Journal of Statistical Software

MMMMMM YYYY, Volume VV, Issue II. *http://www.jstatsoft.org/*

Parameter Estimation of ARMA Models with GARCH/APARCH Errors An R and SPlus Software Implementation

Diethelm Würtz[1], **Yohan Chalabi**[2], **Ladislav Luksan**[3]

[1,2]Institute for Theoretical Physics
Swiss Federal Institute of Technology, Zurich
[3]Academy of Sciences of the Czech Republic
Institute of Computer Science, Praha

Abstract

We report on concepts and methods to implement the family of ARMA models with GARCH/APARCH errors introduced by Ding, Granger and Engle. The software implementation is written in S and optimization of the constrained log-likelihood function is achieved with the help of a SQP solver. The implementation is tested with Bollerslev's GARCH(1,1) model applied to the DEMGBP foreign exchange rate data set given by Bollerslev and Ghysels. The results are compared with the benchmark implementation of Fiorentini, Calzolari and Panattoni. In addition the MA(1)-APARCH(1,1) model for the SP500 stock market index analyzed by Ding, Granger and Engle is reestimated and compared with results obtained from the Ox/G@RCH and SPlus/Finmetrics software packages. The software is part of the Rmetrics open source project for computational finance and financial engineering. Implementations are available for both software environments, R and SPlus.

Keywords: Time Series Analysis, Heteroskedasticity, ARCH, GARCH, APARCH, constrained maximum log-likelihood, SQP solver, R, Rmetrics, SPlus, Finmetrics, Ox, G@RCH.

1. Introduction

GARCH, *Generalized Autoregressive Conditional Heteroskedastic*, models have become important in the analysis of time series data, particularly in financial applications when the goal is to analyze and forecast volatility. For this purpose, we describe functions for simulating, estimating and forecasting various univariate GARCH-type time series models in the conditional variance and an ARMA specification in the conditional mean. We present a numerical

implementation of the maximum log-likelihood approach under different assumptions, Normal, Student-t, GED errors or their skewed versions. The parameter estimates are checked by several diagnostic analysis tools including graphical features and hypothesis tests. Functions to compute n-step ahead forecasts of both the conditional mean and variance are also available.

The number of GARCH models is immense, but the most influential models were the first. Beside the standard ARCH model introduced by Engle [1982] and the GARCH model introduced by Bollerslev [1986], we consider also the more general class of asymmetric power ARCH models, named APARCH, introduced by Ding, Granger and Engle [1993]. APARCH models include as special cases the TS-GARCH model of Taylor [1986] and Schwert [1989], the GJR-GARCH model of Glosten, Jaganathan, and Runkle [1993], the T-ARCH model of Zakoian [1993], the N-ARCH model of Higgins and Bera [1992], and the Log-ARCH model of Geweke [1986] and Pentula [1986].

Coupled with these models was a sophisticated analysis of the stochastic process of data generated by the underlying process as well as estimators for the unknown model parameters. Theorems for the autocorrelations, moments and stationarity and ergodicity of GARCH processes have been developed for many of the important cases. There exist a collection of review articles by Bollerslev, Chou and Kroner [1992], Bera and Higgins [1993], Bollerslev, Engle and Nelson [1994], Engle [2001], Engle and Patton [2001], and Li, Ling and McAleer [2002] that give a good overview of the scope of the research.

2. Mean and Variance Equation

We describe the mean equation of an univariate time series x_t by the process

$$x_t \;=\; \mathrm{E}(x_t|\Omega_{t-1}) + \varepsilon_t \;, \tag{1}$$

where $\mathrm{E}(\cdot|\cdot)$ denotes the conditional expectation operator, Ω_{t-1} the information set at time $t-1$, and ε_t the innovations or residuals of the time series. ε_t describes uncorrelated disturbances with zero mean and plays the role of the unpredictable part of the time series. In the following we model the mean equation as an ARMA process, and the innovations are generated from a GARCH or APARCH process.

ARMA Mean Equation: The ARMA(m,n) process of autoregressive order m and moving average order n can be described as

$$\begin{aligned} x_t &= \mu + \sum_{i=1}^{m} a_i x_{t-i} + \sum_{j=1}^{n} b_j \varepsilon_{t-j} + \varepsilon_t \;, \\ &= \mu + a(\mathcal{B}) x_t + b(\mathcal{B}) \varepsilon_t \;, \end{aligned} \tag{2}$$

with mean μ, autoregressive coefficients a_i and moving average coefficients b_i. Note, that the model can be expressed in a quite comprehensive form using the backshift operator $\mathcal{B}$ defined by $\mathcal{B}x_t = x_{t-1}$. The functions $a(\mathcal{B})$ and $b(\mathcal{B})$ are polynomials of degree m and n respectively in the backward shift operator $\mathcal{B}$. If $n = 0$ we have a pure autoregressive process and if on the other hand $m = 0$ we have a pure moving average process. The ARMA time

series is *stationary* when the series $a(\mathcal{B})$, which is the generating function of the coefficients a_i, converges for $|\mathcal{B}| < 1$ that is, on or within the unit circle.

GARCH Variance Equation: The mean equation cannot take into account for heteroskedastic effects of the time series process typically observed in form of fat tails, as clustering of volatilities, and the leverage effect. In this context Engle [1982] introduced the *Autoregressive Conditional Heteroskedastic* model, named ARCH, later generalized by Bollerslev [1986], named GARCH. The ε_t terms in the ARMA mean equation (2) are the innovations of the time series process. Engle [1982] defined them as an autoregressive conditional heteroscedastic process where all ε_t are of the form

$$\varepsilon_t \;=\; z_t \; \sigma_t \;, \tag{3}$$

where z_t is an *iid* process with zero mean and unit variance. Although ε_t is serially uncorrelated by definition its conditional variance equals σ_t^2 and, therefore, may change over time. All the GARCH models we consider in the following differ only in their functional form for the conditional variance.

The variance equation of the GARCH(p,q) model can be expressed as

$$\begin{aligned}
\varepsilon_t &= z_t\sigma_t \;, \\
z_t &\sim \mathcal{D}_\vartheta(0,1) \;, \\
\sigma_t^2 &= \omega \;+\; \sum_{i=1}^{p} \alpha_i \varepsilon_{t-i}^2 \;+\; \sum_{j=1}^{q} \beta_j \sigma_{t-j}^2 \;, \\
&= \omega \;+\; \alpha(\mathcal{B})\varepsilon_{t-1}^2 \;+\; \beta(\mathcal{B})\sigma_{t-1}^2 \;,
\end{aligned} \tag{4}$$

where $\mathcal{D}_\vartheta(0,1)$ is the probability density function of the innovations or residuals with zero mean and unit variance. Optionally, ϑ are additional distributional parameters to describe the skew and the shape of the distribution. If all the coefficients β are zero the GARCH model is reduced to the ARCH model. Like for ARMA models a GARCH specification often leads to a more parsimonious representation of the temporal dependencies and thus provides a similar added flexibility over the linear ARCH model when parameterizing the conditional variance. Bolerslev [1986] has shown that the GARCH(p,q) process is wide-sense stationary with $\mathrm{E}(\varepsilon_t) = 0$, $\mathrm{var}(\varepsilon_t) = \omega/(1 - \alpha(1) - \beta(1))$ and $\mathrm{cov}(\varepsilon_t, \varepsilon_s) = 0$ for $t \neq s$ if and only if $\alpha(1) + \beta(1) < 1$.

The variance equation of the APARCH(p,q) model can be written as

$$\begin{aligned}
\varepsilon_t &= z_t\sigma_t \;, \\
z_t &\sim \mathcal{D}_\vartheta(0,1) \;, \\
\sigma_t^\delta &= \omega \;+\; \sum_{i=1}^{p} \alpha_i (|\varepsilon_{t-i}| - \gamma_i \varepsilon_{t-i})^\delta \;+\; \sum_{j=1}^{q} \beta_j \sigma_{t-j}^\delta \;,
\end{aligned} \tag{5}$$

where $\delta > 0$ and $-1 < \gamma_i < 1$. This model has been introduced by Ding, Granger, and Engle [1993]. It adds the flexibility of a varying exponent with an asymmetry coefficient to take the

leverage effect into account. A stationary solution exists if $\omega > 0$, and $\Sigma_i \alpha_i \kappa_i + \Sigma_j \beta_j < 1$, where $\kappa_i = \mathrm{E}(|z| + \gamma_i z)^\delta$. Note, that if $\gamma \neq 0$ and/or $\delta \neq 2$ the κ_i depend on the assumptions made on the innovation process. The family of APARCH models includes the ARCH and GARCH models, and five other ARCH extensions as special cases:

- *ARCH Model of Engle* when $\delta = 2$, $\gamma_i = 0$, and $\beta_j = 0$.
- *GARCH Model of Bollerslev* when $\delta = 2$, and $\gamma_i = 0$.
- *TS-GARCH Model of Taylor and Schwert* when $\delta = 1$, and $\gamma_i = 0$.
- *GJR-GARCH Model of Glosten, Jagannathan, and Runkle* when $\delta = 2$.
- *T-ARCH Model of Zakoian* when $\delta = 1$.
- *N-ARCH Model of Higgens and Bera* when $\gamma_i = 0$, and $\beta_j = 0$.
- *Log-ARCH Model of Geweke and Pentula* when $\delta \to 0$.

3. The Standard GARCH(1,1) Model

Code Snippet 1 shows how to write a basic S function named `garch11Fit()` to estimate the parameters for Bollerslev's GARCH(1,1) model. As the benchmark data set we use the daily DEMGBP foreign exchange rates as supplied by Bollerslev and Ghysels [1996] and the results obtained by Fiorentini, Calzolar and Panattoni [1996] based on the optimization of the log-likelihood function using analytic expressions for the gradient and Hessian matrix. This setting is well accepted as the benchmark for GARCH(1,1) models. The function estimates the parameters $\mu, \omega, \alpha, \beta$ in a sequence of several major steps: *(1)* the initialization of the time series, *(2)* the initialization of the model parameters, *(3)* the settings for the conditional distribution function, *(4)* the composition of the log-likelihood function, *(5)* the estimation of the model parameters, and *6)* the summary of the optimization results.

Given the model for the conditional mean and variance and an observed univariate return series, we can use the maximum log-likelihood estimation approach to fit the parameters for the specified model of the return series. The procedure infers the process innovations or residuals by inverse filtering. Note, that this filtering transforms the observed process ε_t into an uncorrelated white noise process z_t. The log-likelihood function then uses the inferred innovations z_t to infer the corresponding conditional variances ${\sigma_t}^2$ via recursive substitution into the model-dependent conditional variance equations. Finally, the procedure uses the inferred innovations and conditional variances to evaluate the appropriate log-likelihood objective function. The MLE concept interprets the density as a function of the parameter set, conditional on a set of sample outcomes. The Normal distribution is the standard distribution when estimating and forecasting GARCH models. Using $\varepsilon_t = z_t \sigma_t$ the log-likelihood function of the Normal distribution is given by

$$\begin{aligned} \mathcal{L}_N(\theta) &= \ln \prod_t \frac{1}{\sqrt{(2\pi\sigma_t^2)}} e^{-\frac{\varepsilon_t^2}{2\sigma_t^2}} = \ln \prod_t \frac{1}{\sqrt{(2\pi\sigma_t^2)}} e^{-\frac{z_t^2}{2}} \\ &= -\frac{1}{2} \sum_t [\log(2\pi) + \log(\sigma_t^2) + z_t^2] \ , \end{aligned} \tag{6}$$

or in general

$$\mathcal{L}_N(\theta) = \ln \prod_t \mathcal{D}_\vartheta(x_t, E(x_t|\Omega_{t-1}), \sigma_t) \ , \tag{7}$$

where $\mathcal{D}_\vartheta$ is the conditional distribution function. The second argument of $\mathcal{D}_\vartheta$ denotes the mean, and the third argument the standard deviation. The full set of parameters θ includes the parameters from the mean equation $(\mu, a_{1:m}, b_{1:n})$, from the variance equation $(\omega, \alpha_{1:p}, \gamma_{1:p}, \beta_{1:q}, \delta)$, and the distributional parameters (ϑ) in the case of a non-normal distribution function. For Bollerslev's GARCH(1,1) model the parameter set reduces to $\theta = \{\mu, \omega, \alpha_1, \beta_1\}$. In the following we will suppress the index on the parameters α and β if we consider the GARCH(1,1) model.

The parameters θ which fit the model best, are obtained by minimizing the "negative" log-likelihood function. Some of the values of the parameter set θ are constrained to a finite or semi-finite range. Note, that $\omega > 0$ has to be constrained on positive values, and that α and β have to be constrained in a finite interval $[0, 1)$. This requires a solver for constrained numerical optimization problems. R offers the solvers `nlminb()` and `optim(method="L-BFGS-B")` for constrained optimization, the first is also part of SPlus. These are R interfaces to underlying Fortran routines from the PORT Mathematical Subroutine Library, Lucent Technologies [1997], and to TOMS Algorithm 778, ACM [1997], respectively. In addition we have implemented a sequential quadratic programming algorithm `"sqp"`, Lucsan [1976], which is more efficient compared to the other two solvers. And additionally, everything is implemented in Fortran, the solver, the objective function, gradient vector, and Hessian matrix.

The optimizer require a vector of initial parameters for the mean μ, as well as for the GARCH coefficients ω, α, and β. The initial value for the mean is estimated from the mean μ of the time series observations x. For the GARCH(1,1) model we initialize $\alpha = 0.1$ and $\beta = 0.8$ by typical values of financial time series, and ω by the variance of the series adjusted by the persistence $\omega = \mathrm{Var}(x) * (1 - \alpha - \beta)$. Further arguments to the GARCH fitting function are the upper and lower box bounds, and optional control parameters.

3.1. How to fit Bollerslev's GARCH(1,1) Model

In what follows we explicitly demonstrate how the parameter estimation for the GARCH(1,1) model with normal innovations can be implemented in S. The argument list of the fitting function `garch11Fit(x)` requires only the time series, the rest will be done automatically step by step:

- *Step 1 - Series Initialization:* We save the time series `x` globally to have the values available in later function calls without parsing the series through the argument list.
- *Step 2 - Parameter Initialization:* In the second step we initialize the set of model parameters $\{\theta\}$, `params`, and the corresponding upper and lower bounds. In the example we use bounds `lowerBounds` and `upperBounds` which are wide enough to serve almost every economic and financial GARCH(1,1) model, and define model parameters which take typical values.
- *Step 3 - Conditional Distribution:* For the conditional distribution we use the Normal distribution `dnorm()`.

```
garch11Fit = function(x)
{
    # Step 1: Initialize Time Series Globally:
    x <<- x

    # Step 2: Initialize Model Parameters and Bounds:
    Mean = mean(x); Var = var(x); S = 1e-6
    params = c(mu = Mean, omega = 0.1*Var, alpha = 0.1, beta = 0.8)
    lowerBounds = c(mu = -10*abs(Mean), omega = S^2, alpha = S, beta = S)
    upperBounds = c(mu = 10*abs(Mean), omega = 100*Var, alpha = 1-S, beta = 1-S)

    # Step 3: Set Conditional Distribution Function:
    garchDist = function(z, hh) { dnorm(x = z/hh)/hh }

    # Step 4: Compose log-Likelihood Function:
    garchLLH = function(parm) {
        mu = parm[1]; omega = parm[2]; alpha = parm[3]; beta = parm[4]
        z = (x-mu); Mean = mean(z^2)
        # Use Filter Representation:
        e = omega + alpha * c(Mean, z[-length(x)]^2)
        h = filter(e, beta, "r", init = Mean)
        hh = sqrt(abs(h))
        llh = -sum(log(garchDist(z, hh)))
        llh }
    print(garchLLH(params))

    # Step 5: Estimate Parameters and Compute Numerically Hessian:
    fit = nlminb(start = params, objective = garchLLH,
        lower = lowerBounds, upper = upperBounds, control = list(trace=3))
    epsilon = 0.0001 * fit$par
    Hessian = matrix(0, ncol = 4, nrow = 4)
    for (i in 1:4) {
        for (j in 1:4) {
        x1 = x2 = x3 = x4 = fit$par
        x1[i] = x1[i] + epsilon[i]; x1[j] = x1[j] + epsilon[j]
        x2[i] = x2[i] + epsilon[i]; x2[j] = x2[j] - epsilon[j]
        x3[i] = x3[i] - epsilon[i]; x3[j] = x3[j] + epsilon[j]
        x4[i] = x4[i] - epsilon[i]; x4[j] = x4[j] - epsilon[j]
        Hessian[i, j] = (garchLLH(x1)-garchLLH(x2)-garchLLH(x3)+garchLLH(x4))/
            (4*epsilon[i]*epsilon[j])
        }
    }

    # Step 6: Create and Print Summary Report:
    se.coef = sqrt(diag(solve(Hessian)))
    tval = fit$par/se.coef
    matcoef = cbind(fit$par, se.coef, tval, 2*(1-pnorm(abs(tval))))
    dimnames(matcoef) = list(names(tval), c(" Estimate",
        " Std. Error", " t value", "Pr(>|t|)"))
    cat("\nCoefficient(s):\n")
    printCoefmat(matcoef, digits = 6, signif.stars = TRUE)
}
```

■ Code Snippet 1: Example script which shows the six major steps to estimate the parameters of a GARCH(1,1) time series model. The same steps are implemented in the "full version" of `garchFit()` which allows the parameter estimation of general ARMA(m,n)-APARCH(p,q) with several types of conditional distribution functions.

- *Step 4 - Log-Likelihood Function:* The function `garchLLH()` computes the "negative" log-likelihood for the GARCH(1,1) model. We use a fast and efficient filter representation for the variance equation, such that no execution time limiting `for` loop will become necessary.
- *Step 5: - Parameter Estimation:* For the GARCH(1,1) model optimization of the log-likelihood function we use the constrained solver `nlminb()` which is available in R and SPlus. To compute standard errors and t-values we evaluate the Hessian matrix numerically.
- *Step 6: - Summary Report:* The results for the estimated parameters together with standard errors and t-values are summarized and printed.

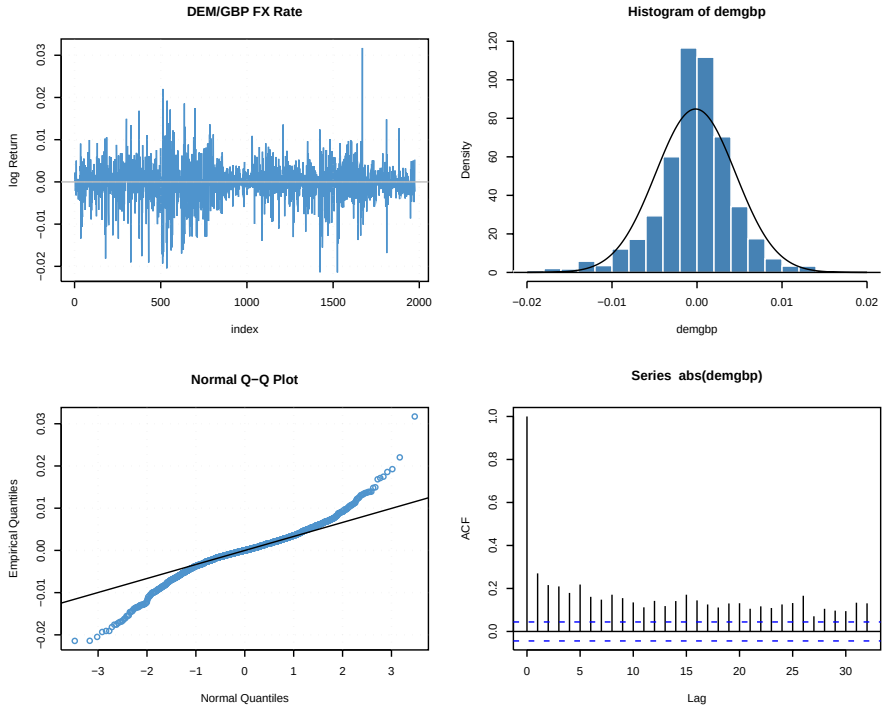
3.2. Case Study: The DEMGBP Benchmark

Through the complexity of the GARCH models it is evident that different software implementations have different functionalities, drawbacks and features and may lead to differences in the numerical results. McCullough and Renfro [1999], Brooks, Burke and Persand [2001], and Laurent and Peters [2003] compared the results of several software packages. These investigations demonstrate that there could be major differences between estimated GARCH parameters from different software packages. Here we use for comparisons the well accepted benchmark suggested by Fiorentini, Calzolari, and Panattoni [1996] to test our implementation.

For the time series data we take the DEMGBP daily exchange rates as published by Bollerslev and Ghysels [1996]. The series contains a total of 1975 daily observations sampled during the period from January 2, 1984, to December 31, 1991. This benchmark was also used by McCullough and Renfro [1999], and by Brooks, Burke, and Persand [2001] in their analysis of several GARCH software packages. Figure 1 shows some stylized facts of the log-returns of the daily DEM/GBP FX rates. The distribution is leptokurtic and skewed to negative values. The log-returns have a mean value of $\mu = -0.00016$, i.e. almost zero, a skewness of $\varsigma = -0.25$, and an excess kurtosis of $\kappa = 3.62$. The histogram of the density and the quantile-quantile plot graphically display this behavior. Furthermore, the autocorrelation function of the absolute values of the returns, which measures volatility, decay slowly.

We use the GARCH(1,1) model to fit the parameters of the time series using the function `garch11Fit()` and compare these results with the benchmark computations of Fiorentini, Calzolari, and Panattoni and with results obtained from other statistical software packages.

```
# Code Snippet 2:  GARCH(1,1) Benchmark for the DEM/GBP Exchange Rates

> data(dem2gbp)
> garch11Fit(x = dem2gbp[, 1])

Coefficient(s):
           Estimate   Std. Error   t value    Pr(>|t|)
mu      -0.00619040   0.00846211  -0.73154  0.46444724
omega    0.01076140   0.00285270   3.77236  0.00016171 ***
alpha    0.15313411   0.02652273   5.77369  7.7552e-09 ***
beta     0.80597365   0.03355251  24.02126  < 2.22e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```



■ Figure 1: Returns, their distribution, the quantile-quantile plot, and the autocorrelation function of the volatility for the DEMGBP benchmark data set.

The estimated model parameters are to more than four digests exactly the same numbers as the FCP benchmark. Note, we used numerically computed derivatives whereas the benchmark used analytically calculated derivatives. The observation that it is not really necessary to compute the derivatives analytically was already made by Doornik [2000] and Laurent and Peters [2001].

	Rmetrics:			FCP Benchmark:			Ox/Garch:			Splus/Finmetrics:		
	Estimate	StdError	t Value	Estimate	StdError	t Value	Estimate	StdError	t Value	Estimate	StdError	t Value
μ	-0.0061904	0.0084621	-0.732	-0.0061904	0.0084621	-0.732	-0.006184	0.008462	-0.731	-0.006053	0.00847	-0.715
ω	0.010761	0.0028527	3.77	0.010761	0.0028527	3.77	0.010761	0.0028506	3.77	0.010896	0.0029103	3.74
α	0.15313	0.026523	5.77	0.15313	0.026523	5.77	0.15341	0.026569	5.77	0.15421	0.026830	5.75
β	0.80597	0.033553	24.0	0.80597	0.033553	24.0	0.80588	0.033542	24.0	0.80445	0.034037	23.6

■ Table 1: Comparison of the results of the GARCH(1,1) parameter estimation for the DEMGBP benchmark data set obtained from several software packages. The first two columns show the results from the GARCH(1,1) example program, the second group from the FCP benchmark results, the third group from Ox' G@RCH, and the last group from SPlus' Finmetrics. The left hand numbers are the parameter estimates and the right hand number the standard errors computed from the Hessian matrix.

4. Alternative Conditional Distributions

Different types of conditional distribution functions $\mathcal{D}$ are discussed in literature. These are the *Normal distribution* which we used in the previous section, the *standardized Student-t distribution* and the *generalized error distribution* and their skewed versions. Since only the symmetric Normal distribution of these functions is implemented in R's base environment and also in most of the other statistical software packages, we discuss the implementation of the other distributions in some more detail.

The default choice for the distribution $\mathcal{D}$ of the innovations z_t of a GARCH process is the *Standardized Normal Probability Function*

$$f^\star(z) \;=\; \frac{1}{\sqrt{2\pi}}\, e^{-\frac{z^2}{2}} \;. \tag{8}$$

The probability function or density is named standardized, marked by a star *, because $f^\star(z)$ has zero mean and unit variance. This can easily be verified computing the *moments*

$$\mu_r^\star \;=\; \int_{-\infty}^{\infty} z^r f^\star(z) dz \;. \tag{9}$$

Note, that $\mu_0^\star \equiv 1$ and $\sigma_1^\star \equiv 1$ are the normalization conditions, that $\mu_1^\star$ defines the mean $\mu \equiv 0$, and $\mu_2^\star$ the variance $\sigma^2 \equiv 1$.

An arbitrary Normal distribution located around a mean value μ and scaled by the standard deviation σ can be obtained by introducing a *location* and a *scale* parameter through the transformation

$$f(z)dz \;\to\; \frac{1}{\sigma} f^\star\Big(\frac{z-\mu}{\sigma}\Big) dz \;=\; \frac{1}{\sigma\sqrt{2\pi}}\, e^{-\frac{(z-\mu)^2}{2\sigma^2}} dz \;. \tag{10}$$

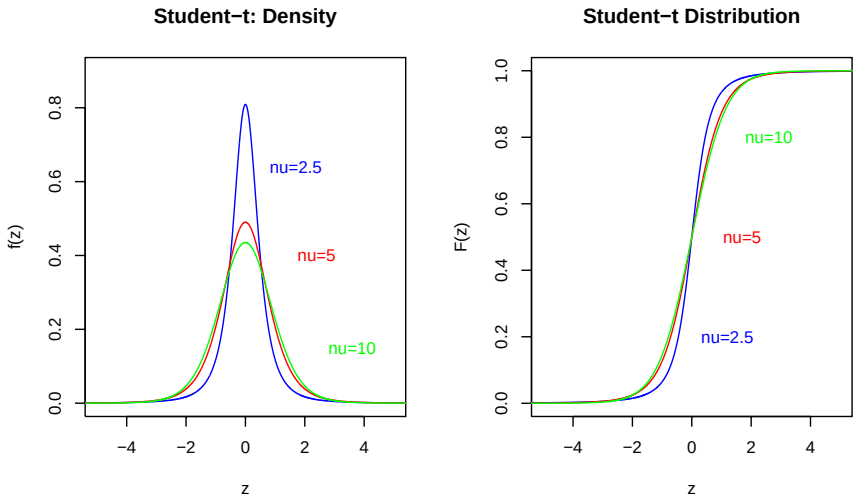
The *central moments* μ_r of $f(z)$ can simply be expressed in terms of the moments $\mu_r^\star$ of the standardized distribution $f^\star(z)$. Odd central moments are zero and those of even order can be computed from

$$\mu_{2r} \;=\; \int_{-\infty}^{\infty} (z-\mu)^{2r} f(z) dz \;=\; \sigma^{2r} \mu_{2r}^\star \;=\; \sigma^{2r}\, \frac{2^r}{\sqrt{\pi}}\, \Gamma\Big(r+\frac{1}{2}\Big) \;, \tag{11}$$

yielding $\mu_2 = 0$, and $\mu_4 = 3$. The degree of asymmetry γ_1 of a probability function, named *skewness*, and the degree of peakedness γ_2, named *excess kurtosis*, can be measured by normalized forms of the third and fourth central moments

$$\gamma_1 = \frac{\mu_3}{\mu_2^{3/2}} = 0 \;, \qquad \gamma_2 = \frac{\mu_4}{\mu_2^2} - 3 = 0 \;. \tag{12}$$

However, if we like to model an asymmetric and/or leptokurtic shape of the innovations we have to draw or to model z_t from a standardized probability function which depends on additional shape parameters which modify the skewness and kurtosis. However, it is important that the probability has still zero mean and unit variance. Otherwise, it would be impossible, or at least difficult, to separate the fluctuations in the mean and variance from the



■ Figure 2: The density and distribution for the Standardized Student-t distribution with unit variance for three different degrees of freedom, $\nu = 2.5, 5, 10$. The graph for the largest value of ν is almost undistinguishable from a Normal distribution.

fluctuations in the shape of the density. In a first step we consider still symmetric probability functions but with an additional shape parameter which models the kurtosis. As examples we consider the generalized error distribution and the Student-t distribution with unit variance, both relevant in modelling GARCH processes.

4.1. Student-t Distribution

Bollerslev [1987], Hsieh [1989)], Baillie and Bollerslev [1989], Bollerslev, Chou, and Kroner [1992], Palm [1996], Pagan [1996)], and Palm and Vlaar [1997] among others showed that the Student-t distribution better captures the observed kurtosis in empirical log-return time series. The density $f^\star(z|\nu)$ of the *Standardized Student-t Distribution* can be expressed as

$$\begin{aligned} f^\star(z|\nu) &= \frac{\Gamma(\frac{\nu+1}{2})}{\sqrt{\pi(\nu-2)}\Gamma(\frac{\nu}{2})} \frac{1}{\left(1+\frac{z^2}{\nu-2}\right)^{\frac{\nu+1}{2}}} \\ &= \frac{1}{\sqrt{\nu-2}\,\mathrm{B}(\frac{1}{2},\frac{\nu}{2})} \frac{1}{\left(1+\frac{z^2}{\nu-2}\right)^{\frac{\nu+1}{2}}}, \end{aligned} \tag{13}$$

where $\nu > 2$ is the shape parameter and $\mathrm{B}(a,b) = \Gamma(a)\Gamma(b)/\Gamma(a+b)$ the Beta function. Note, when setting $\mu = 0$ and $\sigma^2 = \nu/(\nu-2)$ formula (13) results in the usual one-parameter expression for the Student-t distribution as implemented in the S function *dt*.

Again, arbitrary location and scale parameters μ and σ can be introduced via the transformation $z \to \frac{z-\mu}{\sigma}$. Odd central moments of the standardized Student-t distribution are zero

and those of even order can be computed from

$$\mu_{2r} = \sigma^{2r}\mu^{\star}_{2r} \;=\; \sigma^{2r}\;(\nu-2)^{\frac{r}{2}}\;\frac{\mathrm{B}(\frac{r+1}{2},\frac{\nu-r}{2})}{\mathrm{B}(\frac{1}{2},\frac{\nu}{2})}\;. \tag{14}$$

Skewness γ_1 and kurtosis γ_2 are given by

$$\gamma_1 = \frac{\mu_3}{\mu_2^{3/2}} = 0\;, \qquad \gamma_2 = \frac{\mu_4}{\mu_2^2} - 3 = \frac{6}{\nu-4}\;. \tag{15}$$

This result was derived using the recursion relation $\mu_{2r} = \mu_{2r-2}\frac{2r-1}{\nu-2r}$.

We have implemented the functions `[rdpq]std()` to generate random variates, and to compute the density, probability and quantile functions. In the implementation of the distribution function we made use of the relationship to the (non-standardized) Student-t distribution function, `[rdpq]t()`, which is part of R's and SPlus' base installation.

4.2. Generalized Error Distribution

Nelson [1991] suggested to consider the family of *Generalized Error Distributions*, GED, already used by Box and Tiao [1973], and Harvey [1981]. $f^{\star}(z|\nu)$ can be expressed as

$$\begin{aligned} f^{\star}(z|\nu) &= \frac{\nu}{\lambda_\nu 2^{1+1/\nu}\Gamma(1/\nu)}\;e^{-\frac{1}{2}|\frac{z}{\lambda_\nu}|^{\nu}}\;, \\ \lambda_\nu &= \left(\frac{2^{(-2/\nu)}\Gamma\left(\frac{1}{\nu}\right)}{\Gamma\left(\frac{3}{\nu}\right)}\right)^{1/2}\;, \end{aligned} \tag{16}$$

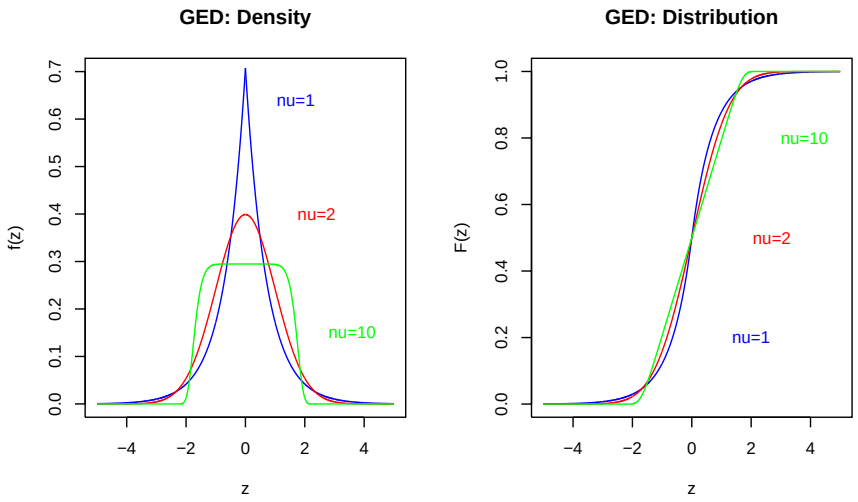
with $0 < \nu \leq \infty$. Note, that the density is standardized and thus has zero mean and unit variance. Arbitrary location and scale parameters μ and σ can be introduced via the transformation $z \to \frac{z-\mu}{\sigma}$. Since the density is symmetric, odd central moments of the GED are zero and those of even order can be computed from

$$\mu_{2r} \;=\; \sigma^{2r}\mu^{\star}_{2r} \;=\; \sigma^{2r}\;\frac{(2^{1/\nu}\lambda_\nu)^{2r}}{\Gamma\left(\frac{1}{\nu}\right)}\Gamma\left(\frac{2r+1}{\nu}\right)\;. \tag{17}$$

Skewness γ_1 and kurtosis γ_2 are given by

$$\gamma_1 = \frac{\mu_3}{\mu_2^{3/2}} = 0\;, \qquad \gamma_2 = \frac{\mu_4}{\mu_2^2} - 3 = \frac{\Gamma\left(\frac{1}{\nu}\right)\Gamma\left(\frac{5}{\nu}\right)}{\Gamma\left(\frac{3}{\nu}\right)^2} - 3\;. \tag{18}$$

For $\nu = 1$ the GED reduces to the Laplace distribution, for $\nu = 2$ to the Normal distribution, and for $\nu \to \infty$ to the uniform distribution as a special case. The Laplace distribution takes the form $f(z) = e^{-\sqrt{2}|z|}/\sqrt{2}$, and the uniform distribution has range $\pm 2\sqrt{3}$. We have implemented functions `[rdpq]ged()` which compute the GED and generate random variates. To compute the distribution function $F(x) = \int_{-\infty}^{x} f(z)dz$ in an efficient way we have transformed $\frac{1}{2}|\frac{z}{\lambda_\nu}|^{\nu} \to z$ and made use of the relationship to the Gamma distribution function which is part of R's and SPlus' base installation.



■ Figure 3: The density and distribution for the GED. Three cases are considered: The leptokurtic *Laplace distribution* with $\nu = 1$, the *Normal distribution* with $\nu = 2$, and the almost *Uniform distribution* for large $\nu = 10$.

4.3. Skewed Distributions

Fernandez and Steel [1998] proposed a quite general approach that allows the introduction of skewness in any continuous unimodal and symmetric distribution by changing the scale at each side of the mode

$$f(z|\xi) \;=\; \frac{2}{\xi + \frac{1}{\xi}} \left[f(\xi z) H(-z) \;+\; f(\frac{z}{\xi}) H(z) \right] \;, \tag{19}$$

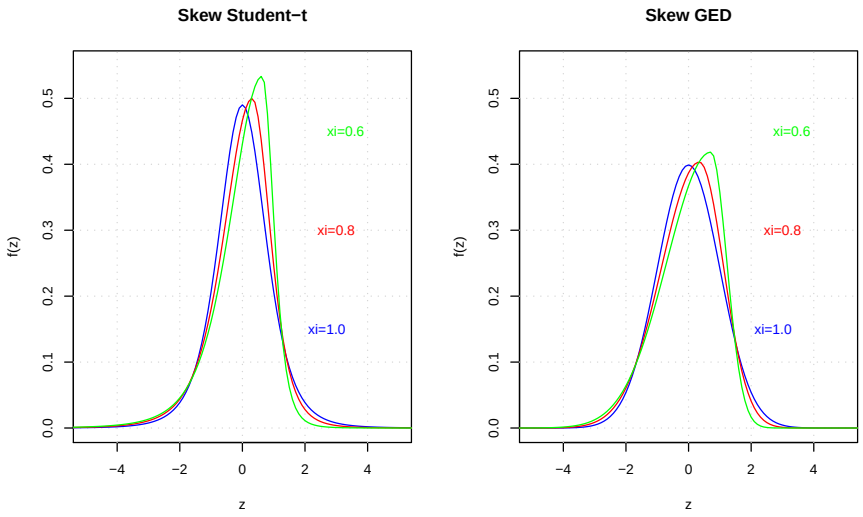
where $0 < \xi < \infty$ is a shape parameter which describes the degree of asymmetry. $\xi = 1$ yields the symmetric distribution with $f(z|\xi = 1) = f(z)$. $H(z) = (1 + \text{sign}(z))/2$ is the Heaviside unit step function. Mean μ_ξ and variance σ_ξ of $f(z|\xi)$ depend on ξ and are given by

$$\begin{aligned} \mu_\xi &= M_1\big(\xi - \frac{1}{\xi}\big) \;, \\ \sigma_\xi^2 &= (M_2 - M_1^2)\big(\xi^2 + \frac{1}{\xi^2}\big) \;, + 2M_1^2 - M_2 \\ M_r &= 2\int_0^\infty x^r \; f(x) \; dx \;, \end{aligned} \tag{20}$$

where M_r is the r-th absolute moment of $f(x)$ on the positive real line. Note, that $M_2 \equiv 1$ if $f(x)$ is a standardized distribution.

Now we introduce a re-parametrization in such a way that the skewed distribution becomes standardized with zero mean and unit variance. Lambert and Laurent [2001] have also reparametrized the density of Fernandez and Steel [1998] as a function of the conditional mean and of the conditional variance in such a way that again the innovation process has zero mean and unit variance. We call a skewed distribution function with zero mean and unit variance *Standardized Skewed Distribution* function.

The probability function $f^\star(z|\xi)$ of a standardized skewed distribution can be expressed in a compact form as



■ Figure 4: The density and distribution for the skew Standardized Student-t with 5 degress of freedom and of the skew GED distribution with $\nu = 2$ with zero mean and unit variance for three different degrees of skewness, $\xi = 1, 0.8, 0.6$.

$$\begin{aligned} f^\star(z|\xi\theta) &= \frac{2\sigma}{\xi + \frac{1}{\xi}}\, f^\star(z_{\mu_\xi\sigma_\xi}|\theta) \\ z_{\mu_\xi\sigma_\xi} &= \xi^{\text{sign}(\sigma_\xi\, z+\mu_\xi)}(\sigma_\xi z + \mu_\xi\ )\ , \end{aligned} \tag{21}$$

where $f^\star(z|\theta)$ may be any standardized symmetric unimodal distribution function, like the standardized Normal distribution (8), the standardized generalized error distribution (16) or the standardized Student-t distribution (13). μ_ξ and σ_ξ can be calculated via equation (20). Transforming $z \to \frac{z-\mu}{\sigma}$ yields skewed distributions, where the parameters have the following interpretation:

- μ - is the mean or location parameter,
- σ - is the standard deviation or the dispersion parameter,
- $0 < \xi < \infty$ - is a shape parameter that models the skewness, and
- θ - is an optional set of shape parameters that model higher moments of even order like ν in the GED and Student-t distributions.

The functions `dsnorm(x, mean, sd, xi)`, `dsged(x, mean, sd, nu, xi)`, and `dsstd(x, mean, sd, nu, xi)` implement the density functions for the skew Normal, the skew GED, and the skew Student-t. Default settings for the mean and standard deviation are `mean=0` and `sd=1`, so that the functions by default are standardized. S functions to compute probabilities or quantiles and to generate random variates follow the common naming conventions used by R and SPlus.

4.4. Fitting GARCH Processes with non-normal distributions

Bollerslev [1987] was the first who modelled financial time series for foreign exchange rates and stock indexes using GARCH(1,1) models extended by the use of standardized Student-t

distributions. In comparison to conditionally normal errors he found that t-GARCH(1,1) errors much better capture the leptokurtosis seen in the data.

It is straightforward to add non-normal distribution functions to our fitting function listed in Code Snippet 1. Then up to two additional parameters which can be either kept fixed or estimated have to be introduced, the `skew` and/or the `shape` parameter. This is left as an exercise to the reader. In the following we use the function `garchFit()` from the Rmetrics software package. The relevant arguments are:

```
garchFit(cond.dist = c("dnorm", "dsnorm", "dstd", "dsstd", "dged", "dsged"),
  skew = 0.9, shape = 4, include.skew = NULL, include.shape = NULL, ...)
```

The argument `cond.dist` allows to select one from six conditional distributions. Three of them are skewed distributions, (the skew-Normal, the skew-Student, and the skew-GED), and four of them have an additional shape parameter (the Student, the GED and their skewed versions) as introduced in the previous section. The value for the skew, ξ, and the shape, ν, are determined through the arguments `skew` and `shape`. Their are two additional undetermined arguments named `include.skew` and `include.shape`. They can be set to `TRUE` or `FALSE`, then the distributional parameters are included or kept fixed during parameter optimization, respectively. If they are undetermined, then an automate selection is done depending on the choice of the other parameters.

```
# Code Snippet 3: Fitting Bollerslev's t-GARCH(1,1) Model

> garchFit(x = "dem2gbp", cond.dist = "dst")
         Estimate  Std. Error  t value Pr(>|t|)
  mu      0.002249    0.006954    0.323   0.7464
  omega   0.002319    0.001167    1.987   0.0469 *
  alpha1  0.124438    0.026958    4.616 3.91e-06 ***
  beta1   0.884653    0.023517   37.617  < 2e-16 ***
  shape   4.118427    0.401185   10.266  < 2e-16 ***
```

Note, that the model has not to be specified explicitly, since the GARCH(1,1) model is the default. Another example shows how to fit a GARCH(1,1) model with Laplace distributed errors. The Laplace distribution is a GED with shape parameter $\nu = 1$. In this example we show the estimated parameters from all three types of implemented solvers:

```
# Code Snippet 4: Fitting Laplace-GARCH(1,1) Model With Different Solvers

> garchFit(x = "dem2gbp", cond.dist = "dged", shape = 1, include.shape = FALSE,
  algorithm = "nlminb+nm")
         Estimate  Std. Error  t value Pr(>|t|)
  mu     0.0030970   0.0002159   14.346  < 2e-16 ***
  omega  0.0040774   0.0018143    2.247   0.0246 *
  alpha1 0.1360974   0.0321703    4.231 2.33e-05 ***
  beta1  0.8661677   0.0304597   28.436  < 2e-16 ***

> garchFit(cond.dist = "dged", shape = 1, include.shape = FALSE,
  algorithm = "lbfgsb+nm")
         Estimate  Std. Error  t value  Pr(>|t|)
  mu     0.0030970   0.0002159   14.346  < 2e-16 ***
  omega  0.0040774   0.0018143    2.247   0.0246 *
  alpha1 0.1360980   0.0321704    4.231 2.33e-05 ***
  beta1  0.8661671   0.0304599   28.436  < 2e-16 ***
```

```
> garchFit(cond.dist = "dged", shape = 1, include.shape = FALSE,
  algorithm = "sqp")
         Estimate  Std. Error  t value  Pr(>|t|)
  mu      0.003098    0.050169    0.062    0.951
  omega   0.004077    0.001831    2.226    0.026 *
  alpha1  0.136075    0.033163    4.103 4.07e-05 ***
  beta1   0.866182    0.031381   27.602  < 2e-16 ***
```

We observe an agreement up to four digits in the GARCH coefficients ω, α, β, and an agreement up to three digits for the ARMA coefficient μ.

5. ARMA(m,n) Models with GARCH(p,q) Errors

The next natural extension of the function `garchFit()` is to allow to model ARMA(m,n) time series processes with GARCH(p,q) errors. Both, ARMA and GARCH may have general orders m, n, p, q. For the specification we make use of the *formula* description of the S language. The implementation of this extension requires three points to be considered:

- How to initialize the iteration of the ARMA and GARCH recursion formula?
- What is an efficient optimization algorithm?
- How to implement efficiently the ARMA and GARCH recursion formula?

For the initialization of the recursion formula we have implemented for the normal GARCH model using the SQP algorithm two different types of startups which we call the *"mu-current-iteration"*, briefly `init.rec="mci"`, and *"unconditional-expected-variances"*, briefly `method="uev"`. For all other models and algorithms the `"mci"` type of initialization will be used.

To explore the convergence behavior and efficiency of the optimization algorithms we have implemented the R solvers `nlminb()`, `optim("L-BFGS-B")` which are available in R's base distribution, and have added an SQP solver implemented in a S function calling an underlying Fortran subroutine. For the first two implementations written entirely in S we used a numerically effective `filter()` implementation without `for`-loops, whereas in the Fortran version of the SQP algorithm, the log-likelihood function and the formula for the numerical evaluation of the Hessian are also implemented internally in Fortran.

The major arguments in the `garchFit()` function are

```
garchFit(formula.mean = ~arma(0,0), formula.var = ~garch(1,1),
   init.rec = c("mci", "uev"), algorithms = c("sqp", "nlminb", "bfgs"))
```

the remaining optional arguments will be discussed later.

5.1. The Recursion Initialization

In the DEM2GBP GARCH(1,1) Benchmark, FCP use the mean and variance initialization for the innovations z and conditional variances h. We have implemented in Rmetrics the following scheme

$$z_{1:\ell} = 0 \ , \quad h_{1:\ell} = \omega + \wp\Upsilon \ , \tag{22}$$

where ω is the current value of the GARCH coefficient under optimization, $\wp$ the persistence, and Υ the variance computed from one of the following two expressions

$$\begin{aligned} \Upsilon &= (1/T)\Sigma_1^T z_t^2 \quad \text{for "mci"}, \\ \Upsilon &= (1-\wp)\omega \quad \text{for "uev"}, \end{aligned} \tag{23}$$

In the case of the normal GARCH(p,q) we have $\wp = \sum \alpha_i + \sum \beta_j$.

```
# Code Snippet 5: Comparing "mci" and "uev" Recursion Initialization

> garchFit(series = "dem2gbp")@fit$coef
          mu        omega       alpha1        beta1
-0.006190408  0.010761398  0.153134060  0.805973672

> garchFit(series = "dem2gbp", init.rec = "uev")@fit$coef
          mu        omega       alpha1        beta1
-0.006269318  0.010983393  0.148699664  0.805808563
```

The results are compared in Table 2 with those obtained from the software package TSP 4.4, which is one of the few packages which allows to specify how to initialize the recursion of the variance formula.

	Rmetrics "mci" Estimate	TSP 4.4 "mci" Estimate	Rmetrics "uev" Estimate	TSP 4.4 "uev" Estimate
μ	-0.00619041	-0.00619041	-0.00626932	-0.00626932
ω	0.010761	0.010761	0.010983	0.010983
α	0.153134	0.153134	0.148700	0.148700
β	0.805974	0.805974	0.805809	0.805809
LLH	1106.608	1106.608	1106.949	1106.949

■ Table 2: Comparison of the parameter estimates for the DEMGBP normal-GARCH(1,1) benchmark model as obtained from Rmetrics and TSP 4.4. `"mci"` denotes the *"mu-current-iteration"* startup, and "uev" denotes the *"unconditional-expected-variances"*, startup. Note, that the results agree in all printed digits!

5.2. The Solvers

The fact why the function `garchFit()` has implemented different optimization algorithms has historical reasons. When the first version was written the only solver implemented in R was the AMC TOMS 778 algorithm `optim(method="L-BFGS-B")`. Later when the function `nlminb()` was added to R's base package, interfacing the PORT Mathematical Software Library, we implemented it in `garchFit()`. Unfortunately, we found that for both solvers it was not always possible to find the optimal values for the set of model parameters even if we tried to adapt the control parameters. This problem was solved in many cases using a two stage approach: First find a (near optimal) solution with one of the two constrained algorithms, and then start with this solution a second optimization step with the (unconstrained) simplex algorithm of Nelder-Mead, `optim(method="Nelder-Mead")`. The result is that one finds with

this "hybrid approach" or "mixed method" in many cases an improved solution. The idea of two step approaches is also mentioned in McCullough and Renfro [1999].

The idea to implement the *Sequential Quadratic Programming* algorithm, SQP, was inspired by the fact that this algorithm is also part of the Ox, Doormik [1999], Matlab and Gauss, Schoenberg [1999], software packages. SQP based algorithms represent very efficient nonlinear programming methods. SQP algorithms have been implemented and tested, Schittkowski [1999], that outperform significantly other methods in terms of accuracy and efficiency over a large number of test problems. Based on work of Biggs [1999], Han [1999], and Powell [1999], [1999], the method allows to closely mimic constrained optimization just as it is done for unconstrained optimization. An overview of SQP methods can be found in Fletcher [1999], Gill et al. [1999], Powell [1999], and Schittkowski [1999]. We have interfaced to Rmetrics the SQP Fortran program written by Luksan [1999], which is a recursive quadratic programming method with the BFGS variable metric update for general nonlinear programming problems. This `algorithm="sqp"` is the default solver used for the optimization of all GARCH models.

5.3. Iteration of the Recursion Formulas

When we have written the `garchFit()` function entirely in S using the R solver `nlminb()` and `optim("L-BFGS-B")` the bottleneck appeared in the computation of the log-likelihood function. The computation seems to require a double `for`-loop in its most simple implementation.

```
# Code Snippet 6: APARCH - Computing Conditional Variances Effectively

> N = 10000; eps = round(rnorm(N), digits = 2) / 10
> omega = 0.1; alpha = c(0.1, 0.05); gamma = c(0, 0); beta = c(0.4, 0.3); delta = 2
> u = length(alpha); v = length(beta); uv = max(u,v); h = rep(0.1, uv)

# Case I: Conditional Variances as Double for-Loop:
> for (i in(uv+1):N ) {
+   ed = 0
+   for (j in 1:u) {
+     ed = ed+alpha[j]*(abs(eps[i-j])-gamma[j]*eps[i-j])^delta
+   }
+   h[i] = omega + ed + sum(beta*h[i-(1:v)])
+ }
```

The usage of the very time consuming double loop could be prevented using a (tricky) filter representation of the processes build on top of the S function `filter()`. The following lines of code are an excellent example how to avoid loops in S scripts and writing fast and efficient S code.

```
# Code Snippet 7: Using R's Filter Representation

# Case II: Conditional Variances in Filter Representation - Loopless:
> edelta = (abs(eps)-gamma*eps)^delta
> edeltat = filter(edelta, filter = c(0, alpha), sides = 1)
> c = omega/(1-sum(beta))
> h = c( h[1:uv], c + filter(edeltat[-(1:uv)], filter = beta,
+   method = "recursive", init = h[uv:1]-c))
```

We like to remark, that the computation of the conditional variances by the filter representation speeds up the computation of the log-likelihood function by one to two orders of magnitude in time depending on the length of the series.

In the case of the SQP Fortran algorithm the computation of the log-likelihood function, of the gradient vector, of the Hessian matrix, and of the conditinal distributions is implemented entirely in Fortran. This results in a further essential speedup of execution time.

5.4. Tracing the Iteration Path

The parameter estimation is by default traced printing information about the model initialization and about the iteration path

```
# Code Snippet 8: Tracing the "t-MA(1)-GARCH(1,2)" Model Fit

> garchFit(~arma(0,1), ~garch(1,2), cond.dist = "dstd")

# Partial Output ...

Series Initialization:
 ARMA model:                  arma
 Formula mean:                ~ arma(0, 1)
 GARCH model:                 garch
 Formula var:                 ~ garch(1, 2)
 ...
 Recursion Init:              mci

Parameter Initialization:
 Initial Parameters:          $params
 Limits of Transformations:   $U, $V
 Which Parameters are Fixed?  $includes
 Parameter Matrix:
                      U           V       params includes
   mu     -1.642679e-01   0.1642679 -0.016426142     TRUE
   ma1    -9.999990e-01   0.9999990  0.009880086     TRUE
   omega   2.211298e-07  22.1129849  0.022112985     TRUE
   alpha1  1.000000e-06   0.9999990  0.100000000     TRUE
   gamma1 -9.999990e-01   0.9999990  0.100000000    FALSE
   beta1   1.000000e-06   0.9999990  0.400000000     TRUE
   beta2   1.000000e-06   0.9999990  0.400000000     TRUE
   delta   0.000000e+00   2.0000000  2.000000000    FALSE
   skew    1.000000e-02 100.0000000  1.000000000    FALSE
   shape   1.000000e+00 100.0000000  4.000000000     TRUE
 Index List of Parameters to be Optimized:
   mu    ma1  omega alpha1  beta1  beta2  shape
    1      2      3      4      6      7     10
 ...

Iteration Path:
 SQP Algorithm

 X and LLH improved to:
 [1] -1.642614e-02  9.880086e-03  2.211298e-02 1.000000e-01  4.000000e-01
 [6]  4.000000e-01  4.000000e+00  1.034275e+03
 ...
 X and LLH final values:
```

```
 [1] 3.119662e-03 3.341551e-02 2.847845e-03 1.721115e-01 2.998233e-01
 [6] 5.407535e-01 4.139274e+00 9.852278e+02
 ...

Control Parameters:
 IPRNT   MIT   MFV   MET   MEC   MER   MES
     1   200   500     2     2     1     4
  XMAX  TOLX  TOLC  TOLG  TOLD  TOLS   RPF
 1e+03 1e-16 1e-06 1e-06 1e-06 1e-04 1e-04

Time to Estimate Parameters:
 Time difference of 7 secs

Hessian Matrix:
...

Coefficients and Error Analysis:
         Estimate  Std. Error  t value Pr(>|t|)
 mu      0.003120    0.007177    0.435 0.663797
 ma1     0.033416    0.023945    1.396 0.162864
 omega   0.002848    0.001490    1.911 0.056046 .
 alpha1  0.172111    0.033789    5.094 3.51e-07 ***
 beta1   0.299823    0.147459    2.033 0.042026 *
 beta2   0.540753    0.144052    3.754 0.000174 ***
 ---
 Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Log Likelihood:
 985.2278    normalized:  0.4991022
```

In summary, the report gives us information about *(i)* the series initialization, *(ii)* the parameter initialization, *(iii)* the iteration path of the optimization, *(iv)* the Hessian matrix, and *(v)* the final estimate including values for the the standard errors and the t-values.

6. APARCH(p,q) - Asymmetric Power ARCH Models

The last extension we consider is concerned with the Taylor effect and the leverage effect.

6.1. The Taylor-Schwert GARCH Model

Taylor [1986] replaced in the variance equation the conditional variance by the conditional standard deviation, and Schwert [1989] modeled the conditional standard deviation as a linear function of lagged absolute residuals. This lead to the so called Taylor-Schwert GARCH model. The model can be estimated fixing the value of δ, i.e. setting `delta=1` and `include.delta=FALSE`, and excluding `leverage=FALSE` terms:

```
# Code Snippet 9: Fitting the TS-GARCH(1,1) Model

> garchFit(formula.var = ~aparch(1,1), delta = 1, include.delta = FALSE, leverage = FALSE)

LLH: 1104.411
          mu        omega       alpha1        beta1
-0.005210079  0.030959213  0.166849469  0.808431234
```

6.2. The GJR GARCH Model

The second model we consider is the model introduced by Glosten, Jagannathan and Runkle [1993]. It is a variance model, i.e. fixed $\delta = 2$ and allows for leverage effects:

```
# Code Snippet 10: Fitting the GJR-GARCH(1,1) Model

> garchFit(formula.var = ~aparch(1,1), delta = 2, include.delta = FALSE)

LLH: 1106.101
          mu        omega       alpha1       gamma1        beta1
-0.007907355  0.011234020  0.154348236  0.045999936  0.801433933
```

6.3. The DGE GARCH Model

The last example in this series shows how to fit an APARCH(1,1) model introduced by Ding, Granger and Engle [1999]:

```
# Code Snippet 11: Fitting the DGE-GARCH(1,1) Model

> garchFit(formula.var = ~aparch(1,1))

LLH: 1101.369
          mu        omega       alpha1       gamma1        beta1        delta
-0.009775186  0.025358108  0.170567970  0.106648111  0.803174547  1.234059172
```

7. An Unique GARCH Modelling Approach

So far we have considered parameter estimation for heteroskedastic time series in the framework of GARCH modelling. However, parameter estimation is not the only one aspect in the analysis of GARCH models, several different steps are required in an unique approach: The *specification* of a time series model, the *simulation* of artificial time series for testing purposes, as already mentioned the *parameter estimation*, the *diagnostic analysis*, and the computation of *forecasts*. In addition methods are needed for printing, plotting, and summarizing the results. Rmetrics provides for each step in the modelling process a function:

- `garchSpec()` - specifies a GARCH model. The function creates a specification object of class `"garchSpec"` which stores all relevant information to specify a GARCH model.
- `garchSim()` - simulates an artificial GARCH time series. The function returns a numeric vector with an attribute defined by the specification structure.
- `garchFit()` - fits the parameters to the model using the maximum log-likelihood estimator. This function estimates the time series coefficients and optionally the distributional parameters of the specified GARCH model.
- `print`, `plot`, `summary`, - are S3 methods for an object returned by the function `garchFit()`. These functions print and plot results, create a summary report and perform a diagnostic analysis.
- `predict` - is a generic function to forecast one step ahead from an estimated model. This S3 method can be used to predict future volatility from a GARCH model.

7.1. The Specification Structure

The function `garchSpec()` creates a S4 object called *specification structure* which specifies a time series process from the ARMA-APARCH family and which can be extended easily to include further type of GARCH models. The specification structure maintains information that defines a model used for time series simulation, parameter estimation, diagnostic analysis, and forecasting:

```
# garchSpec Class Representation:
  setClass("garchSpec",
    representation(
    call = "call",
    formula = "formula",
    model = "list",
    presample = "matrix",
    distribution = "character")
  )
```

The slots of an S4 `garchSpec` object include the `@call`, a `@formula` expression that describes symbolically the model, a list with the `@model` coefficients and the distributional parameters, a `@presample` character string which specifies how to initialize the time series process, and the name of the conditional `@distribution` function. The specification structure is a very helpful object which can be attributed to other objects like the result of a time series simulation or a parameter fit, so that always all background information is available. A specification structure can be in principle created from scratch using the function `new`, but it is much more comfortable to use the function `garchSpec()`. The meaning of the arguments of this function and its default values

```
> args(garchSpec)
function (model = list(omega = 1.0e-6, alpha = 0.1, beta = 0.8), presample = NULL,
  cond.dist = c("rnorm", "rged", "rstd", "rsnorm", "rsged", "rsstd"), rseed = NULL)
```

is summarized in the following list:

- `model` - a list with the model parameters as entries
  - `omega` - the variance value for the GARCH/APARCH specification,
  - `alpha` - a vector of autoregressive coefficients of length `p` for the GARCH/APARCH specification,
  - `beta` - a vector of moving average coefficients of length `q` for the GARCH/APARCH specification,
  - *further optional arguments:*
  - `gamma` - an optional vector of leverage coefficients of length `p` for the APARCH specification,
  - `mu` - the mean value (optional) for ARMA/GARCH specification,
  - `ar` - a vector of autoregressive coefficients (optional) of length `m` for the ARMA specification,
  - `ma` - a vector of moving average coefficients (optional) of length n for the ARMA specification,
  - `delta` - the (optional) exponent value used in the variance equation,
  - `skew` - a numeric value specifying the skewness ξ of the conditional distribution,
  - `shape` - a numeric value specifying the shape ν of the conditional distribution.
- `presample` - either `NULL` or a numeric `"matrix"` with 3 columns and at least $\max(m, n, p, q)$ rows. The first column holds the innovations, the second the conditional variances, and the last the time series. If `presample=NULL` then a default presample will be generated.
- `cond.dist` - a character string selecting the desired distributional form of the innovations either
  `"dnorm"` for the Normal,
  `"dged"` for the Generalized Error,
  `"dstd"` for the standardized Student-t, or

"dsnorm" for the skewed normal,
"dsged" for the skewed GED, or
"dsstd" for the skewed Student-t distribution.

- rseed - NULL or an integer value. If set to an integer, the value is the seed value for the random number generation of innovations.

The function garchSpec() takes the inputs and derives the formula object from the model arguments. Then the remaining values are assigned to the slots of the S4 object of class "garchSpec" and returned.

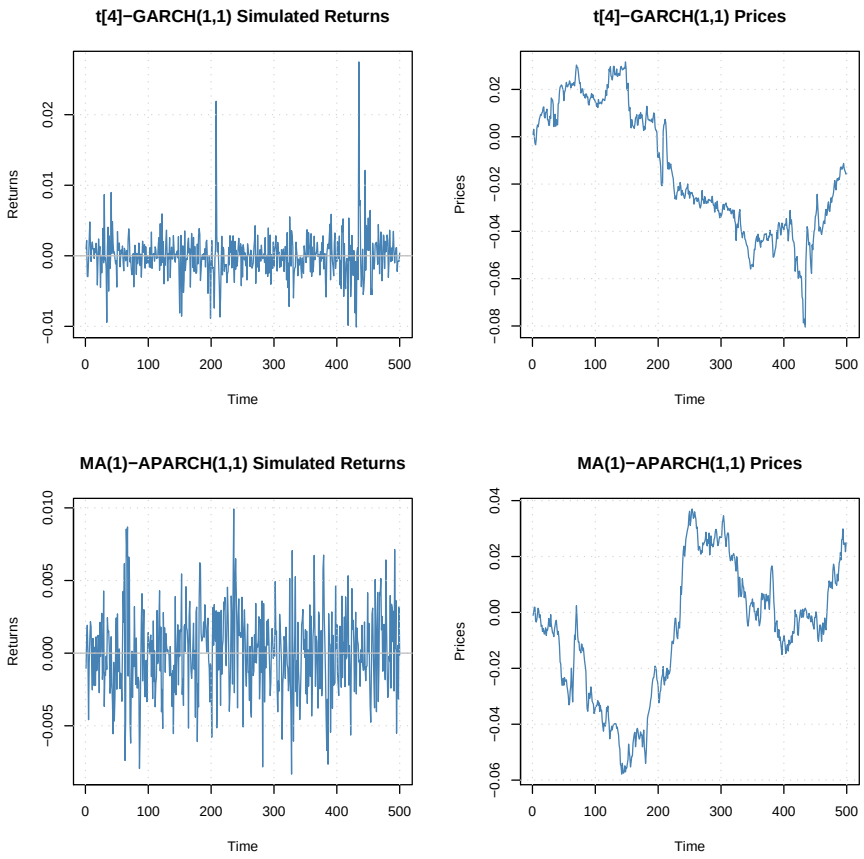
The model is specified by default as a GARCH(1,1) process with $\omega = 10^{-6}$, $\alpha = 0.1$, $\beta = 0.8$, and with normal innovations. In general only a minimum of entries to the model list have to be declared, missing values will be replaced by default settings. The model list can be retrieved from the @model slot. Until now allowed models include the "ARCH", the "GARCH", and the "APARCH" type of heteroskedastic time series processes. A formula object is automatically created from the model list and available through the @formula slot, which is a list with two formula elements named formula.mean and formula.var, most likely returned as arma(m,n) and garch(p,q), where m, n, p, and q are integers denoting the model order. "arma" can be missing in the case of *iid* innovations or can be specified as "ar(m)" or "ma(n)" in the case of innovations from pure autoregressive or moving average models. garch(p,q) may be alternatively specified as "arch(p)" or aparch(p,q). Note, that the conditional distribution used for the innovations cannot be extracted from the @formula or @model slots, its name is available from the @distribution slot, but the distributional parameters are part of the model list.

The next code snippet shows the specification and printing of a t-MA(1)-GARCH(1,1) model where innovations are taken from a Student-t conditional distribution function:

```
# Code Snippet 12: Specifying an t-MA(1)-GARCH(1,1) Model

> garchSpec(model = list(mean = 0.1, ma = 0.3, omega = 2.0e-6, alpha = 0.12,
    beta = 0.84, skew = 1.1, shape = 4), cond.dist = "rsstd", rseed = 4711)

Formula:
   ~ ma(1) + garch(1, 1)
Model:
   ma:    0.3
   omega: 2.0e-6
   alpha: 0.12
   beta:  0.84
Distribution:
   rsstd
Distributional Parameters:
   nu = 4  xi = 1.1
Random Seed:
   4711
Presample:
     time        z     h y
   0    0 1.819735 5e-05 0
```

7.2. Simulation of Artificial Time Series

The function garchSim() creates an artificial ARMA time series process with GARCH or APARCH errors. The function requires the model parameters which can also be an object of



■ Figure 5: Returns and prices for two simulated GARCH processes. The upper two graphs represent Bollerslev's `t[4]-GARCH(1,1)` model with a conditional distribution derived from the Student-t with 4 degrees of freedom. The lower two graphs represent a normal `MA(1)-APARCH(1,1)` model as introduced by Ding, Granger and Engle.

class `garchSpec`, the length `n` of the time series, a `presample` to start the iteration, and the name of `cond.dist`, the conditional distribution to generate the innovations. A presample will be automatically created by default if not specified. The function `garchSim()` returns the sample path for the simulated return series. The model specification is added as an attribute.

```
# Code Snippet 13: Simulating Bollerslev's GARCH(1,1) Model

> model = list(omega = 1.0e-6, alpha = 0.13, beta = 0.81)
> garchSim(model, n = 100)

Time Series:
Start = 1
End = 100
Frequency = 1
  [1]  7.708289e-04  3.201469e-03  5.455730e-03  1.092769e-03  9.551689e-03
  [6] -1.602702e-03  1.893069e-03 -4.404711e-03 -3.380807e-03 -2.287276e-03
 [11] -2.202586e-03  4.725309e-03 -1.817997e-03  ...
attr(,"spec")
```

```
Formula:
~ garch(1, 1)
Model:
 omega: 1e-06
 alpha: 0.13
 beta:  0.81
Distribution:
 rnorm
Presample:
   time         z            h y
 0    0 0.4157423 1.666667e-05 0

# Note, garchSim() also accepts a Specification Structure:
> spec = garchSpec(model)
> garchSpec(model = spec, n = 100)
```

In the same way we can use the Rmetrics function `garchSim()` to simulate more complex ARMA-GARCH/APARCH models. Here are some examples:

```
# Code Snippet 14: Simulating More Complex ARMA-GARCH Models

# ARMA-APARCH Simulation - Show Argument List:
> args(garchSim)
  function (model = list(omega = 1e-06, alpha = 0.1, beta = 0.8), n = 100, n.start = 100,
    presample = NULL, cond.dist = c("rnorm", "rged", "rstd", "rsnorm", "rsged", "rsstd"),
    rseed = NULL)

# Specify ARCH(2) Model:
> model = list(omega = 1e-6, alpha = c(0.1, 0.3), beta = 0)
> garchSim(model)

# Specify Bollerslev t[4]-GARCH(1,1) Model:
> model <- list(omega = 1e-6, alpha = 0.15, beta = 0.75, shape = 4)
> garchSim(model, cond.dist = "rstd")

# Specify Ding-Engle-Granger MA(1)-APARCH(1,1) Model:
> model = list(ma = 0.1, omega = 1e-6, alpha = 0.15, gamma = 0.3, beta = 0.75, delta = 1.3))
> garchSim(model)
```

7.3. Tailored Parameter Estimation:

How to estimate the parameters of an ARMA-GARCH/APARCH model was already shown in detail in the previous sections. In this section we discuss some technical details concerning the S function `garchFit()` and explain how one can tailor the parameter estimation. The estimation process is structured by the successive call of internal functions:

- `garchFit` - Main parameter estimation function
- `.garchInitSeries` - Initializes time series
- `.garchInitParameters` - Initializes the parameters to be optimized
- `.garchSetCondDist` - Defines the conditional distribution
- `.garchOptimizeLLH` - Optimizes the log-likelihood function
- `.garchLLH` - Computes the log-likelihood function
- `.garchHessian` - Computes the Hessian matrix

All functions are written entirely in S. There is one exception concerned with the function `.garchOptimizeLLH()` which is implemented in two different ways. The first type of implementation using the R solvers coming with R's base package is written entirely in S. The second type of implementation (the default) uses the Fortran SQP solver with the gradient, the likelihood function and the conditional distribution functions also implemented in Fortran. For the user who is not interested in implementation details, there is no difference in using one or the other type of implementation.

garchFit: Argument List

The dot functions are internal functions which are called by the main function `garchFit()` with options specified in the argument list. Some of the arguments were already described in previous sections, here we give a brief summary of all arguments:

- `formula.mean` - a formula object for the ARMA(m,n) mean specification
- `formula.var` - a formula object for the GARCH/APARCH(p,q) variance specification
- `series` - a numeric vector specifying the time series
- `init.rec` - a character string naming the type of initialization of recurrence
- `delta` - a numeric value specifying the exponent delta
- `skew` - a numeric value specifying the optional skewness parameter
- `shape` - a numeric value specifying the optional shape parameter
- `cond.dist` - a numeric value specifying the name of the conditional distribution
- `include.mean` - a logical value, should the mean value be estimated ?
- `include.delta` - a logical value, should the exponent be estimated ?
- `include.skew` - a logical value, should the skewness parameter be estimated ?
- `include.shape` - a logical value, should the shape parameter be estimated ?
- `leverage` - a logical value, should the leverage factors be estimated ?
- `trace` - a logical value, should the optimization be traced ?
- `algorithm` - a character string naming the optimization algorithm
- `control` - a list of control parameters for the selected solver
- `title` - an optional project title string
- `description` - an optional project description string

fGARCH: Class Representation

The function `garchFit()` returns an S4 object of class `"fGARCH"` which has the following representation:

```
# fGARCH Class Representation:
setClass("fGARCH",
  representation(
    call = "call",
    formula = "list",
    method = "character",
    data = "list",
    fit = "list",
    residuals = "numeric",
    fitted.values = "numeric",
    sigma.t = "numeric",
    title = "character",
    description = "character")
)
```

The "fGARCH" class representation has 10 slots. @call is a character string telling how the function was invoked, @formula is a list with two formula entries, the formula.mean and formula.var, @method is a string describing the parameter estimation, @data is a list holding the empirical data set, @fit is a list with all information and parameters from the parameter fit, @residuals, @fitted.values, @sigma.t are three numeric vectors with residuals, fitted values, and conditional standard deviations from the time series, and @title and @description are slots for additional information.

The slot named @fit holds the results as a list object from the parameter estimation depending on the solver used: "sqp", "nlminb", or "lbfgsb". The entries of the list are:

- @fit$coef - the estimated parameters
- @fit$se.par - the standard errors of the parameters
- @fit$llh - the value of the log-likelihood function
- @fit$grad - the value of the gradient vector
- @fit$hessian - the hessian matrix
- @fit$cvar - the covariance matrix
- @fit$ics - the values of information criterion statistics
- @fit$series - a list with series information
- @fit$params - a list with parameter information

The list @fit$series provides information about the time series and has the following major entries:

- @fit$series$model - the model formulas
- @fit$series$order - the ARMA-GARCH/APARCH model orders
- @fit$series$init.rec - the type of recursion initialization

The list @fit$params provides information about the model parameters and has the following major entries:

- @fit$params$params - all model parameters including the fixed
- @fit$params$U - the lower box bounds of the model parameters
- @fit$params$V - the upper box bounds of the model parameters
- @fit$params$includes - logical vector identifying fixed parameters
- @fit$params$index - index vector of included parameters
- @fit$params$cond.dist - name of the conditional distribution
- @fit$params$control - list of control parameters of the solver

As already mentioned 5 different algorithms are implemented for parameter estimation. The selected algorithm can be tailored by the user through a list of control parameters. Common to all algorithms are the following two entries:

- control$fscale - if set to TRUE, then the log likelihood function will be standardized by the length of the time series to be fitted.
- control$xscale - if set to TRUE, then the time series x will be standardized by the standard deviation of x.

The first setting can be considered as an *objective function scaling*, and the second setting as a *parameter scaling*. The coefficients a, and b of the ARMA formula, the coefficients *alpha*, *gamma* and *beta* of the GARCH/APARCH formula, and the distributional parameters are not influenced by this transformation. Only, the constants of the mean formula μ, and the variance formula ω have to be properly rescaled after parameter estimation. In many examples we have observed that this kind of scaling may have a significant influence on the execution time of the estimation process. This may not always be the case, especially if the selected optimization algorithm itself allows for an explicit function and parameter scaling.

sqp: Sequential Quadratic Programming Algorithm

The default `algorithm="sqp"` is a sequential quadratic programming algorithm, Luksan [1999], allowing for general nonlinear constraints. Here we use upper and lower bounds on the parameters, which are chosen automatically. The SQP algorithm, the log likelihood objective function, and the Hessian are written entirely in Fortran 77, guaranteeing a fast and efficient estimate of the parameters. The argument `control` in the function `garchFit()` allows to tailor scaling, step-size selection, and convergence parameters. The set of integer valued control parameters is given by

- `control$MIT=200` - the maximum number of iterations, by default 200.
- `control$MVF=500` - the maximum number of function evaluations, by default 500.
- `control$MET=2` - an identifyer which specifies scaling strategy,
  `MET=1` means no scaling,
  `MET=2` means preliminary scaling in 1st iteration,
  `MET=3` means controlled scaling,
  `MET=4` means interval scaling, and
  `MET=5` means permanent scaling in all iterations.
- `control$MEC=2` - an identifier which allows correction for negative curvature,
  `MEC=1` means no correction, and
  `MEC=2` means Powell correction, the default setting.
- `control$MER=1` - an identifier which controls the restart after unsuccessful variable metric updates,
  `MER=0` means no restarts, and
  `MER=1` means standard restart, the default setting.
- `control$MES=4` - an identifier which selects the interpolation method in a line search,
  `MES=1` means bisection,
  `MES=2` means two point quadratic interpolation,
  `MES=3` means three point quadratic interpolation, and
  `MES=4` - three point cubic interpolation, the default setting.

and the set of real valued control parameters is

- `control$XMAX=1000` - the value of the maximum stepsize.
- `control$TOLX=1.0e-16` - the tolerance parameter for the change of the parameter vector.
- `control$TOLC=1.0e-6` - the tolerance parameter for the constraint violation.
- `control$TOLG=1.0e-6` - the tolerance parameter for the Lagrangian function gradient.
- `control$TOLD=1.0e-6` - the tolerance parameter for a descent direction.
- `control$TOLS=1.0e-4` - the tolerance parameter for a function decrease in the line search.
- `control$RPF=0.001` - the value of the penalty coeffient.

The choice of the control parameters `control$XMAX` and `control$RPF` is rather sensitive. If the problem is badly scaled, then the default value `control$XMAX=1000` can bee too small. On the other hand, a lower value, say `control$XMAX=1`, can sometimes prevent divergence of the iterative process. The default value `control$RPF=0.001` is relatively small. Therfore a larger value, say `control$RPF=1` should sometimes be used. We highly recommend to adapt these two control parameters if convergence problems arise in the parameter estimation process.

nlminb: BFGS Trust Region Quasi Newton Method

The algorithm selected by `algorithm="nlminb"` is available through the S function `nlminb()`. Implemented is a variation on Newton's method, which approximates the Hessian (if not specified) by the BFGS secant (quasi-Newton) updating method. The underlying Fortran routine is part of the Fortran PORT library. To promote convergence from poor starting guesses, the routine uses a model/trust technique, Gay [1983] with box bounds, Gay [1984].

Possible names in the `control` list and their default values are:

- `control$eval.max=200` - the maximum number of function evaluations.
- `control$iter.max=150` - the maximum number of iterations allowed.
- `control$trace=0` - the iteration is printed every trace'th iteration.
- `control$abs.tol=1.0e-20` - the value for the absolute tolerance.
- `control$rel.tol=1.0e-10` - the value for the relative tolerance.
- `control$x.tol=1.0e-8` - the value for the X tolerance.
- `control$step.min=2.2e-14` - the minimum step size.

lbfgs: BFGS Limited Memory Quasi Newton Method

The algorithm selected by the argument `algorithm="lbfgsb"` and implemented in the S function `optim(method="L-BFGS-B)"` is that of Byrd, Lu, Nocedal and Zhu [1995]. It uses a limited-memory modification of the BFGS quasi-Newton method subject to box bounds on the variables. The authors designed this algorithm especially for problems in which information on the Hessian matrix is difficult to obtain, or for large dense problems. The algorithm is implemented in Fortran 77, Zhu, Byrd, Lu, Nocedal [1997].

The `control` argument allows to tailor tracing, scaling, step-size selection, and convergence parameters:

- `control$trace` - an integer, higher values give more information from iteration.
- `control$fnscale` - an overall scaling for the objective function and gradient.
- `control$parscale` - a vector of scaling values for the parameters.
- `control$ndeps=1.0e-3` - a vector of step sizes for the gradient.
- `control$maxit=100` - the maximum number of iterations.
- `control$abstol` - the absolute convergence tolerance.
- `control$reltol` - the relative convergence tolerance.
- `control$lmm=5` - an integer giving the number of BFGS updates.
- `control$factr=1.0e7` - controls the reduction in the objective function.
- `control$pgtol` - controls the tolerance on the projected gradient.

+nm: Nelder-Mead Algorithm with BFGS Start Values

In many cases of practical parameter estimation of ARMA-GARCH and ARMA-APARCH models using the `"nlminb"` and `"lbfgsb"` optimization algorithms with default control parameter settings, we observed, that the iteration process got stuck close to the optimal values. Instead of adapting the control parameters, we found out, that a second step optimization using the Nelder-Mead [1965] algorithm can solve the problem in many cases starting from the final values provided by the `"nlminb"` and `"lbfgsb"` algorithms. This approach can be applied setting the argument `algorithm` in the function `garchFit()` either to `algorithm="nlminb+nm"` or to `algorithm="lbfgs+nm"`. The Nelder-Mead method searches then for a local optimum in an unconstrained optimization problem combining the simplex, a generalized n-dimensional triangle, with specific search rules. The reflection, contraction and expansion factor for the simplex can be controlled by the following parameters:

- `control$alpha=1` - the reflection factor.
- `control$beta=0.5` - the contraction factor.
- `control$gamme=2.0` - the expansion factor.

The additional control parameters for the Nelder-Mead algorithm `control$trace`, `control$fnscale`, `control$parscale`, `control$maxit`, `control$ndeps`, `control$abstol` are the same as specified by the control parameters of the `"nlminb"` and `"lbfgs"` algorithms.

For any details concerning the control parameters we refer to the R help page.

7.4. Print, Summary and Plot Method

The `print()`, `summary()` and `plot()` methods create reports and graphs from an object of class `"fGARCH"` created by the function `garchFit()`. The print and summary report lists the function call, the mean and variance equation, the conditional distribution, the estimated coefficients with standard errors, t values, and probabilities, and also the value of the log likelihood function. In additions the summary report returns a diagnostic analysis of the residuals. The plot function creates 13 plots displaying properties of the time series and residuals.

Print Method: Model Parameters, Standard Errors and t-Values

A very useful feature of the log-likelihood is that second derivatives of the log-likelihood function can be used to estimate the standard errors of the model and distributional parameters. Specifically, we have to compute the Hessian matrix. Taking the negative expectation of the Hessian yields the so called information matrix. Inverting this matrix yields a matrix containing the variances of the parameters on its diagonal, and the asymptotic covariances of the parameters in the off-diagonal positions. The square root of the diagonal elements yields the standard errors.

Beside the estimated model parameters and their standard errors, alltogether, the `print()` method returns the following information:

- `@title` - the title string,
- `@call` - the function call,
- `@formula` - the mean and variance equation,

- `@fit$params$cond.dist` - the name of the conditional distribution,
- `@fit$par` - the vector of estimated coefficients,
- `@fit$matcoef` - the coefficient matrix, where the four columns return
  the parameter estimates,
  the standard errors,
  the t-values, and
  the probabilities,
- `@fit$value` - the value of the log likelihood for the estimated parameters,
- `@description` - the description string.

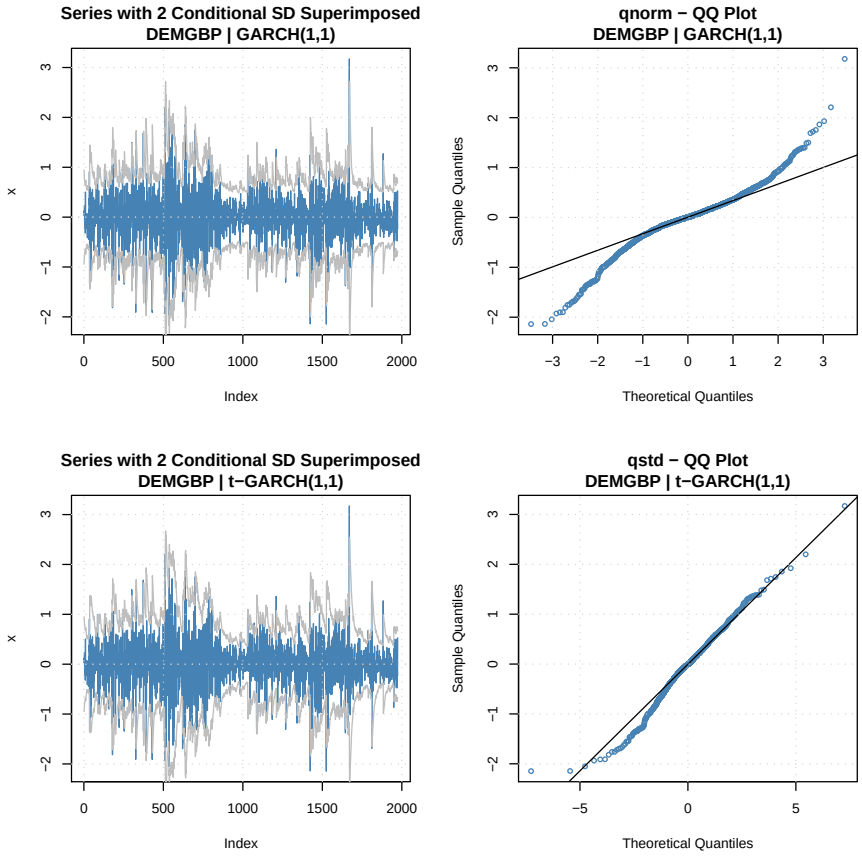
The estimated parameters represent the computer's answers to a solution where the log-likelihood function becomes optimal. The standard error gives then a measure how sure one can be about the estimated parameters. Note, that the standard error for one parameter cannot be compared effortlessly with the standard error of another parameter. For this the t-value are computed which are the ratios of the estimate divided by the standard error. The ration allows a comparison across all parameters. It is just another and better way of measuring how sure one can be about the estimate.

Summary Method: Analysis of Residuals

The summary method allows the analysis of standardized residuals, and thus provides additional information on the quality of the fitted parameters. The summary report adds to the print report the following information:

```
# Code Snippet 15: Summarizing the Results from Parameter Estimates

# Estimate Parameters:
> fit = garchFit()

# Partial Summary Report ...

Standadized Residuals Tests:
                              Statistic p-Value
 Jarque-Bera Test   R    Chi^2  1059.851  0
 Shapiro-Wilk Test  R    W      0.9622817 0
 Ljung-Box Test     R    Q(10)  10.12142  0.4299065
 Ljung-Box Test     R    Q(15)  17.04350  0.3162709
 Ljung-Box Test     R    Q(20)  19.29764  0.5025616
 Ljung-Box Test     R^2  Q(10)  9.062553  0.5261776
 Ljung-Box Test     R^2  Q(15)  16.07769  0.3769074
 Ljung-Box Test     R^2  Q(20)  17.50715  0.6198391
 LM Arch Test       R    TR^2   9.771212  0.6360242

Information Criterion Statistics:
       AIC       BIC       SIC      HQIC
 -1.117131 -1.105808 -1.117139 -1.112970
```

The Jarque-Bera and the Shapiro-Wilk test allow to test for normal distributed residuals, the Ljung-Box test can be performed to test whether the residuals and squared residuals have significant autocorrelations or not, and the Lagrange-Multiplier ARCH test allows to test whether the residuals have conditional heteroskedasticity or not.



■ Figure 6: The upper row shows graphs for the series with 2 conditional standard deviations superimposed (Selection:13) and for the normal quantile-quantile plot (Selection: 13). The lower row shows the same graphs for the parameter estimation with a Student-t conditional distribution.

For the comparison of different fitted models we can follow the same procedures as applied in linear time series analysis. We can compare the value of the log likelihood for the estimated parameters and we can compute information criterion statistics, like AIC and/or BIC statistics to find out which model fits best.

Plot Method: Graphical Plots

The `plot()` method provides 13 different types of plots. (Nota bene, these are the same as created by the SPlus/Finmetrics module.) The user may select from a menu which displays the plot on the screen.

```
# Code Snippet 16: Creating Diagnostic Plots from Parameter Estimates

# Estimate Parameters:
> fit = garchFit()

# Diagnostic Plots:
> plot(fit)
```

```
Make a plot selection (or 0 to exit):

 1:   Time Series
 2:   Conditional SD
 3:   Series with 2 Conditional SD Superimposed
 4:   ACF of Observations
 5:   ACF of Squared Observations
 6:   Cross Correlation
 7:   Residuals
 8:   Conditional SDs
 9:   Standardized Residuals
10:   ACF of Standardized Residuals
11:   ACF of Squared Standardized Residuals
12:   Cross Correlation between r^2 and r
13:   QQ-Plot of Standardized Residuals

Selection:
```

Note, that an explorative data analysis of the residuals is a very useful investigation, since it gives a first idea on the quality of the fit.

7.5. Forecasting Heteroskedastic Time Series

One of the major aspects in the investigation of heteroskedastic time series is to produce forecasts. Expressions for forecasts of both the conditional mean and the conditional variance can be derived.

Forecasting the Conditional Mean:

To forecast the conditional mean we use just R's base function `arima()` and its `predict()` method. This approach, predicting from the ARMA model, is also used for example in the Ox/G@RCH software package using the ARMA prediction from Ox.

Forecasting the Conditional Variance:

The conditional variance can be forecasted independently from the conditional mean. For a GARCH(p,q) process, the h-step-ahead forecast of the conditional variance $\hat{\omega}^2_{t+h|t}$ is computed recursively from

$$\hat{\sigma}^2_{t+h|t} \;=\; \hat{\omega} \;+\; \sum_{i=1}^{q} \hat{\alpha}_i \varepsilon^2_{t+h-i|t} \;+\; \sum_{j=1}^{p} \hat{\beta}_j \sigma^2_{t+h-j|t} \;, \tag{24}$$

where $\varepsilon^2_{t+i|t} = \sigma^2_{t+i|t}$ for $i > 0$ while $\varepsilon^2_{t+i|t} = \varepsilon^2_{t+i}$ and $\sigma^2_{t+i|t} = \sigma^2_{t+i}$ for $i \leq 0$.

For an APARCH(p,q) process the distribution of the innovations may have an effect on the forecast, the optimal h-step-ahead forecast of the conditional variance $\widehat{\omega}^2_{t+h|t}$ is computed recursively from

$$\hat{\sigma}^{\delta}_{t+h|t} \;=\; E(\sigma^{\delta}_{t+h}|\Omega_t) \tag{25}$$

$$\;=\; \hat{\omega} + \sum_{i=1}^{q} \hat{\alpha}_i E[(|\varepsilon_{t+h-i}| - \hat{\gamma}_i \varepsilon_{t+h-i})^{\hat{\delta}}|\Omega_t] + \sum_{j=1}^{p} \hat{\beta}_j \sigma^{\hat{\delta}}_{t+h-j|t} \tag{26}$$

where $E[(|\varepsilon_{t+h-i}| - \hat{\gamma}_i\varepsilon_{t+h-i})^{\hat{\delta}}|\Omega_t] = \kappa_i\sigma^{\hat{\delta}}_{t+k|t}$ for $k > 1$, and $\kappa_i = E(|z| - \gamma_i z)^{\hat{\delta}}$.
Here we have reprinted the formulas for the mean variance forecasts as summarized in the paper of Laurent and Lambert [2002]. The following Code Snippet shows the 10 step ahead forecast for the DEMGBP exchange returns modeled by Bollerslev's GARCH(1,1) model.

```
# Code Snippet 17: Forecasting Mean and Variance

# Estimate Parameters:
> fit = garchFit()

# Forecast 10 step ahead
> predict(fit)

   meanForecast   meanError   standardDeviation
1  -0.006190408   0.4702368           0.3833961
2  -0.006190408   0.4702368           0.3895422
3  -0.006190408   0.4702368           0.3953472
4  -0.006190408   0.4702368           0.4008358
5  -0.006190408   0.4702368           0.4060303
6  -0.006190408   0.4702368           0.4109507
7  -0.006190408   0.4702368           0.4156152
8  -0.006190408   0.4702368           0.4200402
9  -0.006190408   0.4702368           0.4242410
10 -0.006190408   0.4702368           0.4282313
```

In addition Table 3 compares results obtained from SPlus/Finmetrics and Ox/G@RCH.

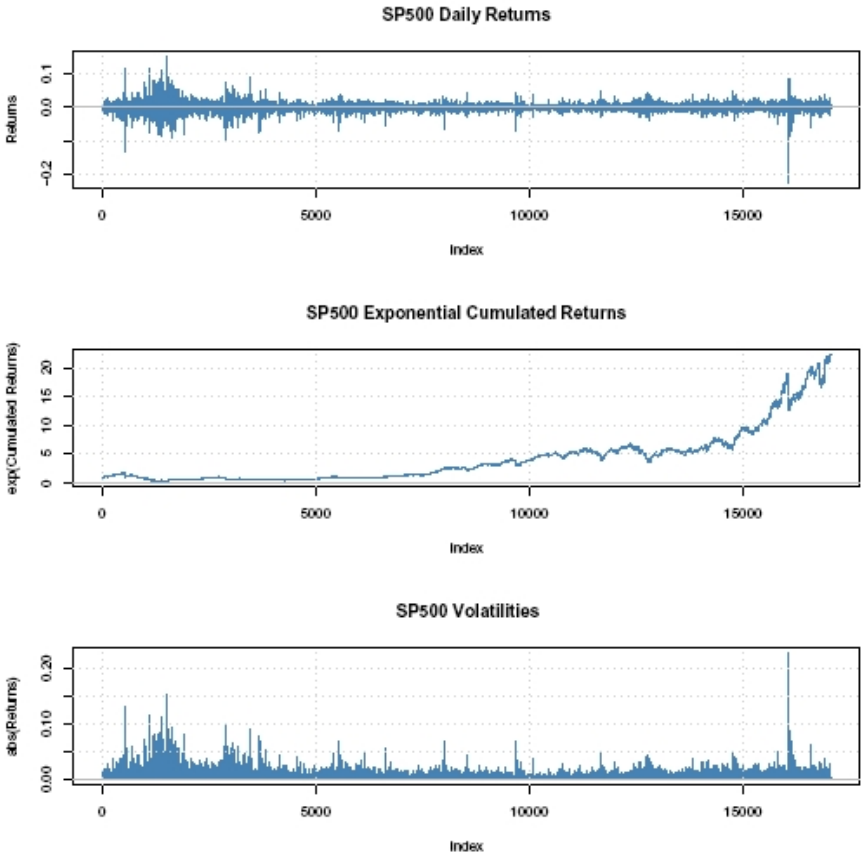
	R/Rmetrics		Splus/Finmetrics		Ox/G@RCH	
Steps:	**Mean**	**StDev**	**Mean**	**StDev**	**Mean**	**StDev**
1	-0.006190	0.3834	-0.006053	0.3838	-0.006183	0.3834
2	-0.006190	0.3895	-0.006053	*0.3900*	-0.006183	0.3895
3	-0.006190	0.3953	-0.006053	0.3959	-0.006183	0.3953
4	-0.006190	0.4008	-0.006053	0.4014	-0.006183	0.4007
5	-0.006190	0.4060	-0.006053	0.4067	-0.006183	0.4060

■ Table 3: Comparison of the 5-step ahead forecast results obtained from Rmetrics, Finmetrics and G@RCH. Note, that the forecasted "Mean" differs for all three packages since the parameter estimate for μ also differs. The standard deviations "StDev" derived from the forecasted variances are in agreement for Rmetrics and G@RCH.

7.6. SP500 Case Study: MA(1)-APARCH(1,1) Modelling

As a last example we analyze the SP500 Index returns as discussed in the famous paper of Ding, Granger and Engle [1993], DGE. The SP500 data are closing index values ranging from January 3, 1928 to August 30, 1991.

In their paper DGE have estimated the parameters for three MA(1) time series models with Bollerslev's GARCH(1,1), with Taylor-Schwert's GARCH(1,1) and with APARCH(1,1) errors. The first order moving average term accounts for the positive first order autocorrelation for the return series. In the following we re-estimate the MA(1)-APARCH(1,1) model and compare



■ Figure 7: From top to bottom are shown the SP500 daily returns, the exponentially cumulated returns, and the volatilities computed as absolute values of the returns. The three graphs reproduce figures 2.1 - 2.3 in Ding, Granger and Engle [1993].

the results with the coefficients published by DGE. In addition we also estimate the parameters using the SPlus/Finmetrics and Ox/G@RCH software for comparison.

```
# Code Snippet 18: Estimating the Parameters for DGE's SP500 Model

# DGE MA(1)-GARCH(1,1) Model Parameter Estimation:
> data(sp500dge)
# Percentual returns result in better scaling and faster convergence ...
> x = 100*sp500dge[, 1]

# R/Rmetrics:
> garchFit(~arma(0,1), ~aparch(1,1))
             Estimate   Std. Error     t value    Pr(>|t|)
mu           0.020646     0.006346       3.253    0.00114 **
ma1          0.144745     0.008357      17.319    < 2e-16 ***
omega        0.009988     0.001085       9.203    < 2e-16 ***
alpha1       0.083803     0.004471      18.742    < 2e-16 ***
gamma1       0.373092     0.027995      13.327    < 2e-16 ***
beta1        0.919401     0.004093     224.622    < 2e-16 ***
delta        1.435124     0.067200      21.356    < 2e-16 ***
```

```
# Rescale:
scale = 1/100; mu = 0.020646; mu*scale
[1] 0.00020646
omega = 0.009988; delta = 1.435124; omega / (1/scale)^(2/delta)
[1] 1.630283e-05

# SPlus/Finmetrics:
# BHHH with Tailored Control Scaled, for use under S-Plus, only
> module(finmetrics)
> x = 100*as.vector(sp500)
> fit = garch(~arma(0,1), ~pgarch(1,1), series = x, leverage = TRUE,
+   control = bhhh.control(tol=1e-6, delta=1e-6, n.iter=10000), trace = TRUE)
# Use Hessian Matrix ...
> coef = fit$coef
> se.coef = sqrt(diag(solve(-fit$cov$A)))
> t.value = coef/se.coef
> data.frame(Estimate = coef, StdError = se.coef, "t value" = t.value)
              Estimate     StdError     t.value
       C    0.02084031  0.006330720    3.291934
   MA(1)    0.14470177  0.008294756   17.444971
       A    0.01002876  0.001091768    9.185798
 ARCH(1)    0.08374599  0.004448664   18.824976
  LEV(1)   -0.37098826  0.027775705  -13.356574
GARCH(1)    0.91954293  0.004078342  225.469798
   POWER    1.42901650  0.067071355   21.305914
# Rescale:
mu = 0.02084; mu*scale
# [1] 0.0002084
omega = 0.01003; delta = 1.42902; omega / (1/scale)^(2/delta)
# [1] 1.592868e-05

# Try: Ox/G@RCH using Rmetrics Interface
> garchFit(~arma(0,1), ~aparch(1,1))
            Coefficient   Std.Error     t-value   t-prob
Cst(M)         0.020375   0.0063657       3.201   0.0014
MA(1)          0.144631   0.0083808       17.26   0.0000
Cst(V)         0.009991   0.0010827       9.228   0.0000
ARCH(Alpha1)   0.083769   0.0044350       18.89   0.0000
APARCH(Gamma1) 0.376495    0.028137       13.38   0.0000
GARCH(Beta1)   0.919863   0.0040708       226.0   0.0000
APARCH(Delta)  1.416169    0.066176       21.40   0.0000
# Rescale:
scale = 1/100; mu = 0.020375; mu*scale
[1] 0.00020375
omega = 0.009991; delta = 1.416169; omega / (1/scale)^(2/delta)
[1] 1.496536e-05
```

The standard errors and t-values from Rmetrics and G@arch which are shown in the above code snippet are calculated from the Hessian matrix. Note, that SPlus prints by default t-values computed from the OPG matrix. To make the results comparable we computed standard errors and t-values from the SPlus output. The list element `fit$cov$A` returns the Hessian matrix. Note, since we have scaled the time series by the scale factor $s = 0.01$, we have to rescale the parameters for the original series in the following way: $\mu = \mu_s s$ and $\omega = \omega_s s^{2/\delta}$. In addition the `summary()` method performs diagnostic checks and the `plot()` creates several diagnostic plots.

	DGE Paper	R Rmetrics	rescaled	Splus Finmetrics	rescaled	Ox G@RCH	rescaled
μ	0.00021	0.02065	0.000207	0.02084	0.000208	0.02038	0.000204
a	0.145	0.1447		0.1447		0.1446	
ω	0.000014	0.009988	0.0000163	0.01003	0.0000159	0.009991	0.0000150
α	0.083	0.08380		0.08375		0.08377	
γ	0.373	0.3731		-0.3710		0.3765	
β	0.920	0.9194		0.9195		0.9199	
δ	1.43	1.435		1.429		1.416	

■ Table 4: Summary and comparison of the parameter estimates as obtained from DGE, Rmetrics, Finmetrics, and SPlus.

Table 4 summarizes and compares the obtained results with those given in the paper of Ding, Granger and Engle.

8. Summary and Outlook

In this paper we have presented and discussed the implementation of S functions for modeling univariate time series processes from the ARMA-APARCH family allowing for (skew) Normal, GED and Student-t conditional distributions. Through the modular concept of the estimation procedure the software can easily be extended to other GARCH and GARCH related models.

The functions listed in Table 5 are part of the R package `fSeries` which is part of the Rmetrics Software environment. Rmetrics is the premier open source solution for financial market analysis and valuation of financial instruments. With hundreds of functions build on modern and powerful methods Rmetrics combines explorative data analysis and statistical modeling with object oriented rapid prototyping. Rmetrics is embedded in R, both building an environment which creates especially for students and researchers in the third world a first class system for applications in statistics and finance. Rmetrics allows you to study all the source code, so you can find out precisely which variation of the algorithm or method has been implemented. This let's you understand what the source code does. Thus Rmetrics can be considered as a unique platform ideally suited for teaching financial engineering and computational finance.

The Rmetrics packages are downloadable from the CRAN server, *cran.r-project.org*, and they are also part of most Debian distributions including for example the Quantian Linux Live CD, *www.quantian.org*. The results presented in this paper were obtained using the R Version 2.2.1 and Rmetrics Version 221.10064. The SPlus version is available through Finance Online GmbH, a Zurich based ETH spin-off company, *www.finance.ch*.

What's next? The software will be extended to further GARCH models including integrated GARCH models, see Engle and Bollerslev [1986], EGARCH models, see Nelson [1991] and Bollerslev and Mikkelsen [1996], GARCH-in-Mean models as well as fractionally integrated GARCH models. In the next version also the fixing of individual ARMA-GARCH/APARCH coefficients will become available. To improve execution times numerically evaluated gradients of the log likelihood function will be replaced by the analytical derivatives. Furthermore, we will provide additional conditional distribution functions including for example members from the family of the hyperbolic distribution, Barndorff-Nielsen [1977]. Concerning perfor-

GARCH Modelling and Utility Functions

Rmetrics' Functions:	fSeries
	GARCH Classes:
garchSpec	S4 Class represenatation for specification object
fGARCH	S4 Class represenatation for GARCH/APRACH models
	Simulation and Parameter Estimation:
garchSpec	Speciifes a GARCH/APARCH model
garchSim	Simulates from a GARCH/APARCH model
garchFit	Fits a GARCH/APARCH model to a univariate time series ARCH, GARCH, ARMA-GARCH, APARCH, ARMA-APARCH Models
garchKappa	Computes Expection for APARCH Models
	Methods:
print	S3: Print method for an object of class fGARCH S3: Print method for an object of class garchSpec
plot	S3: Plot method for an object of class fGARCH
summary	S3: Summary method, diagnostic analysis
predict	S3: Predict method, n-step ahead forecasts
residuals	S3: Residuals method for an object of class fGARCH
fitted	S3: Fitted values method for an object of class fGARCH
	Distribution Functions:
[dpqr]norm	Normal Distribution Function (from Base Installation)
[dpqr]snorm	Skew Normal Distribution
[dpqr]ged	Generalized Error Distribution
[dpqr]sged	Skew Generalized Error Distribution
[dpqr]std	Standardized Student-t Distribution
[dpqr]sstd	Skew standardized Student-t Distribution
	Utility Functions used by Skewed Distributions:
H	Heaviside unit step function
Sign	Just another signum function
	Benchmark Data Set:
dem2gbp	DEM-GBP foreign exchange rates data set
sp500dge	SP500 stock market index data set

■ Table 5: Listing of S functions for analyzing, modeling and forecasting heteroskedastic time series processes from the ARMA-APARCH family.

mance analysis we plan to offer in addition several performance measures for the residuals like Theil's [1967] inequality coefficient, and Mincer and Zarnowitz's [1969] R2 coefficient. Concerning hypothesis testing we will add some specific tests including Engle's [1982] LM ARCH test to test the presence of ARCH effects, Engle and Ng's [1993] test to investigate possible misspecifications of the conditional variance equation, and Nyblom's [1989] test to check the constancy of model parameters over time.

References

[1] Barndorff-Nielsen, O.E., (1977); *Exponentially Decreasing Distributions for the Logarithm of Particle Size*, Proceedings of the Royal Society London A353, 401–419.

[2] Bera A.K., Higgins H.L., (1993); *A Survey of ARCH Models: Properties, Estimation and Testing*, Journal of Economic Surveys 7, 305–366.

[3] Bollerslev T., (1986); *Generalised Autoregressive Conditional Heteroskedasticity*, Journal of Econometrics 31, 307–327.

[4] Bollerslev T., Chou R.Y., Kroner K.F., (1992); *ARCH Modelling in Finance: A Review of the Theory and Empirical Evidence*; Journal of Econometrics 52(5), 5–59.

[5] Bollerslev T., Engle R.F., NelsonD.B., (1994); *ARCH Model*, Handbook of Econometrics Volume IV, Chapter 49, 2961–3031 edited by Engle and McFadden, Elsevier Science.

[6] Bollerslev T., Ghysels E., (1996); *Periodic Autoregressive Conditional Heteroscedasticity*, Journal of Business and Economic Statistics 14, 139–151.

[7] Box G.E.P., Tiao G.C., (1973); *Bayesian Inference in Statistical Analysis*, Addison-Wesley, Publishing Reading, MA.

[8] Brooks R.D., Faff R.W., McKenzie M.D., Mitchell H., (2000); *A Multi-Country Study of Power ARCH Models and National Stock Market Return*, Journal of International Money and Finance 19, 377–397. PDF

[9] Brooks C., Burke S.P., Persand G., (2001); *Benchmarks and the Accuracy of GARCH Model Estimation*, International Journal of Forecasting 17, 45-56. LNK

[10] Byrd R. H., Lu P., Nocedal J., Zhu C., (1995); *A Limited Memory Algorithm for Bound Constrained Optimization*, SIAM Journal of Scientific Computing, 1190–1208.

[11] Dennis J.E., Schnabel R.B., (1983); *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice-Hall, Englewood Cliffs, NJ.

[12] Ding, Z., Granger C.W.J., Engle R.F. (1993); *A Long Memory Property of Stock Market Returns and a New Model*, Journal of Empirical Finance 1, 83–106.

[13] Engle R.F., (1982); *Autoregressive Conditional Heteroscedasticity with Estimates of the Variance of United Kingdom Inflation*, Econometrica 50, 987–1007.

[14] Engle R. F., Bollerslev T., (1986); *Modelling the Persistence of Conditional Variances* Econometric Reviews 5, 1–50.

[15] Engle R.F., Ng V. (1986); *Measuring and Testing the Impact of News on Volatility* Journal of Finance 48, 1749–1778.

[16] Engle R.F., (2001); *GARCH 101: An Introduction to the Use of ARCH/GARCH models in Applied Econometrics*, forthcoming Journal of Economic Perspectives. PDF

[17] Engle R.F., (2002); *New Frontiers for ARCH Models* NYU Preprint, 29 pp. PDF

[18] Fiorentini G., Calzolari G., Panattoni L., (1996); *Analytic derivatives and the computation of GARCH estimates*, Journal of Applied Econometrics 11, 399–417.

[19] Gay D.M., (1983); *ALGORITHM 611 Ů Subroutines for Unconstrained Minimization Using a Model/Trust-Region Approach*, ACM Trans. Math. Software 9, 503–524.

[20] Gay D.M., (1984); *A Trust Region Approach to Linearly Constrained Optimization*, in: Numerical Analysis, Lecture Notes in Mathematics, edited by D.F. Griffiths, Springer Verlag Berlin.

[21] Geweke J. (1986); *Modelling the Persistence of Conditional Variances: A Comment*, Econometric Review 5, 57–61.

[22] Glosten L., Jagannathan R., Runkle D., (1993); *On the Relation Between Expected Value and the Volatility of the Nominal Excess Return on Stocks*, Journal of Finance 48, 1779–1801.

[23] Hansen B., (1994); *Autoregressive Conditional Density Estimation*, International Economic Review 35, 705–730.

[24] Harvey A.C., (1981); *The Econometric Analysis of Time Series*, Oxford.

[25] Higgins M.L., Bera A.K., (1992); *A Class of Nonlinear Arch Models International*, Economic Review 33, 137–158.

[26] Lambert P., Laurent S., (2000); *Modelling Skewness Dynamics in Series of Financial Data*, Discussion Paper, Institut de Statistique, Louvain-la-Neuve. PDF

[27] Lambert P., Laurent S., (2001); *Modelling Financial Time Series Using GARCH-Type Models and a Skewed Student Density*, Mimeo, Universite de Liége.

[28] Laurent S., Lambert. P., (2002); *A Tutorial for GARCH 2.3, a Complete Ox Package for Estimating and Forecasting ARCH Models*, GARCH 2.3 Tutorial, 71 pp. PDF

[29] Laurent S., (2003); *Analytical derivates of the APARCH model*, PrEprint, University of Namur, 8 pp. PDF

[31] Ling S., McAleer M., (2002); *Stationarity and the Existence of Moments of a Family of GARCH processes*, Journal of Econometrics 106, 109–117.

[31] Ling S., McAleer M., (2002); *Necessary and Sufficient Moment Conditions for the GARCH(r,s) and Asymmetric Power GARCH(r,s) Models*, Econometric Theory 18, 722–729.

[32] Lombardi M., Gallo G., (2001); *Analytic Hessian Matrices and the Computation of FI-GARCH Estimates*, Manuscript, Universita degli studi di Firenze. PDF

[33] McCullough B.D., Renfro, C.G., (1999); *Benchmarks and Software Standards: A Case Study of GARCH Procedures*, Journal of Economic and Social Measurement 25, 59–71. PDF

[34] Mincer J., Zarnowitz V., (1969); *The Evaluation of Economic Forecasts*, in: Economic Forecasts and Expectations, edited by J. Mincer, New York, National Bureau of Economic Research.

[35] Nelder J.A., Mead R., (1965); *A Simplex Algorithm for Function Minimization*, Computer Journal 7, 308–313.

[36] Nelson D.B., (1991); *Conditional Heteroscedasticity in Asset Returns: A New Approach*, Econometrica, 59, 347–370.

[37] Nocedal J., Wright S.J., (1999); *Numerical Optimization*, Springer Verlag, Berlin.

[38] Nyblom J., (1989); *Testing the Constancy of Parameters over Time*, Journal of the American Statistical Society 84 223-230.

[39] Pentula S., (1986); *Modelling the Persistence of Conditional Variances: A Comment*, Econometric Review 5, 71–74.

[40] Peters J.P., (2001); *Estimating and Forecasting Volatility of Stock Indices Using Asymmetric GARCH Models and (Skewed) Student-t Densities*, Preprint, University of Liege, Belgium, 20 pp. PDF

[41] Poon S.H., Granger C.W.J., (2001); *Forecasting Financial Market Volatility: A Review*, Manuscript, Department of Economics, UCSD. PDF

[42] Rabemananjara R., Zakoian J.M., (1993); *Threshold Arch Models and Asymmetries in Volatility*, Journal of Applied Econometrics 8, 31–49.

[43] Schnabel R.B., Koontz J.E., Weiss B.E., (1985); *A Modular System of Algorithms for Unconstrained Minimization*, ACM Trans. Mathematical Software 11, 419–440.

[44] Schwert W., (1990); *Stock Volatility and the Crash of Š87*, Review of Financial Studies 3, 77–102.

[45] Taylor S., (1986); *Modelling Financial Time Series*, Wiley, New York.

[46] Theil H., (1967); *Economics and Information Theory*, North Holland, Amsterdam.

[47] Zakoian J.M., (1994); *Threshold Heteroskedasticity Models*, Journal of Economic Dynamics and Control 15, 931–955.

[48] Zhu C., Byrd R.H., Lu P., Nocedal J., (1994); *Algorithm 778: L-BFGS-B: Fortran Subroutines for Large-Scale Bound-Constrained Optimization*, ACM Transactions on Mathematical Software 23, 550–560.

Software Versions:

1st Version and draft: November 2003
2nd Version including forecast: August 2004
3rd Version adding nlminb solver: July 2005
4th Version adding SQP solver: December 2005
this Version: January 2006

The software is downloadable from *www.cran.r-project.org*
A script with all code snippets can be found *www.rmetrics.org*

Affiliation:

Diethelm Würtz
Institut für Theoretische Physik
Eidgenössische Technische Hochschule Zürich 8093 Zürich, Hönggerberg, Switzerland
E-mail: wuertz@itp.phys.ethz.ch
URL: http://www.itp.phys.ethz.ch,http://www.rmetrics.org

Journal of Statistical Software *Submitted:* yyyy-mm-dd
MMMMMM YYYY, Volume VV, Issue II. *Accepted:* yyyy-mm-dd
http://www.jstatsoft.org/