CHALMERS | UNIVERSITY OF GOTHENBURG

Financial Time Series TMS088/MSA410 – LP4 2020/21

Project 2: Some tips if you plan on using R

A few words on fitting GARCH models using R

There are few options when it comes to fitting GARCH models on data[1]. But I strongly advise you to use the R package `rugarch` as it is actively maintained and a great deal of documentation is available for it[2]. This package allows to fit a great variety of GARCH models. We will focus however on the standard GARCH(p,q) models, which are defined for $p, q \geq 0$ as:

$$X_t = \mu + \sigma_t Z_t, \tag{1}$$

with

$$Z \sim IID(0,1)$$

and

$$\sigma_t^2 = \omega + \sum_{j=1}^{p} \alpha_j (X_{t-j} - \mu)^2 + \sum_{k=1}^{q} \beta_k \sigma_{t-k}^2 \tag{2}$$

where the coefficients μ, $\omega > 0$, $\alpha_j \geq 0$, $\beta_k \geq 0$ will be determined by the fit. Note in particular that taking $p = 0$ (resp. $q = 0$) will correspond to remove the sum over j (resp. over k) from the definition of σ_t^2.

Performing a fit is done in two steps:

1. Specify the GARCH(p,q) model you want to fit on your data using the function `ugarchspec`. You have to specify both a GARCH model for the variance term σ_t^2 and an ARMA model for the mean term μ. Since we want μ to be a constant the mean model should be included as an ARMA(0,0) process. Here is an example for the specification of a GARCH(2,3) model:

```
GFSpec=ugarchspec(variance.model = list(model="sGARCH",garchOrder=c(2,3)),
                  mean.model = list(armaOrder=c(0,0),include.mean=TRUE))
```

Note: the function `ugarchspec` also has an argument that allows to set the distribution of the terms Z_t (default is Gaussian distribution). See the help of this function for more information.

2. To fit the model you just specified to the dataset `Xtr`, use the function `ugarchfit`. In particular, I advise you to use the solver `nloptr` when fitting your model, with the parameters given below. Here is an example of the resulting call to the fiiting function:

```
GF=ugarchfit(spec=GFSpec, data = Xtr,
             solver="nloptr", # Solver choice
             solver.control = list(xtol_rel=1e-6,ftol_rel=1e-6,solver=3)) # Solver parameters
```

This line of code stores in the variable `GF` a special object that contains all the information and the results from the fit. These can be extracted by calling respectively `GF@model` and `GF@fit` or by using dedicated functions (see the help of the function `ugarchfit` and of the object class `uGARCHfit`).

[1] cf. this blog post.

[2] See the reference manual and the vignette here.

A few words on fitting ARMA models using R

You can once again use the `rugarch` package to fit your ARMA models. We will focus however on the standard ARMA(p,q) models, which are defined for $p, q \geq 0$ as:

$$(X_t - \mu) - \sum_{j=1}^{p} \alpha_j (X_{t-j} - \mu) = Z_t + \sum_{k=1}^{q} \beta_k Z_{t-k}, \tag{3}$$

with

$$Z \sim WN(0, \sigma^2)$$

where the coefficients μ, $\omega > 0$, $\alpha_j \geq 0$, $\beta_k \geq 0$ will be determined by the fit.

Performing a fit is done in two steps:

1. Specify the ARMA(p,q) model you want to fit on your data using the function `arfimaspec`. Here is an example for the specification of a ARMA(2,3) model with a possibly non-zero mean μ:

```
GFSpec=arfimaspec(mean.model = list(armaOrder=c(2,3),include.mean=TRUE,arfima=FALSE))
```

Note: the function `arfimaspec` also has an argument that allows to set the distribution of the terms Z_t (default is Gaussian distribution). See the help of this function for more information.

2. To fit the model you just specified to the dataset `Xtr`, use the function `ugarchfit`. In particular, I advise you to use the solver `nloptr` when fitting your model, with the parameters given below. Here is an example of the resulting call to the fitting function:

```
GF=arfimafit(spec=GFSpec,data = Xdat,
             solver="nloptr", # Solver choice
             solver.control = list(xtol_rel=1e-6,ftol_rel=1e-6,solver=3)) # Solver parameters
```

This line of code stores in the variable `GF` a special object that contains all the information and the results from the fit. These can be extracted by calling respectively `GF@model` and `GF@fit` or by using dedicated functions (see the help of the function `arfimafit` and of the object class `ARFIMAfit`).

Some useful functions in R

- `residuals` - Compute the residuals associated with a ARMA model (the option `standardize` allws you to standardize them).
- `ugarchforecast` - Computes a forecast of the value and the conditional variance of a GARCH-model.
- `qnorm` - Finds critical values of the normal distribution.
- `qt` - Finds critical values of the t-distribution.
- `acf` - Computes the sample ACF.
- `pacf` - Computes the sample PACF.
- `tryCatch` - Useful for catching errors thrown by estimate in loops.
- `which(M == min(M), arr.ind = TRUE)` - Returns the row and column indices of the smallest entry of a matrix `M`.
- Finally, here are a few lines of code allowing you to compute the QQ-plot of a dataset with respect to a given distribution.

```
### qqplotfunc creates the QQ-plot of a dataset with respect to a given distribution.

### Arguments
```

```r
## Xdat : Vector containing the data for which you want to compute a QQ-plot

## distrQuant : R function returning the quantiles of a distribution,
#               eg. qnorm for a Normal distribution, or qt for a t-Student distribution

## ... : Replace by additional parameters needed by the quantile function.
#        For instance, replace by the argument df of the qt function when you
#        work with a t-Student distribution

qqplotfunc<-function(Xdat,distrQuant,...){

  ## Create a vector containing the values at which the quantiles
  #  will be evaluated
  npts=length(Xdat)
  qvec=seq(from=0,to=1,length.out = npts)[-c(1,npts)]

  ## Plot theoretical quantile VS sample quantiles
  plot(distrQuant(qvec,...), quantile(Xdat,qvec),
       main=paste0("QQ-Plot"),
       xlab="Theoretical Quantiles",ylab="Sample Quantiles",
       pch=19,cex=0.75)
}

#### Example: QQ-Plot for a vector X with respect to the t-Student distribution
#             with 4 degrees of freedom

## Create some data
X=rnorm(1000)
## Compute the QQ-Plot
qqplotfunc(Xdat=X,distrQuant=qt,df=4)
```