garchx: Flexible and Robust GARCH-X Modeling

by Genaro Sucarrat

Abstract The **garchx** package provides a user-friendly, fast, flexible, and robust framework for the estimation and inference of GARCH(p, q, r)-X models, where p is the ARCH order, q is the GARCH order, r is the asymmetry or leverage order, and 'X' indicates that covariates can be included. Quasi Maximum Likelihood (QML) methods ensure estimates are consistent and standard errors valid, even when the standardized innovations are non-normal or dependent, or both. Zero-coefficient restrictions by omission enable parsimonious specifications, and functions to facilitate the non-standard inference associated with zero-restrictions in the null-hypothesis are provided. Finally, in the formal comparisons of precision and speed, the **garchx** package performs well relative to other prominent GARCH-packages on CRAN.

1 Introduction

In the Autoregressive Conditional Heteroscedasticity (ARCH) class of models proposed by Engle (1982), the variable of interest ϵ_t is decomposed multiplicatively as

$$\epsilon_t = \sigma_t \eta_t, \tag{1}$$

where $\sigma_t > 0$ is the standard deviation of ϵ_t, and η_t is a real-valued standardized innovation with mean zero and unit variance (e.g., the standard normal). Originally, Engle (1982) interpreted ϵ_t as the error term of a dynamic regression of inflation so that σ_t is the uncertainty of the inflation forecast. However, ARCH models have also proved to be useful in many other areas. The field in which they have become most popular is finance. There, ϵ_t is commonly interpreted as a financial return, either raw or mean-corrected (i.e., ϵ_t has mean zero) so that σ_t is a measure of the variability or volatility of return. In Engle and Russell (1998), it was noted that the ARCH framework coul also be used to model non-negative variables, say, the trading volume of financial assets, the duration between financial trades, and so on. Specifically, suppose y_t denotes a non-negative variable, say, volume, and μ_t is the conditional expectation of y_t. Engle and Russell (1998) noted that in the expression $\epsilon_t^2 = \sigma_t^2 \eta_t^2$ implied by the ARCH model, if you replace ϵ_t^2 with y_t and σ_t^2 with μ_t, then it follows straightforwardly that μ_t is the conditional expectation of y_t. This is justified theoretically since the underlying estimation theory does not require that ϵ_t has mean zero. The observation made by Engle and Russell (1998) spurred a new class of models, which is known as the Multiplicative Error Model (MEM); see Brownlees et al. (2012) for a survey. The practical implication of all this is that ARCH-software can, in fact, be used to estimate MEMs by simply feeding the package in question with $\sqrt{y_t}$ rather than ϵ_t. The fitted values of σ_t^2 become the fitted values of μ_t, the error term is defined by y_t/μ_t, and the inference theory and other statistical results usually carry over straightforwardly. In conclusion, the ARCH-class of models provides a flexible framework that can be used in a very wide range of empirical applications.

Prominent GARCH-packages on CRAN

Although a large number of specifications of σ_t have been proposed, the most common in empirical applications are variants of the Generalised ARCH (GARCH) proposed by Bollerslev (1986). In particular, the plain GARCH(1,1) is ubiquitous:

$$\sigma_t^2 = \omega + \alpha_1 \epsilon_{t-1}^2 + \beta_1 \sigma_{t-1}^2, \qquad \omega > 0, \quad \alpha_1 \geq 0, \quad \beta_1 \geq 0. \tag{2}$$

By analogy with an ARMA(1,1), the conditional variance σ_t^2 is modeled as a function of the recent past, where the ϵ_{t-1}^2 is referred to as the ARCH(1) term, and σ_{t-1}^2 is referred to as the GARCH(1) term. The non-negativity of ϵ_t^2, together with the constraints on the parameters ω, α_1 and β_1, ensure σ_t^2 is strictly positive. Another way of ensuring that σ_t^2 is strictly positive is by modeling its logarithm, $\ln \sigma_t^2$, as for example in the log-ARCH class of models proposed by Geweke (1986), Pantula (1986), and Milhøj (1987). Here, however, the focus is exclusively on non-logarithmic specifications of σ_t. Also, multivariate GARCH specifications are not covered.

The most prominent packages on CRAN that are commonly used to estimate variants of (2) are **tseries** (Trapletti and Hornik, 2019), **fGarch** (Wuertz et al., 2020), and **rugarch** (Ghalanos, 2020). In

tseries, the function `garch()` enables estimation of the GARCH(p,q) specification

$$\sigma_t^2 = \omega + \sum_{i=1}^{p} \alpha_i \epsilon_{t-i}^2 + \sum_{j=1}^{q} \beta_j \sigma_{t-j}^2, \qquad \omega > 0, \quad \alpha_i \geq 0, \quad \beta_j \geq 0. \tag{3}$$

Notable features of `garch()` include simplicity and speed. With respect to simplicity, it is appealing that a plain GARCH(1,1) can be estimated by the straightforward and simple command `garch(eps)`, where `eps` is the vector or series in question, i.e., ϵ_t in (1). As for speed, it is the fastest among the packages compared here, and outside the R universe, it is also likely to be one of the fastest. Indeed, a formal speed comparison (see Section 4) reveals the relative speed provided by `garch()` can be important, particularly if the number of observations is large or if many models have to be estimated (as in simulations). A notable limitation of (3) is that it does not allow for asymmetry terms, e.g., $I_{\{\epsilon_{t-1}<0\}}\epsilon_{t-1}^2$, also known as 'leverage', or covariates. Asymmetry effects are particularly common in daily stock returns, where its presence implies that volatility in day t is higher if the return on the previous day, ϵ_{t-1}, is negative. Often, such asymmetry effects are attributed to leverage.

In **fGarch**, asymmetry effects are possible. Specifically, the function `garchFit()` enables estimation of the Asymmetric Power GARCH (APARCH) specification

$$\sigma_t^\delta = \omega + \sum_{i=1}^{p} \alpha_i |\epsilon_{t-i}|^\delta + \sum_{j=1}^{q} \beta_j \sigma_{t-j}^\delta + \sum_{k=1}^{r} \gamma_k I_{\{\epsilon_{t-k}<0\}} |\epsilon_{t-k}|^\delta, \qquad \gamma_k \geq 0, \tag{4}$$

where $\delta > 0$ is the power parameter, and the γ_k's are the asymmetry parameters. The power parameter δ is rarely different from 2 in empirical applications, but it does provide the added flexibility of modeling, say, the conditional standard deviation ($\delta = 1$) directly if the user wishes to do so. Another feature of `garchFit()` is that other densities than the normal can be used in the ML estimation, for example, the skewed normal or the skewed Student's t. In theory, this provides more efficient estimates asymptotically if η_t is skewed or more heavy-tailed than the normal. In finite samples, however, the actual efficiency may be more dependent on how estimation is carried out numerically. Also, additional density parameters may increase the possibility of numerical problems. To alleviate this potential problem, the package offers a non-normality robust coefficient-covariance along the lines of Bollerslev and Wooldridge (1992) in combination with normal ML. The coefficient-covariance of Bollerslev and Wooldridge (1992) does not, however, provide robustness to the dependence of the η_t's. Finally, **fGarch** also offers the possibility of specifying the mean equation as an ARMA model. That is, $\epsilon_t = y_t - \mu_t$, where μ_t is the ARMA specification. Theoretically, joint estimation of μ_t and σ_t may improve the asymptotic efficiency compared with, say, a two-step estimation approach, where μ_t is estimated in the first step, and σ_t is estimated in the second using the residuals from the first step. In practice, however, the joint estimation may, in fact, reduce the actual efficiency. The reasons for this are the increase in the number of parameters to be estimated and the increased possibility of numerical problems due to the increase in the number of parameters to be estimated.

A limitation of **fGarch** is that it does not allow for additional covariates ('X') in (4). This can be a serious limitation since additional conditioning variables like $high - low$, realized volatility, interest rates, and so on may help to predict or explain volatility in substantial ways. The **rugarch** package remedies this. Most of the non-exponential specifications offered by **rugarch** are contained in

$$\sigma_t^\delta = \omega + \sum_{i=1}^{p} \alpha_i |\epsilon_{t-i}|^\delta + \sum_{j=1}^{q} \beta_j \sigma_{t-j}^\delta + \sum_{k=1}^{r} \gamma_k I_{\{\epsilon_{t-k}<0\}} |\epsilon_{t-k}|^\delta + \sum_{l=1}^{s} \lambda_l x_{l,t-1}, \quad \lambda_l \geq 0, \quad x_{l,t-1} \geq 0, \tag{5}$$

where the $x_{l,t-1}$'s are the covariates. However, it should be mentioned that the package also enables the estimation of additional models, e.g., the Component GARCH model and the Fractionally Integrated GARCH model, amongst others. These additional models are not the focus here. Note that the covariates in (5) need not enter as lagged of order 1. That is, $x_{l,t-1}$ may denote a variable that is lagged of order 2, say, w_{t-2}, and so on. A variable may also enter as unlagged, w_t. However, it is not clear what the theoretical requirements are for consistent estimation, in this case, due to simultaneity issues. Just as in **fGarch**, the **rugarch** package also enables a non-normality robust coefficient-covariance, ML estimation with non-normal densities, and the joint estimation of an ARMA specification in the mean together with σ_t. To the best of my knowledge, no other CRAN package offers more univariate GARCH specifications than **rugarch**.

What does garchx offer that is not already available?

The **garchx** package[1] aims to provide a simple, fast, flexible, and robust – both theoretically and numerically – framework for GARCH-X modeling. The specifications that can be estimated are all contained within

$$\sigma_t^2 = \omega + \sum_{i=1}^{p} \alpha_i \epsilon_{t-i}^2 + \sum_{j=1}^{q} \beta_j \sigma_{t-j}^2 + \sum_{k=1}^{r} \gamma_k I_{\{\epsilon_{t-k}<0\}} \epsilon_{t-k}^2 + \sum_{l=1}^{s} \lambda_l x_{l,t-1}. \qquad (6)$$

While this implies a restriction of $\delta = 2$ compared with the **rugarch** package, **garchx** enables several additional features that are not available in the above-mentioned packages:

i) *Robustness to dependence*. Normal ML estimation is usually consistent when the η_t's are dependent over time, see e.g., Escanciano (2009) and Francq and Thieu (2018). This is useful, for example, when the conditional skewness, conditional kurtosis, or conditional zero-probability of η_t is time-varying and dependent on the past in unknown ways. In these cases, however, the non-normality robust coefficient-covariance of Bollerslev and Wooldridge (1992) is not valid. Optionally, **garchx** offers the possibility of using the dependence (and non-normality) robust coefficient-covariance derived by Francq and Thieu (2018).

ii) *Inference under nullity*. In applied work, it is frequently of interest to test whether a coefficient differs from zero. The permissible parameter-space of GARCH models, however, is bounded from below by zero. Accordingly, non-standard inference is required when the value of a null-hypothesis lies on the zero-boundary; see Francq and Thieu (2018). The **garchx** package offers functions to facilitate such tests, named `ttest0()` and `waldtest0()`, respectively, based on the results by Francq and Thieu (2018).

iii) *Zero-constrained coefficients by omission*. If one or more coefficients are indeed zero, then it may be desirable to obtain estimates under zero-constraints on these coefficients. For example, if ϵ_t is the error term in a regression of quarterly inflation, then it may be desirable to estimate a GARCH(4,4) model in which the parameters associated with orders 1, 2, and 3 are restricted to zero. That is, it is desirable to estimate

$$\sigma_t^2 = \omega + \alpha_4 \epsilon_{t-4}^2 + \beta_4 \sigma_{t-4}^2.$$

Another example is the non-exponential Realized GARCH of Hansen et al. (2012), which is simply a GARCH(0,1)-X. That is, the ARCH(1) coefficient is set to zero. Zero-constrained coefficients do not only provide a more parsimonious characterization of the process in question. They may also make estimation more efficient and stable numerically since fewer parameters need to be estimated. **rugarch** offers a feature in which coefficients can be fixed to zero. However, its approach is not by omission. In other words, using `coef` in **rugarch** to extract the coefficients in the GARCH(4,4) example above will return a vector of length 9 rather than of length 3, while the coefficient-covariance returned by **rugarch** will be 3×3. This makes multiple hypothesis testing with Wald tests tedious in constrained models. In **garchx**, by contrasts, Wald tests in constrained models are straightforward since the zeros are due to omission. The vector returned by `coef` is of length 3 and the coefficient-covariance is 3×3.

iv) *Computation of the asymptotic coefficient-covariance*. Knowing the value of the theoretical, asymptotic coefficient-covariance matrix is needed for a formal evaluation of an estimator. For GARCH models, these expressions are not available in explicit form. The **garchx** offers a function, `garchxAvar()`, that computes them by combining simulation and numerical differentiation. To illustrate the usage of `garchxAvar()`, a small Monte Carlo study is undertaken. While the results of the study suggest all four packages return approximately unbiased estimates in large samples, they also suggest **tseries** and **rugarch** are less robust numerically than **fGarch** and **garchx** under default options. In addition, the simulations reveal the standard errors of **tseries** can be substantially biased downwards when η_t is non-normal. A bias is expected since **tseries** does not offer a non-normality robust coefficient-covariance. However, the bias is larger than suggested by the underlying estimation theory.

Table 1 provides a summary of the features offered by the four packages.

The rest of the article is organised as follows.[2] The next section provides an overview of the **garchx** package and its usage. Thereafter, the `garchxAvar()` function is illustrated by means of a Monte Carlo study of the large sample properties of the packages. Next, a speed comparison of the packages is undertaken. While **tseries** is the fastest for the specifications it can estimate, **garchx** is notably faster than **fGarch** and **rugarch** in all the experiments that are conducted. Finally, the last section concludes.

[1]On CRAN since 9 April 2020.

[2]All commands and code-executions were carried out in R version 3.6.3 on a Windows 10 64bit machine.

	tseries	**fGarch**	**rugarch**	**garchx**
GARCH(p,q)	Yes	Yes	Yes	Yes
Asymmetry		Yes	Yes	Yes
Power GARCH		Yes	Yes	
Covariates (X)			Yes	Yes
Additional GARCH models			Yes	
Non-normality robust vcov		Yes	Yes	Yes
Dependence robust vcov				Yes
Computation of asymptotic vcov				Yes
Constrained estimation			Yes	
Zero-constraints by omission				Yes
Inference under null-restrictions				Yes
Normal (Q)ML	Yes	Yes	Yes	Yes
Non-normal ML		Yes	Yes	
ARMA in the mean		Yes	Yes	

Table 1: A feature-based comparison of selected R packages that offer GARCH-estimation: **tseries** version 0.10-47 (Trapletti and Hornik, 2019), **fGarch** version 3042.83.2 (Wuertz et al., 2020), **rugarch** version 1.4-2 (Ghalanos, 2020), and **garchx** version 1.1 (Sucarrat, 2020).

2 The garchx package

Estimation theory

Let $\mathcal{F}_{t-1}$ denote the sigma-field generated by past observables. Formally, in the **garchx** package, ϵ_t is expected to satisfy $\epsilon_t^2 = \sigma_t^2\eta_t^2$, (6) and

$$E(\eta_t^2|\mathcal{F}_{t-1}) = 1 \qquad \text{for all } t. \tag{7}$$

The conditional unit variance assumption in (7) is very mild since it does not require that the distribution of η_t is identical over time, nor that η_t is independent of the past. In particular, the assumption is compatible with a time-varying conditional skewness $E(\eta_t^3|\mathcal{F}_{t-1})$ that depends on the past in unknown ways, a time-varying conditional kurtosis $E(\eta_t^4|\mathcal{F}_{t-1})$ that depends on the past in unknown ways, and even a time-varying conditional zero-probability $Pr(\eta_t = 0|\mathcal{F}_{t-1})$ that depends on the past in unknown ways. Empirically, such forms of dependence are common; see e.g., Hansen (1994) and Sucarrat and Grønneberg (2020). GARCH models in which the η_t's are dependent are often referred to as semi-strong after Drost and Nijman (1993).

Subject to suitable regularity conditions, the normal ML estimator provides consistent and asymptotically normal estimates of semi-strong GARCH models; see Francq and Thieu (2018). Specifically, they show that

$$\sqrt{T}(\widehat{\boldsymbol{\vartheta}} - \boldsymbol{\vartheta}_0) \xrightarrow{d} N(\mathbf{0}, \boldsymbol{\Sigma}), \qquad \boldsymbol{\Sigma} = \boldsymbol{J}^{-1}\boldsymbol{I}\boldsymbol{J}^{-1}, \quad \boldsymbol{J} = E\left(\frac{\partial^2 l_t(\boldsymbol{\vartheta}_0)}{\partial\boldsymbol{\vartheta}\partial\boldsymbol{\vartheta}'}\right), \quad \boldsymbol{I} = E\left(\frac{\partial l_t(\boldsymbol{\vartheta}_0)}{\partial\boldsymbol{\vartheta}}\frac{\partial l_t(\boldsymbol{\vartheta}_0)}{\partial\boldsymbol{\vartheta}'}\right), \tag{8}$$

where

$$\widehat{\boldsymbol{\vartheta}} = \arg\min_{\boldsymbol{\vartheta}} \frac{1}{T}\sum_{t=1}^{T} l_t(\boldsymbol{\vartheta}), \qquad l_t(\boldsymbol{\vartheta}) = \frac{\epsilon_t^2}{\sigma_t^2(\boldsymbol{\vartheta})} + \ln\sigma_t^2(\boldsymbol{\vartheta}), \tag{9}$$

is the (normal) Quasi ML (QML) estimate of the true parameter $\boldsymbol{\vartheta}_0$. If the η_t's are independent of the past, then

$$\boldsymbol{\Sigma} = \left(E(\eta_t^4) - 1)\right)\boldsymbol{J}^{-1}, \tag{10}$$

This is essentially the univariate version of the non-normality robust coefficient-covariance of Bollerslev and Wooldridge (1992). It is easily estimated since the standardized residuals can be used to obtain an estimate of $E(\eta_t^4)$, and a numerical estimate of the Hessian $\boldsymbol{J}$ is returned by the optimizer. In the **garchx** package, the estimate of (10) is referred to as the `"ordinary"` coefficient-covariance. Of course, the expression returned by the software is the estimate of the finite sample counterpart $\boldsymbol{\Sigma}/T$, where T is the sample size. In other words, the standard errors are equal to the square root of the diagonal of the estimate $\widehat{\boldsymbol{\Sigma}}/T$. If, instead, the η_t's are not independent of the past, then

$$\boldsymbol{\Sigma} = \boldsymbol{J}^{-1}\boldsymbol{I}\boldsymbol{J}^{-1}, \qquad \boldsymbol{I} = E\left[\left\{E\left(\frac{\epsilon_t^4}{\sigma_t^4(\boldsymbol{\vartheta}_0)}\middle|\mathcal{F}_{t-1}\right) - 1\right\}\frac{1}{\sigma_t^4(\boldsymbol{\vartheta}_0)}\frac{\partial\sigma_t^2(\boldsymbol{\vartheta}_0)}{\partial\boldsymbol{\vartheta}}\frac{\partial\sigma_t^2(\boldsymbol{\vartheta}_0)}{\partial\boldsymbol{\vartheta}'}\right]. \tag{11}$$

In the **garchx** package, the estimate of this expression is referred to as the "robust" coefficient-covariance. Again, the expression returned by the software is the estimate of the finite sample counterpart $\boldsymbol{\Sigma}/T$. It should be noted that the estimation of (11) is computationally much more demanding than (10) since an estimate of $\partial\sigma_t^2(\boldsymbol{\vartheta}_0)/\partial\boldsymbol{\vartheta}$ in $\boldsymbol{I}$ is computed at each t. More details about how this is implemented is contained in the Appendix.

Basic usage of garchx

For illustration, the spyreal dataset in the **rugarch** package is used, which contains two daily financial time series: The SPDR SP500 index open-to-close return and the realized kernel volatility. The data are from Hansen et al. (2012) and goes from 2002-01-02 to 2008-08-29. The following code loads the data and stores the daily return – in percent – in an object named eps:

```
library(xts)
data(spyreal, package = "rugarch")
eps <- spyreal[,"SPY_OC"]*100
```

Note that the data spyreal is an object of class **xts** (Ryan and Ulrich, 2014). Accordingly, the object eps is also of class xts.

The basic interface of **garchx** is similar to that of garch() in **tseries**. For example, the code

```
garchx(eps)
```

estimates a plain GARCH(1,1), and returns a print of the result (implicitly, print.garchx() is invoked):

```
Date: Wed Apr 15 09:19:41 2020
Method: normal ML
Coefficient covariance: ordinary
Message (nlminb): relative convergence (4)
No. of observations: 1661
Sample: 2002-01-02 to 2008-08-29

              intercept      arch1     garch1
Estimate:   0.005945772 0.05470749 0.93785529
Std. Error: 0.002797459 0.01180603 0.01349976

Log-likelihood: -2014.6588
```

Alternatively, the estimation result can be stored to facilitate the subsequent extraction of information:

```
mymod <- garchx(eps)
coef(mymod)       #coefficient estimates
fitted(mymod)     #fitted conditional variance
logLik(mymod)     #log-likelihood (i.e., not the average log-likelihood)
nobs(mymod)       #no. of observations
predict(mymod)    #generate predictions of the conditional variance
print(mymod)      #print of estimation result
quantile(mymod)   #fitted quantile(s), the default corresponds to 97.5% value-at-risk
residuals(mymod)  #standardized residuals
summary(mymod)    #summarise with summary.default
toLatex(mymod)    #LaTeX print of result (equation form)
vcov(mymod)       #coefficient-covariance
```

The series returned by fitted, quantile, and residuals are of class **zoo** (Zeileis and Grothendieck, 2005).

To control the ARCH, GARCH, and asymmetry orders, the argument order, which takes a vector of length 1, 2, or 3, can be used in a similar way to as in the garch() function of **tseries**:

- order[1] controls the GARCH order
- order[2] controls the ARCH order
- order[3] controls the asymmetry order

For example, the following code estimate, respectively, a GARCH(1,1) with asymmetry and a GARCH(2,1) without asymmetry:

```
garchx(eps, order = c(1,1,1))  #garch(1,1) w/asymmetry
garchx(eps, order = c(1,2))    #garch(2,1)
```

To illustrate how covariates can be included via the `xreg` argument, the lagged realized volatility from the `spyreal` dataset can be used:

```
x <- spyreal[,"SPY_RK"]*100
xlagged <- lag(x)  #this lags, since x is an xts object
xlagged[1] <- 0    #replace NA-value with 0
```

The code

```
garchx(eps, xreg = xlagged)
```

estimates a GARCH(1,1) with the lagged realized volatility as covariate, i.e.,

$$\sigma_t^2 = \omega + \alpha_1\epsilon_{t-1}^2 + \beta_1\sigma_{t-1}^2 + \lambda_1 x_{1,t-1}, \tag{12}$$

and returns the print

```
Date: Wed Apr 15 09:26:46 2020
Method: normal ML
Coefficient covariance: ordinary
Message (nlminb): relative convergence (4)
No. of observations: 1661
Sample: 2002-01-02 to 2008-08-29

             intercept      arch1      garch1     SPY_RK
Estimate:   0.01763853 0.00000000 0.71873142 0.28152520
Std. Error: 0.01161863 0.03427413 0.09246282 0.08558003

Log-likelihood: -1970.247
```

The estimates suggest the ARCH parameter α_1 is 0. In a t-test with $\alpha_1 = 0$ as the null hypothesis, the parameter lies on the boundary of the permissible parameter space under the null. Accordingly, inference is non-standard and below I illustrate how this can be carried out with the `ttest0()` function. Note that if α_1 is, indeed, 0, then the specification reduces to the non-exponential Realised GARCH of Hansen et al. (2012). Below I illustrate how it can be estimated by simply omitting the ARCH term, i.e., by imposing a zero-coefficient restriction via omission.

The `"ordinary"` coefficient-covariance is the default. To instead use the dependence robust coefficient-covariance, set the `vcov.type` argument to `"robust"`:

```
garchx(eps, xreg = xlagged, vcov.type = "robust")
```

The associated print

```
Date: Wed Apr 15 09:31:12 2020
Method: normal ML
Coefficient covariance: robust
Message (nlminb): relative convergence (4)
No. of observations: 1661
Sample: 2002-01-02 to 2008-08-29

             intercept      arch1    garch1    SPY_RK
Estimate:   0.01763853 0.00000000 0.7187314 0.2815252
Std. Error: 0.01864470 0.04569981 0.1507067 0.1136347

Log-likelihood: -1970.247
```

reveals the standard errors change, but not dramatically. If the estimation result had been stored in an object with, say, the command `mymod <-garchx(eps,xreg = xlagged)`, then the robust coefficient-covariance could instead have been extracted by the code `vcov(mymod,vcov.type = "robust")`.

Inference under nullity

If the value of a parameter is zero under the null hypothesis, then it lies on the boundary of the permissible parameter space. In these cases, the non-standard inference is required, see Francq and

Thieu (2018). The **garchx** package offers two functions to facilitate such non-standard tests, `ttest0()`, and `waldtest0()`.

Recall that $\boldsymbol{\vartheta}_0$ denotes the d-dimensional vector of true parameters. In a plain GARCH(1,1), for example, $d = 3$. Next, let $\boldsymbol{e}_k$ denote a $d \times 1$ vector whose elements are all 0 except element k, which is 1. The function `ttest0()` undertakes the following t-test of parameter $k \geq 2$:

$$H_0 : \boldsymbol{e}_k'\boldsymbol{\vartheta}_0 = 0 \quad \text{and} \quad \boldsymbol{e}_l'\boldsymbol{\vartheta}_0 > 0 \ \forall l \neq k \qquad \text{against} \qquad H_A : \boldsymbol{e}_k'\boldsymbol{\vartheta}_0 > 0.$$

Note that, in this test, all parameters – except parameter k – are assumed to be greater than 0 under the null. While the test-statistic is the usual one, the p-value is obtained by only considering the positive part of the normal distribution. To illustrate the usage of `ttest0`, let us revisit the GARCH(1,1)-X model in (12):

```
mymod <- garchx(eps, xreg = xlagged)
```

In this model, the non-exponential Realized GARCH of Hansen et al. (2012) is obtained when the ARCH(1)-parameter α_1 is 0. This is straightforwardly tested with `ttest0(mymod,k = 2)`, which yields

```
      coef  std.error t-stat p-value
arch1    0 0.03427413      0     0.5
```

In other words, at the most common significance levels, the result supports the claim that $\alpha_1 = 0$. Finally, note that if the user does not specify k, then the code `ttest0()` returns a t-test of all the coefficients except the intercept ω.

The function `waldtest0()` can be used to test whether one or more coefficients are zero. Let $\boldsymbol{r}$ denote the restriction vector of dimension $r_0 \times 1$, and let $\boldsymbol{R}$ denote the combination matrix of dimension $r_0 \times d$. Assuming that $\boldsymbol{R}$ has full row-rank, the null and alternative hypotheses in the Wald-test are given by

$$H_0 : \boldsymbol{R}\boldsymbol{\vartheta}_0 = \boldsymbol{r} \qquad \text{against} \qquad H_A : \boldsymbol{R}\boldsymbol{\vartheta}_0 \neq \boldsymbol{r}.$$

The associated Wald test-statistic has the usual form, but the distribution is non-standard (Francq and Thieu, 2018):

$$W_T = (\boldsymbol{R}\widehat{\boldsymbol{\vartheta}} - \boldsymbol{r})'\boldsymbol{R}(\widehat{\boldsymbol{\Sigma}}/T)\boldsymbol{R}'(\boldsymbol{R}\widehat{\boldsymbol{\vartheta}} - \boldsymbol{r}), \qquad W_T \xrightarrow{d} W = ||\boldsymbol{R}\boldsymbol{Z}||^2, \qquad \boldsymbol{Z} \sim N(\boldsymbol{0}, \boldsymbol{\Sigma}).$$

Critical values are obtained by parametric Bootstrap. First, the sequence

$$\left\{||\boldsymbol{R}\widehat{\boldsymbol{Z}}_i||^2, i = 1, \ldots, n\right\}$$

is simulated, where the $\widehat{\boldsymbol{Z}}_i$'s are independent and identically distributed $N(\boldsymbol{0}, \widehat{\boldsymbol{\Sigma}})$ vectors. In `waldtest0()`, the default is $n = 20000$. Next, the critical value associated with significance level $\alpha \in (0,1)$ is obtained by computing the empirical (1-α)-quantile of the simulated values. To illustrate the usage of `waldtest0()`, let us reconsider the GARCH(1,1)-X model in (12). Specifically, let us test whether both the ARCH and GARCH coefficients are zero: $H_0 : \alpha_1 = 0$ and $\beta_1 = 0$. This means

```
r <- cbind(c(0,0))
R <- rbind(c(0,1,0,0),c(0,0,1,0))
```

Next, the command `waldtest0(mymod,r = r,R = R)` performs the test and returns a list with the statistic and critical values:

```
$statistic
[1] 72.95893

$critical.values
10%       5%       1%
41.79952 57.97182 97.15217
```

In other words, the Wald statistic is 72.96 and the critical values associated with the 10%, 5%, and 1% levels, respectively, are 41.80, 57.97, and 97.15. So H_0 is rejected at the 10% and 5% levels, but not at the 1% level. If the user wishes to do so, the significance levels can be changed via the `level` argument.

Zero-coefficient restrictions via omission

The ARCH, GARCH, and asymmetry orders can be specified in two ways. Either via the `order` argument as illustrated above or via the `arch`, `garch`, and `asym` arguments whose defaults are all `NULL`. If any of their values is not `NULL`, then it takes precedence over the corresponding component in `order`. For example, the code

```
garchx(eps, order = c(0,0), arch = 1, garch = 1)
```

estimates a GARCH(1,1) since the values of arch and garch override those of order[2] and order[1], respectively. Similarly, garchx(eps,asym = 1) estimates a GARCH(1,1) with asymmetry, and garchx(eps,garch = 0) estimates a GARCH(1,0) model.

To estimate higher-order models with the arch, garch, and asym arguments, the lags must be provided explicitly. For example, to estimate the GARCH(2,2) model $\sigma_t^2 = \omega + \alpha_1\epsilon_{t-1}^2 + \alpha_2\epsilon_{t-2}^2 + \beta_1\sigma_{t-1}^2 + \beta_2\sigma_{t-2}^2$, use

```
garchx(eps, arch = c(1,2), garch = c(1,2))
```

Zero-coefficient constraints, therefore, can be imposed by simply omitting the lags in question. For example, to estimate the GARCH(2,2) model with $\alpha_1 = \beta_1 = 0$, use

```
garchx(eps, arch = 2, garch = 2)
```

This returns the print

```
Date: Wed Apr 15 09:34:04 2020
Method: normal ML
Coefficient covariance: ordinary
Message (nlminb): relative convergence (4)
No. of observations: 1661
Sample: 2002-01-02 to 2008-08-29

              intercept      arch2     garch2
Estimate:   0.009667606 0.07533534 0.91392791
Std. Error: 0.004494075 0.01636917 0.01899654

Log-likelihood: -2033.7251
```

To estimate the non-exponential Realized GARCH of Hansen et al. (2012), use

```
garchx(eps, arch = 0, xreg = xlagged)
```

The returned print shows that the ARCH(1) term has not been included during the estimation.

Finally, a caveat is in order. The flexibility provided by the arch, garch, and asym arguments is not always warranted by the underlying estimation theory. For example, if the ARCH-parameter α_1 in a plain GARCH(1,1) model is restricted to zero, then the normal ML estimator is invalid. The garchx() function, nevertheless, tries to estimate it if the user provides the code garchx(eps,arch = 0). Currently, the function garchx() does not undertake any checks of whether the zero-coefficient restrictions are theoretically valid.

Numerical optimization

The two optimization algorithms in base R that work best for GARCH estimation are, in my experience, the "Nelder-Mead" method in the optim() function and the nlminb() function. The latter enables bounded optimization, so it is the preferred algorithm here since the parameters of the GARCH model must be non-negative. The "L-BFGS-B" method in optim() also enables bounded optimization, but it does not work as well in my experience. When using the garchx() function, the call to nlminb() can be controlled and tuned via the arguments initial.values, lower, upper, and control. In nlminb(), the first argument is named start, whereas the other three are equal.

Suitable initial parameter values are important for numerical robustness. In the garchx() function, the user can set these via the initial.values argument. If not, then they are automatically determined internally. In the case of a GARCH(1,1), the default initial values are $\omega = 0.1$, $\alpha_1 = 0.1$, and $\beta_1 = 0.7$. For numerical robustness, it is important that they are not too close to the lower boundary of 0 and that β_1 is not too close to instability, i.e., $\beta_1 \geq 1$. The choice c(0.1,0.1,0.7) works well across a range of problems. Indeed, the Monte Carlo simulations of the large sample properties of the packages (see Section 3) reveals that the numerical robustness of **tseries** improves when these initial values are used instead of the default initial values. In the list returned by garchx(), the item named initial.values contains the values used. For example, the following code extracts the initial values used in the estimation of a GARCH(1,1) with asymmetry:

```
mymod <- garchx(eps, asym = 1)
mymod$initial.values
```

In each iteration, `nlminb()` calls the function `garchxObjective()` to evaluate the objective function. For additional numerical robustness, checks of the parameters and fitted conditional variance are conducted within `garchxObjective()` at each iteration. The first check is for whether any of the parameter values at the current iteration are equal to `NA`. The second check is for whether any of the fitted conditional variances are `Inf`, 0, or negative. If either of these checks fails, then `garchxObjective()` returns the value of the `logl.penalty` argument in the `garchx()` function, whose default value is that produced by the initial values. To avoid that the term $\ln \sigma_t^2$ in the objective function explodes to minus infinity, the fitted values of σ_t^2 are restricted to be equal or greater than the value provided by the `sigma2.min` argument in the `garchx()` function.

A drawback with `nlminb()` is that it does not return an estimate of the Hessian at the optimum, which is needed to compute the coefficient-covariance. To obtain such an estimate, the `optimHess()` function is used. In `garchx()`, the call to `optimHess()` can be controlled and tuned via the `optim.control` argument. Next, the inverse of the estimated Hessian is computed with `solve()`, whose tolerance for detecting linear dependencies in the columns is determined by the `solve.tol` argument in the `garchx()` function.

3 Checking the large sample properties

The function `garchxAvar()` returns the asymptotic coefficient-covariance of a GARCH-X model. Currently (version 1.1), only non-normality robust versions are available. The aim of this section is to illustrate how it can be used to check whether the large sample properties of the packages correspond to those of the underlying asymptotic estimation theory. Specifically, the aim is to explore whether large sample estimates from Monte Carlo simulations are unbiased, whether the empirical standard errors correspond to the asymptotic ones, and whether the estimate of the non-normality robust coefficient-covariance is unbiased.

The `garchxAvar()` function

To recall, the non-normality robust asymptotic coefficient-covariance is given by

$$\boldsymbol{\Sigma} = \left(E(\eta_t^4) - 1\right) \boldsymbol{J}^{-1}, \qquad \boldsymbol{J} = E\left(\frac{\partial^2 l_t(\boldsymbol{\vartheta}_0)}{\partial\boldsymbol{\vartheta}\partial\boldsymbol{\vartheta}'}\right)$$

when the η_t's are independent of the past. In general, the expression for $\boldsymbol{J}$ is not available in closed form. Accordingly, numerical methods are needed. The `garchxAvar()` function combines simulation and numerical differentiation to compute $\boldsymbol{\Sigma}$. In short, the function proceeds by first simulating n values of ϵ_t (the default is $n = 10$ million), and then the Hessian of the criterion function $n^{-1}\sum_{t=1}^{n} l_t(\boldsymbol{\vartheta})$ about the true value $\boldsymbol{\vartheta}_0$ is obtained by numerical differentiation to compute an estimate of $\boldsymbol{J}$. Internally, the `garchxAvar()` function conducts the simulation with `garchxSim()` and the differentiation with `optimHess()`. If we denote the numerically obtained Hessian as $\tilde{\boldsymbol{J}}$, then the corresponding finite-sample counterpart of the asymptotic coefficient-covariance associated with a sample of size T is given by

$$\frac{1}{T}\left(E(\eta_t^4) - 1\right)\tilde{\boldsymbol{J}}^{-1}. \tag{13}$$

In other words, the square root of the diagonal of this expression is the asymptotic standard error associated with sample size T.

To obtain an idea about the precision of `garchxAvar()`, a numerical comparison is made for the case where the DGP is an ARCH(1) with standard normal innovations:

$$\epsilon_t = \sigma_t\eta_t, \qquad \eta_t \overset{iid}{\sim} N(0,1), \qquad \sigma_t^2 = \omega + \alpha_1\epsilon_{t-1}^2. \tag{14}$$

In this case, it can be shown that

$$\boldsymbol{J} = E\begin{bmatrix} \frac{1}{(\omega+\alpha_1\epsilon_{t-1}^2)^2} & \frac{\epsilon_{t-1}^2}{(\omega+\alpha_1\epsilon_{t-1}^2)^2} \\ \frac{\epsilon_{t-1}^2}{(\omega+\alpha_1\epsilon_{t-1}^2)^2} & \frac{\epsilon_{t-1}^4}{(\omega+\alpha_1\epsilon_{t-1}^2)^2} \end{bmatrix}; \tag{15}$$

see (Francq and Zakoïan, 2019, pp. 180-181). In other words, in this specific case, it is straightforward to obtain a numerical estimate of $\boldsymbol{J}$ without having to resort to numerical derivatives (as in `garchxAvar()`) by simply computing the means of the sample counterparts. For an ARCH(1) with $(\omega, \alpha_1) = (1, 0.1)$, the code :

```
n <- 10000000
```

```
omega <- 1; alpha1 <- 0.1
set.seed(123)
eta <- rnorm(n)
eps <- garchxSim(n, intercept = omega, arch = alpha1, garch = NULL,
        innovations = eta)

epslagged2 <- eps[-length(eps)]^2
epslagged4 <- epslagged2^2
J <- matrix(NA, 2, 2)
J[1,1] <- mean( 1/( (omega+alpha1*epslagged2)^2 ) )
J[2,1] <- mean( epslagged2/( (omega+alpha1*epslagged2)^2 ) )
J[1,2] <- J[1,2]
J[2,2] <- mean( epslagged4/( (omega+alpha1*epslagged2)^2 ) )
Eeta4 <- 3
Avar1 <- (Eeta4-1)*solve(J)
```

computes the asymptotic coefficient-covariance, and stores it in an object named Avar1:

```
Avar1
          [,1]      [,2]
[1,]  3.475501 -1.368191
[2,] -1.368191  1.686703
```

With garchxAvar(), using the same simulated series for η_t, we obtain

```
Avar2 <- garchxAvar(c(omega,alpha1), arch=1, Eeta4=3, n=n, innovations=eta)
Avar2
          intercept     arch1
intercept  3.474903 -1.367301
arch1     -1.367301  1.685338
```

These are quite similar in relative terms since the ratio Avar2/Avar1 shows each entry in Avar2 is less than 0.1% away from those of Avar1.

Bias and standard errors of estimates

To illustrate how garchxAvar() can be used to study the large sample properties of the packages, a Monte Carlo study is undertaken. The DGP in the study is a plain GARCH(1,1) with either $\eta_t \sim N(0,1)$ or $\eta_t \sim$ standardized $t(5)$, and the sample size is $T = 10000$:

$$\epsilon_t = \sigma_t\eta_t, \qquad \eta_t \overset{iid}{\sim} N(0,1) \quad \text{or} \quad \eta_t \overset{iid}{\sim} \text{standardized } t(5), \qquad t = 1,\ldots,T = 10000,$$

$$\sigma_t^2 = \omega + \alpha_1\epsilon_{t-1}^2 + \beta_1\sigma_{t-1}^2, \qquad (\omega,\alpha_1,\beta_1) = (0.2, 0.1, 0.8).$$

The values of $(\omega,\alpha_1,\beta_1)$ are similar to those that are usually found in empirical studies of financial returns. The code

```
n <- 10000000
pars <- c(0.2, 0.1, 0.8)
set.seed(123)
AvarNormal <- garchxAvar(pars, arch=1, garch=1, Eeta4=3, n=n)
eta <- rt(n, df=5)/sqrt(5/3)
Avart5 <- garchxAvar(pars, arch=1, garch=1, Eeta4=9, n=n,
        innovations=eta)
```

computes and stores the asymptotic coefficient-covariances in objects named AvarNormal and Avart5, respectively. They are:

```
AvarNormal
          intercept      arch1    garch1
intercept  7.043653  1.1819890 -4.693843
arch1      1.181989  0.7784797 -1.278153
garch1    -4.693843 -1.2781529  3.616365

Avart5
           intercept     arch1     garch1
intercept  16.234885  3.216076 -11.313749
```

	$m(\widehat{\omega})$	$se(\widehat{\omega})$	$ase(\widehat{\omega})$	$m(\widehat{\alpha}_1)$	$se(\widehat{\alpha}_1)$	$ase(\widehat{\alpha}_1)$	$m(\widehat{\beta}_1)$	$se(\widehat{\beta}_1)$	$ase(\widehat{\beta}_1)$	n(NA)
$N(0,1)$:										
tseries	0.218	0.160	0.027	0.100	0.010	0.009	0.791	0.082	0.019	0
fGarch	0.203	0.027	0.027	0.100	0.009	0.009	0.799	0.019	0.019	0
rugarch	0.204	0.027	0.027	0.100	0.009	0.009	0.797	0.019	0.019	0
garchx	0.203	0.027	0.027	0.100	0.009	0.009	0.798	0.019	0.019	0
$t(5)$:										
tseries	0.218	0.158	0.040	0.101	0.015	0.016	0.791	0.077	0.030	0
fGarch	0.204	0.039	0.040	0.101	0.016	0.016	0.797	0.030	0.030	0
rugarch	0.201	0.037	0.040	0.100	0.014	0.016	0.799	0.027	0.030	2
garchx	0.201	0.037	0.040	0.100	0.015	0.016	0.799	0.028	0.030	0

Table 2: Comparison of the large sample properties of **tseries** version 0.10-47 (Trapletti and Hornik, 2019), **fGarch** version R 3.0.1 (Wuertz et al., 2020), **rugarch** version 1.4-2 (Ghalanos, 2020), and **garchx** version 1.1 (Sucarrat, 2020). $m(\cdot)$: sample average of estimates. $se(\cdot)$: sample standard deviation of estimates. $ase(\cdot)$: asymptotic standard error. n(NA): the number of times estimation failed due to numerical issues.

```
arch1        3.216076  2.483018  -3.647237
garch1     -11.313749 -3.647237   9.239820
```

Next, the asymptotic standard errors associated with sample size $T = 10000$ are obtained with

```
sqrt( diag(AvarNormal/10000) )
sqrt( diag(Avart5/10000) )
```

These values are contained in the columns labelled $ase(\cdot)$ in Table 2.

Table 2 contains the estimation results of the Monte Carlo study (1000 replications). For each package, normal ML estimation is undertaken with default options on initial parameter values, initial recursion values, and numerical control. The columns labelled $m(\cdot)$ contain the sample average across the replications, and $se(\cdot)$ contains the sample standard deviation. Apart from **tseries**, the simulations suggest the packages produce asymptotically unbiased estimates and empirical standard errors that correspond to the asymptotic ones. Closer examination suggests the biases and faulty empirical standard errors of **tseries** are due to outliers. Additional simulations, with non-default initial parameter values, produce results similar to those of the other packages.[3] This underlines the importance of suitable initial parameter values for numerical robustness. The package **rugarch** ran into numerical problems twice for $\eta_t \sim t(5)$, and thus, failed to returned estimates in these two cases. Additional simulations confirmed **rugarch** is less robust numerically than the other packages under its default options when $\eta_t \sim t(5)$, i.e., it always failed at least once. Changing the initial parameter values to those of **garchx** did not resolve the problem. Also, changing the optimizer to a non-default algorithm, `nlminb()`, which is the default algorithm in **fGarch** and the only option available in **garchx**, produced more failures and substantially biased results by **rugarch**.[4]

Coefficient-covariance estimate

In each of the 1000 replications of the Monte Carlo study, the estimate of the asymptotic coefficient-covariance is recorded. For **fGarch**, **rugarch**, and **garchx**, the estimate is of the non-normality robust type. For **tseries**, which does not offer the non-normality robust option, the estimate is under the assumption of normality. Note also that, for **tseries**, the results reported here are with the numerically more robust non-default initial parameter values alluded to above.

Let $\widehat{\mathbf{\Sigma}}_i$ denote the estimate produced by a package in replication $i = 1, \ldots, 1000$ of the simulations. The relative bias in replication i is given by the ratio $\widehat{\mathbf{\Sigma}}_i / \mathbf{\Sigma}$, a 3×3 matrix, which is obtained by dividing the row i column j component in $\widehat{\mathbf{\Sigma}}_i$ by the corresponding component in $\mathbf{\Sigma}$. The average relative bias,

[3]The additional simulations are not reported, but they are readily conducted by minor modifications to the replication files. Specifically, the code `garch(eps)` needs to be modified to `garch(eps, control = garch.control(start = c(0.1, 0.1, 0.7)))`.

[4]In the replication code, these results are reproduced by changing the estimation command from `ugarchfit(data=eps, spec=spec)` to `ugarchfit(data=eps, spec=spec, solver="nlminb")`.

$m(\widehat{\mathbf{\Sigma}}/\mathbf{\Sigma})$, is obtained by taking the average across the 1000 replications for each of the 9 entries. When $\eta_t \sim N(0,1)$, this produces the following averages:

```
##tseries:
          intercept  arch1 garch1
intercept    1.0702 1.0489 1.0656
arch1        1.0489 1.0256 1.0366
garch1       1.0656 1.0366 1.0566

##fGarch:
          intercept  arch1 garch1
intercept    1.0596 1.0335 1.0548
arch1        1.0335 1.0126 1.0229
garch1       1.0548 1.0229 1.0455

##rugarch:
          intercept  arch1 garch1
intercept    1.0869 1.0723 1.0848
arch1        1.0723 1.0280 1.0501
garch1       1.0848 1.0501 1.0748

##garchx:
          intercept  arch1 garch1
intercept    1.0630 1.0350 1.0576
arch1        1.0350 1.0142 1.0244
garch1       1.0576 1.0244 1.0479
```

Three general characteristics are clear. First, the ratios are all greater than 1. In other words, all packages tend to return estimated coefficient-covariances that are too large in absolute terms. In particular, standard errors tend to be too high. Second, the size of the biases is similar across packages. Those of **rugarch** are slightly higher than those of the other packages, but the difference may disappear if a larger number of replications is used. Third, the magnitude of the relative bias is fairly low since they all lie between 1.26% and 8.69%.

When $\eta_t \sim t(5)$, the simulations produce the following averages:

```
##tseries:
          intercept  arch1 garch1
intercept    0.1082 0.1038 0.1088
arch1        0.1038 0.0952 0.1002
garch1       0.1088 0.1002 0.1070

##fGarch:
          intercept  arch1 garch1
intercept    0.9088 1.0198 0.9098
arch1        1.0198 1.0721 0.9858
garch1       0.9098 0.9858 0.9062

##rugarch:
          intercept  arch1 garch1
intercept    0.8423 0.8596 0.8356
arch1        0.8596 0.8361 0.8349
garch1       0.8356 0.8349 0.8263

##garchx:
          intercept  arch1 garch1
intercept    0.9343 0.9017 0.9200
arch1        0.9017 0.8973 0.8903
garch1       0.9200 0.8903 0.9043
```

The downwards relative bias of about 90% produced by **tseries** simply reflects that a non-normality robust option is not available in that package. However, the size of the bias is larger than expected. If it were simply due to $E(\eta_t^4)$ in the expression for $\mathbf{\Sigma}$ being erroneous (3 instead of 9 in the simulations), then the ratios should have been in the vicinity of $(3-1)/(9-1) = 0.29$. Instead, they are substantially lower, since they all lie in the vicinity of 0.10. In other words, the way estimation is implemented by **tseries** induces a downward bias in the standard errors that can be substantially larger than expected when η_t is fat-tailed. The relative bias produced by **fGarch**, **rugarch**, and **garchx** is more moderate since they all lie less than 18% away from the true values. While the relative bias of **rugarch** is slightly larger than those of **fGarch** and **garchx**, the general tendency of the three packages is that the bias is downwards.

4 Comparison of speed

In nominal terms, all four packages are fairly fast. On an average contemporary laptop, for example, estimation of a plain GARCH(1,1) usually takes less than a second if the number of observations is 10000 or less. The reason is that all four packages use compiled C/C++ or Fortran code in the recursion, i.e., the computationally most demanding part. While the nominal speed difference is almost unnoticeable in simple models with small T, the relative difference among the packages is significant. In other words, when T is large or when a large number of models are estimated (as in Monte Carlo simulations), then the choice of a package can be important.

The comparison is undertaken with the **microbenchmark** (Mersmann, 2019) package version 1.4-7, and the average estimation-time of four GARCH models are compared:

$$\epsilon_t = \sigma_t \eta_t, \qquad \eta_t \overset{iid}{\sim} N(0,1),$$

1	GARCH(1,1):	$\sigma_t^2 = \omega + \alpha_1 \epsilon_{t-1}^2 + \beta_1 \sigma_{t-1}^2$
2	GARCH(2,2):	$\sigma_t^2 = \omega + \sum_{i=1}^2 \alpha_i \epsilon_{t-i}^2 + \sum_{j=1}^2 \beta_j \sigma_{t-j}^2$
3	GARCH(1,1) w/asymmetry:	$\sigma_t^2 = \omega + \alpha_1 \epsilon_{t-1}^2 + \beta_1 \sigma_{t-1}^2 + \gamma_1 I_{\{\epsilon_{t-1}<0\}} \epsilon_{t-1}^2$
4	GARCH(1,1)-X:	$\sigma_t^2 = \omega + \alpha_1 \epsilon_{t-1}^2 + \beta_1 \sigma_{t-1}^2 + \lambda_1 x_{t-1}$

The parameters of the Data Generating Processes (DGPs) are

$$(\omega, \alpha_1, \beta_1, \alpha_2, \beta_2, \gamma_1, \lambda_1) = (0.2, 0.1, 0.8, 0.00, 0.00, 0.05, 0.3),$$

and x_t in DGP number 4 is governed by the AR(1) process

$$x_t = 0.5 x_{t-1} + 0.1 u_t, \qquad u_t \overset{iid}{\sim} N(0,1).$$

The comparison is made for sample sizes $T = 1000$ and $T = 2000$.

Table 3 contains the results of the comparison in relative terms. A value of 1.0 means the package is the fastest on average for the experiment in question. A value of 7.15 means the average estimation time of the package is 7.15 times larger than the average of the fastest, and so on. The entry is empty if the GARCH specification cannot be estimated by the package.The overall pattern of the results is clear: **tseries** is the fastest among the models it can estimate, **garchx** is the second fastest, **fGarch** is the third fastest, and **rugarch** is the slowest. Another salient feature is how much faster **tseries** is relative to the other packages. This is particularly striking for the GARCH(2,2), where the second-fastest package – **garchx** – is about 5 to 6 times slower, and the slowest package – **rugarch** – is about 28 to 30 times slower. A third notable characteristic is that the relative differences tend to fall as the sample size T increases. For example, **rugarch** is about 17 times slower than **tseries** when $T = 1000$, but only about 13 times slower when $T = 2000$. As for the specifications that **tseries** are not capable of estimating, **garchx** is the fastest. Notably so compared with **fGarch** and even more so compared with **rugarch**.

5 Conclusions

This paper provides an overview of the package **garchx** and compares it with three prominent CRAN-packages that offer GARCH estimation routines: **tseries**, **fGarch**, and **rugarch**. While **garchx** does not offer all the GARCH-specifications available in **rugarch**, it is much more flexible than **tseries**, and it also offers the important possibility of including covariates. This feature is not available in **fGarch**. The package **garchx** also offers additional features that are not available in the other packages: i) A dependence-robust coefficient covariance, ii) functions that facilitate hypothesis testing under nullity, iii) zero-coefficient restrictions by omission, and iv) a function that computes the asymptotic coefficient-covariance of a GARCH model.

In a Monte Carlo study of the packages, the large sample properties of the normal Quasi ML (QML) estimator were studied. There, it was revealed that **fGarch** and **garchx** are numerically more robust (under default options) than **tseries** and **rugarch**. However, in the case of **tseries**, the study also

DGP	T	**tseries**	**fGarch**	**rugarch**	**garchx**
1 GARCH$(1,1)$:	1000	1.00	7.15	17.42	2.69
	2000	1.00	6.28	12.89	1.85
2 GARCH$(2,2)$:	1000	1.00	10.14	29.78	5.27
	2000	1.00	14.72	27.79	6.27
3 GARCH$(1,1,1)$:	1000		2.26	14.72	1.00
	2000		2.97	9.91	1.00
4 GARCH$(1,1)$-X:	1000			5.90	1.00
	2000			6.36	1.00

Table 3: Relative speed comparison of **tseries** version 0.10-47 (Trapletti and Hornik, 2019), **fGarch** version R 3.0.1 (Wuertz et al., 2020), **rugarch** version 1.4-2 (Ghalanos, 2020), and **garchx** version 1.1 (Sucarrat, 2020). A value of 1.00 means the package is the fastest on average for the experiment in question. A value of 7.15 means the average estimation time of the package is 7.15 times larger than the average of the fastest, and so on. The entry is empty if the GARCH specification cannot be estimated by the package.

revealed how its numerical robustness could be improved straightforwardly by simply changing the initial parameter values. In the case of **rugarch**, it is less clear how the numerical robustness can be improved. The study also revealed that the standard errors of **tseries** could be substantially biased downwards when η_t is non-normal. A bias is expected since **tseries** does not offer a non-normality robust coefficient-covariance. However, the bias is larger than suggested by the underlying estimation theory.

In a relative speed comparison of the packages, it emerged that the least flexible package – **tseries** – is notably faster than the other packages. Next, **garchx** is the second fastest (1.85 to 6.27 times slower in the experiments), **fGarch** is the third fastest, and **rugarch** is the slowest. The experiments also revealed that the difference could be larger in higher-order models. For example, in the estimation of a GARCH(2,2), **rugarch** was about 28 times slower than **tseries**. In estimating a plain GARCH(1,1), by contrast, it was only 13 to 17 times slower. Another finding was that the difference seems to fall in sample size: The larger the sample size, the smaller the difference in speed.

Bibliography

T. Bollerslev. Generalized autoregressive conditional heteroscedasticity. *Journal of Econometrics*, 31: 307–327, 1986. [p335]

T. Bollerslev and J. Wooldridge. Quasi-Maximum Likelihood Estimation and Inference in Dynamic Models with Time Varying Covariances. *Econometric Reviews*, 11:143–172, 1992. [p336, 337, 338]

C. Brownlees, F. Cipollini, and G. Gallo. Multiplicative Error Models. In L. Bauwens, C. Hafner, and S. Laurent, editors, *Handbook of Volatility Models and Their Applications*, pages 223–247. Wiley, New Jersey, 2012. [p335]

F. C. Drost and T. E. Nijman. Temporal Aggregation of Garch Processes. *Econometrica*, 61:909–927, 1993. [p338]

R. Engle. Autoregressive Conditional Heteroscedasticity with Estimates of the Variance of United Kingdom Inflations. *Econometrica*, 50:987–1008, 1982. [p335]

R. F. Engle and J. R. Russell. Autoregressive Conditional Duration: A New Model of Irregularly Spaced Transaction Data. *Econometrica*, 66:1127–1162, 1998. [p335]

J. C. Escanciano. Quasi-maximum likelihood estimation of semi-strong GARCH models. *Econometric Theory*, 25:561–570, 2009. [p337]

C. Francq and L. Q. Thieu. Qml inference for volatility models with covariates. *Econometric Theory*, 2018. doi: https://doi.org/10.1017/S0266466617000512. `https://doi.org/10.1017/S0266466617000512`. [p337, 338, 340, 341, 349]

C. Francq and J.-M. Zakoïan. *GARCH Models*. Wiley, New York, 2019. 2nd. Edition. [p343]

J. Geweke. Modelling the Persistence of Conditional Variance: A Comment. *Econometric Reviews*, 5: 57–61, 1986. [p335]

A. Ghalanos. *rugarch: Univariate GARCH Models*, 2020. URL https://CRAN.R-project.org/package=rugarch. R package version 1.4-2. [p335, 338, 345, 348]

B. E. Hansen. Autoregressive Conditional Density Estimation. *International Economic Review*, 35: 705–730, 1994. [p338]

P. R. Hansen, Z. Huan, and H. H. Shek. Realized GARCH: A Joint Model for Returns and Realized Measures of Volatility. *Journal of Applied Econometrics*, 27:877–906, 2012. [p337, 339, 340, 341, 342]

O. Mersmann. *microbenchmark: Accurate Timing Functions*, 2019. URL https://CRAN.R-project.org/package=microbenchmark. R package version 1.4-7. [p347]

A. Milhøj. A Multiplicative Parametrization of ARCH Models. Research Report 101, University of Copenhagen: Institute of Statistics, 1987. [p335]

S. Pantula. Modelling the Persistence of Conditional Variance: A Comment. *Econometric Reviews*, 5: 71–73, 1986. [p335]

J. A. Ryan and J. M. Ulrich. *xts: eXtensible Time Series*, 2014. URL https://CRAN.R-project.org/package=xts. R package version 0.12-0. [p339]

G. Sucarrat. *garchx: Flexible and Robust GARCH-X Modelling*, 2020. URL https://CRAN.R-project.org/package=garchx. R package version 1.1. [p338, 345, 348]

G. Sucarrat and S. Grønneberg. Risk Estimation with a Time Varying Probability of Zero Returns. *Journal of Financial Econometrics*, 2020. Forthcoming. DOI: https://doi.org/10.1093/jjfinec/nbaa014. [p338]

A. Trapletti and K. Hornik. *tseries: Time Series Analysis and Computational Finance*, 2019. URL https://CRAN.R-project.org/package=tseries. R package version 0.10-47. [p335, 338, 345, 348]

D. Wuertz, T. Setz, Y. Chalabi, C. Boudt, P. Chausse, and M. Miklovac. *fGarch: Rmetrics - Autoregressive Conditional Heteroskedastic Modelling*, 2020. URL https://CRAN.R-project.org/package=fGarch. R package version 3042.83.2. [p335, 338, 345, 348]

A. Zeileis and G. Grothendieck. zoo: S3 infrastructure for regular and irregular time series. *Journal of Statistical Software*, 14(6):1–27, 2005. URL http://www.jstatsoft.org/v14/i06/. [p339]

Genaro Sucarrat
BI Norwegian Business School
Nydalsveien 37
0484 Oslo
Norway
genaro.sucarrat@bi.no

6 Appendix: Estimation of the dependence robust coefficient-covariance

Francq and Thieu (2018) show that

$$\begin{aligned} \boldsymbol{J} &= E\left(\frac{\partial^2 l_t(\boldsymbol{\vartheta}_0)}{\partial\boldsymbol{\vartheta}\partial\boldsymbol{\vartheta}'}\right) = E\left(\frac{1}{\sigma_t^4(\boldsymbol{\vartheta}_0)}\frac{\partial\sigma_t^2(\boldsymbol{\vartheta}_0)}{\partial\boldsymbol{\vartheta}}\frac{\partial\sigma_t^2(\boldsymbol{\vartheta}_0)}{\partial\boldsymbol{\vartheta}'}\right) \\ \boldsymbol{I} &= E\left[\left\{E\left(\eta_t^4|\mathcal{F}_{t-1}\right)-1\right\}\frac{1}{\sigma_t^4(\boldsymbol{\vartheta}_0)}\frac{\partial\sigma_t^2(\boldsymbol{\vartheta}_0)}{\partial\boldsymbol{\vartheta}}\frac{\partial\sigma_t^2(\boldsymbol{\vartheta}_0)}{\partial\boldsymbol{\vartheta}'}\right] \end{aligned}$$

This means

$$\begin{aligned} \boldsymbol{I} &= E\left[\left\{E\left(\eta_t^4|\mathcal{F}_{t-1}\right)\right\}\frac{1}{\sigma_t^4(\boldsymbol{\vartheta}_0)}\frac{\partial\sigma_t^2(\boldsymbol{\vartheta}_0)}{\partial\boldsymbol{\vartheta}}\frac{\partial\sigma_t^2(\boldsymbol{\vartheta}_0)}{\partial\boldsymbol{\vartheta}'}\right] - \boldsymbol{J} \\ &= E\left[\left(\frac{\epsilon_t^2}{\sigma_t^4(\boldsymbol{\vartheta}_0)}\frac{\partial\sigma_t^2(\boldsymbol{\vartheta}_0)}{\partial\boldsymbol{\vartheta}}\right)\left(\frac{\epsilon_t^2}{\sigma_t^4(\boldsymbol{\vartheta}_0)}\frac{\partial\sigma_t^2(\boldsymbol{\vartheta}_0)}{\partial\boldsymbol{\vartheta}}\right)'\right] - \boldsymbol{J} \end{aligned}$$

The computationally challenging part to estimate in $\boldsymbol{I}$ is $\partial\sigma_t^2(\boldsymbol{\vartheta}_0)/\partial\boldsymbol{\vartheta}$ since it entails the computation of a numerically differentiated gradient of a recursion at each t. In **garchx**, this is implemented with `numericDeriv()` in the `vcov.garchx()` function.