Markus Belfrage
markus.belfrage@gmail.com
15.7.2015

ACDm v1.0.1 tutorial

Contents

1 Introduction

This documents is a tutorial for the R package ACDm, used for fitting the Autoregressive Conditional Duration model (Engle and Russell, 1998) and several extensions of it. The tutorial is intended to get you started with the package, but does not cover all of the functionality of it. For example the simulation function is left out and only the standard ACD(1,1) model with exponential errors is fitted. For more detailed information see the reference manual. Although helpful, no prior knowledge of R is assumed.

The tutorial starts off with a "raw" data set and, by using various functions step by step, has us arrive at a fitted ACD model to be exposed to a series of tests. Each major step in this process has its own dedicated chapter:

1. loading the data into R (chapter 3)
2. computing durations and aggregating (chapter 4)
3. performing diurnal adjustments (chapter 5)
4. fitting an ACD model (chapter 6)
5. testing the model (chapter 7)

Step 1 to 4 will each create a new object in R, and consequently each following step is contingent on the previous step. However, the objects created (except for the fitted model in step 4) are already saved as objects in the package, making it possible for the reader to start the tutorial at any of the steps/sections.

1.1 Installing R and RStudio

For instructions on how to install R please go to http://cran.rstudio.com/.

It is highly recommended to also install the user interface RStudio for a smoother use of R. Get it for free at http://www.rstudio.com/products/rstudio/download/.

1.2 Installing and loading the ACDm package

1. Open the R program (or preferably RStudio if you installed it) and just enter the command
   ```
   install.packages("ACDm")
   ```
   to the console and the ACDm package will be downloaded and installed. The package also requires a few other R packages, and these will be downloaded and installed automatically.

2. You then need to also load the package for your R session. This is done by the command
   ```
   library("ACDm")
   ```

2 About the data set used in the examples

The data used in the examples are based a real data set, but has been obfuscated by transforming the dates, price and volume, for proprietary reasons. The real data are transactions of a major company traded in the Nordic stock market, traded sometime during the past 10 years (as of this writing in 2015). The obfuscated data covers two weeks of intraday transactions recorded at 1 second precision, and together with date and time stamp of the transactions, the price and volume (number of stocks traded at each transaction) are available. The total number of transactions is nearly 100 000, however as we will soon see, transaction done on the same second will be aggregated, shrinking the sample size to 35 000.

3 Loading the data file into R

The package requires the data to be in the R class data.frame, with a necessary column named *time*, where each row is the complete date and clock time for one transaction ordered chronologically. The *time* column needs to be in an R time format, preferably in POSIXlt or POSIXct format. It is also possible to supply the time data as a character sting in the format

yyyy-mm-dd hh:mm:ss

such as *2003-12-14 11:47:05*. The columns price and volume can also be given in the data.frame, if these data are available. If so, the package can estimate ACD models, not only for trade durations, but also price durations (time until an absolute change in price of an arbitrary size) and volume durations (time until an arbitrary large cumulative number of traded shares).

The example file used in the package containing the transactions is in CSV format (Comma-Separated Values). To load a CSV file into a *data.frame* object in R use the R function `read.csv()`.

The .csv file is provided separately with the package, so you first need to locate the directory were you saved the file, for example C:\Users\You\Documents\transactionData.csv (if you don't have this file the data is already loaded by the package into R by default). To load the data to a data.frame named, for example, *transData*, type in the command

```
transData <- read.csv("C:/Users/You/Desktop/transactionData.csv")
```

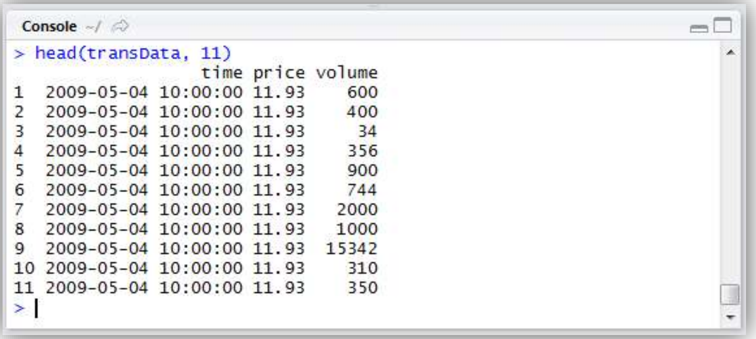
into the R (or Rstudio) console, where `C:/Users/You/Desktop/` should be changed to your personal path where you downloaded the file.

R hints: The assignment operator `<-` in R is made of two symbols, and assigns the value from the right hand expression to left hand variable (or in reverse order if instead `->` is used).

The top 11 rows of the data frame can be view by the command

```
head(transData, 11)
```

as shown in the figure below:



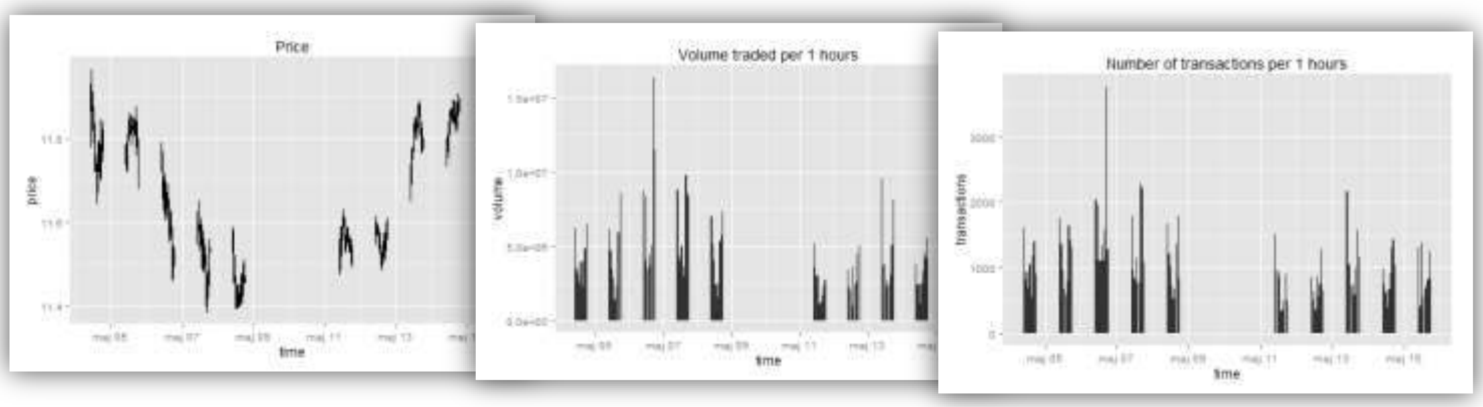
```
> head(transData, 11)
                 time price volume
1  2009-05-04 10:00:00 11.93    600
2  2009-05-04 10:00:00 11.93    400
3  2009-05-04 10:00:00 11.93     34
4  2009-05-04 10:00:00 11.93    356
5  2009-05-04 10:00:00 11.93    900
6  2009-05-04 10:00:00 11.93    744
7  2009-05-04 10:00:00 11.93   2000
8  2009-05-04 10:00:00 11.93   1000
9  2009-05-04 10:00:00 11.93  15342
10 2009-05-04 10:00:00 11.93    310
11 2009-05-04 10:00:00 11.93    350
> |
```

Other ways of viewing parts of the data are with the command `tail(transData)`, or for example `transData[35001:35100, ]` to view the next 100 rows starting from row 35 001.

We may also want to plot the data. This can be done with the usual ways of plotting in R (see for example the function *plot* from the base package or the graphics package *ggplot2*), but there is also a function called `plotDescTrans()` in the ACDm package dedicated to plotting the price, volume, and number of transactions from the transactions data. Running the code

```
plotDescTrans(transData, windowunit = "hours", window = 1)
```

gives the following graphs:



One may want to change the window sizes for the last two graphs, for example with `plotDescTrans(transData, windowunit = "mins", window = 20)` to plot 20 minutes intervals instead, or by setting *windowunit = "day"* to easily see the differences between days.

3.1 An example of importing a different data set

Note! This section should probably be skipped. This example presented here is intended to be of help if you have any problems getting your own data set to work with the ACDm package.

The timestamps in a set of data can be in many different formats, and getting the program to understand them might need some work. If the time is in a non-standard format, this can be a bit of a hassle. The example here will guide you through one way of doing this in R, using a real high frequency data set made available online by Ruey S. Tsay, which he used in examples in his book *Analysis of Financial Time Series* (2010) and are the same data used in Engle and Russell's (1998) original ACD paper . The data file is available (at the time of writing, 2015) at

http://faculty.chicagobooth.edu/ruey.tsay/teaching/fts/ibm.txt

and consists of IBM transaction data covering 3 months between 1990 and 1991. Download this file to your hard drive and have a look at it, for example by opening it in Notepad. The file consists of unnamed columns with whitespace between them, and from Tsay's webpage at http://faculty.chicagobooth.edu/ruey.tsay/teaching/fts he informs us that the "columns are date/time, volume, bid quote, ask quote, and transaction price".

Now load it to R, again using the read.csv() function. For example with

```
IBMtrans <- read.csv("C:/Users/You/Desktop/ibm.txt", header = FALSE, sep
= "")
```

where `C:/Users/You/Documents/` should be changed to your local path. This time we need to add the argument `header = FALSE` to inform R that the data file is lacking a header (column names). By also adding `sep = ""`, the whitespaces around the columns will be removed. To give the columns names, run the command

```
names(IBMtrans) <- c("time", "volume", "bid quote", "ask quote",
"price")
```

By viewing the first few rows of the data.frame `IBMtrans` with head(IBMtrans) we see a strange time format, with the timestamp '90110134228' for the first transaction. One might deduce that the first 6 digits must refer to the date, and the last 5 to the number of seconds after midnight. We now need some way of transforming the timestamps into the format POSIXlt used in R and the ACDm package. To do this we will use the R function `substr()` to "cut" the timestamps in two parts, date and time, and the function `strptime()` to convert to the POSIXlt format. To first extract the date, i.e. the first 6 digits of the string, and save it as a vector named 'time', use

```
time <- IBMtrans[ , 1]
time <- substr(time, 1, 6)
```

The first line of code uses R's extraction operator `[ , ]` that is used to extract individual elements from a data.frame (and other data object such as a matrix). The first argument specifies the row index and the second argument column index. By leaving one argument blank, all of the elements in that dimension are selected. In our case, with `IBMtrans[ , 1]`, this means that the first column vector is returned. In the next row of code, the `substr()` function will work on all elements in this vector of strings individually. By giving the last two arguments of `substr()` the values 1 and 6 we retrieve the first six characters (digits) of each string, as a new vector.

The next step is then to convert this new vector of strings to the POSIXlt format, using `strptime()`. For this, run the code

```
time <- strptime(time, format = "%y%m%d")
```

The function needs to be told the date format of the date strings. The documentation of the `substr()` function (which can be read by `?substr` in R) gives details on how to specify the time and date format, which for this case should be "%y%m%d". Next we will add the last 5 digits of the original time stamps, representing seconds after midnight, to the time vector. It is possible to add seconds directly to a POSIXlt object with the `+` operator, as we will do here. The last 5 digits first need to be extracted with `substr()` and then converted from a string to a number representation by using the function `as.numeric()`. In one line of code we can do all of that, by running

```
time <- time + as.numeric(substr(IBMtrans[ ,1], 7, 11))
```

By using the `+` operator the class of the data object is converted to POSIXct instead of POSIXlt. The latter format is good to use as the time is internally represented by the different units of time, days, hours etc., separately. Therefor as the final step we will convert it back to POSIXlt and at the same time overwrite the old time column in the data.frame with the code

```
IBMtrans$time <- as.POSIXlt(time)
```

where the `$time` part is used to specify the column 'time' in `IBMtrans`.

The `IBMtrans` data.frame is now in the correct format to be used by ACDm. Unfortunately though, the data set has two days of long periods without trades, due to a halt in the market. This was recognized by Engle and Russell (1998) and they dealt with it simply by removing these two days, November 23 and December 27. If you would like to do the same, to then have a ready data set to use with the ACDm package, this can be done with the following code:

```
IBMtrans <- IBMtrans[IBMtrans[ ,1]$yday != strptime("901227", format =
"%y%m%d")$yday, ]

IBMtrans <- IBMtrans[IBMtrans[ ,1]$yday != strptime("901123", format =
"%y%m%d")$yday, ]
```

An approximation of what Engle and Russell (1998) then did concerning the diurnal adjustment and removal of trades at the ends of the trading day can be done with:

```
IBMDurations <- computeDurations(IBMtrans, open = "10:00:00", close =
"16:00:00")

IBMAdjDurations <- diurnalAdj(IBMDurations, aggregation = "all", method
= "cubicSpline", nodes = c(seq(600, 900, 60), 930, 960))
```

4 Computing durations from transaction data

As we are interested in modeling the time between trades (or price- or volume durations) we now need to compute the durations. When the data have been recorded with one second precision, it is common practice to aggregate transactions completed at the same second, making no zero duration possible (this can partly be motivated by so called split transactions, where a single large trade is executed as several smaller transactions in a short time frame). We will follow this procedure here.

Durations can be computed with the function `computeDurations()`.

This function has the following arguments:

- **transactions** – the data frame containing your transactions
- **open** – a string with the market opening time in format "hh:mm:ss"
- **close –** the closing time, e.g. "18:25:00"
- **rm0dur** – remove zero durations? TRUE/FALSE
- **type** – "trade" for transaction durations, "price" for price durations, "volume" for cumulated volume durations
- **priceDiff** – used if the **type** = "price" is used
- **cumVol** – used if the **type** = "volume" is used

To get transaction durations in a new data frame called *durData* just type in

```
durData <- computeDurations(transData)
```

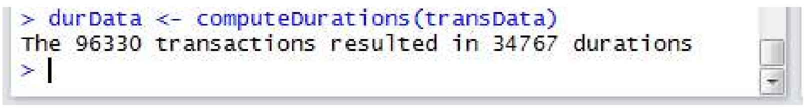
The left out arguments will then take the default values. This yields the same result as the command

```
durData <- computeDurations(transactions = transData, open = "10:00:00",
close = "18:25:00", rm0dur = TRUE, type = "trade")
```

R hints: The name of the arguments (the left hand side of =) in a function can be left out if the arguments are entered in the predefined order.

Since the stock exchange, where our example data where traded, is open for continuous trading between 10.00 and 18:25 local time, the default values suits our needs.

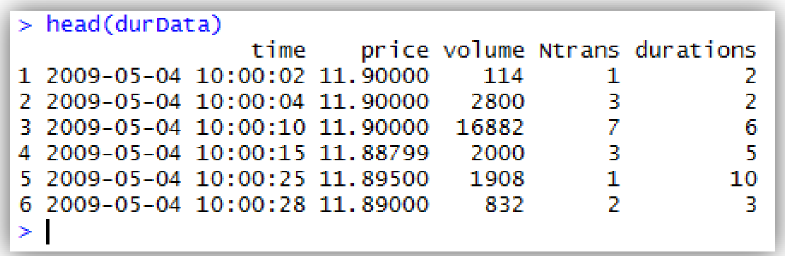
Running the above command returns



```
> durData <- computeDurations(transData)
The 96330 transactions resulted in 34767 durations
> |
```

informing us that after aggregation and removing of trades done outside the opening hours of continuous trading, we are left with just short of 35 000 durations.

Again we may want to have a look at our new data object. The new object is of class data.frame, so the same functions used previously to show the transactions can be used here, for example:



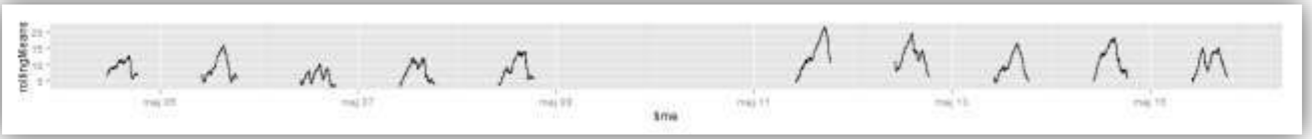
```
> head(durData)
                 time    price volume Ntrans durations
1 2009-05-04 10:00:02 11.90000    114      1         2
2 2009-05-04 10:00:04 11.90000   2800      3         2
3 2009-05-04 10:00:10 11.90000  16882      7         6
4 2009-05-04 10:00:15 11.88799   2000      3         5
5 2009-05-04 10:00:25 11.89500   1908      1        10
6 2009-05-04 10:00:28 11.89000    832      2         3
> |
```

The last column *durations* are the durations, which we are mainly interested in. The column *price* is the volume weighted price of the aggregated transactions, the column *volume* is the aggregated traded shares, and Ntrans gives us the number of transaction aggregated for each row.

One way of plotting the durations is to use a rolling mean. The function `plotRollMeanAcd()`can be used for this. Running

```
plotRollMeanAcd(durData,  window = 500)
```

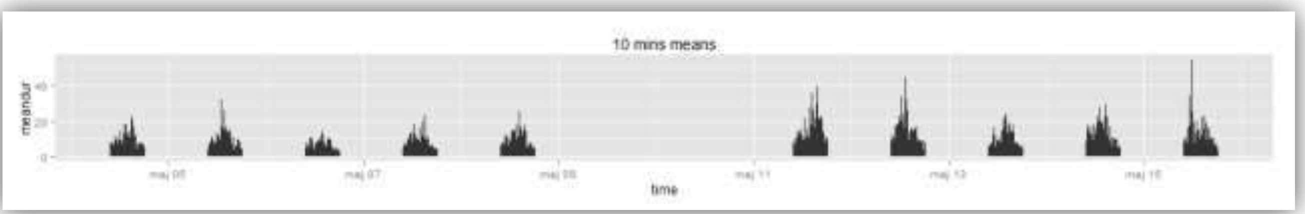
will yield the following graph:



Using a smaller window size will give a more jagged curve. Another way of graphing the durations is by using the mean duration over a specified interval length in a bar plot. This can be done with the function `plotHistAcd()`, for example

```
plotHistAcd(durData,  windowunit = "mins", window = 10)
```

will return:



5 Diurnal adjustment of the durations

The ACD models available in in the package all require a stationary duration process. However, trade durations have a clear daily pattern, arising from the fact that traders' level of activity varies over the trading day, notably being higher in the beginning and end of the trading day. This makes the unconditional mean durations time varying, a daily seasonal variation that in the ACD literature predominantly has been modeled as a deterministic term. To get a stationary duration process this deterministic factor can be estimated and then removed from the durations. This is done by calculating the diurnally adjusted durations $\tilde{x}_i$ as

$$\tilde{x}_i = \frac{x_i}{\hat{s}_i},$$

where $\hat{s}_i$ is the estimated value of this seasonal component at the time of the transaction done at the start of duration x_i. There is however no consensus in the literature on how this diurnal factor should be estimated, and numerous methods have been proposed. The ACDm package currently has implemented 4 different methods of calculating$\hat{s}_i$, and each method has several parameters/arguments to be predetermined by the practitioner. Things are further complicated by the fact that the diurnal pattern seems to differ between weekdays, leading some authors to estimate the diurnal factor separately for each day of the week (eg., Bauwens and Giot, 2000). The package allows the user to choose between no aggregation (each date has its own diurnal factor), daily aggregation (each weekday has its own diurnal factor), and total aggregation (all days have the same diurnal factor).

In the package the function `diurnalAdj()` is used to create a diurnally adjusted duration series. The function arguments are

dur – the duration data frame

method – how the deterministic diurnal component is calculated, either *cubicSpline, supsmu* (super smoother), *smoothSpline* (cubicSpline with a smoothing parameter), or *FFF (Flexible Fourier Form).*

For the *cubicSpline* and *smoothSpline* methods:

nodes – a vector of the nodes (in minutes after midnight) used by the *cubicSpline* method

For the *smoothSpline* method:

spar – “smoothing parameter, typically (but not necessarily) in (0,1]”

For the *supsmu* method:

span – a paramter sent to the *supsmu* function, “the fraction of the observations in the span of the running lines smoother, or "cv" to choose this by leave-one-out cross-validation”

For the *FFF* method:

Q – the number of trigonometric function pairs

aggregation – should the diurnal adjustment be done pooled for *all* days*,* done separately for *weekdays* or done with no *(none)* aggregation*?*

The command

```
diurnalAdj(durData)
```

is equivalent to the command

```
diurnalAdj(dur = durData, method = "cubicSpline", nodes = c(seq(600,
1110, 60), 1100), aggregation = "all")
```

as these arguments are the default values.

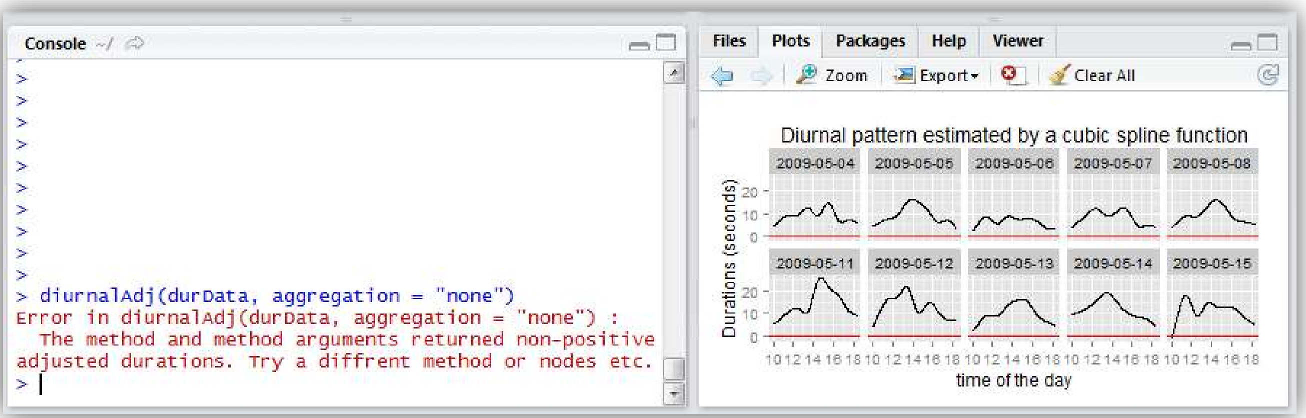
R hints:

- The function `seq(from, to, by)` creates a sequence returned as a vector. For example `seq(5, 10, 2)` returns the vector [5 7 9]
- The function `c()` combines or concatenates its vector arguments into a larger vector (in R any scalar is considered a vector with a single element)
- To get more information of a function, use the comand `?` followed by the function name, for example `?seq` or `?diurnalAdj`

Let's start by trying the `diurnalAdj()` function by changing the aggregation from "all" to "none", while keeping the other default values , with the command

➡ `diurnalAdj(durData, aggregation = "none")`

The output is shown in the figure underneath.

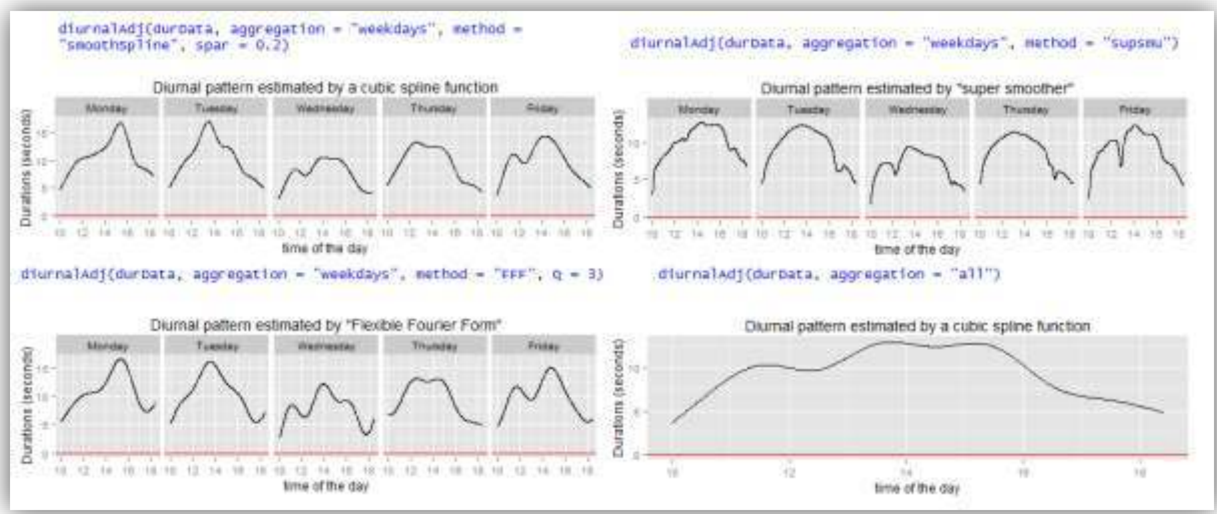


```
>
>
>
>
>
>
>
>
>
>
> diurnalAdj(durData, aggregation = "none")
Error in diurnalAdj(durData, aggregation = "none") :
  The method and method arguments returned non-positive
adjusted durations. Try a diffrent method or nodes etc.
> |
```

A few interesting things can be observed from the output. First, to the right we see the graph of the estimated diurnal pattern. It is clear that the durations are larger (longer) in the middle of the day and shorter at the end of the trading day when traders are more active. Other than that, the pattern seems to differ largely between the dates, even for dates occurring on the same weekday. Therefore one may prefer to aggregate all of the dates. This, however, would force the weekdays to have the same diurnal pattern. On the other hand, it could be argued that aggregating over weekdays separately is using a model with too many parameters and over fits the data. Unfortunately, the ACD literature has touched upon this issue too little to give us any clear guidelines.

Secondly, in the left part of the figure we see an error message. If we take a closer look at the beginning of the last day (Feb. 19), we see that a small part of the curve goes below the red line, indicating negative $\hat{s}_i$ components. It turns out that this method used on this data set produced a negative fitted line, probably due to a sharp shift in mean durations the first two hours of that day. If any of the $\hat{s}_i$'s are negative, this in turn would yield negative adjusted durations $\tilde{x}_i$, and go against the assumptions of the ACD model and preventing further estimation. The program will therefore not allow this particular method with the given method parameters for this data set.

Some further examples of diurnal adjustments of the data set are given in the figure below. With the exceptions of the upper left panel, variants of these have all been used in the literature.



```
diurnalAdj(durData, aggregation = "weekdays", method =
"smoothSpline", spar = 0.2)
```

```
diurnalAdj(durData, aggregation = "weekdays", method = "supsmu")
```

```
diurnalAdj(durData, aggregation = "weekdays", method = "FFF", q = 3)
```

```
diurnalAdj(durData, aggregation = "all")
```

Without further motivation, we decide to aggregate all days and use a spline function with nodes set hourly. The diurnal factor is depicted in the figure above in the lower right corner. This decision can certainly be criticized and one might want to investigate how a different choice would have affected the estimation in the next section.

To finally save the diurnally adjusted durations, run the command

```
adjDurData <- diurnalAdj(durData, aggregation = "all")
```

This will add a column named adjDur with the adjusted durations to the previous data.frame. You may want to take a look at the new data.frame, again with the function `head()`. Another way of exploring the data.frame object is with the function `str()`.

R hint: the function `str()` displays the internal **str**ucture of any R object

6 Fitting an ACD model

To estimate various ACD models, use the function `acdFit()`. The estimation is done by maximum likelihood estimation. Currently the package can estimate the following ACD models (see the appendix for model specifications):

- ACD
- LACD1 or LACD2
- AMACD
- ABACD
- SNIACD or LSNIACD

The ACD models can be of different orders, such as LACD1(2, 1) for example. The possible distributions of the error terms are the following (the text in *italic* is the respective keyword to be passed as the argument value):

- Exponential distribution: *"exponential"*
- Weibull distribution*: "weibull"*
- Burr distribution*: "burr"*
- Generalized Gamma distribution: *"gengamma"*
- Generelized F: *"genf"*
- q-Weibull: *"qweibull"*
- Finite mixture of inverse Gaussian Distributions: *"mixinvgauss"*

The function `acdFit()` takes the following arguments (among others):

durations – the durations data.frame, with the 'adjDur' column or 'Durations' column (can also be a vector of durations).

model – the model for the conditional expected duration. Either "ACD", "LACD1", "LACD2", "AMACD", "BACD", "ABACD", "SNIACD" or "LSNIACD" (see appendix for model specifications).

dist – the assumed distribution of the error term, either "exponential", "weibull", "burr", "gengamma ", "genf ", "qweibull", or "mixinvgauss".

startPara – a vector with parameter values from which the MLE algorithm starts.

order – a vector with the lag orders of the model, for example **order = c(1, 1)** for an ACD(1,1) model, or for example **order = c(1, 1, 2)** for an AMACD(1, 1, 2) model.

dailyRestart – if given the value 1, the conditional expected duration is reset at the start of every day (to the mean value of the whole sample).

optimFnc and **method** – Specifies which optimization function to use for the estimation. "optim", " nlminb", "solnp", and "optimx" are available. **method** is used to specify the optimization algorithm for the optim and optimx functions.

6.1 Estimation example

Now that we have prepared the data by removing the diurnal component we are ready to estimate different ACD models.

We will first start with the simple ACD(1, 1) model with exponentially distributed errors. When assuming that the errors are exponentially distributed, we are essentially using QML (Quasi Maximum Likelihood), as the parameter estimates of the conditional durations are consistent, regardless of the assumed error term distribution possibly being wrong (assuming that the rest of the model is correctly specified).

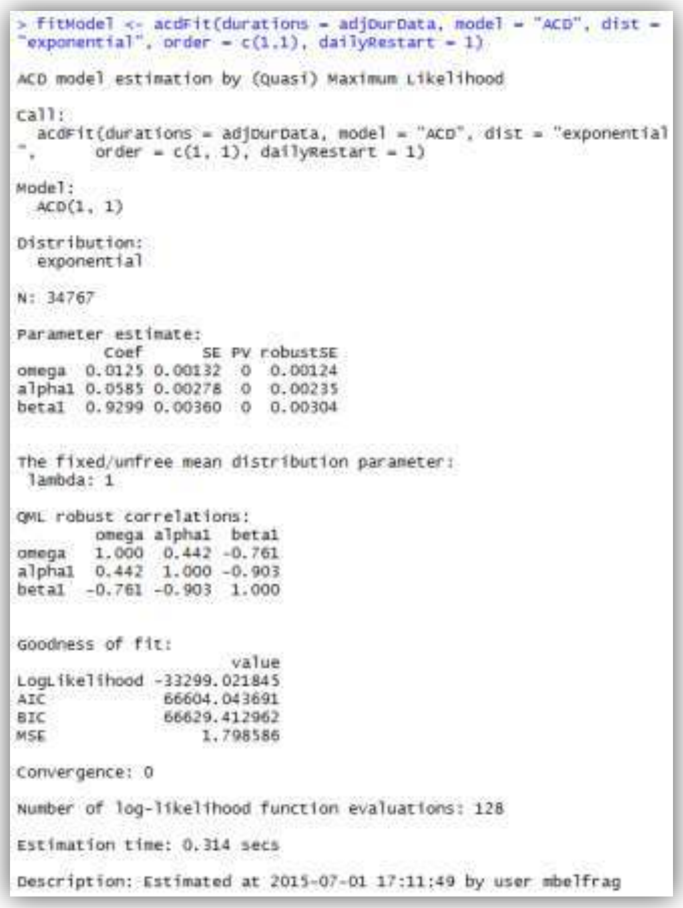
In all of the examples here, the conditional duration μ_i is set to restart each morning by setting the argument *dailyRestart* = 1. This way the previous evening's durations won't have any direct impact on the conditional mean durations at the start of the day.

In mathematic notation, the durations x_i are modeled as

$$x_i = \mu_i * \varepsilon_i,$$
$$\mu_i = \omega + \alpha_1 x_{i-1} + \beta_1 \mu_{i-1},$$
$$\varepsilon_i \sim \exp(1).$$

To estimate this model, type (or copy-paste) in the console:

```
fitModel <- acdFit(durations =
adjDurData, model = "ACD",
dist = "exponential", order =
c(1,1), dailyRestart = 1)
```



```
> fitModel <- acdFit(durations = adjDurData, model = "ACD", dist =
"exponential", order = c(1,1), dailyRestart = 1)

ACD model estimation by (Quasi) Maximum Likelihood

Call:
  acdFit(durations = adjDurData, model = "ACD", dist = "exponential
",       order = c(1, 1), dailyRestart = 1)

Model:
  ACD(1, 1)

Distribution:
  exponential

N: 34767

Parameter estimate:
          Coef      SE PV robustSE
omega   0.0125 0.00132  0  0.00124
alpha1  0.0585 0.00278  0  0.00235
beta1   0.9299 0.00360  0  0.00304


The fixed/unfree mean distribution parameter:
 lambda: 1

QML robust correlations:
        omega alpha1  beta1
omega   1.000  0.442 -0.761
alpha1  0.442  1.000 -0.903
beta1  -0.761 -0.903  1.000


Goodness of fit:
                      value
LogLikelihood -33299.021845
AIC            66604.043691
BIC            66629.412962
MSE                1.798586

Convergence: 0

Number of log-likelihood function evaluations: 128

Estimation time: 0.314 secs

Description: Estimated at 2015-07-01 17:11:49 by user mbelfrag
```

Figure 1 ACD(1, 1) with exp. errors

The output is shown in Figure 1. Note that the part fitModel <- could be left out to get the output from the estimate (as done in the figure), but saving the fitted model is needed for some of the later tests, and lets you extract the residuals $\hat{\varepsilon}_i$ or the fitted conditional durations $\hat{\mu}_i$. It may also be a good idea to save the fitted model if you later want to compare several different models.

Convergence: 0 from the output tells us that the maximum likelihood optimization algorithm managed to converge at a local maximum. There are, however, no guarantees that the found estimate is the global maximum. It can therefore be a good idea to perform the estimation across different starting parameters. This is especially important when estimating the more advanced models with more parameters to estimate.

The residuals and the fitted conditional durations $\hat{\mu}_i$ can be retrieved from the estimated model, saved as `fitModel`, with the command `fitModel$residuals`, and `fitModel$muHats` respectively.

7 Plots and tests

"The three golden rules of econometrics are test, test and test."

David Hendry

The examples in this section is based on the ACD(1, 1) with exponential errors fitted to the data used in the previous sections. To save space this object is not included in the R package, so unless you performed the examples in the previous section, you will need to run the estimation first before continuing. Just run the code

```
fitModel <- acdFit(durations = adjDurData, model = "ACD", dist =
"exponential", order = c(1,1), dailyRestart = 1)
```

and you will have the fitted model object used in the following examples saved as *fitModel.*

7.1 plotScatterAcd()

This is a general function to graphically investigating (mainly) the possible need for nonlinear specifications of the conditional mean duration, as well as finding potential issues with the diurnal adjustment. As the name of the function implies, it does this by a simple scatterplot between two variables, but has a possible third variable represented by a color scale. The variables can be lagged as well. On top of this is a curve showing the estimated (by ggplot2) conditional mean of the y-variable given the x-variable.

The possible variables from a fitted model to be plotted are:

- muHats
- residuals
- durations
- adjDur
- daytime
- time
- index (the $i's$)

In the first example we want to investigate whether the estimated conditional means $\hat{\mu}_i$ under- or over predicts the mean size of the upcoming duration $\tilde{x}_i$, for certain values of $\hat{\mu}_i$. This can be done by plotting the residuals $\hat{\varepsilon}_i = \tilde{x}_i/\hat{\mu}_i$ against $\hat{\mu}_i$. As the error terms, given correct specification, are independent of $\hat{\mu}_i$ we should expect the mean of the residuals to be about the same for all values of $\hat{\mu}$. The plot is created with the following command:

```
plotScatterAcd(fitModel, x = "muHats", y = "residuals", colour = NULL,
ylag = 0, xlim = NULL, ylim = NULL, alpha = 1/10, smoothMethod =
"auto")
```

There will be a warning when running this function, informing us what smoothing method was used, so don't worry about that. The resulting output is shown in Figure 2. Noticeably we see a smaller mean of the residuals at small and large $\hat{\mu}$, indicating that the predicted $\hat{\mu}_i$:s are too large at the ends of $\hat{\mu}$:s' range, something a nonlinear model could perhaps overhaul. Another interesting thing of note is that a large amount of the residuals seems to be placed together on curves (hard to see on the figure with low resolution, but very clear if one zoom in closer). This is due to the discretization (one second precision) of the original duration series. In fact, if the durations were not diurnally adjusted beforehand, all of the residuals would lie on exact curves like those (try it out by estimating an unadjusted duration series).

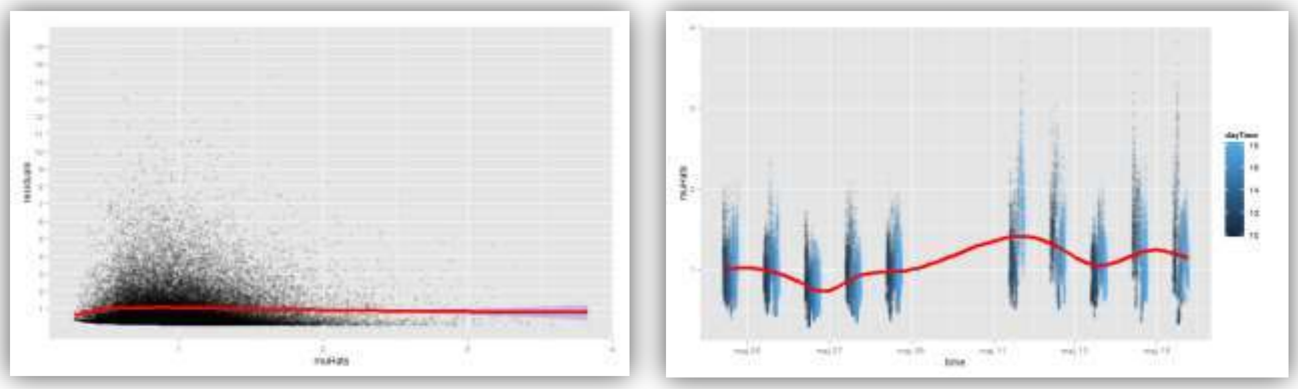


Figure 2 two example uses of the function plotScatterACD()

The second example, shown in the right panel of Figure 2, was created by running plotScatterAcd() with the following arguments:

```
plotScatterAcd(fitModel, x = "time", y = "muHats", colour = "dayTime",
ylag = 0, xlim = NULL, ylim = NULL, alpha = 1/10, smoothMethod =
"auto")
```

This example shows the use of the *colour* argument field, here with the value *daytime.*

7.2 acf_acd()

This function plots the autocorrelations of one or several of the durations, the adjusted durations, or the residuals. Continuing with our example, we can call this function with

```
acf_acd(fitModel, conf_level = 0.95, max = 50)
```

returning the plot in Figure 3 below. The blue bands in the figure are 95 % confidence bands (conf_level=0.95). In our example we see that the durations are auto correlated with a slowly decreasing correlation for larger lags. After removing the diurnal component the correlations are declining somewhat. The large reduction in autocorrelations however comes for the residuals after removing the estimated ACD(1, 1) dependency.

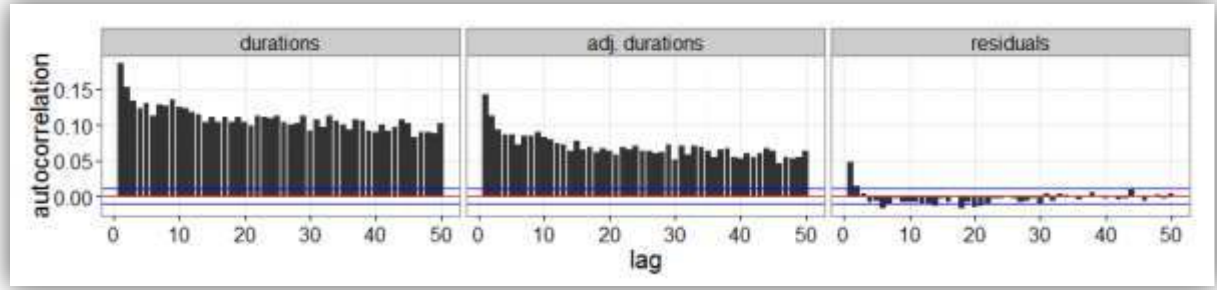


Figure 3 Autocorrelogram of durations, diurnally adjusted durations, and residuals

7.3 plotHazard()

As the name suggests this test graphically asses the hazard function of the error terms. This is done by plotting a nonparametric estimate of the hazard function from the residuals, as well as the hazard function implied from the model estimate.

To use the test you need to first estimate a model and save the estimate, for instance with the name fitModel, as in the example estimate above. The test is then called by

```
plotHazard(fitModel)
```

The output using the example data is shown in Figure 4 below. In the top left corner we see the hazard function when the estimated model is the ACD(1, 1) with exponential errors we earlier estimated. The black line is the empirical hazard function of the residuals from the non-parametric estimator (slightly modified/improved) used by Engle and Russell (1998). The red line instead depicts the hazard function as implied by the estimated parameters of the error distribution. For the exponential distribution the hazard is flat, in this case at 1 as the distribution is forced to have unit expectation. The other two figures have used estimated models assuming Weibull- and Burr distributions respectively. Clearly the closest match seems to be the Burr distribution, as it allows for a non-monotonic hazard, while still not being a perfect match.

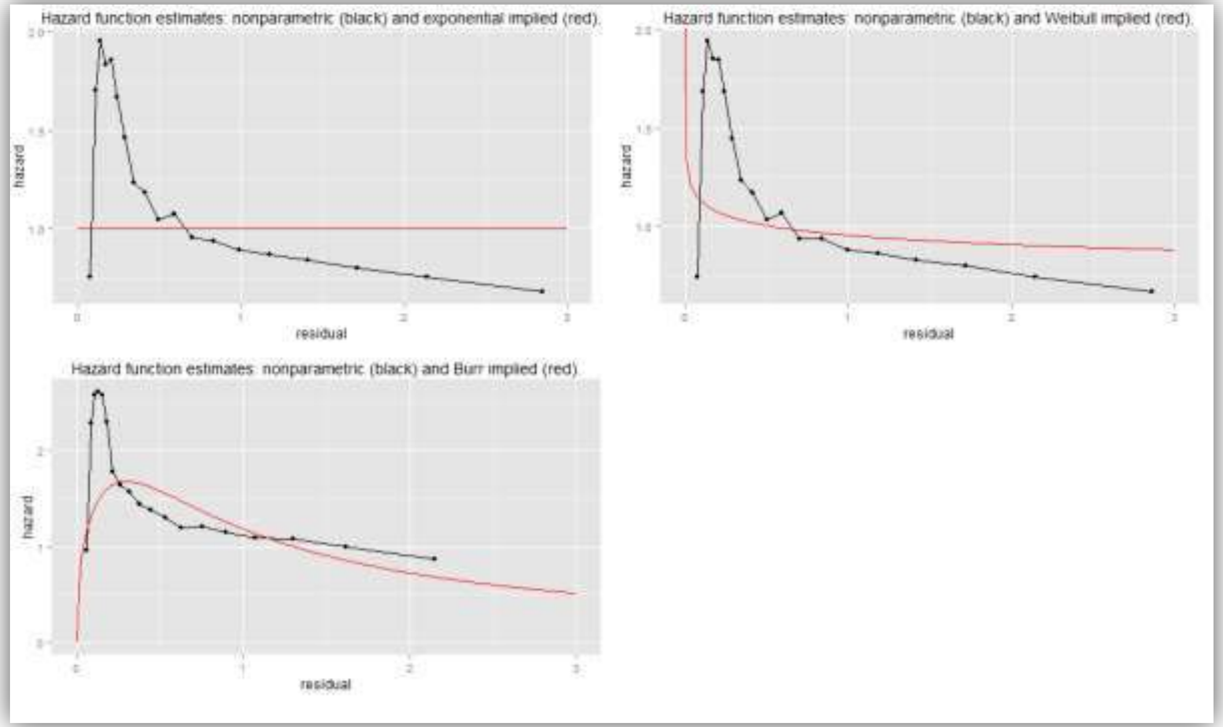


Figure 4 Hazard functions of ACD(1, 1) estimation with various error distributions

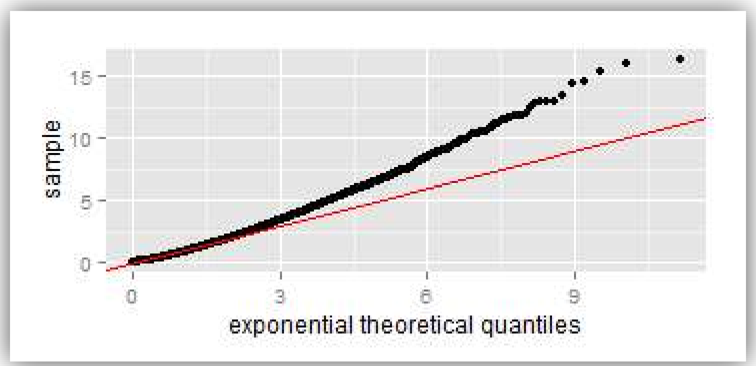
7.4 qqplotAcd()

Another common graphical method of assessing the closeness between a theoretical distribution and an empirical sample is the QQ-plot. Here the quantiles of the sample is plotted against the quantiles of the theoretical quantiles. If the sample is in fact a realization from the theoretical distribution, those should not deviate from each other too much and the plotted curve should be straight.

To get the QQ-plot simply run the code

```
qqplotAcd(fitModel)
```

where fitModel as usual refers to a fitted ACD model. In our example the residual quantiles are plotted against the exponential distribution, since this was the assumed error term distribution in the estimation, as seen in the figure below. The fact that the sample quantiles lies above the straight red line to the right in the figure shows that the empirical distribution has a thicker tail than the exponential distribution.



7.5 Meitz and Teräsvirta's (2006) LM-tests

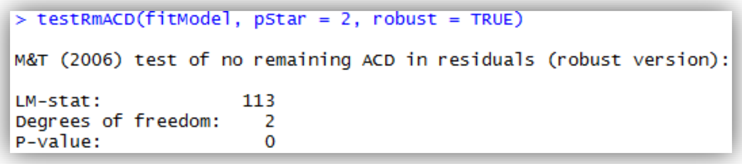
Meitz and Teräsvirta (2006) developed a series of LM-tests (Lagrange Multiplier tests) for ACD-models. A few of those can be run through the ACDm package, as described in the subsequent subchapters. The tests are however currently only available for standard ACD(p, q) conditional duration specification (as the null hypothesis), estimated with QMLE (i.e. estimated with exponential errors).

testRmACD() – test for no remaining ACD

If the model is correctly specified, the error terms are independent and accordingly the residuals should not show any further ACD structured dependency. This is tested with the function `testRmACD()`, a test that is asymptotically equivalent to the Li and Yu test (Meitz and Teräsvirta, 2006). We can run this test on our fitted model by calling

```
testRmACD(fitModel, pStar = 2, robust = TRUE)
```

Here *pStar* is the number of α-parameters in the possibly remaining ACD structure and letting *robust = TRUE* makes the test robust to possible misspecifications of the error term (see Meitz and Teräsvirta, 2006).



```
> testRmACD(fitModel, pStar = 2, robust = TRUE)

M&T (2006) test of no remaining ACD in residuals (robust version):

LM-stat:             113
Degrees of freedom:    2
P-value:               0
```

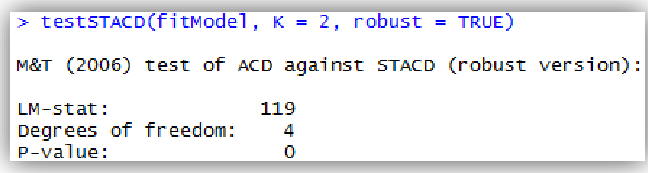
testSTACD() – test against smooth transition ACD models

The linearity imposed by the standard ACD(p, q) model may not be enough to describe the dynamics of durations. Engle and Russell (1998) noticed that the residuals $\hat{\varepsilon}_i$ from their model estimation were too small when the previous durations $\tilde{x}_{i-1}$ were either very small or large (a similar conclusion was made earlier with the plotScatterAcd() function on page 16). A nonlinear model where the ACD parameters are functions of lagged durations might therefore be valid. Meitz and

Teräsvirta (2006) suggested the smooth transition ACD model for this purpose, and derived a LM test testing an ACD(p, q) against this STACD model. The test can be called by:

```
testSTACD(fitModel, K = 2, robust = TRUE)
```

Here K determines the general shape of the transition function (see Meitz and Teräsvirta, 2006). Running the test with our example gives us the output:



```
> testSTACD(fitModel, K = 2, robust = TRUE)

M&T (2006) test of ACD against STACD (robust version):

LM-stat:              119
Degrees of freedom:     4
P-value:                0
```

testTVACD() – test against time varying ACD models

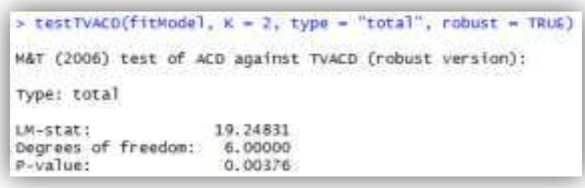
While the STACD model uses the smooth transition function with the transition variable being the lagged durations, one could instead let the transition variable be the time of the transaction, which led Meitz and Teräsvirta's (2006) to propose the time varying ACD model (TVACD). The TVACD model lets the ACD parameters vary over time.

In one specification, the time variable is *total* time, and a test rejecting the null in favor of this alternative specification would indicate that the ACD parameters are changing over time over the total sample.

The other specification lets the parameters be *intraday* varying, by letting the transition variable be the time of the day. Failing this test could indicate that the diurnal adjustment was inadequate at removing any diurnal component. Let's perform these tests by copy-pasting the following two lines to R:

```
testTVACD(fitModel, K = 2, type = "total", robust = TRUE)
testTVACD(fitModel, K = 2, type = "intraday", robust = TRUE)
```

Once again K indicates the general shape of the transition function. The output is shown below:

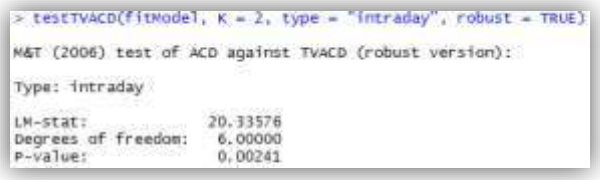


```
> testTVACD(fitModel, K = 2, type = "total", robust = TRUE)

M&T (2006) test of ACD against TVACD (robust version):

Type: total

LM-stat:             19.24831
Degrees of freedom:   6.00000
P-value:              0.00376
```



```
> testTVACD(fitModel, K = 2, type = "intraday", robust = TRUE)

M&T (2006) test of ACD against TVACD (robust version):

Type: intraday

LM-stat:             20.33576
Degrees of freedom:   6.00000
P-value:              0.00241
```

Both tests are rejecting the null, though not as extremely as the previous tests.

8 References

Bauwens, L., and P. Giot (2000) The logarithmic ACD model: an application to the bid-ask quote process of three NYSE stocks. *Annales d'Economie et de Statistique*, 60, 117–149.

Engle, R.F., and Russell, J.R. (1998) Autoregressive Conditional Duration: A New Model for Irregularly Spaced Transaction Data. *Econometrica*, 66(5): 1127-1162

Grammig, J., and Maurer, K.-O. (2000) Non-monotonic hazard functions and the autoregressive conditional duration model. *Econometrics Journal* 3: 16–38.

Hautsch, N. (2012) *Econometrics of Financial High-Frequency Data*. Berlin, Heidelberg: Springer.

Li, W.K. and Yu, P.L.H. (2003) On the residual autocorrelation of the autoregressive conditional duration model. *Economics Letters* 79: 169–175.

Lunde, A. (1999): "A generalized gamma autoregressive conditional duration model," Working paper, Aalborg University.

Meitz, M. and Teräsvirta, T. (2006) Evaluating models of autoregressive conditional duration. *Journal of Business and Economic Statistics* 24: 104–124.

Tsay, R. S. (2001), Analysis of Financial Time Series. Wiley, ISBN 0-471-41544-8.

9 Appendix

9.1 Different ACD conditional duration specifications:

For all of the following models it is assumed that the duration x_i is

$$x_i = \mu_i * \varepsilon_i,$$

where the error term ε_i is I.I.D. and either exponential-, Weibull- or Burr distributed with a mean $E(\varepsilon_i) = 1$. The conditional duration μ_i depends on the specific ACD-model.

ACD(p, q) (Engle and Russell, 1998)

$$\mu_i = \omega + \sum_{j=1}^{p} \alpha_j \, x_{i-j} + \sum_{j=1}^{q} \beta_j \, \mu_{i-j}$$

LACD₁(p, q) (logarithmic ACD type 1) (Bauwens and Giot, 2000)

$$\ln \mu_i = \omega + \sum_{j=1}^{p} \alpha_j \ln \varepsilon_{i-j} + \sum_{j=1}^{q} \beta_j \ln \mu_{i-j}$$

LACD₂(p, q) (logarithmic ACD type 2) (Lunde, 1999)

$$\ln \mu_i = \omega + \sum_{j=1}^{p} \alpha_j \, \varepsilon_{i-j} + \sum_{j=1}^{q} \beta_j \ln \mu_{i-j}$$

AMACD(p, r, q) (Additive and Multiplicative ACD) (Hautsch , 2012)

$$\mu_i = \omega + \sum_{j=1}^{p} \alpha_j \, x_{i-j} + \sum_{j=1}^{r} \nu_j \, \varepsilon_{i-j} + \sum_{j=1}^{q} \beta_j \, \mu_{i-j}$$

ABACD(p, q) (Augmented Box-Cox ACD) (Hautsch, 2012)

$${\mu_i}^{\delta_1} = \omega + \sum_{j=1}^{p} \alpha_j \left(\left|\varepsilon_{i-j} - \nu\right| + c_j\left|\varepsilon_{i-j} - b\right|\right)^{\delta_2} + \sum_{j=1}^{q} \beta_j \, {\mu_{i-j}}^{\delta_1}$$

SNIACD(p, q, M) (Spline News Impact ACD) (Hautsch, 2012, with a slight difference)

$$\mu_i = \omega + \sum_{j=1}^{p} (\alpha_j + c_0) \, \varepsilon_{i-j} + \sum_{j=1}^{p} \sum_{k=1}^{M} (\alpha_j + c_k) \, \mathbb{1}\left(\varepsilon_{i-j} \le \bar{\varepsilon}_k\right)\left(\varepsilon_{i-j} - \bar{\varepsilon}_k\right) + \sum_{j=1}^{q} \beta_j \, \mu_{i-j},$$

where $\alpha_1 = 0$, $\mathbb{1}(\cdot)$ is an indicator function and $\{\bar{\varepsilon}_1, \bar{\varepsilon}_2, \dots, \bar{\varepsilon}_M\}$ is a set of breakpoints.

Starting parameters format is $\left(\omega,\ c_0,\ \dots,\ c_M,\ \alpha_2,\ \dots,\ \alpha_p,\ \beta_q,\ \dots,\ \beta_q\right)$

9.2 Error distributions

Weibull

$\varepsilon \sim W(\theta, \gamma), \qquad \theta, \gamma > 0$

$$f_\varepsilon(\varepsilon) = \theta\gamma\varepsilon^{\gamma-1}e^{-\theta\varepsilon^\gamma}$$

$$E(\varepsilon) = \theta^{-\gamma^{-1}}\Gamma(\gamma^{-1}+1)$$

Forcing $E(\varepsilon) = 1$ gives:

$$\theta = [\Gamma(\gamma^{-1}+1)]^\gamma$$

Hazard function:

$$h(\varepsilon) = \theta\gamma\varepsilon^{\gamma-1}$$

Burr (as in Grammig and Maurer, 2000)

$\varepsilon \sim Burr(\theta,\kappa,\sigma^2), \qquad \theta,\kappa,\sigma^2 > 0, \quad \kappa > \sigma^2$

$$f_\varepsilon(\varepsilon) = \frac{\theta\kappa\varepsilon^{\kappa-1}}{(1+\sigma^2\theta\varepsilon^\kappa)^{\frac{1}{\sigma^2}+1}}$$

$$E(\varepsilon^s) = \theta^{-\frac{s}{\kappa}} \times \frac{\Gamma\left(1+\frac{s}{\kappa}\right)\Gamma\left(\frac{1}{\sigma^2}-\frac{s}{\kappa}\right)}{\sigma^{2\left(1+\frac{s}{\kappa}\right)}\Gamma\left(\frac{1}{\sigma^2}+1\right)}$$

Forcing $E(\varepsilon) = 1$ gives:

$$\theta = \left(\frac{\Gamma\left(1+\frac{1}{\kappa}\right)\Gamma\left(\frac{1}{\sigma^2}-\frac{1}{\kappa}\right)}{\sigma^{2\left(1+\frac{1}{\kappa}\right)}\Gamma\left(\frac{1}{\sigma^2}+1\right)}\right)^\kappa$$

Generalized Gamma

$\varepsilon \sim GG(\gamma,\kappa,\lambda), \qquad \varepsilon,\gamma,\kappa,\lambda > 0$

$$f_\varepsilon(\varepsilon) = \frac{\gamma\varepsilon^{\kappa\gamma-1}}{\lambda^{\kappa\gamma}\Gamma(\kappa)}\exp\left\{-\left(\frac{\varepsilon}{\lambda}\right)^\gamma\right\}$$

$$E(\varepsilon^s) = \lambda^s\frac{\Gamma(\kappa+s/\gamma)}{\Gamma(\kappa)}$$

Forcing $E(\varepsilon) = 1$ gives:

$$\lambda = \frac{\Gamma(\kappa)}{\Gamma\left(\kappa+\frac{1}{\gamma}\right)}.$$

Log likelihood for the full ACD model:

$$l_i = -\ln(\mu_i) + \ln(\gamma) + (\kappa\gamma - 1)\ln(\varepsilon_i) + (\kappa\gamma)\ln(\lambda) - \ln\big(\Gamma(\kappa)\big) - \left(\frac{\varepsilon_i}{\lambda}\right)^{\gamma},$$

where $\varepsilon_i = x_i/\mu_i$ and λ is defined as above.

Generalized F (Hautsch, 2012)

$\varepsilon \sim GG(\kappa, \eta, \gamma, \lambda), \qquad \varepsilon, \kappa, \eta, \gamma, \lambda > 0$

$$f_\varepsilon(\varepsilon) = \frac{\gamma \varepsilon^{\kappa\gamma-1}[\eta + (\varepsilon/\lambda)^{\gamma}]^{-\eta-\kappa}\eta^{\eta}}{\lambda^{\kappa\gamma}B(\kappa,\eta)},$$

where $B(\kappa,\eta) = \frac{\Gamma(\kappa)\Gamma(\eta)}{\Gamma(\kappa+\eta)}$.

$$E(\varepsilon^s) = \lambda^s \eta^{s/\gamma}\frac{\Gamma(\kappa + s/\gamma)\Gamma(\eta - s/\gamma)}{\Gamma(\kappa)\Gamma(\eta)}$$

Forcing $E(\varepsilon) = 1$ gives:

$$\lambda = \frac{\Gamma(\kappa)\Gamma(\eta)}{\eta^{1/\gamma}\Gamma(\kappa + 1/\gamma)\Gamma(\eta - 1/\gamma)}.$$

Log likelihood for the full ACD model:

$$l_i = -\ln(\mu_i) + \ln(\gamma) + (\kappa\gamma - 1)\ln(\varepsilon_i) + (-\eta - \kappa)\ln\left(\eta + \left(\frac{\varepsilon_i}{\lambda}\right)^{\gamma}\right) + \eta\ln(\eta) - (\kappa\gamma)\ln(\lambda) - \ln\big(B(\kappa,\eta)\big),$$

where $\varepsilon_i = x_i/\mu_i$ and λ is defined as above.

q-Weibull

$\varepsilon \sim qW(a, q, b), \qquad \varepsilon, a, b > 0, \qquad 1 < q < 2$

$$f_\varepsilon(\varepsilon) = (2-q)\frac{a}{b^a}\varepsilon^{a-1}\exp_q\left[-\left(\frac{\varepsilon}{b}\right)^{a}\right],$$

where $\exp_q[x] = [1 + (1-q)x]^{1/(1-q)}$.

Forcing $E(\varepsilon) = 1$ gives:

$$b = \frac{(q-1)^{\frac{1+a}{a}}}{2-q}\frac{\mathrm{a}\Gamma\left(\frac{1}{q-1}\right)}{\Gamma\left(\frac{1}{a}\right)\Gamma\left(\frac{1}{q-1} - \frac{1}{a} - 1\right)}.$$

Log likelihood for the full ACD model:

$$l_i = -\ln(\mu_i) + (a-1)\ln(\varepsilon_i) + \frac{1}{1-q}\ln\left[1 - (1-q)\left(\frac{\varepsilon_i}{b}\right)^{a}\right] + \ln\left[\frac{a(2-q)}{b^a}\right],$$

where $\varepsilon_i = x_i/\mu_i$ and b is defined above.