and compare the three models' NI curves:

```
egarch11=ugarchspec(variance.model = list(model = c("sGARCH","gjrGARCH","eGARCH","fGARCH","apARCH
mean.model = list(armaOrder = c(1, 0), include.mean = TRUE),
distribution.model = c("norm", "snorm", "std", "sstd", "ged", "sged", "nig", "ghyp", "jsu")[3])
# Now to fit, I use as.matrix and the data - this way the plot functions we will use later will u
garchfit3 = ugarchfit(spec = egarch11,data=as.matrix(porteqw))
```

Now let's define and then plot the NIS':

```
# Just to be fancy (and to give you a nice illustration
# of how to use functions for doing repetitive things...):

NICurveMaker <- function(Fit, Name) {
  NI <-newsimpact(z = NULL, Fit)
  NI <- cbind(NI$zx, NI$zy)
  colnames(NI) <- c(paste0("Epsilon_", Name), paste0("Sigma_", Name) )
  NI %>% data.frame() %>% tbl_df()
}

NI1 <- NICurveMaker(garchfit1, "GARCH11")
NI2 <- NICurveMaker(garchfit2, "GJR")
NI3 <- NICurveMaker(garchfit3, "EGARCH")

# Note the Epsilon columns are similar, so let's remove them and gather the Sigmas...
NI <-
  cbind(NI1, NI2, NI3) %>% gather(Model, Sigma, starts_with("Sigma")) %>%
  rename("Epsilon" = Epsilon_GARCH11) %>% select(-Epsilon_GJR, -Epsilon_EGARCH )

ggplot(NI) + geom_line(aes(x = Epsilon, y = Sigma, colour = Model)) +
```

```
ggtitle("News Impact Curves") + theme_hc()
```



See if you can wrap all of the above into a function that makes a comparison for two or more model inputs. Create a contingency (if function) that does not bind if only one model has been provided. In this case, it should simply plot one NIC.

5.1.1 Loop for best fitting GARCH model

As another illustration of simple automation, let's create a loop for finding the best fitting GARCH model given our possible selections:

```
models = 1:4
model.list = list()

for (p in models) {
garchfit=ugarchspec(
variance.model = list(model = c("sGARCH","gjrGARCH","eGARCH","apARCH")[p], garchOrder = c(1, 1)),
mean.model = list(armaOrder = c(1, 0), include.mean = TRUE),
distribution.model = c("norm", "snorm", "std", "sstd", "ged", "sged", "nig", "ghyp", "jsu")[1])

garchfit1 = ugarchfit(spec = garchfit,data=as.numeric(porteqw))

model.list[[p]] = garchfit1
}

names(model.list) <- c("sGARCH","gjrGARCH","eGARCH","apARCH")
fit.mat = sapply(model.list, infocriteria)
# Note: sapply can apply a function (infocriteria here) to a list...
rownames(fit.mat) = rownames(infocriteria(model.list[[1]]))
kable(fit.mat)
```

	sGARCH	gjrGARCH	eGARCH	apARCH
Akaike	2.850929	2.842611	2.841420	2.843373
Bayes	2.863481	2.857673	2.856482	2.860945
Shibata	2.850920	2.842597	2.841406	2.843354
Hannan-Quinn	2.855507	2.848104	2.846914	2.849782

The table suggests that the eGARCH model performs the best (lowest AIC).

For the apARCH model, we can interpret the parameters *gamma* as the leverage effect and d as the integration parameter. Let's again consider the APARCH mathematical construct to undertsand these parameters. Consider the APARCH(1,1):

$$\sigma_t^d = \alpha_0 + \sum_{i=1}^{p} \alpha_i \left(|\epsilon_{t-i}| - \gamma_i \epsilon_{t-i}\right)^d + \sum_{j=1}^{q} \beta_1 \sigma_{t-j}^d \tag{5.1}$$

From equation 5.1 we note that γ is the leverage parameter, so that if $\gamma_i > 0$, we have leverage and negative returns increases σ more than positive.

Also, if $d = 2$, we have an ordinary GARCH model with leverage (GJR), while if $d = 1$ we are effectively studying squared σ. See the output, from it we note leverage, and the assumption that $d \neq 2$ cannot be rejected (adding S.E. to parameter includes 2).

```
# Below follows the summary of the apARCH model's output:
model.list[4]
```

```
## $apARCH
##
## *---------------------------------*
## *          GARCH Model Fit        *
## *---------------------------------*
##
## Conditional Variance Dynamics
## -----------------------------------
## GARCH Model  : apARCH(1,1)
## Mean Model   : ARFIMA(1,0,0)
## Distribution : norm
##
```

```
## Optimal Parameters
## ------------------------------------
##         Estimate  Std. Error  t value Pr(>|t|)
## mu     -0.008044    0.019181 -0.41938 0.674938
## ar1    -0.028420    0.021935 -1.29566 0.195093
## omega   0.023386    0.005737  4.07613 0.000046
## alpha1  0.066954    0.011974  5.59138 0.000000
## beta1   0.910644    0.017358 52.46275 0.000000
## gamma1  0.421712    0.220985  1.90833 0.056349
## delta   1.710065    0.533655  3.20444 0.001353
##
## Robust Standard Errors:
##         Estimate  Std. Error  t value Pr(>|t|)
## mu     -0.008044    0.019142 -0.42023 0.674319
## ar1    -0.028420    0.019826 -1.43345 0.151728
## omega   0.023386    0.006420  3.64264 0.000270
## alpha1  0.066954    0.013544  4.94325 0.000001
## beta1   0.910644    0.033300 27.34694 0.000000
## gamma1  0.421712    0.488131  0.86393 0.387625
## delta   1.710065    1.143590  1.49535 0.134824
##
## LogLikelihood : -3240.131
##
## Information Criteria
## ------------------------------------
##
## Akaike       2.8434
## Bayes        2.8609
```

```
## Shibata      2.8434
## Hannan-Quinn 2.8498
##
## Weighted Ljung-Box Test on Standardized Residuals
## ------------------------------------
##                         statistic   p-value
## Lag[1]                    0.02056 0.8859771
## Lag[2*(p+q)+(p+q)-1][2]   5.21602 0.0002633
## Lag[4*(p+q)+(p+q)-1][5]  10.40880 0.0007277
## d.o.f=1
## H0 : No serial correlation
##
## Weighted Ljung-Box Test on Standardized Squared Residuals
## ------------------------------------
##                         statistic p-value
## Lag[1]                     0.6307  0.4271
## Lag[2*(p+q)+(p+q)-1][5]    2.3249  0.5439
## Lag[4*(p+q)+(p+q)-1][9]    3.4767  0.6784
## d.o.f=2
##
## Weighted ARCH LM Tests
## ------------------------------------
##             Statistic Shape Scale P-Value
## ARCH Lag[3]     1.320 0.500 2.000  0.2506
## ARCH Lag[5]     1.593 1.440 1.667  0.5682
## ARCH Lag[7]     1.930 2.315 1.543  0.7321
##
## Nyblom stability test
```

```
## -------------------------------------
## Joint Statistic:  2.2234
## Individual Statistics:
## mu     0.05473
## ar1    0.64415
## omega  0.48876
## alpha1 0.51089
## beta1  0.49627
## gamma1 0.07230
## delta  0.67370
##
## Asymptotic Critical Values (10% 5% 1%)
## Joint Statistic:          1.69 1.9 2.35
## Individual Statistic:     0.35 0.47 0.75
##
## Sign Bias Test
## -------------------------------------
##                    t-value   prob sig
## Sign Bias           0.9996 0.3176
## Negative Sign Bias  1.2050 0.2283
## Positive Sign Bias  0.4982 0.6184
## Joint Effect        2.6562 0.4477
##
##
## Adjusted Pearson Goodness-of-Fit Test:
## -------------------------------------
##   group statistic p-value(g-1)
## 1    20     45.04    0.0006773
```

```
## 2    30      52.19    0.0051901
## 3    40      61.74    0.0116250
## 4    50      69.24    0.0299629
##
##
## Elapsed time : 1.029059
```

Let's now look at ways that we can use the second order persistence model to forecast volatility into the future, and also compare in-sample forecasting performance of GARCH models.

6 Forecasting using GARCH

Let's use our GARCH models to forecast volatility. This is particularly important for forward estimating expected volatility for options pricing, or for predicting forward VaR estimates. *rugarch* offers a robust method of doing so. Let's compare the 10 period ahead volatility forecast of our three models:

```
load_pkg("xtable")
garchf.1 <- ugarchforecast(garchfit1,n.ahead=10)
garchf.2 <- ugarchforecast(garchfit2,n.ahead=10)
garchf.3 <- ugarchforecast(garchfit3,n.ahead=10)
slotNames(garchf.1)
```

[1] "forecast" "model"

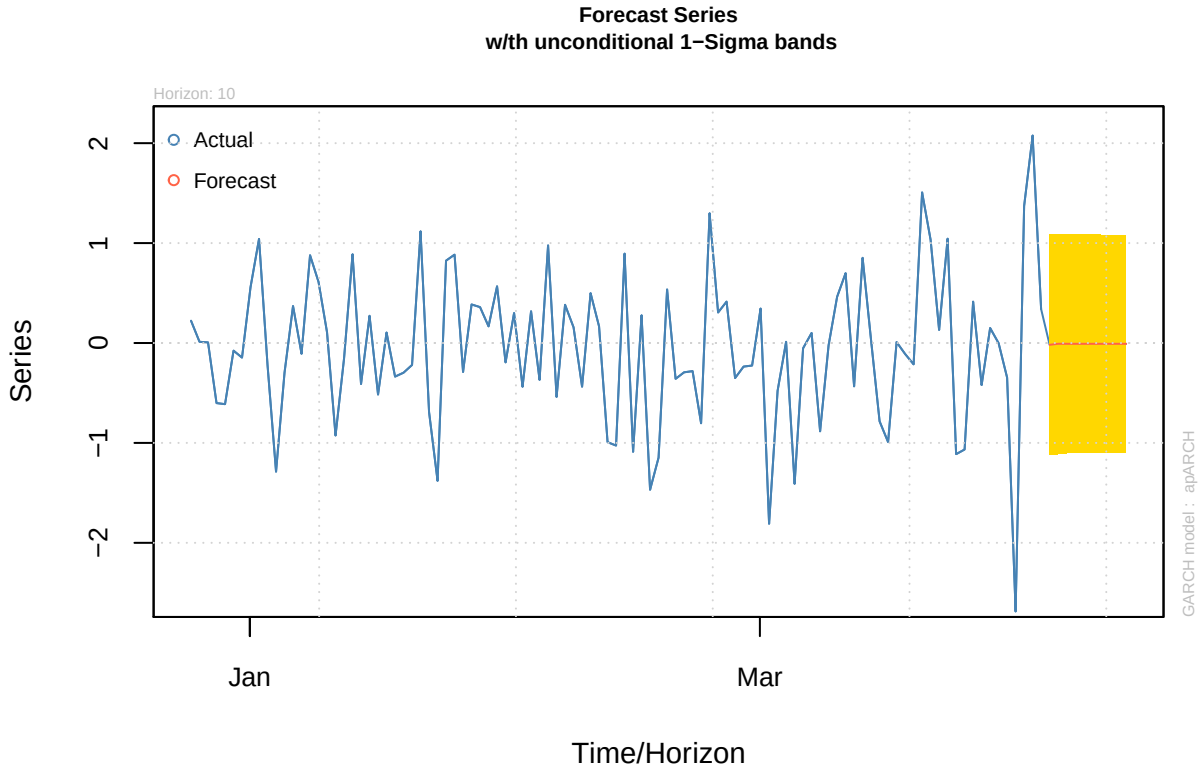
```
names(garchf.1@forecast)
```

[1] "n.ahead" "N" "n.start" "n.roll" "sigmaFor" "seriesFor"

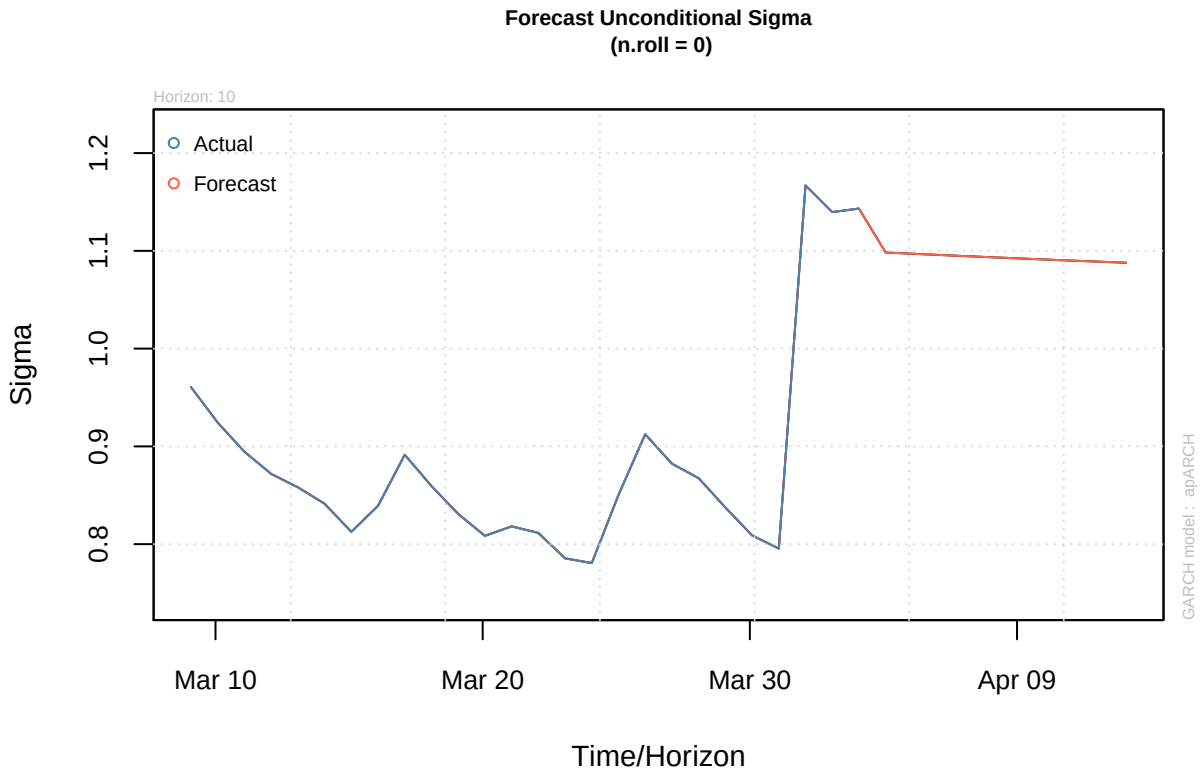
```
names(garchf.1@model)
```

[1] “modelinc” “modeldesc” “modeldata” “pars” “start.pars” [6] “fixed.pars” “maxOrder” “pos.matrix” “fmodel” “pidx”
[11] “n.start”

```
# As these models are again S4 classes with their own given structures,
# let's use the built-in plotting functionalities:
plot(garchf.1,which=1)
```



```
plot(garchf.1,which=3)
```




```
# Let's create a table of forecast values:
# First: Save the forecast sigmas:
f1 <-as.data.frame(sigma(garchf.1))
f2 <-as.data.frame(sigma(garchf.2))
f3 <-as.data.frame(sigma(garchf.3))

# Save The fitted forecasts:
series1 <-fitted(garchf.1)
series2 <-fitted(garchf.2)
```

```
series3 <-fitted(garchf.3)


sigmaf <- cbind(f1,f2,f3)
fitf <- cbind(series1,series2,series3)
vol <-sigmaf^2
colnames(vol) = c("garch","gjrgarch","egarch")


tv = xtable(vol)
# For your Latex paper format:
  # print(tv, type = "latex", file = "forc.tex")


# in HTML:
kable(vol)
```

	garch	gjrgarch	egarch
T+1	1.206321	1.300123	1.023954
T+2	1.203541	1.295182	1.023151
T+3	1.200823	1.290351	1.022364
T+4	1.198165	1.285627	1.021594
T+5	1.195566	1.281007	1.020840
T+6	1.193024	1.276490	1.020102
T+7	1.190540	1.272072	1.019379
T+8	1.188110	1.267753	1.018671
T+9	1.185734	1.263529	1.017978
T+10	1.183411	1.259399	1.017300

6.1 Forward VaR estimating

Now we know that we can use the values in the above table of vol forecasts to forward estimate the VaR's we fitted in the previous practical:

```
# as series1 is the estimate of forward mean. and f1 sigma fcast from garch11, we can use:
var95 <- series1 + f1*qnorm(0.05)

# Hence we expect at 95% surety our EW portfolio's return
# not to dip below 1.9% for each day t to t+10.
```

To now compute the 10 day VaR forecast (and estimate the 5% most extreme downside for the next 10 days for the equal weighted portfolio) we have to sum the forecast sigmas:

```
sigma10d <- sqrt(sum(f1^2))
sigma10d
```

```
## [1] 3.456188
```

```
var9510d <-10*series1[1] + sigma10d*qnorm(0.05)
var9510d
```

```
## [1] -5.864117
```

```
# Thus we expect the returns of the portfolio to exceed -5.9% at 95%
```

7 Rolling Forecasts

In order to backtest the accuracy of a model in terms of its ability to provide accurate forecasts, we need to re-estimate the model each period (or every few periods, defined by the modeller) to capture the

impact of data changes on parameter fits. *rugarch* offers a procedure, *ugarchroll* which does just that - it allows for the generation of 1-step ahead forecasts and the periodic re-estimation of model parameters. We can either choose a moving window (with fixed window size) or an expanding window (where the first period remains fixed and the sample used for parameter estimation grows each period).

It then allows an incredibly useful means of assessing the ability of the chosen GARCH model to predict accurate VaR estimates. It now gives us the expected and actual exceedences of the VaR estimates - and calculate the Kupiec unconditional coverage ratio Kupiec and the Christoffersen conditional coverage ratios (Christoffersen, Hahn, and Inoue 2001) which can be used to test the H_0 of correct exceedence levels. We did so for SA economic sectors and provide detailed account of the measures in Katzke and Garbers (2015).

```
cl = makePSOCKcluster(10)
spec = ugarchspec(variance.model = list(model = 'gjrGARCH', garchOrder = c(1, 1)),
                  mean.model = list(armaOrder = c(1, 0), include.mean = TRUE),
                  distribution ='norm')
# Thus the model spec is a ARIMA(1,1,0)-GJRGARCH(1,1), with normal distribution

roll = ugarchroll(spec,
                  porteqw,
                  forecast.length = 1000,
                  refit.every = 50,
                  refit.window = 'moving',
                  window.size = 1200,
                  calculate.VaR = TRUE,
                  VaR.alpha = c(0.01, 0.05),
                  keep.coef = TRUE,
                  cluster = cl)
```

```
# For this, only 1-step ahead can be done automatically.
show(roll)
```

```
##
## *-----------------------------------*
## *              GARCH Roll           *
## *-----------------------------------*
## No.Refits        : 20
## Refit Horizon    : 50
## No.Forecasts : 1000
## GARCH Model      : gjrGARCH(1,1)
## Distribution : norm
##
## Forecast Density:
##                  Mu  Sigma Skew Shape Shape(GIG) Realized
## 2011-07-12 -0.0178 0.7383    0     0          0   0.3472
## 2011-07-13 -0.0051 0.7227    0     0          0   0.2067
## 2011-07-14 -0.0088 0.7058    0     0          0  -0.0503
## 2011-07-15 -0.0156 0.6889    0     0          0  -0.0284
## 2011-07-18 -0.0150 0.6732    0     0          0  -0.9983
## 2011-07-19 -0.0406 0.7377    0     0          0  -0.2840
##
## ........................
##                  Mu  Sigma Skew Shape Shape(GIG) Realized
## 2015-07-03 -0.0082 0.8052    0     0          0  -0.0021
## 2015-07-06  0.0028 0.7828    0     0          0  -0.3461
## 2015-07-07  0.0276 0.7711    0     0          0  -2.6869
## 2015-07-08  0.1968 1.1877    0     0          0   1.3706
```

```
## 2015-07-09 -0.0965 1.1434    0     0          0   2.0765
## 2015-07-10 -0.1475 1.1093    0     0          0   0.3394
##
## Elapsed: 15.93491 secs
```

```
report(roll, type="VaR", VaR.alpha = 0.01, conf.level = 0.95)
```

```
## VaR Backtest Report
## ===========================================
## Model:               gjrGARCH-norm
## Backtest Length: 1000
## Data:
##
## ===========================================
## alpha:               1%
## Expected Exceed: 10
## Actual VaR Exceed:   15
## Actual %:            1.5%
##
## Unconditional Coverage (Kupiec)
## Null-Hypothesis: Correct Exceedances
## LR.uc Statistic: 2.189
## LR.uc Critical:      3.841
## LR.uc p-value:       0.139
## Reject Null:     NO
##
## Conditional Coverage (Christoffersen)
## Null-Hypothesis: Correct Exceedances and
```

```
##                  Independence of Failures
## LR.cc Statistic: NaN
## LR.cc Critical:      5.991
## LR.cc p-value:       NaN
## Reject Null:     NA
```

```
report(roll, type="fpm")
```

```
##
## GARCH Roll Mean Forecast Performance Measures
## ---------------------------------------------
## Model        : gjrGARCH
## No.Refits    : 20
## No.Forecasts: 1000
##
##      Stats
## MSE 0.8189
## MAE 0.6890
## DAC 0.5070
```

8 Adding regressors

To add regressors to the mean and variance equations, use the following in your model specifications:

- Garch-in-Mean: **archm** (To see if volatility is priced into the mean equation - implying a risk premium...)[1]
- *mxreg1* for the regressor in the mean equation (include in mean.model)

[1] In the ugarchspec, include archm=TRUE as: mean.model = list(armaOrder = c(1, 0), include.mean = TRUE, archm=TRUE)

  - Implying that X_t has a mean level effect on the series.

- *vxreg1* for the regressor in the variance equation (include in variance.model).

  - Implying that X_t has an effect on the volatility of the series.

9 References

Bauwens, Luc, Sébastien Laurent, and Jeroen VK Rombouts. 2006. "Multivariate GARCH Models: A Survey." *Journal of Applied Econometrics* 21 (1). Wiley Online Library: 79–109.

Christoffersen, Peter, Jinyong Hahn, and Atsushi Inoue. 2001. "Testing and Comparing Value-at-Risk Measures." *Journal of Empirical Finance* 8 (3). Elsevier: 325–42.

Hentschel, Ludger. 1995. "All in the Family Nesting Symmetric and Asymmetric Garch Models." *Journal of Financial Economics* 39 (1). Elsevier: 71–104.

Katzke, Nico, and Chris Garbers. 2015. "Do Long Memory and Asymmetries Matter When Assessing Downside Return Risk?" *Stellenosch Working Paper Series*.

Ruppert, David. 2011. *Statistics and Data Analysis for Financial Engineering*. Springer.

Silvennoinen, Annastiina, and Timo Teräsvirta. 2009. "Multivariate GARCH Models." In *Handbook of Financial Time Series*, 201–29. Springer.

Tsay, Ruey S. 2012. *An Introduction to Analysis of Financial Data with R*. Wiley Publishing.

———. 2014. *An Introduction to Analysis of Financial Data with R*. John Wiley & Sons.