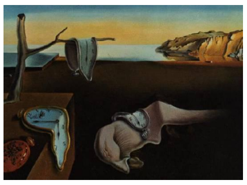

Journal of Statistical Software

September 2023, Volume 107, Issue 5. *doi: 10.18637/jss.v107.i05*

ARCHModels.jl: Estimating ARCH Models in Julia

Simon A. Broda
Lucerne University of
Applied Sciences and Arts

Marc S. Paolella
University of Zurich
Swiss Finance Institute

Abstract

This paper introduces **ARCHModels.jl**, a package for the Julia programming language that implements a number of univariate and multivariate autoregressive conditional heteroskedasticity models. This model class is the workhorse tool for modeling the conditional volatility of financial assets. The distinguishing feature of these models is that they model the latent volatility as a (deterministic) function of past returns and volatilities. This recursive structure results in loop-heavy code which, due to its just-in-time compiler, Julia is well-equipped to handle. As such, the entire package is written in Julia, without any binary dependencies. We benchmark the performance of **ARCHModels.jl** against popular implementations in MATLAB, R, and Python, and illustrate its use in a detailed case study.

Keywords: ARCH, GARCH, CCC, DCC, Value at Risk, Julia.

1. Introduction

Financial returns data at daily or higher frequency display a number of *stylized facts*, including volatility clustering (large, in absolute value, returns tend to cluster together), heavy tails, and statistical leverage, among others. Modeling these lies at the heart of much of financial econometrics, because the volatility and conditional distribution of an asset (or a group of assets) are key ingredients in applications such as risk management, portfolio optimization, and derivative pricing. Autoregressive conditional heteroskedasticity (ARCH) models form the most widely used class of models for capturing these features. In an ARCH-type model, the latent volatility σ_t of the return r_t of an asset is modeled in terms of past returns and volatilities. For example, given a sample of daily asset returns $\{r_t\}_{t\in\{1,\dots,T\}}$, the basic generalized ARCH (GARCH) model, due to Bollerslev (1986), is

$$r_t = \sigma_t z_t, \quad z_t \sim \mathrm{N}(0,1), \quad \sigma_t^2 = \omega + \alpha r_{t-1}^2 + \beta\sigma_{t-1}^2, \quad \omega, \alpha, \beta > 0, \quad \alpha + \beta < 1,$$

where the *stationarity condition* $\alpha + \beta < 1$ keeps the variance from exploding. The GARCH

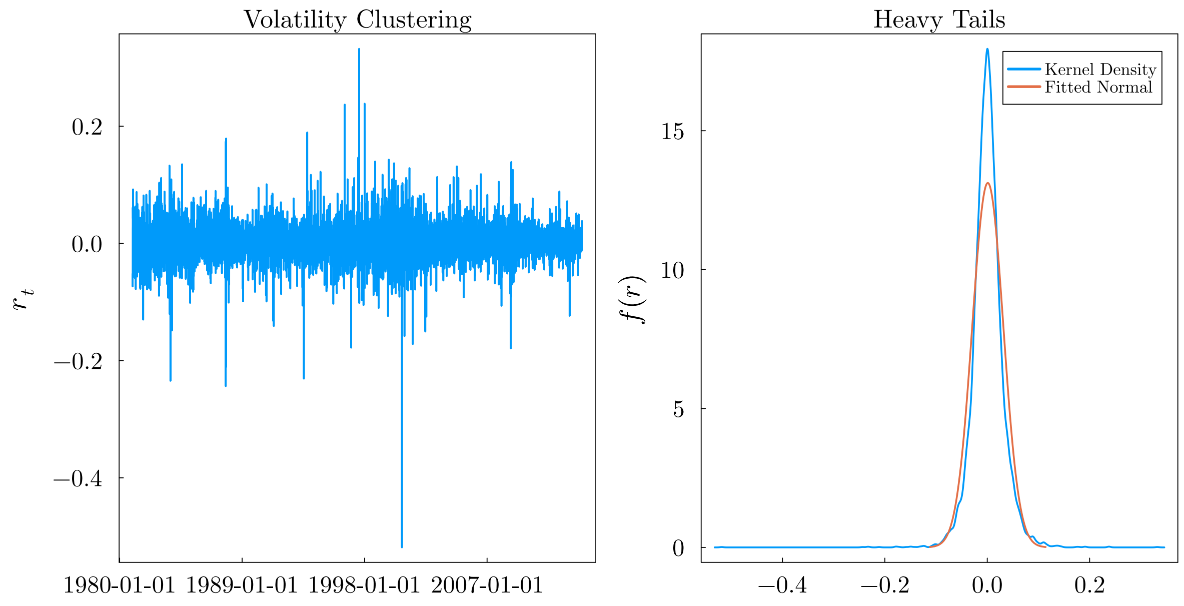

Figure 1: Volatility clustering (left panel) and heavy tails (right panel) illustrated by way of daily returns on Apple stock.

model extends the ARCH model (in which $\beta = 0$) of Engle (1982), who in 2003 was awarded a Nobel Memorial Prize in Economic Sciences for its development.

Due to their popularity, the ARCH and GARCH models and their various extensions have been implemented in many commercial and free programming environments; examples include Ghalanos's (2022) **rugarch** package for R (R Core Team 2023), Sheppard *et al.*'s (2022) **arch** package for Python (Van Rossum 1995), and MATLAB's (The MathWorks Inc. 2021) **Econometrics** toolbox. These implementations all outsource the evaluation of the likelihood function to a compiled language, because the recursive nature of (1) defies any attempt at "vectorizing" it, a common recommendation for performant implementations in interpreted languages. Julia (Bezanson, Edelman, Karpinski, and Shah 2017) is unique among these high-level languages because its just-in-time compiler allows us to keep even the tight loops in Julia itself without sacrificing performance. In fact, we show in Section 6 below that our implementation outperforms those mentioned above. This is partly due to the use of automatic differentiation for the computation of gradients in maximizing the likelihood, via **Optim.jl** (Mogensen and Riseth 2018; Mogensen, Riseth *et al.* 2023) and **ForwardDiff.jl** (Revels, Lubin, and Papamarkou 2016; Revels, Lubin, Papamarkou *et al.* 2023).

A registered Julia package distributed under the permissive MIT License, **ARCHModels.jl** is easily installed with Julia's package manager using the commands below.

```
julia> using Pkg
julia> Pkg.add("ARCHModels")
```

ARCHModels.jl implements estimation, model selection, simulation, and Value at Risk calculations for a variety of univariate and multivariate ARCH models, for different choices of standardized innovation distributions. The conditional mean can be specified as either zero, an intercept, a linear regression model, or an autoregressive moving average model. The package is designed to be easy to extend with other volatility specifications and distributions, and integrates with the relevant parts of the Julia ecosystem, such as **DataFrames.jl** (Bouchet-Valat and Kamiński 2023; Kamiński, White, Bouchet-Valat *et al.* 2023), **Distributions.jl** (Besançon *et al.* 2021; Lin *et al.* 2023b), **GLM.jl** (Bates *et al.* 2023), **HypothesisTests.jl** (Kornblith *et al.* 2023), and **StatsBase.jl** (Lin *et al.* 2023a).

The remainder of the paper is as follows: Section 2 outlines the theory of ARCH models. Section 3 describes the type hierarchy of **ARCHModels.jl**, and hence the available models. Section 4 provides details on the implementation. Section 5 presents a case study detailing the use of the package. Section 6 offers a comparison with implementations in other languages. Section 7 concludes.

2. Theoretical background

2.1. ARCH models

Consider a sample of daily asset returns $\{r_t\}_{t\in\{1,\ldots,T\}}$. All models covered in this package share the same basic structure, in that they decompose the return into a conditional mean and a mean-zero innovation. In the univariate case,

$$r_t = \mu_t + a_t, \quad \mu_t \equiv \mathbb{E}[r_t \mid \mathcal{F}_{t-1}], \quad \sigma_t^2 \equiv \mathbb{E}[a_t^2 \mid \mathcal{F}_{t-1}],$$

$z_t \equiv a_t/\sigma_t$ is identically and independently distributed according to some law with mean zero and unit variance, and $\{\mathcal{F}_t\}$ is the natural filtration of $\{r_t\}$, i.e., it encodes information about past returns.

A complete model requires specifying the conditional mean μ_t, the density of z_t, and the conditional volatility σ_t (Section 3.1 details the possible choices for each). ARCH models specify the latter in terms of past returns, conditional (co-)variances, and potentially other variables. For concreteness, let $z_t \sim \mathrm{N}(0,1)$ and $\mu_t \equiv 0$ for now, and let

$$\sigma_t^2 = \omega + \alpha a_{t-1}^2 + \beta\sigma_{t-1}^2, \quad \omega,\alpha,\beta > 0, \quad \alpha+\beta < 1. \tag{1}$$

This yields the GARCH(1, 1) model in (1) (the GARCH(p, q) model generalizes (1) by including more lags of σ_t^2 and a_t^2). It is evident that large (in absolute value) returns $r_t = a_t$ increase the next day's conditional return variance σ_t^2. This allows the model to capture the volatility clustering that is present in the returns of most daily financial returns series. The left panel of Figure 1 shows this for the daily returns on Apple stock.

A notable feature of the GARCH model, and in fact all ARCH-type models, is that the conditional volatility σ_t is completely determined using information available at time $t-1$, i.e., σ_t is measurable with respect to $\mathcal{F}_{t-1}$. This feature distinguishes ARCH models from stochastic volatility models, another popular model class, and makes them comparatively straightforward to estimate by maximum likelihood. As an aside, a related class of models that also feature an $\mathcal{F}_{t-1}$-measurable volatility is that of Generalized Autoregressive Score, or GAS, models, introduced by Creal, Koopman, and Lucas (2013) and Harvey (2013). The package **ScoreDrivenModels.jl** (Bodin, Saavedra, Fernandes, and Street 2020, 2022) implements these in Julia.

Let $f_z(z;\theta)$ denote the density of z_t, parameterized by the (possibly empty) tuple θ, and let Θ denote a tuple containing θ along with the parameters in the specifications for μ_t and σ_t. For example, in the GARCH(1, 1) model with standard Gaussian errors and $\mu_t = 0$, $\Theta = (\omega, \alpha, \beta)$. Further, let τ denote the number of pre-sample values needed for computing μ_t and σ_t; e.g., in the GARCH(1, 1) model, $\tau = 1$. Then the log-likelihood, conditional on these τ pre-sample values, is

$$\begin{aligned}\ell(\Theta) &\equiv \log f_{\mathbf{r}_{\tau+1:t}|\mathbf{r}_{1:\tau}}(\mathbf{r}_{\tau+1:t};\Theta)\\ &= \log \prod_{t=\tau+1}^{T} f_{r_t|\mathbf{r}_{1:t-1}}(r_t;\Theta)\\ &= \sum_{t=\tau+1}^{T} \ell_t(\Theta), \qquad (2)\end{aligned}$$

where

$$\ell_t(\Theta) = -\log \sigma_t + \log f_z([r_t - \mu_t]/\sigma_t; \theta)$$

and $\mathbf{r}_{s:t} \equiv [r_s, r_{s+1}, \dots, r_t]$ denotes a vector containing the returns from period s through t. The log-likelihood at (2) can be maximized numerically; see Section 4 for details. Notice that the recursive structure of σ_t common to all ARCH-type models implies that the computation of (2) cannot be "vectorized" into a single library call to, say, **NumPy** (Harris *et al.* 2020), a common technique for speeding up numerical calculations in interpreted languages such as Python. Instead, it requires a loop over t. This is the reason that the **rugarch** package for R, the **arch** package for Python, and MATLAB's **Econometrics** toolbox all implement (2) in a compiled low-level language: Cython (Behnel, Bradshaw, Citro, Dalcin, Seljebotn, and Smith 2011) in the case of **arch**, C/C++ for **rugarch** and MATLAB. The just-in-time compiled nature of Julia obviates this need: **ARCHModels.jl** is entirely implemented in Julia, and nevertheless typically faster than the above-mentioned packages, see Section 6.

Many generalizations of the basic (G)ARCH model have been proposed in the literature, and **ARCHModels.jl** implements a number of these. For example, one limitation of the standard GARCH model is that the conditional variance reacts equally strongly to positive and negative return shocks, because a_t only enters as its square in (1). Typically however, volatility reacts more strongly to a negative shock. This phenomenon is known as the *statistical leverage effect* and has led to the development of *asymmetric* ARCH models. Section 3 describes the two that are available in **ARCHModels.jl**.

An important aspect in ARCH modeling is model selection, in particular, selecting the right lag orders. Two widely used approaches are based on minimizing some information criterion, and diagnostic testing. Information criteria embody a tradeoff between model fit and parsimony, similarly to the adjusted R^2 in regression. For example, the AIC (Akaike information criterion, Akaike 1974) and BIC (Bayesian information criterion, Schwarz 1978) are respectively defined as

$$\text{AIC} \equiv -2\ell(\Theta^*) + 2|\Theta| \quad \text{and} \quad \text{BIC} \equiv -2\ell(\Theta^*) + |\Theta|\log(T),$$

where Θ^* denotes the maximum likelihood estimator (MLE) and $|\Theta|$ represents the number of parameters in the model. Given a collection of candidate models, one selects the one that minimizes one of these. Note that the BIC puts a greater penalty on additional parameters, and hence tends to select smaller models.

Diagnostic testing of volatility models generally relies on the estimated conditional volatilities $\hat{\sigma}_t$ and standardized residuals $\hat{z}_t \equiv (r_t - \hat{\mu}_t)/\hat{\sigma}_t$. For example, the ARCH-LM (Lagrange multiplier) test of Engle (1982) is based on the R^2 of the auxiliary regression

$$\hat{z}_t^2 = \gamma_0 + \sum_{i=1}^{p} \gamma_i \hat{z}_{t-i}^2 + \varepsilon_t,$$

where p must be specified by the user. The null hypothesis of the test is that no volatility clustering remains in the standardized residuals. The test rejects for large values of $T \cdot R^2$.

2.2. Value at Risk

A prominent application of ARCH models, and volatility models in general, is estimating the Value at Risk (VaR). The VaR of an asset (or portfolio of assets) is a measure of its risk. For a given level α, it is defined in terms of a quantile of the return distribution, i.e., it solves

$$\mathbb{P}[r_{t+1} \leq -\operatorname{VaR}_{t+1}^{(\alpha)} \mid \mathcal{F}_t] = \alpha.$$

Intuitively, if $\operatorname{VaR}_{t+1}^{(\alpha)} = 3\%$, then the loss on the asset or portfolio will only exceed 3% in the worst $100 \cdot \alpha\%$ of cases. It should be noted that different definitions of the Value at Risk exist. The one used here is sometimes known as the return-VaR, because it is defined in terms of returns, rather than a monetary amount. It is easily converted to what is sometimes known as the \$-VaR by multiplying it by the exposure, i.e., the value of the asset or portfolio.

Banks are mandated to report the 1% Value at Risk of their portfolio to the relevant regulatory body every two weeks. They can either use the model defined in the Basel Accords to estimate the VaR, or use their own internal models. In the latter case, they have to demonstrate the correctness of the model to the regulatory body by *backtesting* it on historical data. This amounts to producing VaR forecasts

$$\left\{ \widehat{\operatorname{VaR}}_{t+1}^{(\alpha)} \right\}$$

for a set of historical returns, and then analyzing the *VaR violations*, i.e., those days in the sample on which the relative loss exceeded the estimated VaR. If the model is correct, then this should happen on $100 \cdot \alpha\%$ of trading days, on average. In addition, it is desirable that these violations not cluster in time, i.e., they should not be autocorrelated.

One way to test the correctness of the VaR specification is to use the dynamic quantile test of Engle and Manganelli (2004). Let

$$I_t \equiv \mathbf{1}\left\{ r_t < -\widehat{\operatorname{VaR}}_t^{(\alpha)} \right\} - \alpha.$$

The indicator function in the expression above takes the value one if a VaR violation occurred on day t, and zero otherwise. Engle and Manganelli's (2004) test regresses the I_t on an intercept and p lags of both I_t and the VaR forecasts themselves. The null hypothesis that the VaR specification is correct is rejected if the coefficients in this model are jointly significant.

2.3. Multivariate models

Multivariate ARCH models generalize the theory to deal with several assets at once. They aim to model the joint evolution of the conditional volatilities of several assets, as well as their conditional covariances. These quantities play important roles in, e.g., modeling the VaR of a portfolio of assets. The general structure of a multivariate ARCH model is a straightforward extension of the univariate case. It reads

$$r_t = \mu_t + a_t, \quad \mu_t \equiv \mathbb{E}[r_t \mid \mathcal{F}_{t-1}], \quad \Sigma_t \equiv \mathbb{E}[a_t a_t^\top \mid \mathcal{F}_{t-1}], \tag{3}$$

where now $r_t \in \mathbb{R}^d$. In principle, a specific multivariate model is obtained by simply specifying the dynamics of Σ_t (along with μ_t and an error distribution). For example, the natural generalization of (1) to the multivariate case would be to allow each element of Σ_t to depend on each element of both Σ_{t-1} and $a_{t-1}a_{t-1}^\top$. This results in the so-called VECH model (Bollerslev, Engle, and Wooldridge 1988)

$$\mathrm{vech}(\Sigma_t) = a_0 + A_1 \mathrm{vech}(a_{t-1}a_{t-1}^\top) + B_1 \mathrm{vech}(\Sigma_{t-1}),$$

where A_1 and B_1 have dimension $k \times k$, $k = d(d+1)/2$, and a_0 is a k-vector. The model takes its name from the operator $\mathrm{vech}(A)$, which stacks the lower-triangular elements of symmetric matrix A into a vector. In principle, this model could then be estimated by maximum likelihood as in the univariate case. Unfortunately, the number of parameters in this model is $O(d^4)$, so this becomes infeasible beyond three or four assets (the *curse of dimensionality*). In addition, it is difficult to ensure that the resulting matrices $\{\Sigma_t\}$ are positive definite, as is required of covariance matrices.

Much research on multivariate ARCH models has focused on solving these two issues, by imposing more structure on the dynamics of Σ_t. The class of so-called conditional correlation models has been particularly successful, and **ARCHModels.jl** implements two of these: The CCC (constant conditional correlation) model of Bollerslev (1990), and the DCC (dynamic conditional correlation) model of Engle (2002); see Section 3.2 for details. One of the two estimators for the DCC model implemented in the package, due to Engle, Ledoit, and Wolf (2019), makes estimation feasible for a large number of assets (in the thousands).

3. Available models and type hierarchy

3.1. Univariate type hierarchy

This package represents a univariate ARCH model as an instance of `UnivariateARCHModel`, which implements the interface of `StatisticalModel` from **StatsBase.jl**.

An instance of `UnivariateARCHModel` contains a vector of data (such as equity returns), and encapsulates information about the volatility specification (e.g., ARCH or GARCH), the mean specification (e.g., whether an intercept is included), and the error distribution. Hence, the constructor for `UnivariateARCHModel` takes two mandatory arguments: An instance of a subtype of `UnivariateVolatilitySpec` (see below), and a vector of returns. The mean specification and error distribution can be changed via the keyword arguments `meanspec` and `dist`, which respectively default to a zero mean and a standard Gaussian. For example, to construct a GARCH(1, 1) model for a vector of returns `data`, one would run the following.

```
julia> spec = GARCH{1, 1}([1., .9, .05])
julia> am = UnivariateARCHModel(spec, data)
```

It should, however, rarely be necessary to construct a `UnivariateARCHModel` manually via its constructor. Typically, instances of it are created by calling `fit`, `selectmodel`, or `simulate`. Details will be given in Section 5 below.

Since `UnivariateARCHModel` implements the interface of `StatisticalModel` from **StatsBase.jl**, one may call `aic`, `bic`, `coef`, `coefnames`, `confint`, `dof`, `informationmatrix`, `isfitted`, `loglikelihood`, `nobs`, `score`, `stderror`, `vcov`, etc. on its instances. Other useful methods include `means`, `volatilities`, `residuals`, and `predict`.

Volatility specifications

Volatility specifications describe the evolution of σ_t. They are modeled as subtypes of `UnivariateVolatilitySpec`. There is one type for each class of (G)ARCH model, parameterized by the number(s) of lags (e.g., p, q for a GARCH(p, q) model).

The simplest volatility specification is given by the ARCH(q) model, due to Engle (1982). It reads

$$\sigma_t^2 = \omega + \sum_{i=1}^{q} \alpha_i a_{t-i}^2, \quad \omega, \alpha_i > 0, \quad \sum_{i=1}^{q} \alpha_i < 1. \tag{4}$$

The corresponding type is `ARCH{q}`.

The GARCH(p, q) model, due to Bollerslev (1986), generalizes the ARCH(q) model by including lagged values of the squared volatility on the right hand side of (4). This renders the conditional variance as

$$\sigma_t^2 = \omega + \sum_{i=1}^{p} \beta_i \sigma_{t-i}^2 + \sum_{i=1}^{q} \alpha_i a_{t-i}^2, \quad \omega, \alpha_i, \beta_i > 0, \quad \sum_{i=1}^{\max p,q} \alpha_i + \beta_i < 1. \tag{5}$$

It is available as `GARCH{p, q}`.

The ARCH and GARCH models are special cases of a more general class of models, known as threshold GARCH (TGARCH), due to Glosten, Jagannathan, and Runkle (1993). The model is also known as GJR-GARCH in the literature after the initials of the authors, to avoid confusion with the TGARCH model of Zakoian (1994). The latter essentially uses the absolute value in place of the square of the residuals in (6) below.

The model takes the form

$$\sigma_t^2 = \omega + \sum_{i=1}^{o} \gamma_i a_{t-i}^2 \mathbf{1}\left\{a_{t-i} < 0\right\} + \sum_{i=1}^{p} \beta_i \sigma_{t-i}^2 + \sum_{i=1}^{q} \alpha_i a_{t-i}^2, \tag{6}$$

where

$$\omega, \alpha_i, \beta_i, \gamma_i > 0 \quad \text{and} \quad \sum_{i=1}^{\max o,p,q} \alpha_i + \beta_i + \gamma_i/2 < 1.$$

The TGARCH model allows the volatility to react differently (typically more strongly) to negative shocks, a feature known as the (statistical) leverage effect. It is available as `TGARCH{o, p, q}`.

Finally, the exponential GARCH (EGARCH) volatility specification, due to Nelson (1991), is

$$\log(\sigma_t^2) = \omega + \sum_{i=1}^{o} \gamma_i z_{t-i} + \sum_{i=1}^{p} \beta_i \log(\sigma_{t-i}^2) + \sum_{i=1}^{q} \alpha_i (|z_{t-i}| - \sqrt{2/\pi}), \quad z_t = a_t/\sigma_t, \quad \sum_{i=1}^{p} \beta_i < 1.$$

Like the TGARCH model, it can account for the leverage effect. The corresponding type is `EGARCH{o, p, q}`.

The constructors for the volatility specifications take a coefficient vector as input, where the order of the parameters is such that all parameters pertaining to the first type parameter (e.g., p for the GARCH(p, q) model) appear before those pertaining to the second (q), and so on. For example, an EGARCH(1, 1, 1) model with $\omega = -0.003$, $\gamma_1 = -0.03$, $\beta_1 = 0.99$ and $\alpha_1 = 0.2$ is obtained as follows.

```
julia> EGARCH{1, 1, 1}([-0.003, -0.03, 0.99, 0.2])
```

Explicitly creating instances of volatility specifications is only necessary for simulation (see Section 5). For fitting, passing the type is sufficient.

Mean specifications

Mean specifications serve to specify μ_t. They are modeled as subtypes of `MeanSpec`. Their instances contain the parameters as (possibly empty) vectors. Convenience constructors are provided where appropriate, though as with volatility specifications, constructing them explicitly is only required for simulation, not for fitting. The following specifications are available.

A zero mean, i.e., $\mu_t = 0$, is available as `NoIntercept`. An intercept ($\mu_t = \mu$) is obtained as `Intercept`. `Regression` allows one to specify a linear regression model, as in

$$\mu_t = \mathbf{x}_t^\top \boldsymbol{\beta}.$$

Unlike the other mean specifications, a regression requires external data, which the constructor expects as a matrix with observations in rows and variables in columns, as follows.

```
julia> reg = Regression(ones(100, 1))
```

This creates a regression model containing one regressor, given by a column of ones. This is equivalent to including an intercept in the model (the latter is however more memory efficient, as no design matrix needs to be stored). Another way to create a linear regression with ARCH errors is to pass a `LinearModel` or `TableRegressionModel` from **GLM.jl**; see Section 5 for an example.

Finally, an ARMA (autoregressive moving average) model specifies the conditional mean as

$$\mu_t = c + \sum_{i=1}^{p} \varphi_i r_{t-i} + \sum_{i=1}^{q} \theta_i a_{t-i}. \tag{7}$$

ARMA models are popular in time series econometrics because they are able to approximate the autocorrelation structure of *any* stationary process. It should be mentioned, however, that financial returns typically do not exhibit much autocorrelation; in fact, the weak form market efficiency hypothesis (Fama 1970) implies its absence. Nevertheless, the ARMA model (7) is available as `ARMA{p, q}`. The special cases of pure autoregressive and moving average models are also available, as `AR{p}` and `MA{q}`, respectively.

Distributions

Different standardized (mean zero, variance one) distributions for z_t are available as subtypes of `StandardizedDistribution`. `StandardizedDistribution` in turn inherits from

`Distribution{Univariate, Continuous}` from **Distributions.jl**, though not the entire interface need necessarily be implemented. Instances of `StandardizedDistribution` again hold their parameters as vectors, but convenience constructors are provided. Available distributions include the standard normal (`StdNormal`) with density

$$f_N(x) \propto \exp(-x^2/2),$$

the standardized Student's t (`StdT`) with density

$$f_{\mathrm{StdT}}(x;\nu) \propto (1 + x^2/(\nu-2))^{-(\nu+1)/2},$$

the standardized generalized error distribution (`StdGED`) with density

$$f_{\mathrm{GED}}(x;p) \propto \exp(-|x \cdot s|^p), \quad s = \sqrt{\Gamma(3/p)/\Gamma(1/p)},$$

and Hansen's (1994) skewed t distribution (`StdSkewT`) with density

$$f_{\mathrm{SkewT}}(x;\nu,\lambda) \propto \left(1 + \frac{(bx+a)^2}{(\nu-2)\big(1+\mathrm{sign}(x+a/b)\lambda\big)^2}\right)^{-(\nu+1)/2},$$

where

$$c = \frac{\Gamma((\nu+1)/2)}{\sqrt{\pi(\nu-2)}\Gamma(\nu/2)}, \quad a = 4\lambda c(\nu-2)/(\nu-1), \quad \text{and } b = \sqrt{1+3\lambda^2-a^2}.$$

In addition, it is possible to wrap a continuous univariate distribution from **Distributions.jl** in the `Standardized` wrapper type. Below, we use this feature to re-implement the standardized normal distribution.

```
julia> using Distributions: Normal
julia> const MyStdNormal = Standardized{Normal}
```

`MyStdNormal` can be used everywhere that a built-in distribution could, albeit with a speed penalty. Note also that if the underlying distribution (such as `Normal` in the example above) contains location and/or scale parameters, then these are no longer identifiable, which implies that the estimated covariance matrix of the estimators will be singular. A final remark concerns the domain of the parameters: The estimation process relies on a starting value for the parameters of the distribution, say $\theta \equiv (\theta_1, \ldots, \theta_p)^\top$. For a distribution wrapped in `Standardized`, the starting value for θ_i is taken to be a small positive value ε. This will fail if ε is not in the domain of θ_i. As an example, the standardized Student's t distribution is only defined for degrees of freedom larger than 2, because a finite variance is required for standardization. In that case, it is necessary to define a method for the (non-exported) function `startingvals` that returns a feasible vector of starting values, as follows.

```
julia> import ARCHModels: startingvals
julia> import Distributions: TDist
julia> const MyStdT = Standardized{TDist}
julia> startingvals(::Type{<:MyStdT}, data::Vector{T}) where T = T[3.]
```

3.2. Multivariate type hierarchy

Analogously to the univariate case, an instance of `MultivariateARCHModel` contains a matrix of data (with observations in rows and assets in columns), and encapsulates information about the covariance specification (CCC or DCC), the mean specification, and the error distribution. Like `UnivariateARCHModel`, it implements most of the interface of `StatisticalModel` and hence supports many of the same methods as `UnivariateARCHModels`, with a few noteworthy differences: The prediction targets for `predict` include covariances and correlations for respectively predicting Σ_t and the correlation matrix R_t, and the functions `covariances` and `correlations` respectively return the in-sample estimates of Σ_t and R_t. Details will be given in Section 5 below.

Covariance specifications

As discussed, the main challenge in multivariate ARCH modeling is the *curse of dimensionality*: Allowing each of the $d(d+1)/2$ free elements of Σ_t to depend on the past returns and covariances of all d other assets requires $O(d^4)$ parameters, unless additional structure is imposed. **ARCHModels.jl** focuses on conditional correlation models to approach this issue. These decompose Σ_t as

$$\Sigma_t = D_t R_t D_t,$$

where R_t is the conditional correlation matrix and D_t is a diagonal matrix containing the volatilities of the individual assets, which are modeled as univariate ARCH processes. The dynamics of Σ_t are modeled as subtypes of `MultivariateVolatilitySpec`.

It remains, then, to specify the dynamics of the conditional correlations R_t. The dynamic conditional correlation (DCC) model of Engle (2002) imposes a GARCH-type structure on these. In particular, for a DCC(p, q) model with covariance targeting,

$$R_{ij,t} = \frac{Q_{ij,t}}{\sqrt{Q_{ii,t}Q_{jj,t}}},$$

where

$$Q_t \equiv \bar{Q}(1-\bar{\alpha}-\bar{\beta}) + \sum_{i=1}^{p} \beta_i Q_{t-i} + \sum_{i=1}^{q} \alpha_i \epsilon_{t-i}\epsilon_{t-i}^\top,$$

$\alpha, \beta > 0$, $\alpha + \beta < 1$, $\bar{\alpha} \equiv \sum_{i=1}^{q} \alpha_i$, $\bar{\beta} \equiv \sum_{i=1}^{p} \beta_i$, $\epsilon_t \equiv D_t^{-1} a_t$, $Q_t = \operatorname{cov}(\epsilon_t \mid F_{t-1})$, and $\bar{Q} = \operatorname{cov}(\epsilon_t)$. It is available as `DCC{p, q}`. Its constructor takes $\bar{Q}$, a vector of coefficients, and a vector of `UnivariateARCHModels` as inputs. For example, a bivariate DCC(1, 1) model with GARCH(1, 1) dynamics for the individual volatilities is obtained as follows.

```
julia> DCC{1, 1}([1. .5; .5 1.], [.9, .05], [GARCH{1, 1}([1., .9, .05])
+   for _ in 1:2])
```

This becomes rather unwieldy if the number of assets is large, but note that as in the univariate case, manually constructing a model is rarely necessary.

The constant conditional correlation (CCC) model of Bollerslev (1990) models $R_t = R$ as constant. It is the special case of the DCC model in which $p = q = 0$. As such, the constructor has the exact same signature, except that the DCC parameters must be passed as a zero-length vector as follows.

```
julia> CCC([1. .5; .5 1.], Float64[], [GARCH{1, 1}([1., .9, .05])
+   for _ in 1:2])
```

Mean specifications

The conditional mean of a `MultivariateARCHModel` is specified by providing vector of univariate `MeanSpecs`.

Multivariate standardized distributions

Multivariate standardized distributions subtype `MultivariateStandardizedDistribution`. Currently, only `MultivariateStdNormal` is available. Note, however, that under mild regularity conditions, the (quasi-)MLE based on the Gaussian assumption estimates the parameters of the dynamic covariance specification consistently even if Gaussianity is violated (Jeantheau 1998).

4. Implementation details

ARCH-type models are typically estimated by maximum likelihood, i.e., by maximizing (2) numerically, conditioned on the τ pre-sample values of r_t. Starting the recursion for σ_t (as in, e.g., (1) for the simple GARCH(1, 1) model) requires specifying the pre-sample values of σ_t^2. **ARCHModels.jl** follows a common convention in setting these to the unconditional sample variance of r_t. Other possibilities would have included backcasting the variance, or substituting the unconditional variance. Different choices made by package authors lead to slight differences in estimation results, see Section 6.

We maximize the likelihood using the **Optim.jl** package, using the BFGS algorithm (Broyden 1970; Fletcher 1970; Goldfarb 1970; Shanno 1970) by default. The user may choose a different algorithm by providing a keyword argument to the `fit` function. The gradients are obtained via automatic differentiation based on **ForwardDiff.jl**, an option exposed by **Optim.jl**.

The robust standard errors provided by **ARCHModels.jl** are constructed according to the general theory of White (1982) and Gourieroux, Monfort, and Trognon (1984), by taking the square root of the diagonal elements of

$$\hat{\Sigma} = \mathcal{J}^{-1}\mathcal{S}\mathcal{J}^{-1}, \tag{8}$$

where, with Θ^* denoting the MLE,

$$\mathcal{S} \equiv \sum_{t=\tau+1}^{T} [\nabla \ell_t(\Theta) \nabla \ell_t(\Theta)^\top]_{\Theta=\Theta^*},$$

and

$$\mathcal{J} \equiv -[\nabla\nabla^\top \ell(\Theta)]_{\Theta=\Theta^*}$$

is the observed Fisher information. The scores $\nabla \ell_t(\Theta)$ and Hessian $\nabla\nabla^\top \ell(\Theta)$ are again obtained by automatic differentiation via **ForwardDiff.jl**. Other implementations may rely on finite differences, or on analytic expressions for the scores and Hessian. For example, Fiorentini, Calzolari, and Panattoni (1996) present the relevant analytic expressions in the

GARCH(1, 1) case. Another alternative is to rely on the robust standard errors of Bollerslev and Wooldridge (1992), which replace the observed information $\mathcal{J}$ with the expected information in (8).

As discussed in Section 3.1, all volatility specifications, such as `GARCH{1, 1}`, are implemented as parametric types, parameterized by the lag orders of the model in question. The same is true for the `ARMA{p, q}` mean specification. Due to the way that Julia works, this implies that specialized methods of, e.g., the likelihood function are compiled for each lag order. This facilitates certain compiler optimizations, such as unrolling some loops over the lag index that appear in the likelihood function. This is one of the reasons for the excellent performance of **ARCHModels.jl** (see Section 6). It does, however, come at the cost of increased compilation time. While this cost is only paid once per new lag order to be estimated, it does become noticeable when estimating many models of the same class but with different lag order parameters, as occurs when using `selectmodel` for model selection. To mitigate this, `selectmodel` estimates so-called subset models. This means that when doing model selection over, say, the class of `TGARCH{o, p, q}` models with o, p, and q ranging from 1 to 3, `selectmodel` will, under the hood, estimate $3^3 = 27$ `TGARCH{3, 3, 3}` models, but with some of the coefficients α_i, β_i, and γ_i in (6) set to zero. This slows down the estimation of each individual model, but more than makes up for it in reduced compile time.

Turning to multivariate models, the DCC model is typically estimated in two steps, by first fitting univariate ARCH models to the individual assets and saving the standardized residuals $\{\hat{\epsilon}_t\}$, and then estimating the DCC parameters from those, by maximizing the correlation component of the likelihood

$$-\frac{1}{2}\sum_{t=\tau}^{T} \log|R_t| + \hat{\epsilon}^\top R_t^{-1}\hat{\epsilon} - \hat{\epsilon}^\top\hat{\epsilon}$$

over $\{\alpha_i\}_{i=1}^q$ and $\{\beta_i\}_{i=1}^p$, treating the volatility parameters of the univariate models as fixed in this second step. Engle (2002) provides the details and expressions for the standard errors. By default, this package employs an alternative estimator due to Engle *et al.* (2019), which is better suited to large-dimensional problems. It achieves this by i) estimating $\bar{Q}$ with a nonlinear shrinkage estimator instead of the sample covariance of $\hat{\epsilon}_t$, and ii) estimating the DCC parameters by maximizing the sum of pairwise log-likelihoods, rather than the joint log-likelihood over all assets, thereby avoiding the inversion of large matrices during the optimization. The estimation method is controlled by passing the `method` keyword to the `fit` method or the type constructors. Possible values are `:largescale` (the default), and `:twostep`.

5. Illustrations

5.1. Case study: Univariate modeling

Fitting, model selection, and diagnostic testing

We will be using the data from Bollerslev and Ghysels (1996), available as the constant `BG96`. The data consist of daily German mark/British pound exchange rates (1974 observations) and are often used in evaluating implementations of (G)ARCH models (see Section 6). We begin by convincing ourselves that the data exhibit ARCH effects. A quick and dirty way of doing this is to look at the sample autocorrelation function of the squared returns using the `autocor` function that **ARCHModels.jl** re-exports from **StatsBase.jl**. Specifically, the following code returns the vector `[.22, .18, .14, .13]`.

```
julia> autocor(BG96 .^ 2, 1:4, demean = true)
```

Using a critical value of $1.96/\sqrt{1974} = 0.044$, we see that there is indeed significant autocorrelation in the squared series.

A more formal test for the presence of volatility clustering is Engle's (1982) ARCH-LM test. The call `ARCHLMTest(BG96, 1)` shows an observed test statistic of 98.12 with a p value of 10^{-22}, so the null is strongly rejected, again providing evidence for the presence of volatility clustering.

Having established the presence of volatility clustering, we begin our analysis by fitting the workhorse model of volatility modeling, a GARCH(1, 1) with standard normal errors. The call `fit(GARCH{1, 1}, BG96)` returns an instance of `UnivariateARCHModel`, as described in Section 3.1. The model is printed as follows.

```
GARCH{1, 1} model with Gaussian errors, T=1974.

Mean equation parameters:
───────────────────────────────────────────────
      Estimate   Std.Error    z value  Pr(>|z|)
───────────────────────────────────────────────
μ  -0.00616637  0.00920163  -0.670139    0.5028
───────────────────────────────────────────────

Volatility parameters:
─────────────────────────────────────────────
     Estimate   Std.Error   z value  Pr(>|z|)
─────────────────────────────────────────────
ω   0.0107606  0.00649493   1.65677    0.0976
β₁  0.805875   0.0725003   11.1155     <1e-27
α₁  0.153411   0.0536586    2.85903    0.0042
─────────────────────────────────────────────
```

The parameters α_1 and β_1 in the volatility equation are highly significant, again confirming the presence of volatility clustering.

The `fit` method supports a number of keyword arguments. The full signature is given below.

```
fit(
    ::Type{<:UnivariateVolatilitySpec},
    data::Vector;
    dist = StdNormal,
    meanspec = Intercept,
    algorithm = BFGS(),
    autodiff = :forward,
    kwargs...
    )
```

Here, `dist` is a subtype (not instance) of `StandardizedDistribution`. The mean specification is specified via `meanspec` and defaults to `Intercept`. It can be passed as either a subtype of `MeanSpec` or an instance thereof (for specifications that require additional data, such as `Regression`). If the mean specification in question has a notion of sample size (like `Regression`), then this sample size should match that of the data, or an error will be thrown. The remaining keyword arguments are passed on to the optimizer.

As an example, an EGARCH(1, 1, 1) model with an intercept and Student's t errors would be fitted using the code below.

```
julia> fit(EGARCH{1, 1, 1}, BG96; meanspec = Intercept, dist = StdT)
```

An alternative approach to fitting a `UnivariateVolatilitySpec` to `BG96` is to first construct a `UnivariateARCHModel` containing the data, and then using `fit!` to modify it in place.

```
julia> am = UnivariateARCHModel(GARCH{1, 1}([1., 0., 0.]), BG96)
julia> fit!(am)
```

Calling `fit(am)` will return a new instance of `UnivariateARCHModel` instead.

Assuming the **GLM.jl** and **DataFrames.jl** packages are installed, it is also possible to pass a `LinearModel` (or `TableRegressionModel`) to `fit` instead of a data vector. This is equivalent to using a `Regression` as a mean specification. In the following example, we fit a linear model with GARCH(1, 1) errors, where the design matrix consists of a breaking intercept and time trend.

```
julia> using GLM, DataFrames
julia> data = DataFrame(B = [ones(1000); zeros(974)], T = 1:1974, Y = BG96)
julia> model = lm(@formula(Y ~ B * T), data)
julia> fit(GARCH{1, 1}, model)
```

One of the issues in ARCH modeling is selecting the lag order. As discussed in Section 2, one possibility is to make this choice based on an information criterion. **ARCHModels.jl** can automate this procedure, via the `selectmodel` function. Given a model class (i.e., a subtype of `UnivariateVolatilitySpec`), it will return a fitted `UnivariateARCHModel`, with the lag length parameters chosen to minimize the desired criterion. The BIC is used by default. As an example, the following selects the optimal (minimum AIC) EGARCH(o, p, q) model, where $o, p, q \leq 2$, assuming t distributed errors.

```
julia> selectmodel(EGARCH, BG96; criterion = aic, maxlags = 2, dist = StdT)
```

For these data, an EGARCH(1, 1, 2) model is selected. Passing the optional keyword argument `show_trace = true` will show the criterion for each model after it is estimated. Any unspecified lag length parameters in the mean specification (such as p and q for ARMA) will be optimized over as well. For example, the code below returns an ARCH(2)-AR(1) model.

```
julia> selectmodel(ARCH, BG96;  meanspec = AR, maxlags = 2)
```

Note, however, that this can result in an explosion of the number of models that must be estimated. For example, selecting the best model from the class of TGARCH(o, p, q)-ARMA(p, q) models results in `maxlags`^{5} models being estimated. It may thus be preferable to fix the lag length of the mean specification by specifying `meanspec = AR{1}` instead of `meanspec = AR`. Similarly, one may restrict the lag length of the volatility specification and select only among different mean specifications.

The conditional volatilities $\hat{\sigma}_t$ and standardized residuals $\hat{z}_t$ are respectively accessible via `volatilities(::UnivariateARCHModel)` and `residuals(::UnivariateARCHModel)`. The non-standardized residuals $\hat{a}_t$ can be obtained by passing `standardized = false` as a keyword argument to `residuals`.

As discussed in Section 2, one possibility to test a volatility specification is to apply the ARCH-LM test to the standardized residuals. This is achieved by calling `ARCHLMTest` on the estimated `UnivariateARCHModel`.

```
julia> am = fit(GARCH{1, 1}, BG96)
julia> ARCHLMTest(am, 4)
```

By default, the number of lags in the test regression is chosen as the maximum order of the volatility specification (e.g., $\max(p, q)$ for a GARCH(p, q) model). Here, we chose 4 instead. The test does not reject, indicating that a GARCH(1, 1) specification is sufficient for modeling the volatility clustering (a common finding).

Value at Risk prediction

Basic in-sample estimates for the Value at Risk implied by an estimated `UnivariateARCHModel` can be obtained using `VaRs`.

```
julia> am = fit(GARCH{1, 1}, BG96)
julia> vars = VaRs(am, 0.05)
```

The 0.05 here specifies that the 5% VaR is sought. The code snippet below then produces the graph in Figure 2.

```
julia> using Plots
julia> plot(-BG96, legend = :none, xlabel = "\$t\$", ylabel = "\$-r_t\$")
julia> plot!(vars, color = :purple)
```

The `predict(am::UnivariateARCHModel)` method can be used to construct one-step ahead forecasts for a number of quantities. Its signature is given below.

```
predict(
        am::UnivariateARCHModel,
        what = :volatility,
        horizon = 1;
        level = 0.01
        )
```

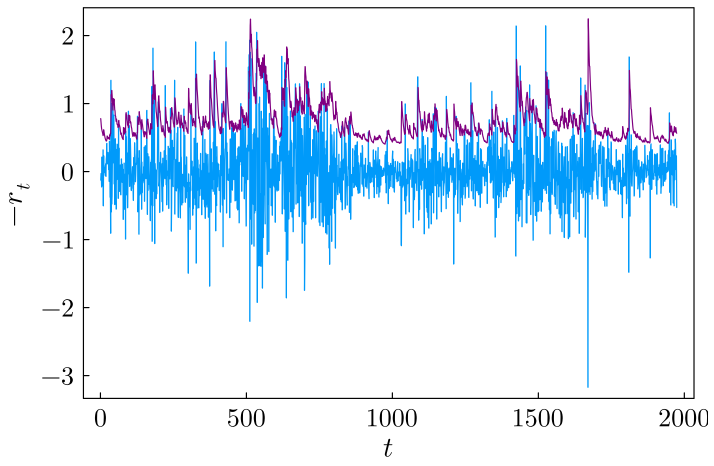

Figure 2: 5% Value at Risk for the Bollerslev and Ghysels (1996) data.

The optional argument `what` controls which object is predicted. The choices are `:volatility` (the default), `:variance`, `:return`, and `:VaR`. The VaR level can be controlled with the keyword argument `level`.

One way to use `predict` is in a backtesting exercise. The following code snippet constructs out-of-sample VaR forecasts $\widehat{\text{VaR}}_t$ for the Bollerslev and Ghysels (1996) data by re-estimating the model in a rolling window fashion.

```
julia> T = length(BG96)
julia> windowsize = 1000
julia> vars = similar(BG96)
julia> for t = windowsize:T-1
           m = fit(GARCH{1, 1}, BG96[t-(windowsize-1):t])
           vars[t+1] = predict(m, :VaR; level = 0.05)
       end
```

Remarkably, estimating these 974 GARCH(1, 1) models and predicting the Value at Risk for each takes but 1.5 seconds in total. Decorating the loop with `Threads.@threads` to enable multithreading reduces this further, to around .15 seconds using 12 threads; see Section 6 for detailed benchmarks. As discussed in Section 2, the correctness of the VaR specification can then be tested using Engle and Manganelli's (2004) dynamic quantile test. The test is available as `DQTest`. Calling the following shows that the test fails to reject its null hypothesis that the specification is correct with a p value of 0.74.

```
julia> DQTest(BG96[windowsize+1:end], vars[windowsize+1:end], 0.05)
```

Simulation

To simulate from a `UnivariateARCHModel`, the `simulate` function is used. It requires specifying the `UnivariateVolatilitySpec` (and optionally the distribution and mean specification), and the desired number of observations T. As an example, simulating a GARCH(1, 1) model with an intercept and errors following a standardized t distribution is achieved as follows.

```
julia> am3 = simulate(GARCH{1, 1}([1., .9, .05]), 1000;
+   warmup = 500, meanspec = Intercept(5.), dist = StdT(3.))
```

Here, the keyword argument `warmup = 500` specifies that 500 pre-sample values should be simulated (and later discarded), 100 being the default. Alternatively, it is possible to pass an existing `UnivariateARCHModel` to `simulate`. In this case, passing T is optional; it defaults to the sample size of the provided model. Hence, `simulate(am3)` returns a new model simulated from `am3` above. Alternatively, `simulate!(am3)` modifies `am3` in place. All simulation functions accept a keyword argument `rng`, to allow the user to specify a random number generator (and thus a seed value) by passing an `AbstractRNG` from the **Random** standard library module.

Care must be taken if the mean specification has a notion of sample size, as in the case of `Regression`: Because the sample size must match that of the data to be simulated, one must pass `warmup = 0`, or an error will be thrown.

5.2. Multivariate modeling

In this section, the percentage returns on 29 stocks from the Dow Jones Industrial Average (DJIA) from 2008-03-19 through 2019-04-11, available as `DOW29`, will be used.

Fitting a multivariate ARCH model proceeds similarly to the univariate case, by passing the type of the multivariate ARCH specification to `fit`. If the lag length (and in the case of the DCC model, the univariate specification) is left unspecified, then these default to 1 (and GARCH). In other words, the invocations `fit(DCC, DOW29)`, `fit(DCC{1, 1}, DOW29)`, and `fit(DCC{1, 1, GARCH{1, 1}}, DOW29)` are all equivalent. All three return an object of type `MultivariateARCHModel`.

As in the univariate case, `fit` supports a number of keyword arguments. The full signature is given below.

```
fit(
    spec,
    data;
    method = :largescale,
    dist = MultivariateStdNormal,
    meanspec = Intercept,
    algorithm = BFGS(),
    autodiff = :forward,
    kwargs...
    )
```

Their meaning is similar to the univariate case. In particular, `meanspec` can be any univariate mean specification. The estimation method can be specified by passing either `:twostep` or `:largescale` for `method`, which respectively refer to the methods of Engle (1982) and Engle *et al.* (2019). As discussed in Section 4, the latter sacrifices some amount of statistical efficiency for much-improved computational speed and is the default. Again paralleling the univariate case, one may also construct a `MultivariateARCHModel` by hand and then call `fit` or `fit!` on it, but this is rather cumbersome, as it requires specifying all the parameters of the covariance specification; in the case of the CCC and DCC models, this includes the parameters of the univariate volatility models.

One-step ahead forecasts of the covariance or correlation matrix are obtained by respectively passing `what = :covariance` (the default) or `what = :correlation` to `predict`.

In the multivariate case, there are three types of residuals that can be considered: The unstandardized residuals, $\hat{a}_t$; the devolatized residuals, $\hat{\epsilon}_t$, where $\hat{\epsilon}_{it} \equiv \hat{a}_{it}/\hat{\sigma}_{it}$; and the decorrelated residuals $\hat{z}_t \equiv \widehat{\Sigma}_t^{-1/2}\hat{a}_t$. When called on a `MultivariateARCHModel`, `residuals` returns $\{\hat{z}_t\}$ by default. Passing `decorrelated = false` returns $\{\hat{\epsilon}_t\}$, and passing `standardized = false` returns $\{\hat{a}_t\}$ (note that `decorrelated = true` implies `standardized = true`).

6. Comparison with other packages

6.1. Univariate: GARCH(1, 1)

In this section, we compare the results produced by **ARCHModels.jl** to those obtained with the **rugarch** package for R, the **arch** package for Python, and with MATLAB's **Econometrics** toolbox. As mentioned, the de-facto standard when comparing implementations of GARCH models is to use the daily German mark/British pound exchange rates from Bollerslev and Ghysels (1996). This goes back to Brooks, Burke, and Persand (2001), whose results we will use as a benchmark.

Brooks *et al.* (2001) fit a GARCH(1, 1) model with an intercept to the data and give the following estimates (rounded to three significant digits, t statistics in parentheses): $\mu = -0.00619(-0.67)$, $\omega = 0.0108(1.66)$, $\alpha_1 = 0.153(2.86)$, and $\beta_1 = 0.806(11.11)$. Note that these values should not necessarily be considered the "correct" estimates, because as discussed in Section 4, the results will generally depend on the implementation (e.g., treatment of presample values, choice of optimization algorithm and starting values).

The estimates obtained by the various packages are given in Table 1. It is seen that the differences in parameter estimates between packages are generally small. MATLAB matches those of Brooks *et al.* (2001) to the given precision, and **ARCHModels.jl** and **rugarch** are generally close. The estimates produced by Python's **arch** package are noticeably different when using the default settings. The reason is that **arch** treats presample values differently by default. However, its `fit` method accepts an optional argument `backcast`. When this is set to the sample variance of the data, then the estimates match those of **ARCHModels.jl** to the given precision.

ARCHModels.jl, R's **rugarch**, and Python's **arch** all produce robust standard errors, and the results do not differ much. **ARCHModels.jl** (and **arch** after passing the same optional argument as before) track Brooks *et al.* (2001) most closely. MATLAB is the exception in that it does not produce robust standard errors, instead relying on the "outer product of gradients" estimate of the covariance matrix, i.e., instead of $\hat{\Sigma} = \mathcal{J}^{-1}\mathcal{S}\mathcal{J}^{-1}$ as in (8), it uses $\hat{\Sigma} = \mathcal{S}^{-1}$.

The most striking observation in Table 1 is the difference in runtime between packages: **ARCHModels.jl** takes 2.69 ms to fit the GARCH(1, 1) model to these 1,974 observations, with Python's **arch** package taking 5.43 times as long, MATLAB taking 9.94 times as long, and R's **rugarch** package taking 32.97 times as long (though to be fair, the output of the latter

	Brooks *et al.*	**ARCHModels.jl**	**rugarch**	**arch**	**arch**[†]	MATLAB
			Coefficients			
$\mu \cdot 10^2$	-0.619	-0.617	-0.618	-0.608	-0.617	-0.619
$\omega \cdot 10^1$	0.108	0.108	0.108	0.099	0.108	0.108
α	0.153	0.153	0.153	0.145	0.153	0.153
β	0.806	0.806	0.806	0.817	0.806	0.806
			t statistics			
μ	-0.67	-0.67	-0.69	-0.66	-0.67	-0.73
ω	1.66	1.66	1.66	1.56	1.66	8.13
α	2.86	2.86	3.11	2.61	2.86	10.96
β	11.11	11.12	11.65	10.97	11.12	48.67
			Runtime			
Absolute (ms)		2.687	88.584	14.600		26.701
Relative		1.0	32.97	5.43		9.94

[†] `arch_model(data).fit(backcast = np.var(data))`

Table 1: Estimation results and runtime for fitting a GARCH(1, 1) to the Bollerslev and Ghysels (1996) data with various packages.

contains a considerable amount of additional information, such as non-robust standard errors and a number of tests). While the differences in *absolute* runtimes may seem small enough to not make these speedups noticeable in practice, it should be noted that there are situations in which a model needs to be estimated thousands (e.g., univariate backtesting) or even millions (e.g., multivariate backtesting of large-dimensional CCC/DCC models) of times.

As all four packages implement the "hot loop", i.e., the log-likelihood function, in a compiled language (Julia in the case of Julia, Cython in the case of **arch**, C/C++ for the others), Julia's advantage likely results from a combination of the use of automatic differentiation for gradients, method specialization on type parameters, and the fact that fewer context switches are needed because both the optimizer and the likelihood function are implemented in Julia.

6.2. Multivariate: DCC(1, 1)

In this section, we compare our implementation of the multivariate DCC model with that of the R package **rmgarch** (Galanos 2022). We exclude MATLAB and Python from this comparison, as MATLAB has no built-in support for multivariate models, and there does not appear to exist a widely used package for Python.

Unlike in the univariate case, there is no generally accepted benchmark dataset. As such, we fit a DCC(1, 1) model to the percentage returns on 29 stocks from the DJIA from 2008-03-19 through 2019-04-11. As discussed, these are available as `DOW29` in **ARCHModels.jl**. A simple GARCH(1, 1) model with an intercept in the mean equation is used for the univariate specifications. We exclude computing the standard errors and t statistics from the runtime; both packages have options for doing this, because the operation is costly. Table 2 shows the results.

	ARCHModels.jl[†]	**ARCHModels.jl**[‡]	**rmgarch**
	Coefficients		
α	0.057	0.008	0.006
β	0.888	0.956	0.965
	t statistics		
α	1.98	3.98	5.83
β	12.98	58.24	105.47
	Runtime (w/o standard errors)		
Absolute (s)	$21.5 \cdot 10^{-3}$	3.91	9.35
Relative	0.006	1.0	2.4

[†] `fit(DCC, DOW29)`
[‡] `fit(DCC, DOW29; method = :twostep)`

Table 2: Estimation results and runtime for fitting a DCC(1, 1) to the DJIA data with **ARCHModels.jl** and **rmgarch**.

As described in Section 4, **ARCHModels.jl** implements two different algorithms for fitting DCC models. The `:twostep` algorithm corresponds to the one used in **rmgarch**, and the estimation results are similar. The difference in runtime between the packages is much smaller than in the univariate case. This is because the runtime is dominated by repeatedly inverting the 29-dimensional correlation matrix within the likelihood (estimating the 29 univariate models is parallelized in both packages). Using the default `:largescale` algorithm in **ARCHModels.jl** is around 180 times faster, because it avoids having to invert large matrices.

7. Summary and discussion

We have introduced **ARCHModels.jl**, a package for estimating, simulating, and testing ARCH-type models in the Julia programming language. The package is entirely written in Julia and makes ample use of its features, including, but not limited to, parametric types and automatic differentiation, while its modular design makes it easy to extend with additional models or error distributions. **ARCHModels.jl** is able to reproduce benchmark estimation results available in the literature, and outperforms some alternative implementations in terms of computational speed.

Computational details

The package versions used throughout the paper are **ARCHModels.jl** 2.3.3 on Julia 1.8.5, **rugarch** 1.4-9 and **rmgarch** 1.3-9, both utilizing compiled Cython binaries, on R 4.2.2, **arch 5.3.1** on Python 3.10.9, and the **Econometrics** toolbox of MATLAB R2021b. All timings, including those in Tables 1 and 2, have been obtained with Ubuntu 20.04 running on an AMD Ryzen 9 3900X with 32 GB of RAM, with Julia, R, and Python running inside Docker (Merkel 2014) containers.

Acknowledgments

Simon Broda has received funding from the European Union's Horizon 2020 research and innovation program under Marie Skłodowska-Curie grant agreement No 750559. The authors would like to thank two anonymous referees for their insightful comments, which have helped improve the paper.

References

Akaike H (1974). "A New Look at the Statistical Model Identification." *IEEE Transactions on Automatic Control*, **19**, 716–723. doi:10.1109/tac.1974.1100705.

Bates D, Noack A, Kornblith S, Bouchet-Valat M, Borregaard MK, Arslan A, White JM, Kleinschmidt D, Alday P, Lynch G, Dunning I, Mogensen PK, Lendle S, Aluthge D, Deffebach P, Calderón JBS, Patnaik A, Born B, Setzler B, Kamiński B (2023). ***GLM.jl**: Generalized Linear Models in Julia*. doi:10.5281/zenodo.7734970. Julia package version 1.8.2, URL https://github.com/JuliaStats/GLM.jl.

Behnel S, Bradshaw R, Citro C, Dalcin L, Seljebotn DS, Smith K (2011). "Cython: The Best of Both Worlds." *Computing in Science & Engineering*, **13**(2), 31–39. doi:10.1109/mcse.2010.118.

Besançon M, Papamarkou T, Anthoff D, Arslan A, Byrne S, Lin D, Pearson J (2021). "**Distributions.jl**: Definition and Modeling of Probability Distributions in the JuliaStats Ecosystem." *Journal of Statistical Software*, **98**(16), 1–30. doi:10.18637/jss.v098.i16.

Bezanson J, Edelman A, Karpinski S, Shah VB (2017). "Julia: A Fresh Approach to Numerical Computing." *SIAM Review*, **59**(1), 65–98. doi:10.1137/141000671.

Bodin G, Saavedra R, Fernandes C, Street A (2020). "**ScoreDrivenModels.jl**: A Julia Package for Generalized Autoregressive Score Models." *Technical Report 2008.05506*, arXiv.org E-Print Archive. doi:10.48550/arXiv.2008.05506.

Bodin G, Saavedra R, Fernandes C, Street A (2022). ***ScoreDrivenModels.jl**: A Julia Package for Generalized Autoregressive Score Models*. Julia package version 0.2.1, URL https://github.com/LAMPSPUC/ScoreDrivenModels.jl.

Bollerslev T (1986). "Generalized Autoregressive Conditional Heteroskedasticity." *Journal of Econometrics*, **31**, 307–327. doi:10.1016/0304-4076(86)90063-1.

Bollerslev T (1990). "Modelling the Coherence in Short-Run Nominal Exchange Rates: A Multivariate Generalized ARCH Model." *The Review of Economics and Statistics*, **72**(3), 498–505. doi:10.2307/2109358.

Bollerslev T, Engle RF, Wooldridge JM (1988). "A Capital Asset Pricing Model with Time-Varying Covariances." *Journal of Political Economy*, **96**(1), 116–131. doi:10.1086/261527.

Bollerslev T, Ghysels E (1996). "Periodic Autoregressive Conditional Heteroscedasticity." *Journal of Business & Economic Statistics*, **14**, 139–151. doi:10.1080/07350015.1996.10524640.

Bollerslev T, Wooldridge JM (1992). "Quasi-Maximum Likelihood Estimation and Inference in Dynamic Models with Time-Varying Covariances." *Econometric Reviews*, **11**, 143–172. doi:10.1080/07474939208800229.

Bouchet-Valat M, Kamiński B (2023). "**DataFrames.jl**: Flexible and Fast Tabular Data in Julia." *Journal of Statistical Software*, **107**(4), 1–32. doi:10.18637/jss.v107.i04.

Brooks C, Burke S, Persand G (2001). "Benchmarks and the Accuracy of GARCH Model Estimation." *International Journal of Forecasting*, **17**, 45–56. doi:10.1016/s0169-2070(00)00070-4.

Broyden CG (1970). "The Convergence of a Class of Double-Rank Minimization Algorithms 1. General Considerations." *IMA Journal of Applied Mathematics*, **6**, 76–90. doi:10.1093/imamat/6.1.76.

Creal D, Koopman SJ, Lucas A (2013). "Generalized Autoregressive Score Models with Applications." *Journal of Applied Econometrics*, **28**(5), 777–795. doi:10.1002/jae.1279.

Engle RF (1982). "Autoregressive Conditional Heteroscedasticity with Estimates of the Variance of United Kingdom Inflation." *Econometrica*, **50**(4), 987–1007. doi:10.2307/1912773.

Engle RF (2002). "Dynamic Conditional Correlation." *Journal of Business & Economic Statistics*, **20**(3), 339–350. doi:10.1198/073500102288618487.

Engle RF, Ledoit O, Wolf M (2019). "Large Dynamic Covariance Matrices." *Journal of Business & Economic Statistics*, **37**(2), 363–375. doi:10.1080/07350015.2017.1345683.

Engle RF, Manganelli S (2004). "CAViaR." *Journal of Business & Economic Statistics*, **22**(4), 367–381. doi:10.1198/073500104000000370.

Fama E (1970). "Efficient Capital Markets: A Review of Theory and Empirical Work." *Journal of Finance*, **25**, 383–417. doi:10.2307/2325486.

Fiorentini G, Calzolari G, Panattoni L (1996). "Analytic Derivatives and the Computation of GARCH Estimates." *Journal of Applied Econometrics*, **11**, 399–417. doi:10.1002/(sici)1099-1255(199607)11:4<399::aid-jae401>3.0.co;2-r.

Fletcher R (1970). "A New Approach to Variable Metric Algorithms." *The Computer Journal*, **13**, 317–322. doi:10.1093/comjnl/13.3.317.

Galanos A (2022). ***rmgarch**: Multivariate GARCH Models.* R package version 1.3-9., URL https://CRAN.R-project.org/package=rmgarch.

Ghalanos A (2022). ***rugarch**: Univariate GARCH Models.* R package version 1.4-9., URL https://CRAN.R-project.org/package=rugarch.

Glosten LR, Jagannathan R, Runkle DE (1993). "On the Relation Between the Expected Value and the Volatility of the Nominal Excess Return on Stocks." *The Journal of Finance*, **48**, 1779–1801. doi:10.1111/j.1540-6261.1993.tb05128.x.

Goldfarb D (1970). "A Family of Variable-Metric Methods Derived by Variational Means." *Mathematics of Computation*, **24**, 23–26. doi:10.1090/s0025-5718-1970-0258249-6.

Gourieroux C, Monfort A, Trognon A (1984). "Pseudo Maximum Likelihood Methods: Theory." *Econometrica*, **52**, 681–700. doi:10.2307/1913471.

Hansen BE (1994). "Autoregressive Conditional Density Estimation." *International Economic Review*, **35**, 705–730. doi:10.2307/2527081.

Harris CR, Millman KJ, Van der Walt SJ, Gommers R, Virtanen P, Cournapeau D, Wieser E, Taylor J, Berg S, Smith NJ, Kern R, Picus M, Hoyer S, Van Kerkwijk MH, Brett M, Haldane A, del Río JF, Wiebe M, Peterson P, Gérard-Marchant P, Sheppard K, Reddy T, Weckesser W, Abbasi H, Gohlke C, Oliphant TE (2020). "Array Programming with **NumPy**." *Nature*, **585**(7825), 357–362. doi:10.1038/s41586-020-2649-2.

Harvey AC (2013). *Dynamic Models for Volatility and Heavy Tails: with Applications to Financial and Economic Time Series*. Econometric Society Monographs. Cambridge University Press, Cambridge. doi:10.1017/cbo9781139540933.

Jeantheau T (1998). "Strong Consistency of Estimators for Multivariate Arch Models." *Econometric Theory*, **14**, 70–86. doi:10.1017/S0266466698141038.

Kamiński B, White JM, Bouchet-Valat M, *et al.* (2023). ***DataFrames.jl**: In-Memory Tabular Data in Julia*. doi:10.5281/zenodo.8129187. Julia package version 1.6.0, URL https://github.com/JuliaData/DataFrames.jl.

Kornblith S, *et al.* (2023). ***HypothesisTests.jl**: Hypothesis Tests for Julia*. Julia package version 0.11.0, URL https://github.com/JuliaStats/HypothesisTests.jl.

Lin D, Byrne S, Noack A, Bates D, White JM, Kornblith S, *et al.* (2023a). ***StatsBase.jl**: Basic Statistics for Julia*. Julia package version 0.34.0, URL https://github.com/JuliaStats/StatsBase.jl.

Lin D, White JM, Byrne S, Bates D, Noack A, Pearson J, Arslan A, Squire K, Anthoff D, Papamarkou T, Besançon M, Drugowitsch J, Schauer M, *et al.* (2023b). ***Distributions.jl**: A Julia Package for Probability Distributions and Associated Functions*. doi:10.5281/zenodo.8102384. Julia package version 0.25.98, URL https://github.com/JuliaStats/Distributions.jl.

Merkel D (2014). "Docker: Lightweight Linux Containers for Consistent Development and Deployment." *Linux Journal*, **2014**(239), 2. doi:10.1007/978-1-4842-5826-2_3.

Mogensen PK, Riseth AN (2018). "**Optim**: A Mathematical Optimization Package for Julia." *Journal of Open Source Software*, **3**(24), 615. doi:10.21105/joss.00615.

Mogensen PK, Riseth AN, *et al.* (2023). ***Optim.jl**: Optimization Functions for Julia*. Julia package version 1.7.6, URL https://github.com/JuliaNLSolvers/Optim.jl.

Nelson DB (1991). "Conditional Heteroskedasticity in Asset Returns: A New Approach." *Econometrica*, **59**, 347–370. doi:10.2307/2938260.

R Core Team (2023). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna. URL https://www.R-project.org/.

Revels J, Lubin M, Papamarkou T (2016). "Forward-Mode Automatic Differentiation in Julia." *Technical Report 1607.07892*, arXiv.org E-Print Archive. doi:https://doi.org/10.48550/arXiv.1607.07892.

Revels J, Lubin M, Papamarkou T, *et al.* (2023). ***ForwardDiff.jl**: Forward Mode Automatic Differentiation for Julia.* Julia package version 0.10.35, URL https://github.com/JuliaDiff/ForwardDiff.jl.

Schwarz G (1978). "Estimating the Dimension of a Model." *The Annals of Statistics*, **6**, 461–464. doi:10.1214/aos/1176344136.

Shanno DF (1970). "Conditioning of Quasi-Newton Methods for Function Minimization." *Mathematics of Computation*, **24**, 647–656. doi:10.1090/s0025-5718-1970-0274029-x.

Sheppard K, *et al.* (2022). ***arch**: ARCH for Python.* doi:10.5281/zenodo.6684078. Python package version 5.3.1, URL https://pypi.org/project/arch/.

The MathWorks Inc (2021). *MATLAB – The Language of Technical Computing, Version R2021a*. Natick. URL https://www.mathworks.com/products/matlab/.

Van Rossum G (1995). "Python Tutorial." *Technical Report CS-R9526*, Centrum voor Wiskunde en Informatica (CWI), Amsterdam. URL https://ir.cwi.nl/pub/5007.

White H (1982). "Maximum Likelihood Estimation of Misspecified Models." *Econometrica*, **50**, 1–25. doi:10.2307/1912526.

Zakoian JM (1994). "Threshold Heteroskedastic Models." *Journal of Economic Dynamics and Control*, **18**(5), 931–955. doi:10.1016/0165-1889(94)90039-6.

Affiliation:

Simon A. Broda
Institute of Financial Services IFZ
Lucerne University of Applied Sciences and Arts
Campus Zug-Rotkreuz
Suurstoffi 1
6343 Rotkreuz, Switzerland
E-mail: simon.broda@hslu.ch
URL: https://www.hslu.ch/de-ch/hochschule-luzern/ueber-uns/personensuche/profile/?pid=4728

Marc S. Paolella
Department of Banking and Finance
University of Zurich
PLM-H 320
Plattenstr. 14
8032 Zürich, Switzerland
E-mail: marc.paolella@bf.uzh.ch
URL: https://www.bf.uzh.ch/de/persons/paolella-marc

Journal of Statistical Software https://www.jstatsoft.org/
published by the Foundation for Open Access Statistics https://www.foastat.org/
September 2023, Volume 107, Issue 5 *Submitted:* 2022-04-08
doi:10.18637/jss.v107.i05 *Accepted:* 2023-03-01