# Financial Econometrics Practical

Practical 7: Multi-variate Volatility Modelling

NF Katzke


## Table of Contents

**1 Introduction** **2**

**2 Import data** **3**

2.1 Calculating Returns and Cleaning . . . . . 3

**3 MV Conditional Heteroskedasticity tests** **4**

3.1 MV modelling: conceptually . . . . . 5

**4 EWMA Models** **6**

**5 BEKK Models** **10**

**6 DCC Models** **11**

6.1 DCC: flexible Univariate specs . . . . . 21

**7 Go-GARCH: Orthogonalizing sources of volatility** **28**

7.0.1 Intuitively... . . . . 29

7.0.2 Why should GO-GARCH be used? . . . . . 31

7.1 Fitting the Go-GARCH . . . . . 31

7.2 Time-varying third and fourth moments . . . . . 36

**8 Other interesting models** **37**

**References** **37**

# 1 Introduction

In this practical, we will be looking at some applications of GARCH models mapped onto the multivariate plane. Particularly, we will be considering applications to the parsimonious estimation of conditional correlation estimates that can be estimated for a matrix of n-series.

Comovement is of particular interest to investors and investment managers, as it provides insight into diversification potential. The higher the correlation between two assets, the lower the diversification. In this practical, we will outline some of the most widely used techniques to calculate this measure. We will be looking at the:

- DCC-GARCH model
- ADCC-GARCH model
- GO-GARCH model

We can then use the time-varying correlation estimates so defined to provide insight into the underlying comovement structures of a portfolio of assets. We can also use our estimates to forecast the expected future comovement between two assets.

We will be using mainly two packages: **MTS** from the Tsay (2014) textbook (where the MV volatility models are discussed in chapter 7), and Ghalanos (2014) **rmgarch**, which is the multivariate extension of the **rugarch** package. **MTS** provides us with excellent goodness-of-fit estimates, while **rmgarch** provides unmatched ease of estimation and forecasting and plotting functionality.

See the brilliant summary papers by Bauwens, Laurent, and Rombouts (2006) and Silvennoinen and Teräsvirta (2009), as well as the canonical paper on DCC modelling by Engle (2002). Various other applications of these MV-GARCH techniques have been written (c.f. Horvath and Poldauf (2012), Christiansen (2007), Johansson (2008), etc.).

## 2 Import data

TOday we will be using the same data as in the previous practical to keep things simple. I will provide you with much more in-depth data for the projects in due course. For now, load and calculate and clean the returns similar to last practical:

```
# Note: we will install the MTS package today
library(rmsfuns)
load_pkg("MTS")
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") )
```

### 2.1 Calculating Returns and Cleaning

```
# 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)


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


  # And clean it using Boudt's technique:
  rtn <- Return.clean(rtn, method = c("none", "boudt", "geltner")[2], alpha = 0.01)
```

So now we have our return data, rtn, with which we will conduct our MV-Volatility modelling.

## 3 MV Conditional Heteroskedasticity tests

Tsay (2014) summarizes how to fit MV Portmanteau tests in chapter 7.1, which motivates the use of heteroskedasticity models such as GARCH. See e.g. the **MarchTest** from **MTS** package (Tsay 2014, 401:407)

```
load_pkg("MTS")
MarchTest(rtn)
```

```
## Q(m) of squared series(LM test):
## Test statistic:  2401.114  p-value:  0
## Rank-based Test:
## Test statistic:  1797.01  p-value:  0
## Q_k(m) of squared series:
## Test statistic:  3545.727  p-value:  0
## Robust Test(5%) :  2718.207  p-value:  0
```

The MARCH test indicates that all the MV portmanteau tests reject the null of no conditional heteroskedasticity, motivating our use of MVGARCH models.

Now there are very many types of multi-variate volatility models that we can fit. Here follows a few that we can use to control for the remaining serial heteroskedasticity in our series- as well as uncover the conditional covariance structure to do further analyses on.

### 3.1 MV modelling: conceptually

Generalization of univariate GARCH models to the multivariate domain is a conceptually simple exercise: Given the stochastic process, $x_t, t = 1, 2, ...T$ of financial returns with dimension $N \times 1$ and mean vector $\mu_t$, given the information set $I_{t-1}$:

$$x_t \, | I_{t-1} = \mu_t + \varepsilon_t \tag{3.1}$$

where the residuals of the process are modelled as:

$$\varepsilon_t = H_t^{1/2} z_t, \tag{3.2}$$

$H_t^{1/2}$ above is a $N \times N$ positive definite matrix such that $\mathbf{H_t}$ is the **conditional covariance** matrix of $\mathbf{x_t}$. $\mathbf{z_t}$ is a $N \times 1$ i.i.d. N(0,1) series.

Now several techniques have been proposed that map the $H_t$ matrix into the multivariate plain:

- VECH, BEKK, CCC-GARCH, DCC-GARCH, GOGARCH, etc.
- A large literature has developed that attempts to map $H_t$ into the multivariate space,

  - But MV Volatility models typically suffer from curse of high dimensionality..

- Studying correlation between a large amount of series, implies a potentially very large set of unique bivariate correlation combinations.

  - This requires a parsimonious and effective means of isolating the conditional comovements.

# 4 EWMA Models

The EWMA offers a simplified means of estimating MVGARCH models.

From our earlier theoretical definitions of the residuals,

$$\hat{\alpha_t} = z_t - \hat{\mu_t}$$

, we define the EWMA model as:

$$\hat{\Sigma_t} = \lambda \hat{\Sigma_{t-1}} + (1-\lambda)\hat{\alpha_{t-1}}\hat{\alpha'_{t-1}} \tag{4.1}$$

where $0 < \lambda < 1$ denotes the decaying rate, or the persistence parameter (Tsay 2014, p414). In practice, the value of $\lambda$ is typically fixed at around 0.96, and not estimated. This follows simply to avoid complicated and long estimation times to reach a value that is anyway quite similar. This makes the model extremely parsimonious and easy to estimate. The problem, though, is that this fit is assumed to hold accross many time-series (something which has been shown in the literature to be an inaccurate assumption).

To fit the EWMA on our data, let's use the MTS package.

Before we fit the data, however, let's follow Tsay (2014) in fitting a VAR(1) mean model to control for remaining serial auto-persistence in the first moment.

After controlling for MV first order persistence, let's use a EWMA first to control for MV second moment persistence (GARCH effects).

Note: for the EWMA estimation, a negative spec for lambda implies estimating lambda, while a positive value sets it. Thus, we could, e.g., to save estimation time simply set lambda to 0.96.

```
rtnv <- VAR(rtn,1)
```

```
## Constant term:
## Estimates:  0.007610328 -0.001661725 0.00524227 0.0007124152 0.01856287 -0.00354616 -0.0029431
## Std.Error:  0.01537472 0.03521303 0.0410719 0.02130332 0.04214582 0.03789043 0.03581563
## AR coefficient matrix
## AR( 1 )-matrix
##          [,1]      [,2]     [,3]     [,4]     [,5]     [,6]     [,7]
## [1,] -0.11377  0.000966 -0.00543  0.04406  0.01552 -0.01158  0.01293
## [2,] -0.04488 -0.144599  0.04858  0.00173  0.01050  0.03929 -0.02245
## [3,]  0.04612 -0.013534 -0.05812 -0.07615  0.01939  0.08393 -0.00804
## [4,]  0.09164 -0.019542  0.01155 -0.14903  0.00600  0.00369  0.00986
## [5,] -0.00881 -0.027861  0.22359 -0.01802 -0.32521  0.11356 -0.00178
## [6,]  0.05356  0.004449  0.08037 -0.03883 -0.00406 -0.08234 -0.05972
## [7,] -0.06991  0.013133  0.07312 -0.00936 -0.02255  0.03130 -0.13600
## standard error
##        [,1]   [,2]   [,3]   [,4]   [,5]   [,6]   [,7]
## [1,] 0.0212 0.0113 0.0144 0.0152 0.0134 0.0130 0.0113
## [2,] 0.0486 0.0259 0.0329 0.0349 0.0307 0.0299 0.0258
## [3,] 0.0566 0.0302 0.0384 0.0407 0.0358 0.0348 0.0301
## [4,] 0.0294 0.0157 0.0199 0.0211 0.0186 0.0181 0.0156
## [5,] 0.0581 0.0310 0.0394 0.0417 0.0368 0.0357 0.0309
## [6,] 0.0522 0.0279 0.0354 0.0375 0.0331 0.0321 0.0277
```

```
## [7,] 0.0494 0.0263 0.0335 0.0355 0.0313 0.0304 0.0262
##
## Residuals cov-mtx:
##             ABSP        BVT        FSR        NBKP         RMH        SBK
## ABSP 0.53763760 0.02758469 0.02022088  0.14793899  0.03878450 0.06993342
## BVT  0.02758469 2.82021456 1.68259542  0.07194826  1.74244553 1.55388751
## FSR  0.02022088 1.68259542 3.83676190  0.01316811  3.24086285 2.55673688
## NBKP 0.14793899 0.07194826 0.01316811  1.03221478 -0.01235463 0.04461899
## RMH  0.03878450 1.74244553 3.24086285 -0.01235463  4.04002870 2.55226218
## SBK  0.06993342 1.55388751 2.55673688  0.04461899  2.55226218 3.26538578
## SLM  0.05829976 1.39897201 1.74406741  0.06673161  1.82889698 1.59913274
##              SLM
## ABSP 0.05829976
## BVT  1.39897201
## FSR  1.74406741
## NBKP 0.06673161
## RMH  1.82889698
## SBK  1.59913274
## SLM  2.91756520
##
## det(SSE) =  13.47983
## AIC =  2.644101
## BIC =  2.767109
## HQ  =  2.688965
```

```
et <- rtnv$residuals  ## Save the VAR(1) model's residuals.
# Now do a GARCH test on remaining series heteroskedasticity:
MarchTest(et)
```

```
## Q(m) of squared series(LM test):
## Test statistic:  2252.919  p-value:  0
## Rank-based Test:
## Test statistic:  1843.421  p-value:  0
## Q_k(m) of squared series:
## Test statistic:  3416.142  p-value:  0
## Robust Test(5%) :  2503.156  p-value:  0
```

```
#The MarchTest shows the presence of MV heteroskedasticity, and motivates
# the use of EWMA (or any other MV vol model).

# Let's fit the EWMA:
EWMA <- EWMAvol(et,lambda = 0.96)

# The available output:
names(EWMA)
```

```
## [1] "Sigma.t" "return"  "lambda"
```

```
# gives the  conditional variance estimates:
Sigma.t <- EWMA$Sigma.t

# The mean returns series (which is the same as et above)
all.equal(EWMA$return, et)
```

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

```
# And the estimate of lambda.


# Model checking could be done using the built-in MCHdiag from Tsay (2014):
modcheck <- MCHdiag(et,Sigma.t)
```

```
## Test results:
## Q(m) of et:
## Test and p-value:  70.00076 4.432288e-11
## Rank-based test:
## Test and p-value:  188.2411 0
## Qk(m) of epsilon_t:
## Test and p-value:  662.4126 3.031273e-07
## Robust Qk(m):
## Test and p-value:  736.5485 3.337441e-12
```

Note the MARCH test (section 7.1 in Tsay (2014)) p-values are all close to zero, indicating the standardized residuals (zt) of EWMA still have remaining conditional heteroskedasticity.

Let's progress to a more generalized estimation model: BEKK

# 5 BEKK Models

BThe BEKK GARCH family of models (Engle and Kroner (1995)), as discussed in class, allow us to consider spill-over effects and study the impact of one variable on another.

We won't spend too much time on this technique, as it typically suffers from very high dimensionality. It is mostly used for studying spill-overs in a constrained framework.

See Tsay (2014) p.417 for code to run this - health warning: it will take long to run on this data set...

# 6 DCC Models

DCC models offer a simple and more parsimonious means of doing MV-vol modelling. In particular, it relaxes the constraint of a fixed correlation structure (assumed by the CCC model), to allow for estimates of time-varying correlation (see Engle (2002) for the original paper).

The DCC model can be defined as:

$$H_t = D_t.R_t.D_t. \tag{6.1}$$

Equation 6.1 splits the varcovar matrix into identical diagonal matrices and an estimate of the time-varying correlation. Estimating $R_T$ requires it to be inverted at each estimated period, and thus a proxy equation is used (Engle (2002)):

$$\begin{aligned} Q_{ij,t} &= \bar{Q} + a\left(z_{t-1}z'_{t-1} - \bar{Q}\right) + b\left(Q_{ij,t-1} - \bar{Q}\right) \\ &= (1-a-b)\bar{Q} + az_{t-1}z'_{t-1} + b.Q_{ij,t-1} \end{aligned} \tag{6.2}$$

- Note the above equation is similar in form to a GARCH(1,1) process, with non-negative scalars $a$ and $b$, and with:
  - $Q_{ij,t}$ the unconditional (sample) variance estimate between series $i$ and $j$,
  - $\bar{Q}$ the unconditional matrix of standardized residuals from each univariate pair estimate.}

We next use eq. 6.2 to estimate $R_t$ as:

$$R_t = diag(Q_t)^{-1/2}Q_t.diag(Q_t)^{-1/2}. \tag{6.3}$$

Which has bivariate elements:

$$R_t = \rho_{ij,t} = \frac{q_{i,j,t}}{\sqrt{q_{ii,t}.q_{jj,t}}} \tag{6.4}$$

The resulting DCC model is then formulated as:

$$\begin{aligned}
\varepsilon_t &\sim N(0, D_t.R_t.D_t) \\
D_t^2 &\sim \text{Univariate GARCH(1,1) processes } \forall \text{ (i,j), i} \neq \text{j} \\
z_t &= D_t^{-1}.\varepsilon_t \\
Q_t &= \bar{Q}(1-a-b) + a(z_t' z_t) + b(Q_{t-1}) \\
R_t &= Diag(Q_t^{-1}).Q_t.Diag({Q_t}^{-1})
\end{aligned} \tag{6.5}$$

Next, the ADCC allows for leverage effects in the above (think GJR GARCH..)

$$\begin{aligned}
\varepsilon_t &\sim N(0, D_t.R_t.D_t) \\
D_t^2 &\sim \text{Univariate GARCH(1,1) processes } \forall \text{ (i,j), i} \neq \text{j} \\
z_t &= D_t^{-1}.\varepsilon_t \\
Q_t &= \bar{Q}(1-a-b-G) + a(z_t' z_t) + b(Q_{t-1}) + G' z_t^{-} z_t'^{-} G \\
R_t &= Diag(Q_t^{-1}).Q_t.Diag({Q_t}^{-1})
\end{aligned} \tag{6.6}$$

Now fitting these techniques to our earlier data implies a two-step approach:

- Fitting univariate GARCH models to each of our VAR series' residuals: $\alpha_t = z_t - \mu_t$. The volatility approximation series so estimated, $h_t$, will then be used in step 2.
- These vol series are then standardized

$$\eta_{i,t} = \frac{\hat{\alpha_{i,t}}}{\hat{\sigma_{i,t}}}$$

and used in fitting a DCC model for $\eta_t$.

```
# dccPre fits the univariate GARCH models to each series in our data frame of returns.
# Let's select a VAR order of zero for the mean equation, and simply use the mean of each series.
# The mean equation is thus in our case simply: Rtn_t = mean(R) + et

# Then, for every series, a standard univariate GARCH(1,1) is run - giving us:
# et and sigmat, which is then used to calculate the standardized resids, zt, which is used in DC
DCCPre <- dccPre(rtn/100, include.mean = T, p = 0)
```

```
## Sample mean of the returns:  7.026744e-05 -1.341082e-05 4.293091e-05 1.296492e-05 0.0001402855
## Component:  1
## Estimates:  9e-06 0.130221 0.712201
## se.coef   :  2e-06 0.01996 0.046624
## t-value   :  4.72973 6.524029 15.2754
## Component:  2
## Estimates:  4e-06 0.065071 0.918916
## se.coef   :  1e-06 0.009466 0.012016
## t-value   :  3.282867 6.873843 76.47236
## Component:  3
## Estimates:  7e-06 0.061327 0.91994
## se.coef   :  2e-06 0.010534 0.01409
## t-value   :  3.21803 5.822016 65.28883
```

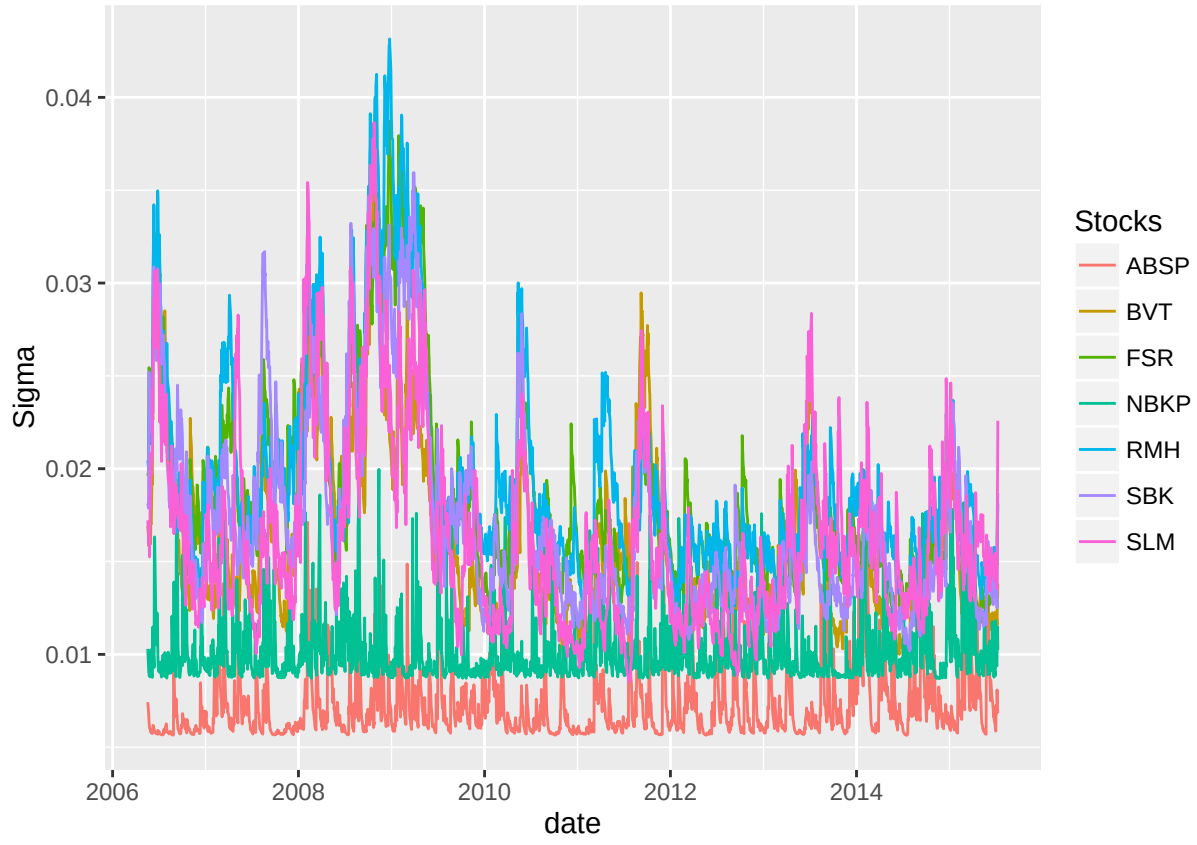
```
## Component:  4
## Estimates:  4.3e-05 0.163184 0.43434
## se.coef   :  1.2e-05 0.03029 0.132008
## t-value   :  3.571244 5.3874 3.290253
## Component:  5
## Estimates:  8e-06 0.067689 0.910828
## se.coef   :  2e-06 0.010387 0.013513
## t-value   :  3.620458 6.516533 67.40482
## Component:  6
## Estimates:  5e-06 0.06914 0.913914
## se.coef   :  1e-06 0.009973 0.012459
## t-value   :  3.457619 6.932742 73.35421
## Component:  7
## Estimates:  7e-06 0.100838 0.875812
## se.coef   :  2e-06 0.015059 0.017988
## t-value   :  3.732725 6.696005 48.68857
```

```
# If you want to fit other univariate garch models for each series, use the fGarch package to do

names(DCCPre)
```

```
## [1] "marVol"  "sresi"   "est"     "se.coef"
```

```
# We now have the estimates of volatility for each series.
# Follow my lead below in changing the output to a usable Xts series for each column in rtn:
Vol <- DCCPre$marVol
colnames(Vol) <- colnames(rtn)
Vol <-
```

```
  data.frame( cbind( date = index(rtn), Vol)) %>% # Add date column which dropped away...
  mutate(date = as.Date(date)) %>%  tbl_df()  # make date column a date column...
TidyVol <- Vol %>% gather(Stocks, Sigma, -date)
ggplot(TidyVol) + geom_line(aes(x = date, y = Sigma, colour = Stocks))
```




```
# From the figure, it is clear that RMH is the most volatile stock...


# ------- Back to DCC:


# After saving now the standardized residuals:
StdRes <- DCCPre$sresi
```

```
# We can now use these sresids to calculate the DCC model.

# BUT FIRST NOTE THIS:
# Now, here follows a CLASSIC example of bad names for package commands.
# If you run the dccFit command, notice that it gives you an error for filter_

# Upon investigation (Cntrl + click on the word dccFit in Rstudio) - you will notice
# the dccFit function uses the command filter. The problem is, dplyr also uses filter.
# The MTS authors should have wrapped the command as stats::filter and not filter, as it
# produces ambiguity between stats::filter and dplyr::filter...
# Try it yourself - Run the following code:
# DCC <- dccFit(StdRes, type="Engle")

# SO... to solve this petty issue, let's detach the tidyr and dplyr packages,
# then run dccFit and then reload tidyr and dplyr...
# (Note this takes a few minutes - go get coffee in the meantime):

detach("package:tidyr", unload=TRUE)
detach("package:dplyr", unload=TRUE)
DCC <- dccFit(StdRes, type="Engle")
```

```
## Estimates:  0.92 0.01664139 8.997275
## st.errors:  0.01825412 0.002799644 0.4624216
## t-values:   50.39957 5.944109 19.45687
```

```
load_pkg(c("tidyr","dplyr"))
```

Finally, we have our DCC model estimated...

So what can we do with it? Note that it is again a S4 class (i.e., not a normal data.frame) - so we need to be guided by the authors what it is that we can gain from it...

From the equations above, we know that we can now calculate the bivariate correlation between all the pairs in our data set. As we have 7 series, this translates to 42 pairs...

Let's plot all the bivariate time-varying correlations with RMH from our DCC model:

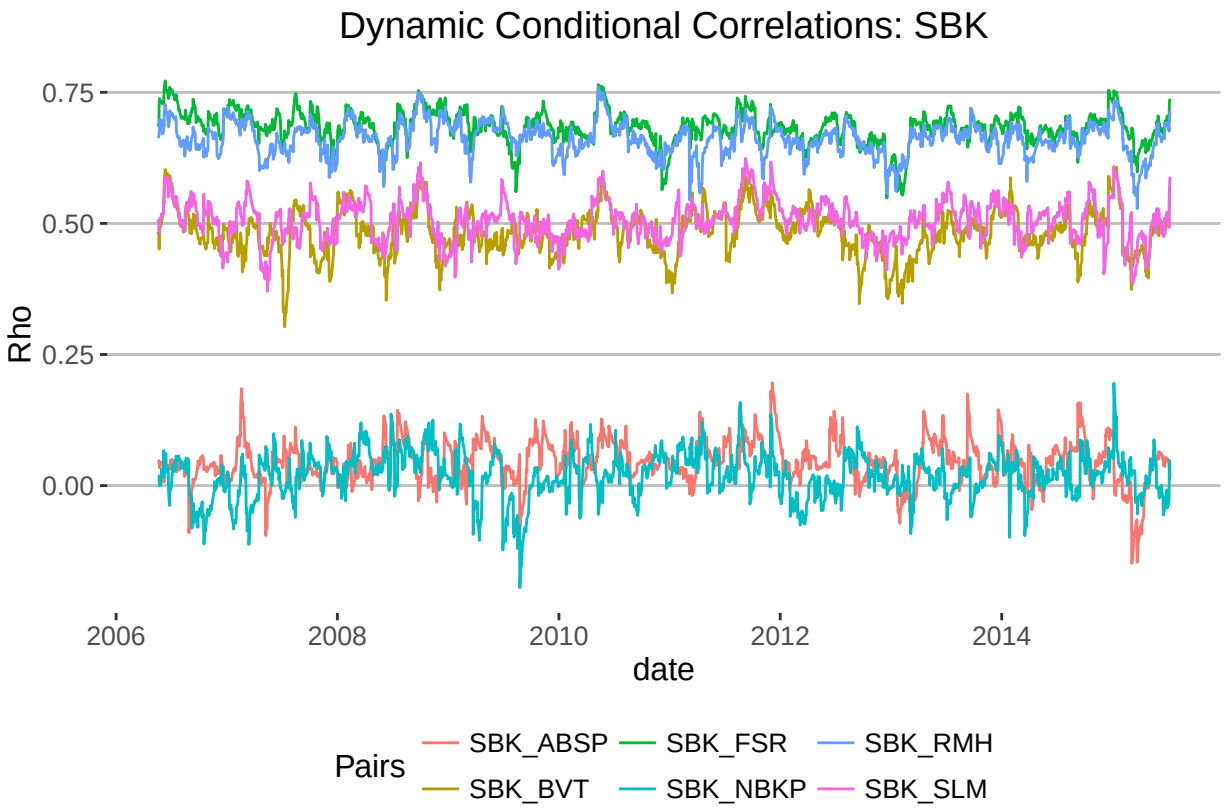
```
Rhot <- DCC$rho.t
# Right, so it gives us all the columns together in the form:
# X1,X1 ; X1,X2 ; X1,X3 ; ....

# So, let's be clever about defining more informative col names.
# I will create a renaming function below:

renamingdcc <- function(ReturnSeries, DCC.TV.Cor) {

ncolrtn <- ncol(ReturnSeries)
namesrtn <- colnames(ReturnSeries)
paste(namesrtn, collapse = "_")

nam <- c()
xx <- mapply(rep, times = ncolrtn:1, x = namesrtn)
# Now let's be creative in designing a nested for loop to save the names corresponding to the col

# TIP: draw what you want to achieve on a paper first. Then apply code.

# See if you can do this on your own first.. Then check us my solution:

nam <- c()
```

```
for (j in 1:(ncolrtn)) {
for (i in 1:(ncolrtn)) {
  nam[(i + (j-1)*(ncolrtn))] <- paste(xx[[j]][1], xx[[i]][1], sep="_")
}
}

colnames(DCC.TV.Cor) <- nam

# So to plot all the time-varying correlations wrt SBK:
 # First append the date column that has (again) been removed...
DCC.TV.Cor <-
    data.frame( cbind( date = index(ReturnSeries), DCC.TV.Cor)) %>% # Add date column which dropp
    mutate(date = as.Date(date)) %>%  tbl_df()

DCC.TV.Cor <- DCC.TV.Cor %>% gather(Pairs, Rho, -date)

DCC.TV.Cor

}

# Let's see if our function works! Excitement!
Rhot <-
  renamingdcc(ReturnSeries = rtn, DCC.TV.Cor = Rhot)

head(Rhot)
```

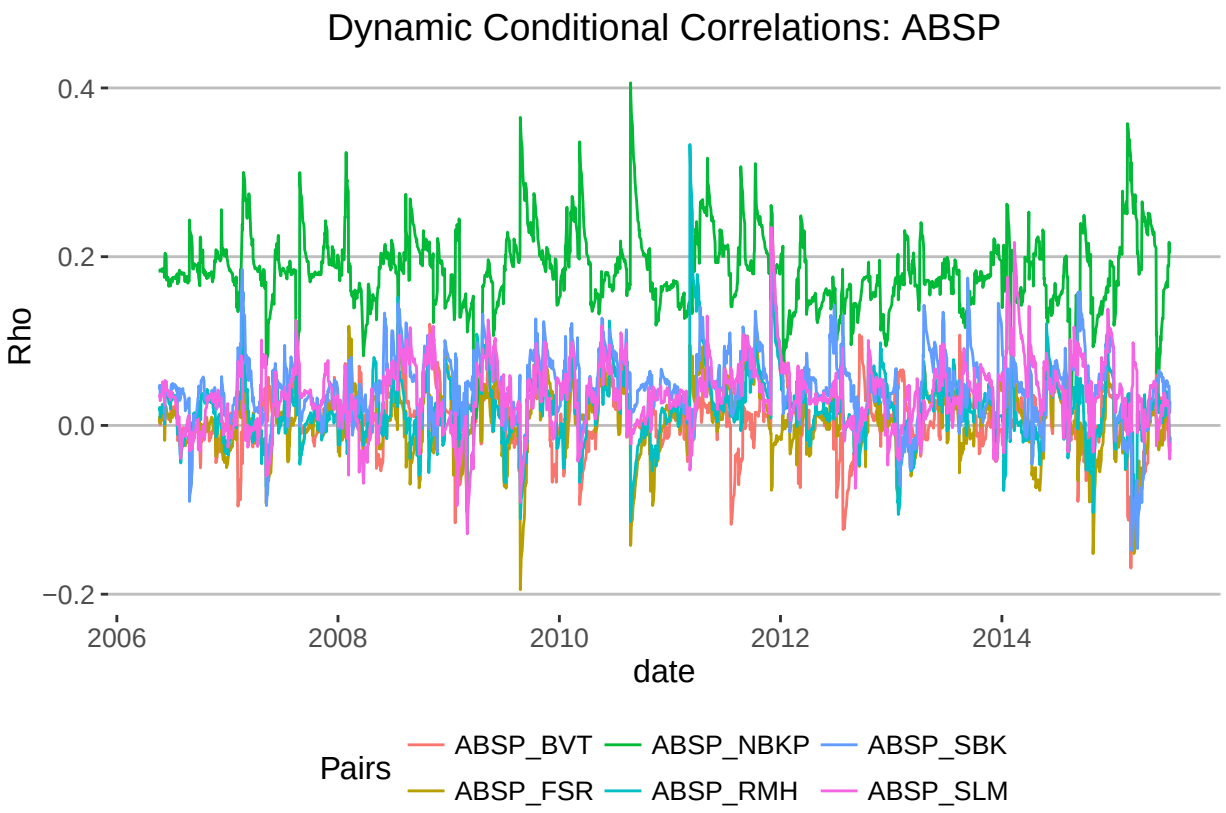
```
## Source: local data frame [6 x 3]
##
```

```
##          date     Pairs   Rho
##        (date)     (chr) (dbl)
## 1 2006-05-19 ABSP_ABSP     1
## 2 2006-05-22 ABSP_ABSP     1
## 3 2006-05-23 ABSP_ABSP     1
## 4 2006-05-24 ABSP_ABSP     1
## 5 2006-05-25 ABSP_ABSP     1
## 6 2006-05-26 ABSP_ABSP     1
```

```r
# Let's now create a plot for all the stocks relative to the other stocks...
g1 <-
  ggplot(Rhot %>% filter(grepl("SBK_", Pairs ), !grepl("_SBK", Pairs)) ) +
  geom_line(aes(x = date, y = Rho, colour = Pairs)) +
  theme_hc() +
  ggtitle("Dynamic Conditional Correlations: SBK")

print(g1)
```




```
g2 <-
  g1 %+% subset(Rhot %>% filter(grepl("ABSP_", Pairs ), !grepl("_ABSP", Pairs))) + ggtitle("Dynam
print(g2)
```




```
# So... try write a function that loops through all your unique ticker names and plots the above.
```

You can now use these measures of time-varying correlations to study the level changes during different periods - particularly assessing whether and by how much correlation changed.

### 6.1 DCC: flexible Univariate specs

To be honest, I am not the biggest fan of constraints in model estimations. Ideally you should be fitting the optimal univariate garch specification to each series. At times it may be infeasible, if you have many series. Nonetheless, here follows a way of specifiying different univariate

```
# Using the rugarch package, let's specify our own univariate functions to be used in the dcc pro

# Step 1: Give the specficiations to be used first:

# A) Univariate GARCH specifications:
uspec <- ugarchspec(variance.model = list(model = "gjrGARCH", garchOrder = c(1, 1)),
                    mean.model = list(armaOrder = c(1, 0), include.mean = TRUE),
                    distribution.model = "sstd")
# B) Repeat uspec n times. This specification should be self-explanatory...
multi_univ_garch_spec <- multispec(replicate(ncol(rtn), uspec))

# Right, so now every series will have a GJR Garch univariate specification. (see ?ugarchspec for

# C) DCC Specs
spec.dcc = dccspec(multi_univ_garch_spec,
                   dccOrder = c(1, 1),
                   distribution = 'mvnorm',
                   lag.criterion = c("AIC", "HQ", "SC", "FPE")[1],
                   model = c("DCC", "aDCC")[1]) # Change to aDCC e.g.

# D) Enable clustering for speed:
cl = makePSOCKcluster(10)

# ------------------------
# Step 2:
# The specs are now saved. Let's now build our DCC models...
# ------------------------
```

```
# First, fit the univariate series for each column:
multf = multifit(multi_univ_garch_spec, rtn, cluster = cl)

# Now we can use multf to estimate the dcc model using our dcc.spec:
fit.dcc = dccfit(spec.dcc,
                 data = rtn,
                 solver = 'solnp',
                 cluster = cl,
                 fit.control = list(eval.se = FALSE),
                 fit = multf)

# And that is our DCC fitted model!

# We can now test the model's fit as follows:
    #   Let's use the covariance matrices to test the adequacy of MV model in fitting mean residu
RcovList <- rcov(fit.dcc) # This is now a list of the monthly covariances of our DCC model series
covmat = matrix(RcovList,nrow(rtn),ncol(rtn)*ncol(rtn),byrow=TRUE)
mc1 = MCHdiag(rtn,covmat)
```

```
## Test results:
## Q(m) of et:
## Test and p-value:  28.84887 0.001318194
## Rank-based test:
## Test and p-value:  81.06984 3.096412e-13
## Qk(m) of epsilon_t:
## Test and p-value:  552.0588 0.02708512
## Robust Qk(m):
```

```
## Test and p-value:  634.2451 1.112388e-05
```

```
# ....Check Tsay (2014 on the interpretation of these Portmanteau tests)....

# Now to save the time-varying correlations as specified by the DCC model,
# it again requires some gymnastics from our side.
# First consider what the list looks like:
dcc.time.var.cor <- rcor(fit.dcc)
print(dcc.time.var.cor[,,1:3])
```

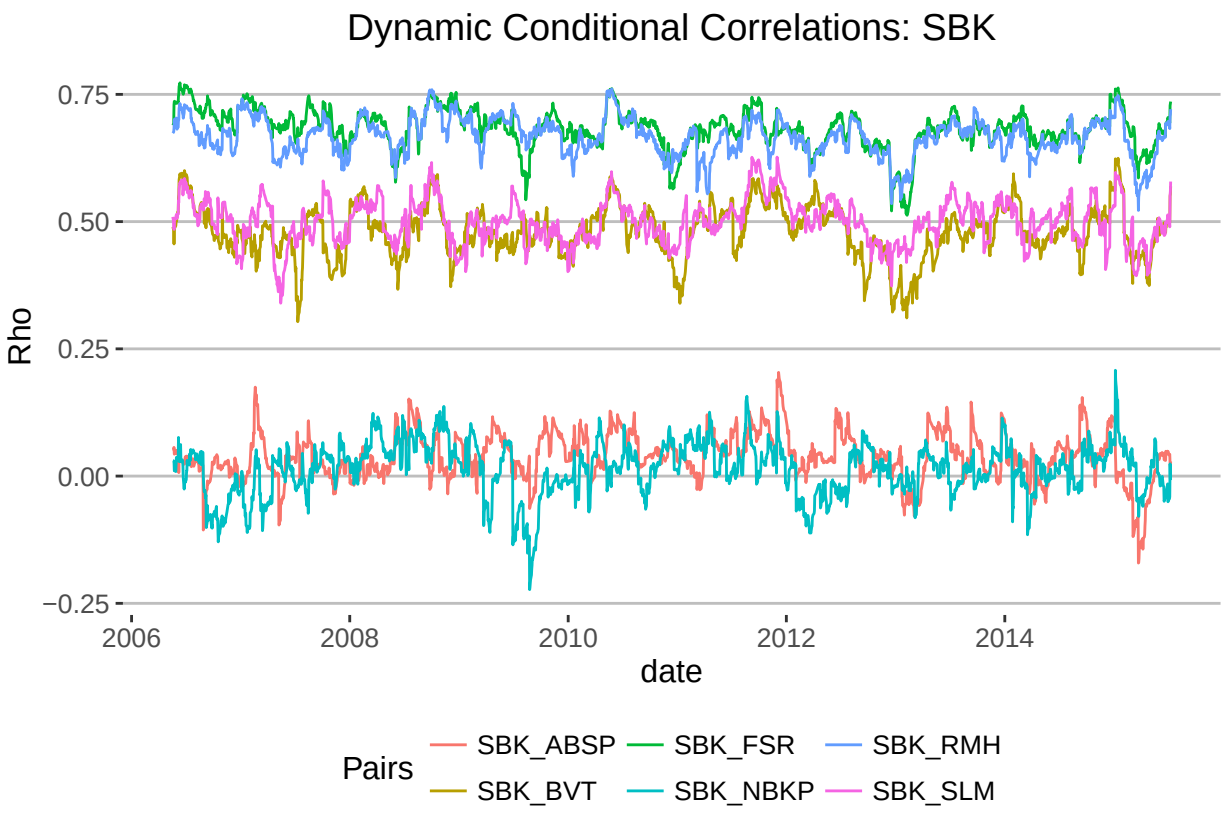
```
## , , 2006-05-19
##
##             ABSP        BVT        FSR        NBKP         RMH        SBK
## ABSP 1.00000000 0.01864161 0.02026426 0.211889330 0.035380223 0.05348617
## BVT  0.01864161 1.00000000 0.49624258 0.047448489 0.503583578 0.48958193
## FSR  0.02026426 0.49624258 1.00000000 0.012384786 0.817725154 0.68930945
## NBKP 0.21188933 0.04744849 0.01238479 1.000000000 0.005074594 0.03029962
## RMH  0.03538022 0.50358358 0.81772515 0.005074594 1.000000000 0.67372682
## SBK  0.05348617 0.48958193 0.68930945 0.030299618 0.673726823 1.00000000
## SLM  0.05197811 0.45957436 0.51067121 0.042455756 0.535230651 0.50929415
##              SLM
## ABSP 0.05197811
## BVT  0.45957436
## FSR  0.51067121
## NBKP 0.04245576
## RMH  0.53523065
## SBK  0.50929415
## SLM  1.00000000
```

```
##
## , , 2006-05-22
##
##             ABSP        BVT        FSR        NBKP         RMH        SBK
## ABSP 1.00000000 0.01797417 0.01959115 0.211296502 0.034729994 0.05274005
## BVT  0.01797417 1.00000000 0.48609065 0.046665240 0.498614773 0.48390744
## FSR  0.01959115 0.48609065 1.00000000 0.012077539 0.818477923 0.69270096
## NBKP 0.21129650 0.04666524 0.01207754 1.000000000 0.004666677 0.02982019
## RMH  0.03472999 0.49861477 0.81847792 0.004666677 1.000000000 0.67626060
## SBK  0.05274005 0.48390744 0.69270096 0.029820192 0.676260596 1.00000000
## SLM  0.05118229 0.45999294 0.49396968 0.041465791 0.525926641 0.49897690
##              SLM
## ABSP 0.05118229
## BVT  0.45999294
## FSR  0.49396968
## NBKP 0.04146579
## RMH  0.52592664
## SBK  0.49897690
## SLM  1.00000000
##
## , , 2006-05-23
##
##             ABSP        BVT        FSR        NBKP         RMH        SBK
## ABSP 1.00000000 0.01745353 0.02540928 0.210755188 0.037258324 0.05638369
## BVT  0.01745353 1.00000000 0.45608063 0.046041255 0.492190905 0.45615712
## FSR  0.02540928 0.45608063 1.00000000 0.012160319 0.813718463 0.72963976
## NBKP 0.21075519 0.04604125 0.01216032 1.000000000 0.004635988 0.02884752
## RMH  0.03725832 0.49219091 0.81371846 0.004635988 1.000000000 0.68353590
```

```
## SBK  0.05638369 0.45615712 0.72963976 0.028847520 0.683535895 1.00000000
## SLM  0.05155671 0.45933722 0.47850869 0.040983390 0.525957473 0.48474832
##              SLM
## ABSP 0.05155671
## BVT  0.45933722
## FSR  0.47850869
## NBKP 0.04098339
## RMH  0.52595747
## SBK  0.48474832
## SLM  1.00000000
```

```r
# Okay, so this format presents a particular challenge in
# getting our time-varying series into a plottable format...
# Every date is a list, and the list a matrix... How to get it in
# a tidy and plottable format?!!

# If you're up for the challenge - see if you can solve this
# little conundrum of getting the lists as bivariate pair columns before using my solution.

# -------------SOLUTION:--------------------

# We will first use the base R function aperm - which transposes any array
# into a list by changing the dimensions
dcc.time.var.cor <- aperm(dcc.time.var.cor,c(3,2,1))
dim(dcc.time.var.cor) <- c(nrow(dcc.time.var.cor), ncol(dcc.time.var.cor)^2)

# And now we can rename our columns the same way as before.
# Luckily we wrote a function so we can use it again...
```

```
dcc.time.var.cor <-
renamingdcc(ReturnSeries = rtn, DCC.TV.Cor = dcc.time.var.cor)

# Note that the figure is very similar to our earlier DCC TV correlations.
# So having a GJR univariate model didn't exactly change much...

g1 <-
  ggplot(dcc.time.var.cor %>% filter(grepl("SBK_", Pairs ), !grepl("_SBK", Pairs)) ) +
  geom_line(aes(x = date, y = Rho, colour = Pairs)) +
  theme_hc() +
  ggtitle("Dynamic Conditional Correlations: SBK")

print(g1)
```




## 7 Go-GARCH: Orthogonalizing sources of volatility

Another different approach to estimating second order persistence models is based on the assumption that returns are generated by a set of unobserved **orthogonal** conditionally heteroskedastic factors (c.f. Van der Weide (2002)). See my presentation on this under the Research Tab, or click here.

These orthogonal components (think PCA on second moment...) are measured by identifying independent and uncorrelated factors that make up the var-covar matrix $H_t$.

The statistical transformations are done as follows:

$$r_t = \mu_t + \varepsilon_t \tag{7.1}$$

$$\varepsilon_t = A.f_t \tag{7.2}$$

with $A$ linking the unobserved uncorrelated components with the observed residual process. $A$: constant and invertible ; $f_t$: normalized, unit variance.

Also: $f_t$ represents the unobserved independent factors assigned to each series (factor weights), such that:

$$f_t = H^{1/2}.z_t \tag{7.3}$$

with $H_T$ and $z_t$ as before and:

$$\begin{aligned} E[f_t] &= 0; & E[f_t f_t'] &= I_N \\ E[\varepsilon_t] &= 0; & E[\varepsilon_t \varepsilon_t'] &= AA' \end{aligned} \tag{7.4}$$

So that the conditional covariance matrix is given by

$$\Sigma_t = AH_tA' \tag{7.5}$$

### 7.0.1 Intuitively...

What is happening is that we are using the residual series of a simple mean return series to decompose the var-covar matrix into a factor loading matrix, A, which is made up of:

- a sample (unconditional) covariance estimate $\Sigma^{1/2}$;
- an orthogonal matrix, $U$, mapping the uncorrelated factors onto the covar matrix: estimated using Independent Component Analysis (ICA).
- Thus it assumes the variance process is driven by independent orthogonal latent components.
- The aim is then to model the independent factors as independent univariate GARCH processes.

  I strongly suggest reading wake me up before you Go-GARCH, which offers a very simple intuitive overview of this technique, as well as a neat application section in section 6 and 7 - a very good source for project ideas...

- An example of a Normally distributed GoGARCH(1,1) model is:

$$\varepsilon_t = A.f_t = \Sigma^{1/2}.U \tag{7.6}$$

$$\tag{7.7}$$

With each uncorrelated component driven by a GARCH(1,1) process:

$$H_t = diag(h_{1,t}, ..., h_{N,t} \tag{7.8}$$

with $h_i$: conditional variances of the factors, $f_t$, so that:

$$h_{i,t} = \gamma_i + \alpha_i.f_{i,t-1}^2 + \beta_i.h_{i,t-1}, \qquad \forall i \tag{7.9}$$

The conditional covariances of $\varepsilon_t$ are then driven by: $V_t = ZH_tZ$

### 7.0.2 Why should GO-GARCH be used?

- The DCC model estimations allow for the **direct modelling** of the conditional correlation.
- Although efficient, this method has some drawbacks, as e.g. it assumes a constant structure to the correlation dynamics.
- The GoGARCH model uses a highly efficient and less parameter intensive estimation technique to decompose the var-covar matrix into orthogonal sources of volatility, which could then be calculated using an additive log-likelihood approach because of the independence of volatility factors.
- Although computationally highly efficient, the assumption of linearly indendent volatility factors is strong.

## 7.1 Fitting the Go-GARCH

We will once again use the rmgarch package to fit the Go-GARCH model. Also see the vignette for technical details (specifically, see section 2.4 for Go-GARCH).

> Tip: search for the rmgarch folder installed on your computer - go to rmgarch/doc, and opening *The_rmgarch_models.Rnw* you will find the source code for the vignette - and by definition all the math equations needed for these models!

So let's fit these models on our data:

```
# Let's keep using our earlier first step individual garch definitions (GJR):
# Now we create a gogarch model specification:
spec.go <- gogarchspec(multi_univ_garch_spec,
                       distribution.model = 'mvnorm', # or manig.
                       ica = 'fastica') # Note: we use the fastICA
cl <- makePSOCKcluster(10)
```

```
multf <- multifit(multi_univ_garch_spec, rtn, cluster = cl)

fit.gogarch <- gogarchfit(spec.go,
                          data = rtn,
                          solver = 'hybrid',
                          cluster = cl,
                          gfun = 'tanh',
                          maxiter1 = 40000,
                          epsilon = 1e-08,
                          rseed = 100)

print(fit.gogarch)
```

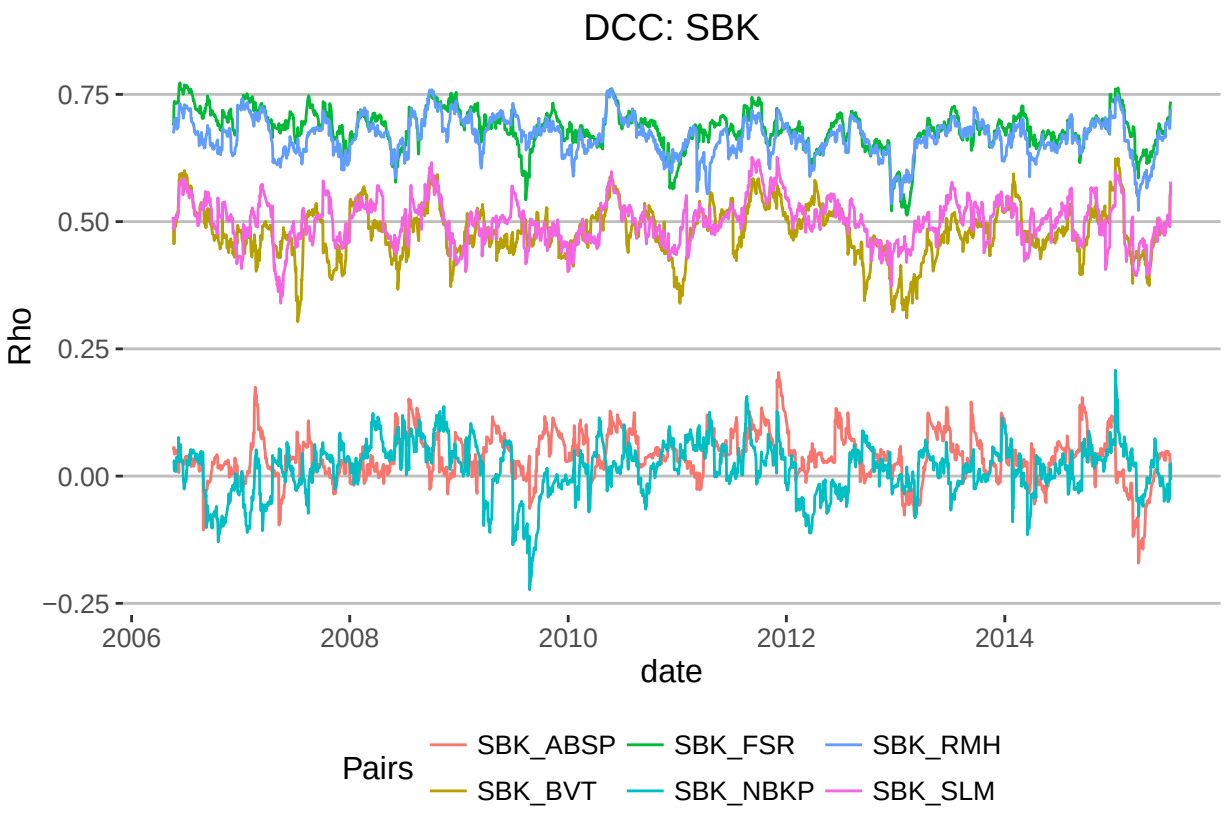
```
##
## *------------------------------*
## *        GO-GARCH Fit          *
## *------------------------------*
##
## Mean Model       : CONSTANT
## GARCH Model      : sGARCH
## Distribution : mvnorm
## ICA Method       : fastica
## No. Factors      : 7
## No. Periods      : 2284
## Log-Likelihood   : -24844.57
## ----------------------------------
##
## U (rotation matrix) :
```

```
##
##           [,1]    [,2]     [,3]   [,4]    [,5]    [,6]   [,7]
## [1,] -0.03261  0.0676 -0.10066  0.237 -0.0122 -0.7778 -0.568
## [2,] -0.04068  0.0197 -0.50668  0.435  0.0557  0.5574 -0.488
## [3,] -0.00398  0.0953 -0.84769 -0.282  0.0429 -0.2368  0.367
## [4,]  0.37756  0.2831  0.00126 -0.701 -0.1580  0.1537 -0.488
## [5,]  0.88808 -0.0688 -0.04172  0.353 -0.2036 -0.0628  0.186
## [6,]  0.02358 -0.9460 -0.10351 -0.231 -0.1072 -0.0155 -0.169
## [7,]  0.25588 -0.0784  0.04589 -0.076  0.9576 -0.0214 -0.055
##
## A (mixing matrix) :
##
##          [,1]     [,2]     [,3]     [,4]     [,5]   [,6]     [,7]
## [1,] -0.0661  0.00154  0.01211 -0.00391  0.73563 0.0144  0.01032
## [2,]  0.0735 -0.11530  0.41237  0.25132  0.00994 1.6108 -0.14023
## [3,]  0.0388 -0.71980  0.38424 -0.91992 -0.02299 1.1282  1.03402
## [4,] -1.0194 -0.01645 -0.00738  0.03984  0.09562 0.0878 -0.00261
## [5,]  0.0683  0.51929  0.48383 -0.91427 -0.00585 1.2440  1.13966
## [6,]  0.0490 -0.23224  0.25956  0.33635  0.05154 0.9569  1.46116
## [7,]  0.0454  0.01090 -1.05026 -0.22992  0.06770 1.2279  0.53844
```

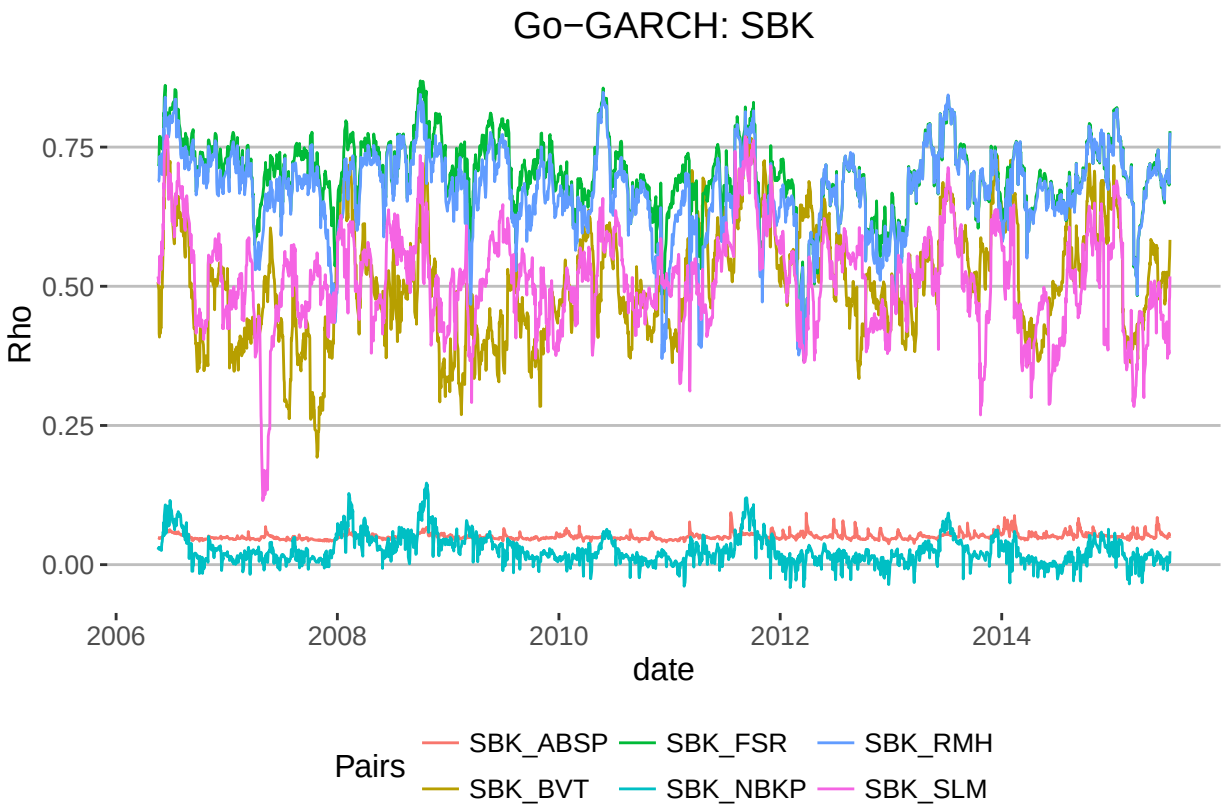
```
# Extracting time-varying conditional correlations: You know the drill...
gog.time.var.cor <- rcor(fit.gogarch)
gog.time.var.cor <- aperm(gog.time.var.cor,c(3,2,1))
dim(gog.time.var.cor) <- c(nrow(gog.time.var.cor), ncol(gog.time.var.cor)^2)
# Finally:
gog.time.var.cor <-
renamingdcc(ReturnSeries = rtn, DCC.TV.Cor = gog.time.var.cor)
```

Now, see the discussion on p.21 of Wake me up before you gogarch, for interpreting the model parameter outputs of the Go-GARCH, as well as the discussion of tv-correlation estimates between DCC and Go-GARCH.

```
g1 <-
  ggplot(dcc.time.var.cor %>% filter(grepl("SBK_", Pairs ), !grepl("_SBK", Pairs)) ) +   geom_lin
  theme_hc() +
  ggtitle("DCC: SBK")


g2 <-
  ggplot(gog.time.var.cor %>% filter(grepl("SBK_", Pairs ), !grepl("_SBK", Pairs)) ) +   geom_lin
  theme_hc() +
  ggtitle("Go-GARCH: SBK")

# DCC tv corr:
print(g1)
```




```
# Go-GARCH tv corr:
print(g2)
```




### 7.2 Time-varying third and fourth moments

In addition to providing us with an estimate of the time-varying second moments, the ACD model allows a closed form solution of the time-varying skewness and kurtosis measures. I would be very interested in someone studying the stability of the third and fourth moment in the domestic market. See here for a step-by-step example of setting up such a tv third and fourth moment model. Important: install the needed packages (racd package) here.

# 8 Other interesting models

The literature on second order moment modelling is fast expanding, and there are very many brilliant techniques that can be applied and used.

I am quite interested in developments in:

- GAS models
- Bayesian GARCH models

I recommend checking out the GAS model page for interesting papers and recommendations on studies. The recent gasmodel R package also provides easily accessible code for fitting these models.. so go play!

# 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.

Christiansen, Charlotte. 2007. "Volatility-Spillover Effects in European Bond Markets." *European Financial Management* 13 (5). Wiley Online Library: 923–48.

Engle, Robert. 2002. "Dynamic Conditional Correlation: A Simple Class of Multivariate Generalized Autoregressive Conditional Heteroskedasticity Models." *Journal of Business & Economic Statistics* 20 (3). Taylor & Francis: 339–50.

Ghalanos, Alexios. 2014. *rmgarch: Multivariate GARCH Models.*

Horvath, Roman, and Petr Poldauf. 2012. "International Stock Market Comovements: What Happened During the Financial Crisis?" *Global Economy Journal* 12 (1).

Johansson, Anders C. 2008. "Interdependencies Among Asian Bond Markets." *Journal of Asian*

*Economics* 19 (2). Elsevier: 101–16.

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

Tsay, Ruey S. 2014. *An Introduction to Analysis of Financial Data with R*. John Wiley & Sons.

Van der Weide, Roy. 2002. "GO-GARCH: A Multivariate Generalized Orthogonal GARCH Model." *Journal of Applied Econometrics* 17 (5). Wiley Online Library: 549–64.