ARMA(1,1)-GARCH(1,1) Estimation and forecast using rugarch 1.2-2

Jesper Hybel Pedersen

8. juni 2013

1 Introduction

First we specify a model ARMA(1,1)-GARCH(1,1) that we want to estimate. Secondly we touch upon the matter of fixing certain paramters of the model. Thirdly we get som data to estimate the model on and estimate the model on the data. After having estimated the model we inspect the created R-object from the fitting of the model.

Then we use the model for making a forecast: 1) A simple forecast and 2) a rolling forecast and 3) a rolling forecast with reestimation of model.

Throughout I make some comments on calculation of VaR.

Reference manual, source code and a *very helpful* vignette is available at:

http://cran.r-project.org/web/packages/rugarch/index.html

I refer to the referencemanual and vignette when relevant so if not for any other reason - although other reasons are plenty - it is a good idea to download and read these.

The author of the rugarchpackage Alexios Ghalanos has a blog:

http://www.unstarched.net/blog/

Im using rugarch: Univariate GARCH models R-package version 1.2-2 by Alexios Ghalanos.

2 Modelspecification - »uGARCHspec«

To fit a GARCH-model the first step is to create an instance of the S4-class »uGARCHspec«. The created object serves the purpose of specifying the model to be estimated. The chosen model can be estimated with all parameters free or alternatively with some parameters fixed.

2.1 Modelspecification (free parameters only)

The »uGARCHspec« class is documented on page 86 of the rugarch referencemanual and the usual helpage is found typing `?ugarchspec` in R-console. On the help page we find the following peace of code:

```
model=ugarchspec(
variance.model = list(model = "sGARCH", garchOrder = c(1, 1)),
mean.model = list(armaOrder = c(1, 1), include.mean = TRUE),
distribution.model = "norm"
)
```

For simplicity I have left out some options we are not going to use which are explicited on the helppage. Also I have added the `model=...` so that our instance of the »uGARCHspec« class i called `model`.

So when creating the `model` we specify the `variance.model`, the `mean.model` and the `distribution.model`. Notice that the first two arguments `mean.model` and `variance.model` takes arguments that are objects of the type `list`.

We have set the argument `model` in the `variance.model` to have the value `"sGARCH"` with `garchOrder=c(1,1)` so that our choice of variancemodel is:

$$\sigma_t^2 = \omega + \alpha\epsilon_{t-1}^2 + \beta\sigma_{t-1}^2 \tag{1}$$

as explained in the vignette on page 6 with `p=1`, `q=1` and `m=0` since we are ignoring external regressors.

The meanmodel is chosen to have `armaOrder=c(1,1)` and we include a mean - constant - by `mean=TRUE` so that our chosen model is:

$$r_t = \mu + \theta_1(r_{t-1} - \mu) + \theta_2\epsilon_{t-1} + \epsilon_t \tag{2}$$

When specifying the distribution we have chosen a normaldistribution using the argument `"norm"` so:

$$\epsilon_t = \sigma_t z_t \qquad z_t \sim N(0,1) \tag{3}$$

often one would choose other distributions due to the stylished fact of financial series having fat tails. One option would be to use the Student-t distribution `distribution.model="std"` or a skewed version of it `"sstd"`. The conditional distributions are discussed on page 13 of the vignette.

2.2 Modelspecification (fixed parameters)

We might want to fix some or all of the parameters in the model. We could fix some of the parameters and estimate the rest for example with the purpose of making likelyhood ratio comparison. We could also fix all parametervalues with the purpose of using the model for forecasting.

To fix the parameters we need to use the argument `fixed.pars = list()` in the `ugarchspec()` see helppage `?ugarchspec`. We also need to find the name of the parameter we want to fix. If you have chosen a modeltype for example the `sGARCH` in our case you can find the name of the relevant parameters by typing `spec@model.pars`. You need to substitute `spec` with the name you gave to your instance of the uGARCHspec class, which in our case was `model` so we simply type `model@model.pars`.

From the top: First we choose the sGARCH-model but lets try it with higher order ARMA and GARCH and a skewed version og the Student-t:

```
model2=ugarchspec(
variance.model = list(model = "sGARCH", garchOrder = c(2, 2)),
mean.model = list(armaOrder = c(2, 2), include.mean = TRUE),
distribution.model = "sstd")
```

Then to inspect the paramternames:

```
model2@model$pars
```

And you should get a printout similar to the one in tabel 1. From the table the names of the parameters are available in the first column and the fourth column takes the value `1` where the relevant parameter is included `0` otherwise. The fifth column - called »Estimate« - indicates whether the parameter is estimated or fixed. So β in our GARCH-model when the `garchOrder=c(1,1)` is probably called `beta1` if there is any justice in the world. So to fix the parameter we do:

```
model=ugarchspec(
variance.model = list(model = "sGARCH", garchOrder = c(1, 1)),
mean.model = list(armaOrder = c(1, 1), include.mean = TRUE),
distribution.model = "norm"
fixed.pars = list(beta1=0.86) )
```

You can try `model@model$pars` to see if the relevant parameter has been set to the chosen value.

Tabel 1: R-print from `model2@model$pars`

	Level	Fixed	Include	Estimate	LB	UB
mu	0	0	1	1	NA	NA
ar1	0	0	1	1	NA	NA
ar2	0	0	1	1	NA	NA
ma1	0	0	1	1	NA	NA
ma2	0	0	1	1	NA	NA
arfima	0	0	0	0	NA	NA
archm	0	0	0	0	NA	NA
mxreg	0	0	0	0	NA	NA
omega	0	0	1	1	NA	NA
alpha1	0	0	1	1	NA	NA
alpha2	0	0	1	1	NA	NA
beta1	0	0	1	1	NA	NA
beta2	0	0	1	1	NA	NA
gamma	0	0	0	0	NA	NA
eta1	0	0	0	0	NA	NA
eta2	0	0	0	0	NA	NA
delta	0	0	0	0	NA	NA
lambda	0	0	0	0	NA	NA
vxreg	0	0	0	0	NA	NA
skew	0	0	1	1	NA	NA
shape	0	0	1	1	NA	NA
ghlambda	0	0	0	0	NA	NA

3 Getting data

Before we can estimate the model we have to get som data. You can use the SP500 index which is included in the rugarchpackage simply type:

```
data(sp500ret)
```

and then the sp500-index is loaded and is called `sp500ret`. Alternatively you can load data from Yahoo using the `get.hist.quote()` function from the package `tseries`:

```
library(tseries)
sp500.prices=get.hist.quote(

instrument = "^GSPC",
quote = "Adj",
provider = c("yahoo"), method = NULL,
origin = "1899-12-30", compression = "d",
retclass = c("zoo"), quiet = FALSE, drop = FALSE
)
sp500=as.data.frame(sp500.prices)
N=length(sp500[,1])
sp500.returns=100*(log(sp500[2:N,])-log(sp500[1:(N-1),]))
```

This downloads the adjusted closing prices - adjusted for dividends - and calculates continously compounded returns.

4 Estimating the model

To estimate the model we use the `model`-object created and the data `sp500ret` and provide them as arguments to the `uGARCHfit()` function:

```
modelfit=ugarchfit(spec=model,data=sp500ret)
```

This creates an object we have chosen to call `modelfit` which is an instance of the S4 class »uGARCHfit« specific to the rugarchpackage as documented on page 65 of the rugarch reference manual.

This object is far more interesting in than the `model` in the sence that it contains information we need to be able to manipulate and as a minimum print. First of all simply typing the object names prints the parameterestimates of the model along with standard errors and a lot of interesting statistics:

```
modelfit
```

I'm not going to delve on these statistics they are explained better elsewhere. References for the relevant litterature can be found in the rugarchvignette. However I will make a comment on S4-objects. Just to repeat: Estimating the model creates the object we have chosen to call `modelfit` as an instance of the class »uGARCHfit« to get the class type:

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

Another useful feature is that you can get the structure of the object by typing `str(modelfit)`. This results in a somewhat »messy« print but if you look for the `@` you should be able to spot the following structure in the printout:

```
Formal class 'uGARCHfit' [package "rugarch"] with 2 slots
..@ fit
..@ model
```

Since we have an S4 object the object has what is called `slots` and in this case there are 2 slots the first is called `fit` the second `model`. This is also apparent typing `slotNames(modelfit)`. Lets think of the slots as places to store data and in each of the slots there are »containers« where every container is indicated by `$` (the containers are offcourse R-objects like vectors, matrices or lists). Reading the print from `str(modelfit)` you will notice that the first `$` is followed by `hessian`:

```
Formal class 'uGARCHfit' [package "rugarch"] with 2 slots
..@ fit :List of 25
.. ..$ hessian
```

This is interesting because it tells you how to access the information stored in the object by typing:

```
modelfit@fit$hessian
```

You could get the parameterestimates by typing:

```
modelfit@fit$coef
```

The fitted values:

```
modelfit@fit$fitted.values
```

The point is that how to access the data is indicated from the `str(modelfit)` printout. However you could also use the designed *methods* which are documented on page 65 of the reference manual:

```
coef(modelfit)
infocriteria(modelfit)
sigma(modelfit)
fitted(modelfit)
residuals(modelfit)
```

To name a few. Be sure to update R and rugarch package since the available methods possibly change from version to version (I had problems with the method quantile used when calculating VaR). To calculate VaR use the quantile method:

```
VaR=quantile(modelfit,0.01)
```

or in our case with standardized normal distribution we could do:

```
VaR=modelfit@fit$sigma*qnorm(0.01)+modelfit@fit$fitted.values
```

Since the model is estimated on the observations from the periods for which we calculate VaR this is called in-sample VaR. As such it may seem uninteresting since what would be the purpose of calculating a riskmeasure of an already realized contingency? Hence later we do forecasting.

5 Forecasting

Having estimated a model we might want to use it for forecasting. Reading the helpfile `?ugarchforecast` it says:

```
ugarchforecast(fitORspec, data = NULL, n.ahead = 10, n.roll = 0,
out.sample = 0)
```

So the first argument is an estimated model - `fit` - or a specified model - `spec`. Using a specified model all parametervalues need to be fixed and the `data` argument cannot be `NULL`. Using a fitted object the data is contained in the fitted object and hence do not need to be supplied, so the `data=NULL` can be used to forecast from the same series as used in estimation.

In our case the fitted model `modelfit` is used for forecasting. First lets do a simple forecast:

```
modelfor=ugarchforecast(modelfit, data = NULL, n.ahead = 10, n.roll
= 0, out.sample = 0)
```

The last observation of the used dataset has the date 2009-01-30 and this is T_0 in the forecast:

```
0-roll forecast [T0=2009-01-30]:
      Series Sigma
T+1 0.0016724 0.02480
T+2 0.0015259 0.02474
T+3 0.0013981 0.02467
T+4 0.0012865 0.02461
T+5 0.0011891 0.02455
T+6 0.0011041 0.02448
T+7 0.0010299 0.02442
T+8 0.0009651 0.02436
T+9 0.0009086 0.02430
T+10 0.0008592 0.02424
```

The object `modelfor` is an instance of the class »uGARCHforecast« page 71 of the referencemanual. The object has two slots with names:

```
slotNames(modelfor)
# [1] "forecast" "model"
```

You can access the forecast of the series with method `fitted(modelfor)` and the sigma (the conditional standard deviation) as `sigma(modelfor)`. Also available are some plots `plot(modelfor)` and then select an appropriate option.

We could forecast with a larger `n.ahead=50`

```
modelfor=ugarchforecast(modelfit, data = NULL, n.ahead = 50, n.roll
= 0, out.sample = 0)
```

An using `str(modelfor)` we can see that the adress for the seriesforecast is `modelfor@forecast$seriesFor` and we can do a simple plot:

```
plot(modelfor@forecast$seriesFor)
```

exhibiting exponential decay to unconditional mean `modelfit@fit$coef["mu"]` or by method `uncmean(modelfit)`.

5.1 Rolling forecast

Using a model for forecast it might be interesting to see how well the model forecasts so we need som observations to compare with the forecasted values. For this reason it is possible to fit the model with the argument `out.sample` different than 0. In this case it becomes possible to do rolling 1-step-head forecasts. To use this option we have to estimate the model *not on all observations* hence we use the `out.sample` argument. For simplicity I choose the `out.sample=2` and I choose a GARCH-model with a simpler meanequation ARMA(0,0):

```
model=ugarchspec (
variance.model = list(model = "sGARCH", garchOrder = c(1, 1)),
mean.model = list(armaOrder = c(0, 0)),
distribution.model = "norm"
)
modelfit=ugarchfit(model,data=sp500ret,out.sample=2)
```

So fx. if we have 500 observations we estimate the model on first 400 since the `out.sample=100`. With the dataset `sp500ret` we have $N = 5523$ observations so the index of the last observation on which the model is estimated on is $T = 5521$:

```
modelfit@model$modeldata$T
# 5521
```

Then we do the forecasts:

```
modelfor=ugarchforecast(modelfit, data = NULL, n.ahead = 1, n.roll
= 2, out.sample = 2)
```

and we look at the forecasted values:

```
sigma(modelfor)
#      2009-01-28 2009-01-29 2009-01-30
# T+1 0.02413003 0.02512952 0.02492642
fitted(modelfor)
#       2009-01-28   2009-01-29   2009-01-30
# T+1 0.0005205228 0.0005205228 0.0005205228
```

The values are forecasted values based on the information of the listed date, *the values are not forecasts for the expected value of the specific date.* Observation $T = 5521$ is the T_0 observation which in `sp500ret` has the date `2009-01-28` and being part of the sample on which the model is estimated there can be no forecast for the date (following conventional terminology and only calling something a forecast if it is out-of-sample).

Anyway since we have a simple model it should be easy to control for this by manual calculation. The mean is constant in our model so the forecast of the returnseries for any period is simply the estimated constant mean `coef(modelfit)["mu"]`. For the period $T = 5521$ we have an estimated residual `residuals(modelfit)[5521]` but for the two periods for which we are forecasting we lack these but can calculate them comparing forecast of the series - the constant mean - to the observed return. With these residuals we can forecast the sigmas.

We could redo the forecasting manually along above lines with the following script:

```
#Getting the expected return from the estimated model
mu=coef(modelfit)["mu"]
#Getting the relevant observed return from last
# two periods of sp500ret
return=sp500ret[5522:5523,]
#Getting residual of period T=5521 from modelfit
e5521=as.vector(residuals(modelfit)[5521])
#Calculating residuals of period T=5522 and 5523
e5522=return[1]-mu
e5523=return[2]-mu
#Taking estimated parameters from modelfit
theta=coef(modelfit)
#Making function for forecast
.fgarch=function(e,sigma_0,theta) {
                    omega=theta["omega"]
                    alpha=theta["alpha1"]
                    beta=theta["beta1"]
                    sigma_1 = sqrt(omega + alpha*e^2 + beta*sigma_0^2)
                    names(sigma_1)=
                    return(sigma_1)
                    }
#Getting estimated sigma for period T=5521 from modelfit
sigma5521=as.vector(sigma(modelfit)[5521])
#Forecast sigma_5522 and comparing with rugarch forecast
sigma5522=.fgarch(e5521,sigma_0=sigma5521,theta)
sigma5522
sigma(modelfor)[1]
#Forecast sigma_5523 and comparing with rugarch forecast
sigma5523=.fgarch(e5522,sigma_0=sigma5522,theta)
sigma5523
sigma(modelfor)[2]
#Forecast sigma_5523 and comparing with rugarch forecast
sigma5524=.fgarch(e5523,sigma_0=sigma5523,theta)
sigma5524
sigma(modelfor)[3]
```

And we are able to get the same values manually as using the rugarch function `ugarchforecast`. Following the script hopefully makes it clear what is being reported calling `sigma(modelfor)` and `fitted(modelfor)` if that was somehow unclear to begin with.

If the meanequation was ARMA(1,1) it would be a little more complicated due to the expected return of the next period not simply being a constant. First we calculate the variance for T_1:

$$\sigma^2_{T_1} = \hat{\omega} + \hat{\alpha}\epsilon^2_{T_0} + \hat{\beta}\sigma^2_{T_0} \tag{4}$$

and we forecast the returnseries using the meanmodel:

$$r_{T_1} = \hat{\mu} + \hat{\theta}_1(r_{T_0} - \hat{\mu}) + \hat{\theta}_2\epsilon_{T_0} \tag{5}$$

Comparing the predicted return r_{T_1} to the observed r_{+1} we can calculate a new residual $\epsilon_{T_1} = r_{+1} - r_{T_1}$. And given this residual we can calculate a new 1-step-head forecast for period T_2 for both the sigma and the returnseries. With the forecast r_{T_2} caculated we can calculate $\epsilon_{T_2} = r_{+2} - r_{T_2}$ and so on by iteration. Since we constantly allow ourselves to use last periods return in forecasting the return for the next period we say `n.ahead=1`, hence conditioning on the information Ω_{t-1}.

Since we have calculated sigmas out-of-sample we could use these to calculate VaR out of sample. Since the density of our model is normal the 1-step-head density is normal with the forecasted standard deviation given that the model is valid. To calculate VaR - using the simple GARCH model with the constant meanequation - we would do:

```
qnorm(0.05)*sigma(modelfor)+coef(modelfit)["mu"]
```

or using the built in function of the rugarchpackage:

```
quantile(modelfor,0.05)
```

If not using the model with the constant mean but instead using the ARMA(1,1) for meaneqation (or another meaneqaution for that matter) do:

```
qnorm(0.05)*sigma(modelfor)+fitted(modelfor)
```

or using the built in function of the rugarchpackage:

```
quantile(modelfor,0.05)
```

Having used the forecast for a period where actual observations are available we gain two new plot options using: `plot(modelfor)` where we now can do the `rolling` options.

6 Rolling forecast with reestimation

If we want to know how well our model does VaR calculation we would probably want to calculate VaR out of sample since this mimics the actual practice closer. Calculating VaR on a dayli basis it probably would not be a problem to reestimate the model every day hence reestimating for each 1-step-head VaR calculation. However if we want to leave 1000 observations out of sample to calculate 1000 out of sample VaR measures we would have to reestimate the model 1000 times and this would be computationally heavy and timecomsuming. On the other hand simply calculating 1000 VaR measures without reestimation would probably not mimic atual practice and would due to changing parameters give bad VaR estimates.

A compromize using the `sp500ret` would be to `out.sample=1000` hence first estimating the model on $5523-1000 = 4523$ observations. Then calculate 50 1-step-ahead forecasts and out-of-sample VaR estimates and the reestimate the model on the observations $1, 2, ..., 4535, ..., 4583$. Based on the new parameterestimates we could then do yet another 50 1-step-ahead forecasts contnuing like this until we have forecasts and VaR calculated for all 1000 out of sample observations.

6.1 Doing a rolling forecast

The rugarchpackage allows this to be done quite easily using the `ugarchroll`. Lets try doing this with the ARMA(1,1)-GARCH(1,1)-model leaving only 100 observations out-of-sample and refitting every 50 observations so as not to be overwhelmed by the complexity of the returned object:

```
model=ugarchspec (
   variance.model = list(model = "sGARCH", garchOrder = c(1, 1)),
   mean.model = list(armaOrder = c(1, 1)),
   distribution.model = "norm"
)
modelroll=ugarchroll (
   spec=model, data=sp500ret, n.ahead = 1, forecast.length = 100,
   n.start = NULL, refit.every = 50, refit.window = c("recursive"),
   window.size = NULL, solver = "hybrid", fit.control = list(),
   solver.control = list(), calculate.VaR = TRUE, VaR.alpha = c(0.01,
   0.05),
   cluster = NULL, keep.coef = TRUE
)
```

So we choose the model using `ugarchspec` and give this as an argument value to `ugarchroll` by `spec=model`. Since the model has not been estimated we have to decide what data to use `data=sp500ret`. We use the `n.ahead=1` and as said on the `?ugarchroll` helppage » Only rolling 1-ahead forecasts are supported spanning the dataset, which should be useful for backtesting models. Anything more complicated should be wrapped by the user by making use of the underlying functions in the package.«

Lets have a look at the returned object:

```
slotNames(modelroll)
#[1] "model"  "forecast"
```

Object returned has two slots and we take a closer look on the second:

```
str(modelroll@forecast)
#List of 2
# $ VaR :'data.frame': 100 obs. of 3 variables:
# ..$ alpha(1%): num [1:100] -0.032 -0.0378 -0.0364 -0.036 -0.0345
# ..$ alpha(5%): num [1:100] -0.0224 -0.0262 -0.0253 -0.0252 -0.0241
# ..$ realized : num [1:100] -0.03473 0.00613 0.01371 0.00212 -0.04828
# $ density:'data.frame': 100 obs. of 6 variables:
# ..$ Mu : num [1:100] 0.000792 0.001651 0.001418 0.001015 0.000935
# ..$ Sigma : num [1:100] 0.0141 0.0169 0.0162 0.0159 0.0152
# ..$ Skew : num [1:100] 0 0 0 0 0 0 0 0 0 0
# ..$ Shape : num [1:100] 0 0 0 0 0 0 0 0 0 0
# ..$ Shape(GIG): num [1:100] 0 0 0 0 0 0 0 0 0 0
# ..$ Realized : num [1:100] -0.03473 0.00613 0.01371 0.00212 -0.04828
```

We see that we get some obejcts with 100 recorde values since we left 100 observations out of sample. We get the Value at Risk measure for the long position $\alpha = 0.01$ and $\alpha = 0.05$ becayse we chose `calculate.VaR = TRUE, VaR.alpha = c(0.01,0.05)` in the `ugarchroll()` call. Also included in the VaR-data-frame is the realized values which is the actual returns from the period associated with the calculated VaR. Nice feature when we have to calculate whether or not a VaR-violation has occured - a VaR-violation being defined as an instance where the realized return is smaller than the VaR-measure (long position). Lets just calculate the rate of VaR-violations for:

```
VaR=modelroll@forecast$VaR[,"alpha(1%)"]
return=modelroll@forecast$VaR[,"realized"]
Hit=return<VaR
sum(Hit)
#5
```

So there are 5 VaR-violations out of 100 observations which is exactly the 5% expected given a $\alpha = 0.05$ (so the model would pass the Kupiec test at least for $\alpha = 0.05$).

We also get a density data-frame where the values of the forecasted expected returned is stored under `modelroll@forecast$density[,"Mu"]` and the forecasted volatility is stored as `modelroll@forecast$density[,"Sigma"]`. Have a look at the reference manual on page 80 for the »uGARCHroll« in order to see methods for this class.

The `Skew`, `Shape` and `Shape(GIG)` are all 0 because they relate to parameters of the chosen distribution of the model and since we are using standadized normal distribution these are all 0.

So one question could be: How do I calculate the VaR of another α-level? One option would be to restimate - hence doing the `ugarchroll` again but as should be apparent from earlier calculations this is unecessary. Simply do:

```
q_st=qnorm(0.025)
sigma=modelroll@forecast$density[,"Sigma"]
mu= modelroll@forecast$density[,"Mu"]
```

```
Var_0025=q_st * sigma + mu
```

If you are using another distribution you should use the `qdist()` function as explained on page 13 of the rugarch vignette. This function should be given the relevant parameter values as explained on page 13-18 of the rugarch vignette. Here is an example using the Skewed Student t distribution in an ARMA(1,1)-GARCH(1,1) model on the `sp500ret` dataset:

```
model=ugarchspec (
variance.model = list(model = "sGARCH", garchOrder = c(1, 1)),
mean.model = list(armaOrder = c(1, 1)),
distribution.model = "sstd"
)
modelroll=ugarchroll (
model, data=sp500ret, n.ahead = 1, forecast.length = 100,
n.start = NULL, refit.every = 50, refit.window = c("recursive"),
window.size = NULL, solver = "hybrid", fit.control = list(),
solver.control = list(), calculate.VaR = TRUE, VaR.alpha = c(0.01,
0.05),
cluster = NULL, keep.coef = TRUE
)
# getting the estimated skew from modelroll
skew.estimate=modelroll@forecast$density[,"Skew"]
# getting the estimated shape from modelroll
shape.estimate=modelroll@forecast$density[,"Shape"]
# using quantile funtion from rugarchpackage
# setting skew and shape according to estimates
# other values kept as specified in vignette
q_st=qdist(distribution="sstd",p=0.05,mu=0,
sigma=1,lambda=-0.5,skew=skew.estimate,shape=shape.estimate)
mu=modelroll@forecast$density[,"Mu"]
sigma=modelroll@forecast$density[,"Sigma"]
VaR=sigma*q_st+mu
```

6.2 Different roll-procedures

Doing a forecast we could choose to specify `forecast.length`,`refit.every` and `refit.window`. Above we used the argument `forecast.length=100` to leave 100 observations out-of-sample leaving the rest 5423 observations a in-sample. As a consequence the first model is estimated on the 5423 in-sample observations. Also we chose to refit the model every 50 observations `refit.every=50` meaning that the first model would be used for 50 forecasts, that is fore period $5423 + 1, 5423 + 2, ..., 5423 + 50$. Then the model is reestimated and another 50 forecasts are made for the periods $5473 + 1, 5473 + 2, ..., 5473 + 50$ after which forecassts have been made for all periods left as out of sample. Leaving out 100 observations but reestimating for every 51 observations results in the first model being used for 51 forecasts and the other for 49 and in general the last model estimated will be used for less forecasts. But on what observations are the models estimated? The `window` could be `recursive` in which case the model is estimated on all observations with lower index than the index of the first period for which the model is used to forecast. The second model in our example would be estimated on observations $1, 2, 3, ..., 5473$ however if instead using a moving window `refit.window="moving"` the second model would only be estimated on observations $51, ..., 5473$ moving forward the first observation by `refit.every=50` every time a new model is estimated. Using the first procedure each model is estimated on an increasing sample size while using the second procedure keeps the sample size constant.

Instead of specifying `forecast.length`,`refit.every` and `refit.window` one could leave out `forecast.length` and instead choose `n.start` indication where in dataset to start rolling forecast. As an example lets say number of observations are $N = 1000$ and we choose `n.start=800` then the forecast length is 200. If you at the same time specify the `forecast.length=100` it will automatically be overwritten to have the value `forecast.length=200`. Otherwise the procedure is the same so with the first model i estimated on $1, ..., 800$ and with `refit.every=50` used to forecast for $800 + 1, ..., 800 + 50$ and then the second model is estimated on $1, ..., 850$ if `refit.window= c("recursive")` and on $51, ..., 850$ if `refit.window=c("moving")`.

Another way the determine the procedure is to specify `forecast.length`, `refit.every` and `window.size` and `refit.window=c("window")`. Assuming we have $N = 1000$ observations and `forecast.length=100` so that 100 observations are left out-of-sample (as we now know only for the initial model so it would be more correct to say that choosing `forecast.length` determines for how many periods forecasts are going to be calculated thus explaining the name of the argument). Assume we choose `refit.every=50` as a consequence the estimation of two models would be suffcient. Assume further that we choose a `window.size=500` this result is that each model is estimated in 500 observations the first observation for the first model being `N-forecast.length-window.size=1000-100-500=400`. The first model is estimated on observations $400, 401, ..., 900$ the second on $450, ..., 950$ because `refit.every=50`. If `refit.window=c("recursive")` the `window.size` is ignored so that the first model is estimated on $1, ..., 900$ and the second on

$1, ..., 950$. But what if the total sample N is to small for the first model to be estimated on a window with the specified window size? Let $N = 1000$ and `forecast.length=300` and `refit.every=50` then 700 observations are available to estimate the first model on and 750 for the second and 800 for the third etc. As a consequence choosing `window.size=770` will result in the first model being estimated on $1, ..., 700$ (hence less than chosen because only 700 are available) and the second model will be estimated on observations $1, ..., 750$ hence also less than chosen and the third model will be estimated on $30, 31, ..., 800$.

Again remember that the created object can be inspected by `str()` so to investigate the consequences of different choices have a look at `str(modelroll @model)` to see how `$rollind` and `$out.sample` among others change depending on how you choose to specify the forecasting procedure.