Dealing with Stochastic Volatility in Time Series Using the R Package stochvol

Gregor Kastner
WU Vienna University of Economics and Business

Abstract

The R package **stochvol** provides a fully Bayesian implementation of heteroskedasticity modeling within the framework of stochastic volatility. It utilizes Markov chain Monte Carlo (MCMC) samplers to conduct inference by obtaining draws from the posterior distribution of parameters and latent variables which can then be used for predicting future volatilities. The package can straightforwardly be employed as a stand-alone tool; moreover, it allows for easy incorporation into other MCMC samplers. The main focus of this paper is to show the functionality of **stochvol**. In addition, it provides a brief mathematical description of the model, an overview of the sampling schemes used, and several illustrative examples using exchange rate data.

Keywords: Bayesian inference, Markov chain Monte Carlo (MCMC), heteroskedasticity, SV, GARCH, forecasting, predictive density, predictive Bayes factor, exchange rates.

Preface

This vignette corresponds to an article of the same name which is published in the Journal of Statistical Software. The code base of the vignette has been updated to reflect the changes in recent versions of **stochvol**. To cite, please use **?**. Further information about citing **stochvol** can be obtained in R by installing the package, e.g., through `install.packages("stochvol")`, and calling `citation("stochvol")`.

1. Introduction

Returns – in particular financial returns – are commonly analyzed by estimating and predicting potentially time-varying volatilities. This focus has a long history, dating back at least to **?** who investigates portfolio construction with optimal expected return-variance trade-off. In his article, he proposes rolling-window-type estimates for the instantaneous volatilities, but already then recognizes the potential for "better methods, which take into account more information".

One approach is to model the evolution of volatility deterministically, i.e., through the (G)ARCH class of models. After the groundbreaking papers of **?** and **?**, these models have been generalized in numerous ways and applied to a vast amount of real-world problems. As an alternative, **?** proposes in his seminal work to model the volatility probabilistically, i.e., through a state-space model where the logarithm of the squared volatilities – the latent states

– follow an autoregressive process of order one. Over time, this specification became known as the *stochastic volatility (SV)* model. Even though several papers (e.g., **???**) provide early evidence in favor of using SV, these models have found comparably little use in applied work. This obvious discrepancy is discussed in **?** who points out two reasons: the variety (and potential incompatibility) of estimation methods for SV models – whereas the many variants of the GARCH model have basically a single estimation method – and the lack of standard software packages implementing these methods.

In **?**, the former issue is thoroughly investigated and an efficient MCMC estimation scheme is proposed. The paper at hand and the corresponding package **stochvol** (**?**) for R (**?**) are crafted to cope with the latter problem: the apparent lack of ready-to-use software packages for efficiently estimating SV models.

2. Model specification and estimation

We begin by briefly introducing the model and specifying the notation used in the remainder of the paper. Furthermore, an overview of Bayesian parameter estimation via Markov chain Monte Carlo (MCMC) methods is given.

2.1. The SV model

Let $\boldsymbol{y} = (y_1, y_2, \ldots, y_n)^\top$ be a vector of returns with mean zero. The intrinsic feature of the SV model is that each observation y_t is assumed to have its "own" contemporaneous variance e^{h_t}, thus relaxing the usual assumption of homoskedasticity. In order to make the estimation of such a model feasible, this variance is not allowed to vary unrestrictedly with time. Rather, its logarithm is assumed to follow an autoregressive process of order one. Note that this feature is fundamentally different to GARCH-type models where the time-varying volatility is assumed to follow a deterministic instead of a stochastic evolution.

The SV model can thus be conveniently expressed in hierarchical form. In its centered parameterization, it is given through

$$y_t | h_t \sim \mathcal{N}(0, \exp h_t)\,, \tag{1}$$

$$h_t | h_{t-1}, \mu, \phi, \sigma_\eta \sim \mathcal{N}\Big(\mu + \phi(h_{t-1} - \mu), \sigma_\eta^2\Big)\,, \tag{2}$$

$$h_0 | \mu, \phi, \sigma_\eta \sim \mathcal{N}\Big(\mu, \sigma_\eta^2/(1-\phi^2)\Big)\,, \tag{3}$$

where $\mathcal{N}\Big(\mu, \sigma_\eta^2\Big)$ denotes the normal distribution with mean μ and variance σ_η^2. We refer to $\boldsymbol{\theta} = (\mu, \phi, \sigma_\eta)^\top$ as the vector of *parameters*: the *level* of log-variance μ, the *persistence* of log-variance ϕ, and the *volatility* of log-variance σ_η. The process $\boldsymbol{h} = (h_0, h_1, \ldots, h_n)$ appearing in Equation 2 and Equation 3 is unobserved and usually interpreted as the latent time-varying *volatility process* (more precisely, the log-variance process). Note that the initial state h_0 appearing in Equation 3 is distributed according to the stationary distribution of the autoregressive process of order one.

2.2. Prior distribution

To complete the model setup, a prior distribution for the parameter vector $\boldsymbol{\theta}$ needs to be

specified. Following **?**, we choose independent components for each parameter, i.e., $p(\boldsymbol{\theta}) = p(\mu)p(\phi)p(\sigma_\eta)$.

The level $\mu \in \mathbb{R}$ is equipped with the usual normal prior $\mu \sim \mathcal{N}(b_\mu, B_\mu)$. In practical applications, this prior is usually chosen to be rather uninformative, e.g., through setting $b_\mu = 0$ and $B_\mu \geq 100$ for daily log returns. Our experience with empirical data is that the exact choice is usually not very influential; see also Section 3.2.

For the persistence parameter $\phi \in (-1, 1)$, we choose $(\phi + 1)/2 \sim \mathcal{B}(a_0, b_0)$, implying

$$p(\phi) = \frac{1}{2B(a_0, b_0)} \left(\frac{1+\phi}{2}\right)^{a_0 - 1} \left(\frac{1-\phi}{2}\right)^{b_0 - 1}, \tag{4}$$

where a_0 and b_0 are positive hyperparameters and $B(x, y) = \int_0^1 t^{x-1}(1-t)^{y-1}\, dt$ denotes the beta function. Clearly, the support of this distribution is the interval $(-1, 1)$; thus, stationarity of the autoregressive volatility process is guaranteed. Its expected value and variance are given through the expressions

$$\begin{aligned} E(\phi) &= \frac{2a_0}{a_0 + b_0} - 1, \\ V(\phi) &= \frac{4a_0 b_0}{(a_0 + b_0)^2 (a_0 + b_0 + 1)}. \end{aligned}$$

This obviously implies that the prior expectation of ϕ depends only on the ratio $a_0 : b_0$. It is greater than zero if and only if $a_0 > b_0$ and smaller than zero if and only if $a_0 < b_0$. For a fixed ratio $a_0 : b_0$, the prior variance decreases with larger values of a_0 and b_0. The uniform distribution on $(-1, 1)$ arises as a special case when $a_0 = b_0 = 1$. For financial datasets with not too many observations (i.e., $n \lesssim 1000$), the choice of the hyperparameters a_0 and b_0 can be quite influential on the shape of the posterior distribution of ϕ. In fact, when the underlying data-generating process is (near-)homoskedastic, the volatility of log-variance σ_η is (very close to) zero and thus the likelihood contains little to no information about ϕ. Consequently, the posterior distribution of ϕ is (almost) equal to its prior, no matter how many data points are observed. For some discussion about this issue, see, e.g., **?** who choose $a_0 = 20$ and $b_0 = 1.5$, implying a prior mean of 0.86 with a prior standard deviation of 0.11 and thus very little mass for nonpositive values of ϕ.

For the volatility of log-variance $\sigma_\eta \in \mathbb{R}^+$, we choose $\sigma_\eta^2 \sim B_{\sigma_\eta} \times \chi_1^2 = \mathcal{G}(1/2, 1/2B_{\sigma_\eta})$. This choice is motivated by **?** who equivalently stipulate the prior for $\pm\sqrt{\sigma_\eta^2}$ to follow a centered normal distribution, i.e., $\pm\sqrt{\sigma_\eta^2} \sim \mathcal{N}(0, B_{\sigma_\eta})$. As opposed to the more common Inverse-Gamma prior for σ_η^2, this prior is not conjugate in the usual sampling scheme. However, it does not bound σ_η^2 away from zero a priori. The choice of the hyperparameter B_{σ_η} turns out to be of minor influence in empirical applications as long as it is not set too small.

2.3. MCMC sampling

An MCMC algorithm such as the one implemented in the package **stochvol** provides its user with draws from the posterior distribution of the desired random variables; in our case with the latent log-variances $\boldsymbol{h}$ and the parameter vector $\boldsymbol{\theta}$. Because these draws are usually dependent, Bayesian inference via MCMC may require careful design of the algorithm and attentive investigation of the draws obtained.

One key feature of the algorithm used in this package is the joint sampling of all instantaneous volatilities "all without a loop" (AWOL), a technique going back at least to **?** and discussed in more detail in **?**. Doing so reduces correlation of the draws significantly and requires auxiliary finite mixture approximation of the errors as in **?** or **?**.

In order to avoid the cost of code interpretation within each MCMC iteration, the core computations are implemented in C. Their output is interfaced to R via the **Rcpp** package (**?**); there, the convenience functions and the user interface are implemented. This combination allows to make use of the well-established and widely accepted ease-of-use of R and its underlying functional programming paradigm. Moreover, existing frameworks for analyzing MCMC output such as **coda** (**?**) as well as high-level visualization tools can easily be used. Last but not least, users with a basic knowledge of R can use the package with a very low entry cost. Nevertheless, despite all these convenience features, the package profits from a highly optimized machine code generated by a compiler at package build time, thus providing acceptable runtime even for larger datasets.

A novel and crucial feature of the algorithm implemented in **stochvol** is the usage of a variant of the "ancillarity-sufficiency interweaving strategy" (ASIS) which has been brought forward in the general context of state-space models by **?**. ASIS exploits the fact that for certain parameter constellations sampling efficiency improves substantially when considering a non-centered version of a state-space model. This is commonly referred to as a reparameterization issue and dealt with extensively in scholarly literature; for an early reference see, e.g., **?**. For the model at hand, a move of this kind can be achieved by transferring the level of log-variance μ and/or the volatility of log-variance σ_η from the state process (Equations 2 and 3) to the observation process (Equation 1) through a simple reparameterization of $\boldsymbol{h}$. However, in the case of the SV model, it turns out that no single superior parameterization exists. Rather, for some underlying processes, the standard parameterization yields superior results, while for other processes non-centered versions are better. To overcome this issue, the parameter vector $\boldsymbol{\theta}$ is sampled twice: once in the centered and once in a noncentered parameterization. This method of "combining best of different worlds" allows for efficient inference regardless of the underlying process with one algorithm. For more details about the algorithm and empirical results concerning sampling efficiency, see **?**.

3. The stochvol package

The usual stand-alone approach to fitting SV models with **stochvol** exhibits the following workflow: (1) Prepare the data, (2) specify the prior distributions and configuration parameters, (3) run the sampler, (4) assess the output and display the results. All these steps are described in more detail below, along with a worked example. For a stepwise incorporation of SV effects into other MCMC samplers, see Section 4.

3.1. Preparing the data

The core sampling function `svsample` expects its input data `y` to be a numeric vector of returns without any missing values (`NA`s) and throws an error if provided with anything else. In the case that `y` contains zeros, a warning is issued and a small offset constant of size `sd(y)/10000` is added to the squared returns before doing the auxiliary mixture sampling (cf. **?**). However, we generally recommend to avoid zero returns altogether, e.g., by demeaning

them beforehand.

Below is an illustration of how to prepare data by using the sample dataset `exrates`[1] included in the package. Figure 1 provides a visualization of a time series from this dataset.

```
R> set.seed(123)
R> library("stochvol")
R> data("exrates")
R> ret <- logret(exrates$USD, demean = TRUE)
R> par(mfrow = c(2, 1), mar = c(1.9, 1.9, 1.9, 0.5), mgp = c(2, 0.6, 0))
R> plot(exrates$date, exrates$USD, type = "l",
+    main = "Price of 1 EUR in USD")
R> plot(exrates$date[-1], ret, type = "l", main = "Demeaned log returns")
```

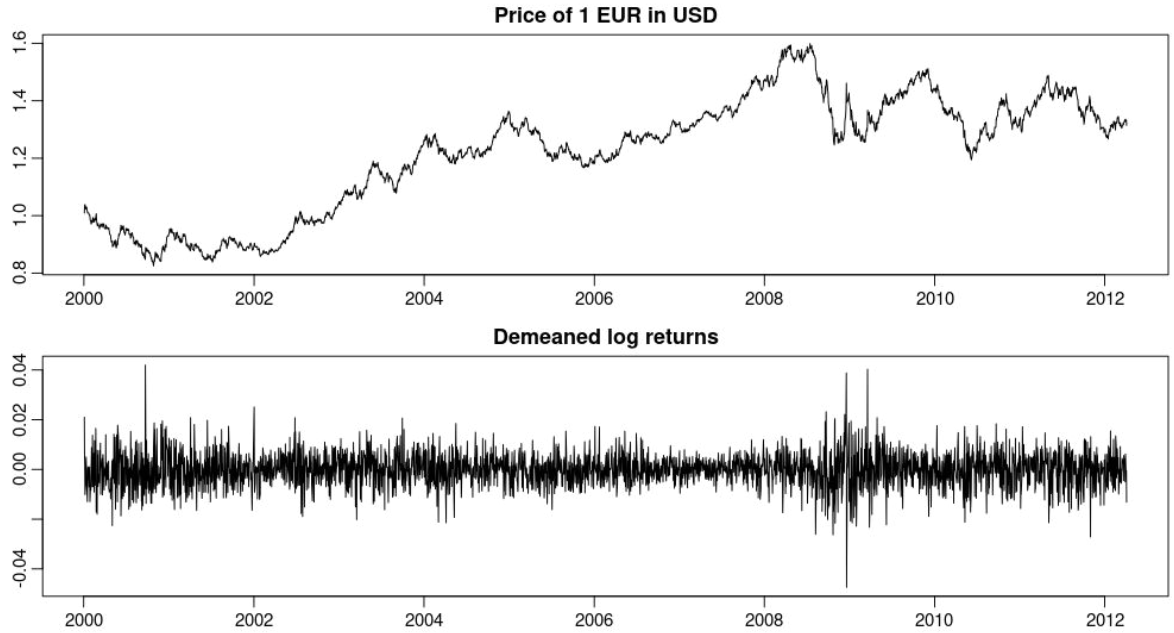

Figure 1: Visualization of EUR-USD exchange rates included in the **stochvol** package.

Additionally to real-world data, **stochvol** also has a built-in data generator `svsim`. This function simply produces realizations of an SV process and returns an object of class `svsim` which has its own `print`, `summary`, and `plot` methods. Exemplary code using `svsim` is given below and the particular instance of this simulated series is displayed in Figure 2.

```
R> sim <- svsim(500, mu = -9, phi = 0.99, sigma = 0.1)
R> par(mfrow = c(2, 1))
R> plot(sim)
```

[1]The dataset – obtained from the European Central Bank's Statistical Data Warehouse – contains the daily bilateral prices of one euro in 23 currencies from January 3, 2000 until April 4, 2012. Conversions to New Turkish lira and Fourth Romanian leu have been incorporated. See `?exrates` for more information.

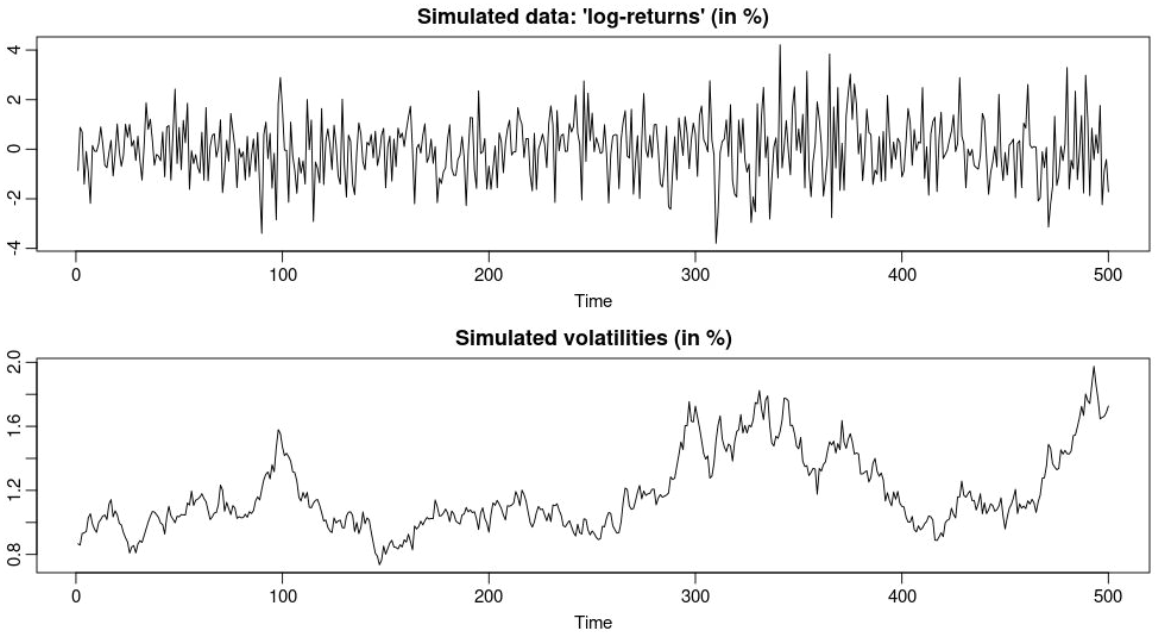

Figure 2: Visualization of a simulated time series as provided by the default `plot` method.

3.2. Specifying prior distributions and configuration parameters

After preparing the data vector `y`, the user needs to specify the prior hyperparameters for the parameter vector $\boldsymbol{\theta} = (\mu, \phi, \sigma_\eta)^\top$ – see also Section 2.2 – and some configuration parameters. The appropriate values are passed to the main sampling function `svsample` as arguments which are described below.

The argument `priormu` is a vector of length 2, containing mean and standard deviation of the normal prior for the level of the log-variance μ. A common strategy is to choose a vague prior here, e.g., `c(0, 100)`, because the likelihood usually carries enough information about this parameter. If one prefers to use (slightly) informative priors, e.g., to avoid outlier draws of μ, one must pay attention to whether *log returns* or *percentage log returns* are analyzed. Assuming daily data, log returns commonly have an unconditional variance of 0.0001 or less and thus the level on the log scale μ lies around $\log(0.0001) \approx -9$. Percentage log returns, on the other hand, have the 100^2-fold unconditional variance (around 1) which implies a level of $\log(1) = 0$. Choices in the literature include `c(0, 10)` (`?`), `c(0, 5)` (`?`), `c(0, sqrt(10))` (`??`) or `c(0, 1)` (`?`). Note that most of these choices are quite informative and clearly designed for percentage log returns.

For specifying the prior hyperparameters for the persistence of log-variance, ϕ, the argument `priorphi` may be used. It is again a vector of length 2, containing a_0 and b_0 specified in Equation 4. As elaborated in Section 2.2, these values can possibly be quite influential, thus we advise to choose them carefully and study the effects of different choices. The default is currently given through `c(5, 1.5)`, implying a prior mean of 0.54 and a prior standard deviation of 0.31.

The prior variance of log-variance hyperparameter B_{σ_η} may be controlled through `priorsigma`.

This argument defaults to `1` if not provided by the user. As discussed in Section 2.2, the exact choice of this value is usually not very influential in typical applications. In general, it should not be set too small unless there is a very good reason, e.g., explicit prior knowledge, to do so.

For specifying the size of the burn-in, the parameter `burnin` is provided. It is the amount of MCMC iterations that are run but discarded to ensure convergence to the stationary distribution of the chain. The current default value for this parameter is 1000 which has turned out to suffice in most situations. Nevertheless, the user is encouraged to check convergence carefully; see Section 3.4 for more details. The amount of iterations which are run after burn-in can be specified through the parameter `draws`, currently defaulting to 10 000. Consequently, the sampler is run for a total of `burnin + draws` iterations.

Three thinning parameters are available which all are 1 if not specified otherwise. The first one, `thinpara`, is the denominator in the fraction of parameter draws (i.e., draws of $\boldsymbol{\theta}$) that are stored. E.g., if `thinpara` equals 10, every 10th draw is kept. The default parameter thinning value of 1 means that all draws are saved. The second thinning parameter, `thinlatent`, acts in the same way for the latent variables $\boldsymbol{h}$. The third thinning parameter, `thintime`, refers to thinning with respect to the time dimension of the latent volatility. In the case that `thintime` is greater than 1, not all elements of $\boldsymbol{h}$ are stored, e.g., for `thintime` equaling 10, only the draws of $h_1, h_{11}, h_{21}, \ldots$ (and h_0) are kept.

Another configuration argument is `quiet` which defaults to `FALSE`. If set to `TRUE`, all output during sampling (progress bar, status messages) is omitted. The arguments `startpara` and `startlatent` are optional starting values for the parameter vector $\boldsymbol{\theta}$ and the latent variables $\boldsymbol{h}$, respectively. All other configuration parameters are summarized in the argument `expert`, because it is not very likely that the end-user needs to mess with the defaults.[2] Please refer to the package documentation and `?` for details.

Any further arguments (`...`) are forwarded to `updatesummary`, controlling the type of summary statistics that are calculated for the posterior draws.

3.3. Running the sampler

At the heart of the package **stochvol** lies the function `svsample` which serves as an R-wrapper for the actual sampler coded in C. Exemplary usage of this function is given in the code snipped below, along with the default output.

```
R> res <- svsample(ret, priormu = c(-10, 1), priorphi = c(20, 1.1),
+    priorsigma = 0.1)

Calling GIS_C MCMC sampler with 11000 iter. Series length is 3139.

  0% [++++++++++++++++++++++++++++++++++++++++++++++++++] 100%
```

[2]Examples of configurations that can be changed with the `expert`-argument include the specification of the (baseline) parameterization (either *centered* or *noncentered*) and the possibility to turn off interweaving. Moreover, some algorithmic details such as the number of blocks used for the parameter updates or the possibility of using a random walk Metropolis-Hastings proposal (instead of the default independence proposal) can be found here.

```
Timing (elapsed): 12.92 seconds.
851 iterations per second.

Converting results to coda objects... Done!
Summarizing posterior draws... Done!
```

As can be seen, this function calls the main MCMC sampler and converts its output to **coda**-compatible objects. The latter is done mainly for reasons of compatibility and in order to have straightforward access to the convergence diagnostics checks implemented there. Moreover, some summary statistics for the posterior draws are calculated. The return value of `svsample` is an object of type `svdraws` which is a named list with eight elements, holding (1) the parameter draws in `para`, (2) the latent log-volatilities in `latent`, (3) the initial latent log-volatility draw in `latent0`, (4) the data provided in `y`, (5) the sampling runtime in `runtime`, (6) the prior hyperparameters in `priors`, (7) the thinning values in `thinning`, and (8) summary statistics of these draws, alongside some common transformations thereof, in `summary`.

3.4. Assessing the output and displaying the results

Following common practice, `print` and `summary` methods are available for `svdraws` objects. Each of these has two optional parameters, `showpara` and `showlatent`, specifying which output should be displayed. If `showpara` is `TRUE` (the default), values/summaries of the parameter draws are shown. If `showlatent` is `TRUE` (the default), values/summaries of the latent variable draws are shown. In the example below, the summary for the parameter draws only is displayed.

```
R> summary(res, showlatent = FALSE)


Summary of 'svdraws' object

Prior distributions:
mu        ~ Normal(mean = -10, sd = 1)
(phi+1)/2 ~ Beta(a = 20, b = 1.1)
sigma^2   ~ Gamma(shape = 0.5, rate = 5)
nu        ~ Infinity
rho       ~ Constant(value = 0)

Stored 10000 MCMC draws after a burn-in of 1000.
No thinning.

Posterior draws of SV parameters (thinning = 1):
              mean      sd       5%      50%     95%  ESS
mu         -10.1287 0.22603 -10.4673 -10.1324 -9.7819 4283
phi          0.9937 0.00278   0.9887   0.9939  0.9978  284
sigma        0.0651 0.01014   0.0495   0.0646  0.0831  143
exp(mu/2)    0.0064 0.00075   0.0053   0.0063  0.0075 4283
sigma^2      0.0043 0.00138   0.0025   0.0042  0.0069  143
```

There are several plotting functions specifically designed for objects of class `svsample` which are described in the following paragraphs.

(1) `volplot`: Plots posterior quantiles of the latent volatilities in percent, i.e., empirical quantiles of the posterior distribution of $100 \exp(h_t/2)$, over time. Apart from the mandatory `svsample`-object itself, this function takes several optional arguments. Only some are mentioned here; for an exhaustive list please see the corresponding help document accessible through `?volplot` or `help(volplot)`. Selected optional arguments that are commonly used include `forecast` for n-step-ahead volatility prediction, `dates` for labels on the x-axis, alongside some graphical parameters. The code snipped below shows a typical example and Figure 3 displays its output.

```
R> volplot(res, forecast = 100, dates = exrates$date[-1])
```

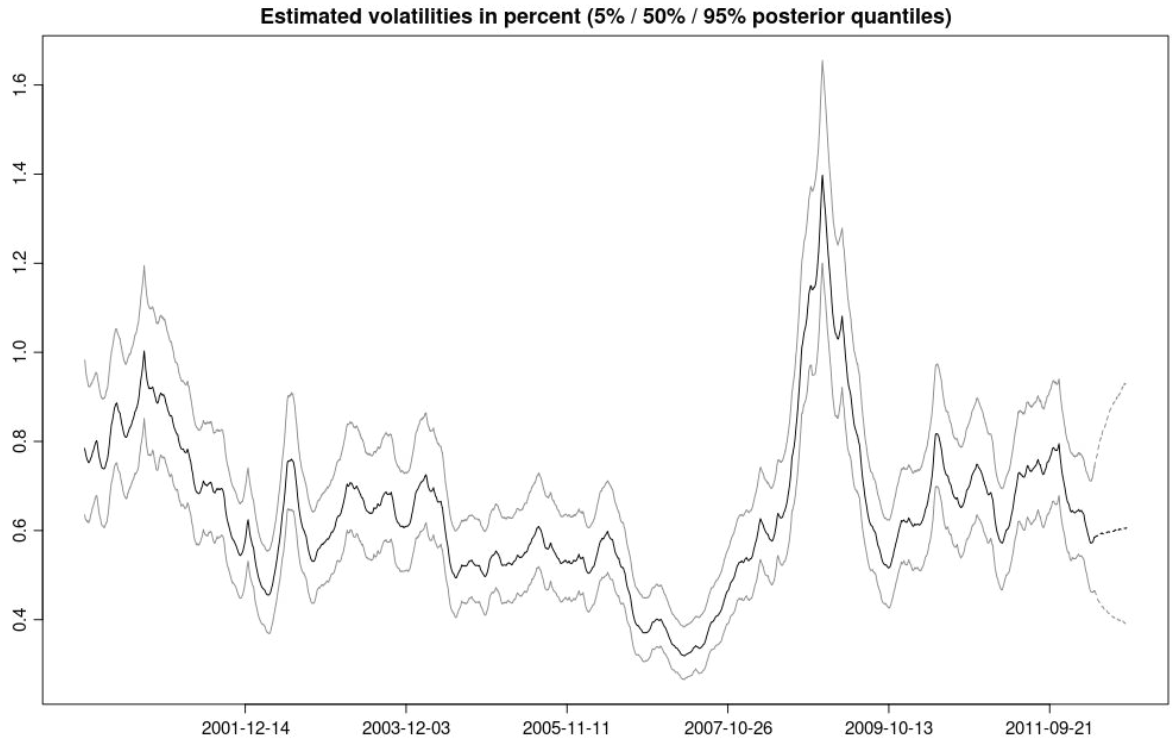

Figure 3: Visualization of estimated contemporaneous volatilities of EUR-USD exchange rates, as provided by `volplot`. If not specified otherwise, posterior medians and 5%/95% quantiles are plotted. The dotted lines on the right indicate predicted future volatilities.

In case the user wants to display different posterior quantiles, the `updatesummary` function has to be called first. See the code below for an example and Figure 4 for the corresponding plot.

```
R> res <- updatesummary(res, quantiles = c(0.01, 0.1, 0.5, 0.9, 0.99))
R> volplot(res, forecast = 100, dates = exrates$date[-1])
```

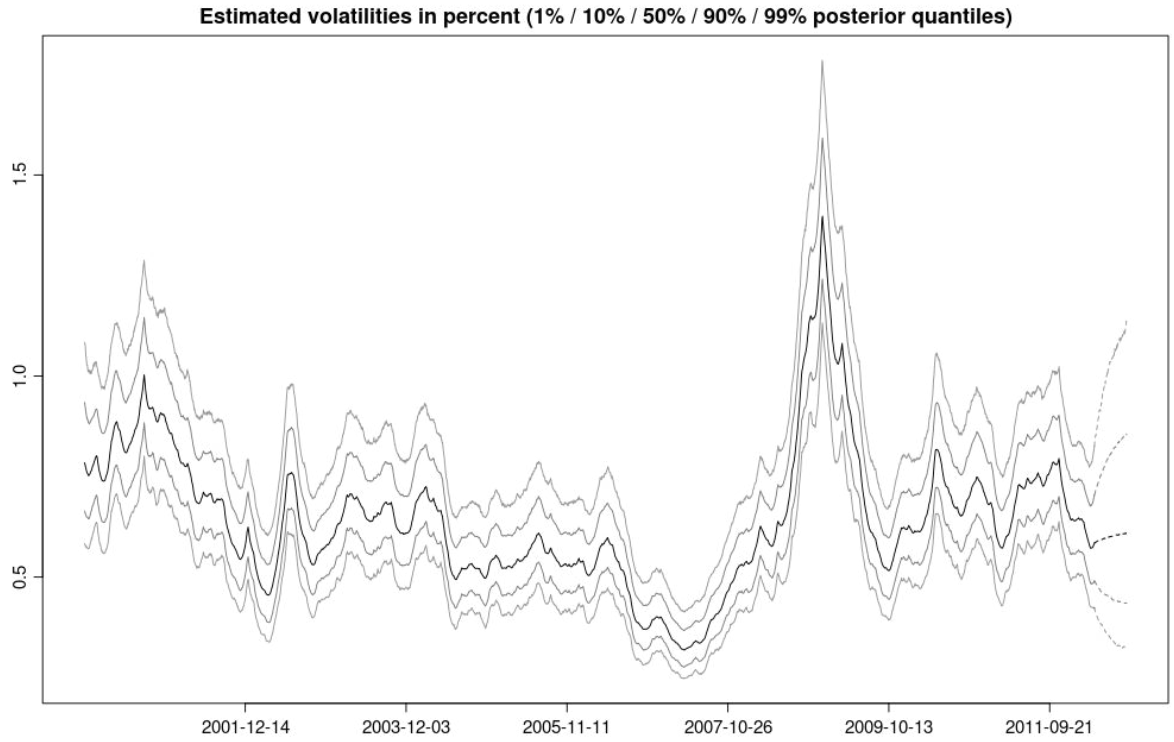

Figure 4: As above, now with medians (black line) and 1%/10%/90%/99% quantiles (gray lines). This behavior can be achieved through a preceding call of `updatesummary`.

(2) `paratraceplot`: Displays trace plots for the parameters contained in $\boldsymbol{\theta}$. Note that the burn-in has already been discarded. Figure 5 shows an example.

```
R> par(mfrow = c(3, 1))
R> paratraceplot(res)
```

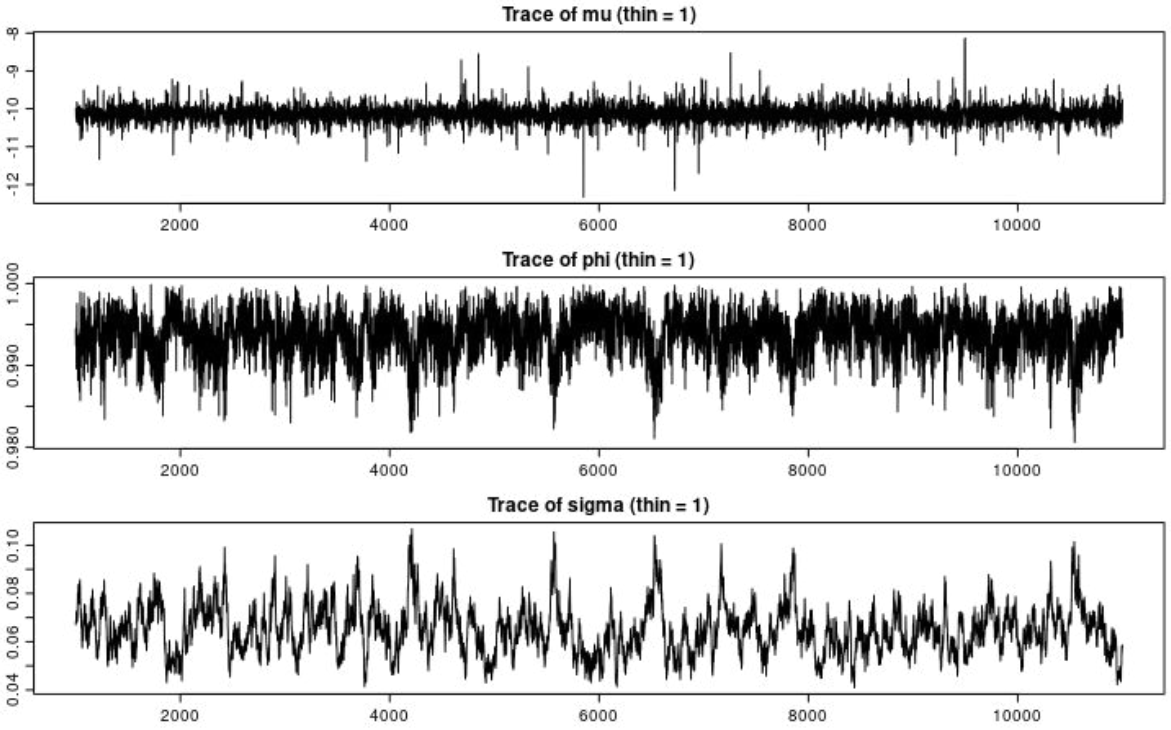

Figure 5: Trace plots of posterior draws for the parameters μ, ϕ, σ_η.

(3) `paradensplot`: Displays a kernel density estimate for the parameters contained in $\boldsymbol{\theta}$. If the argument `showobs` is `TRUE` (which is the default), individual posterior draws are indicated through a rug, i.e., short vertical lines along the x-axis. For quicker drawing of large posterior samples, this argument should be set to `FALSE`. If the argument `showprior` is `TRUE` (which is the default), the prior distribution is indicated through a dashed gray line. Figure 6 shows a sample output for the EUR-USD exchange rates obtained from the `exrates` dataset.

```
R> par(mfrow = c(1, 3))
R> paradensplot(res, showobs = FALSE)
```

The generic `plot` method for `svdraws` objects combines all above plots into one plot. All arguments described above can be used. See `?plot.svsample` for an exhaustive summary of possible arguments and Figure 7 for an example.

```
R> plot(res, showobs = FALSE)
```

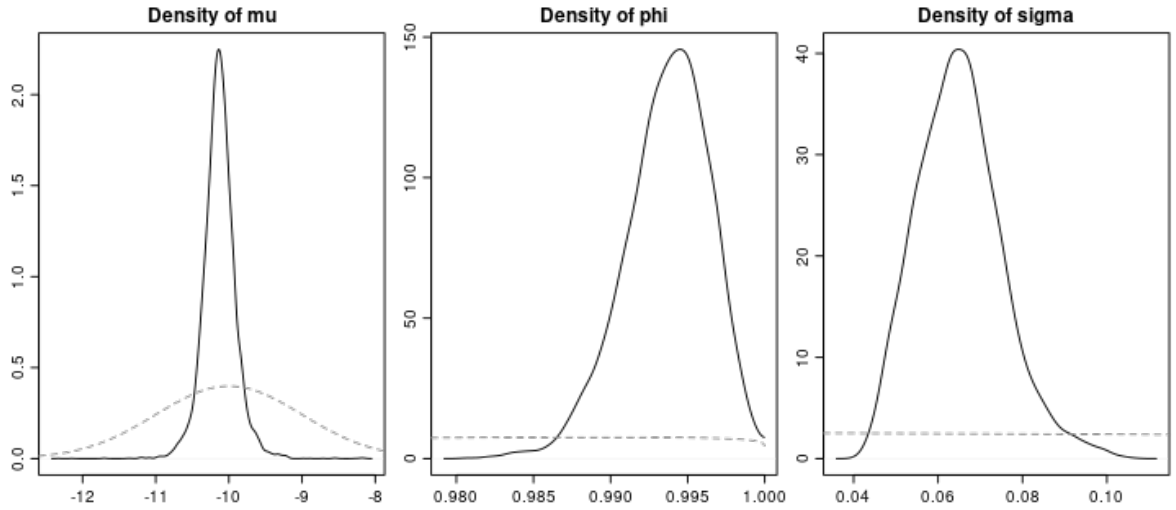

Figure 6: Posterior density estimates (black solid lines) along with prior densities (dashed gray lines). Individual posterior draws are indicated by the underlying rug.

For extracting standardized residuals, the `residuals`/`resid` method can be used on a given `svdraws` object. With the optional argument `type`, the type of summary statistic may be specified. Currently, `type` is allowed to be either `"mean"` or `"median"`, where the former corresponds to the default value. This method returns a real vector of class `svresid` which contains the requested summary statistic of standardized residuals for each point in time. There is also a `plot` method available, providing the option of comparing the standardized residuals to the original data when given through the argument `origdata`. See the code below for an example and Figure 8 for the corresponding output.

```
R> myresid <- resid(res)
R> plot(myresid, ret)
```

4. Using stochvol within other samplers

We demonstrate how the **stochvol** package can be used to incorporate stochastic volatility into any given MCMC sampler. This is particularly easy when the sampler itself is coded in R or C/C++ and applies, e.g., to many of the specialized procedures listed in the CRAN task view about Bayesian inference, available at http://cran.r-project.org/web/views/Bayesian.html. For the sake of simplicity, we explain the procedure using a "hand-coded" Gibbs-sampler for the Bayesian normal linear model with n observations and $k = p - 1$ predictors, given through

$$\boldsymbol{y}|\boldsymbol{\beta}, \boldsymbol{\Sigma} \sim \mathcal{N}(\boldsymbol{X}\boldsymbol{\beta}, \boldsymbol{\Sigma})\,. \tag{5}$$

Here, $\boldsymbol{y}$ denotes the $n \times 1$ vector of responses, $\boldsymbol{X}$ is the $n \times p$ design matrix containing ones in the first column and the predictors in the others, and $\boldsymbol{\beta} = (\beta_0, \beta_1, \ldots, \beta_{p-1})^\top$ stands for the

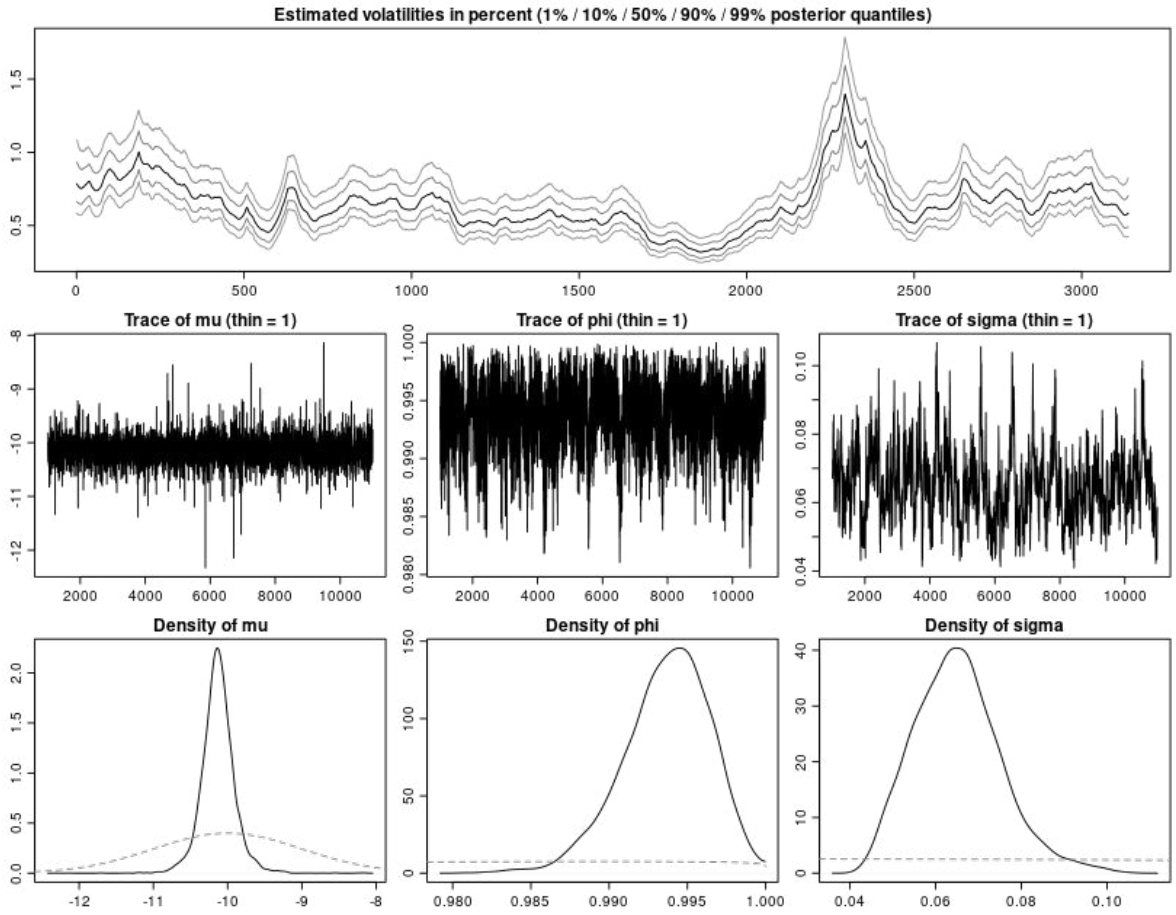

Figure 7: Illustration of the default `plot` method for `svdraws`-objects. This visualization combines `volplot` (Figure 4), `traceplot` (Figure 5), and `paradensplot` (Figure 6) into one single plot.

$p \times 1$ vector of regression coefficients. In the following sections, we discuss two specifications of the $n \times n$ error covariance matrix $\boldsymbol{\Sigma}$.

4.1. The Bayesian normal linear model with homoskedastic errors

The arguably simplest specification of the error covariance matrix in Equation 5 is given by $\boldsymbol{\Sigma} \equiv \sigma_\epsilon^2 \boldsymbol{I}$, where $\boldsymbol{I}$ denotes the n-dimensional unit matrix. This specification is used in many applications and commonly referred to as the linear regression model with homoskedastic errors. To keep things simple, let model parameters $\boldsymbol{\beta}$ and σ_ϵ^2 be equipped with the usual conjugate prior $p(\boldsymbol{\beta}, \sigma_\epsilon^2) = p(\boldsymbol{\beta}|\sigma_\epsilon^2)p(\sigma_\epsilon^2)$, where

$$\begin{aligned} \boldsymbol{\beta}|\sigma_\epsilon^2 &\sim \mathcal{N}\left(\boldsymbol{b}_0, \sigma_\epsilon^2 \boldsymbol{B}_0\right), \\ \sigma_\epsilon^2 &\sim \mathcal{G}^{-1}(c_0, C_0). \end{aligned}$$

Commonly, samples from the posterior distribution of this model are obtained through a Gibbs-algorithm, where draws are generated in turn from the full conditional distributions

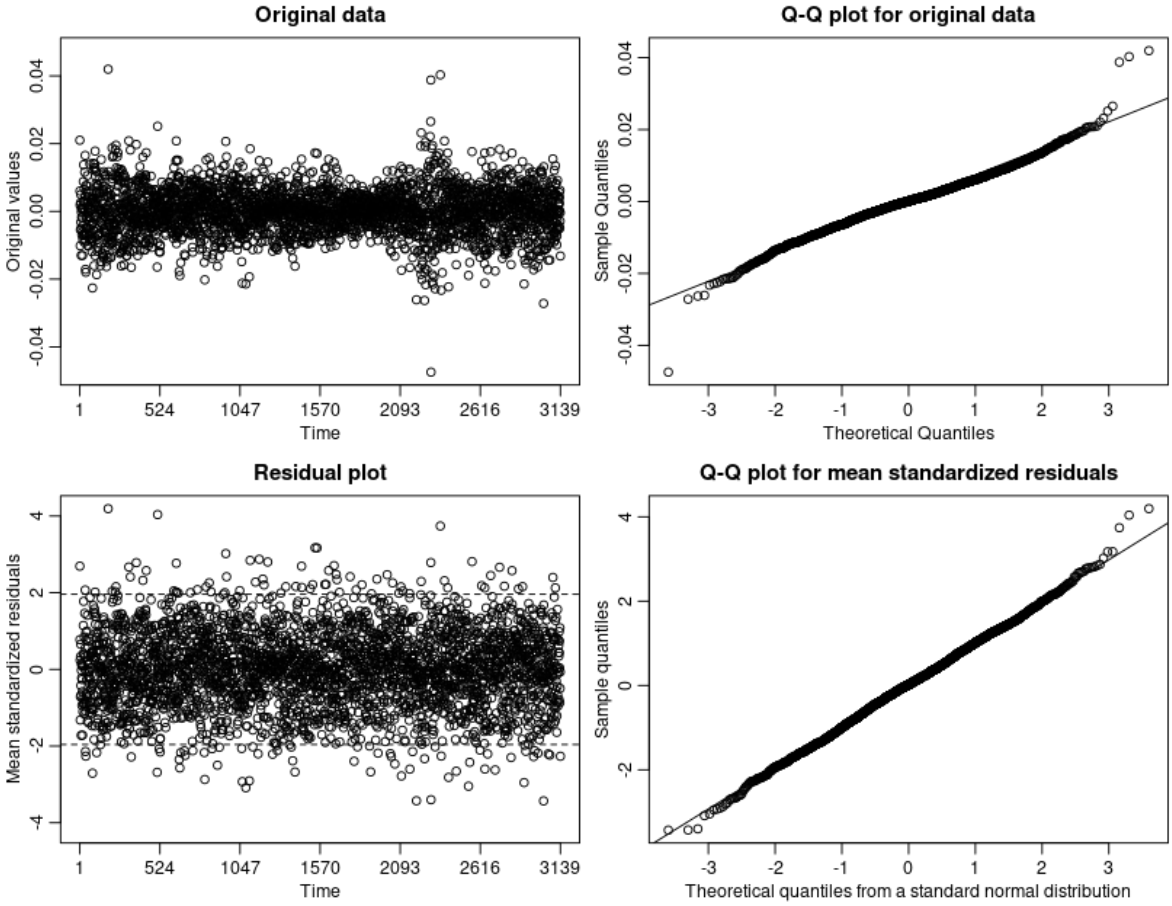

Figure 8: Mean standardized residual plots for assessing the model fit, as provided by the corresponding `plot` method. The dashed lines in the bottom left panel indicate the 2.5%/97.5% quantiles of the standard normal distribution.

$\boldsymbol{\beta}|\boldsymbol{y}, \sigma_\epsilon^2 \sim \mathcal{N}(\boldsymbol{b}_T, \boldsymbol{B}_T)$ with

$$\boldsymbol{b}_T = \left(\boldsymbol{X}^\top \boldsymbol{X} + \boldsymbol{B}_0^{-1}\right)^{-1} \left(\boldsymbol{X}^\top \boldsymbol{y} + \boldsymbol{B}_0^{-1} \boldsymbol{b}_0\right), \qquad \boldsymbol{B}_T = \sigma_\epsilon^2 \left(\boldsymbol{X}^\top \boldsymbol{X} + \boldsymbol{B}_0^{-1}\right)^{-1},$$

and $\sigma_\epsilon^2|\boldsymbol{y}, \boldsymbol{\beta} \sim \mathcal{G}^{-1}(c_n, C_n)$ with

$$c_n = c_0 + \frac{n}{2} + \frac{p}{2}, \qquad C_n = C_0 + \frac{1}{2}\left((\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^\top (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}) + (\boldsymbol{\beta} - \boldsymbol{b}_0)^\top \boldsymbol{B}_0^{-1} (\boldsymbol{\beta} - \boldsymbol{b}_0)\right).$$

In R, this can straightforwardly be coded as follows:

- Set the seed to make results reproducible and simulate some data from the underlying model:

```
R> set.seed(123456)
R> n <- 1000
```

```
R> beta.true <- c(0.1, 0.5)
R> sigma.true <- 0.01
R> X <- matrix(c(rep(1, n), rnorm(n, sd = sigma.true)), nrow = n)
R> y <- rnorm(n, X %*% beta.true, sigma.true)
```

- Specify the size of the burn-in and the number of draws thereafter; set the prior parameters:

```
R> burnin <- 100
R> draws <- 5000
R> b0 <- matrix(c(0, 0), nrow = ncol(X))
R> B0inv <- diag(c(10^-10, 10^-10))
R> c0 <- 0.001
R> C0 <- 0.001
```

- Pre-calculate some values outside the main MCMC loop:

```
R> p <- ncol(X)
R> preCov <- solve(crossprod(X) + B0inv)
R> preMean <- preCov %*% (crossprod(X, y) + B0inv %*% b0)
R> preDf <- c0 + n/2 + p/2
```

- Assign some storage space for holding the draws and set an initial value for σ_ϵ^2:

```
R> draws1 <- matrix(NA_real_, nrow = draws, ncol = p + 1)
R> colnames(draws1) <- c(paste("beta", 0:(p-1), sep = "_"), "sigma")
R> sigma2draw <- 1
```

- Run the main sampler: Iteratively draw from the conditional bivariate Gaussian distribution $\boldsymbol{\beta}|\boldsymbol{y}, \sigma_\epsilon^2$, e.g., through the use of **mvtnorm** (**?**), and the conditional Inverse Gamma distribution $\sigma_\epsilon^2|\boldsymbol{y}, \boldsymbol{\beta}$.

```
R> for (i in -(burnin-1):draws) {
+    betadraw <- as.numeric(mvtnorm::rmvnorm(1, preMean,
+      sigma2draw * preCov))
+    tmp <- C0 + 0.5 * (crossprod(y - X %*% betadraw) +
+      crossprod((betadraw - b0), B0inv) %*% (betadraw - b0))
+    sigma2draw <- 1 / rgamma(1, preDf, rate = tmp)
+    if (i > 0) draws1[i, ] <- c(betadraw, sqrt(sigma2draw))
+  }
```

- Calculate posterior means in order to obtain point estimates for the parameters:

```
R> colMeans(draws1)
```

```
    beta_0     beta_1      sigma
0.09991649 0.50472433 0.01027775
```

- Visualize the draws through **coda**'s native `plot` method:

```
R> par(mar = c(3.1, 1.8, 1.9, .5), mgp = c(1.8, .6, 0))
R> plot(coda::mcmc(draws1[seq(1L, nrow(draws1), by = plotevery),]))

R> plot(coda::mcmc(draws1))
```

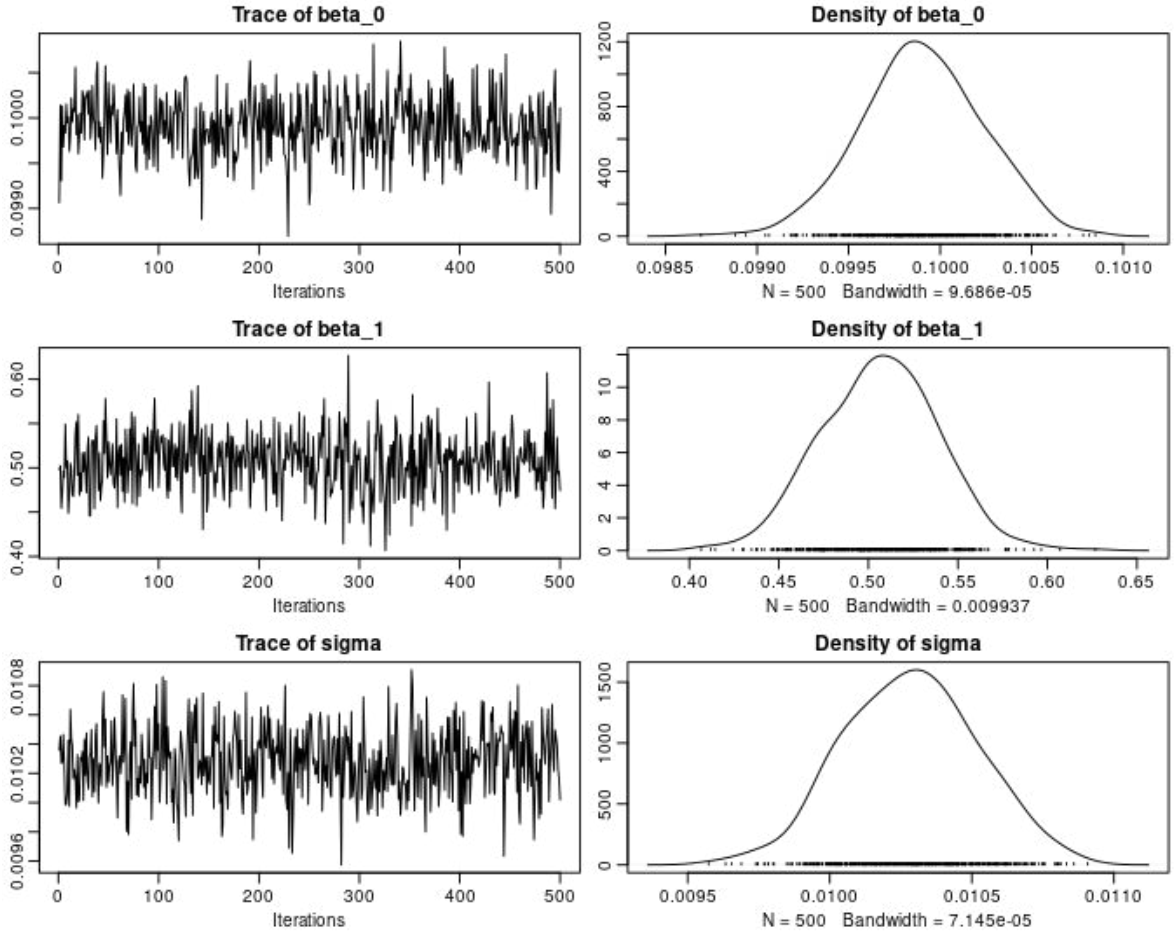

Figure 9: Trace plots and kernel density estimates for some draws from the marginal posterior distributions in the regression model with heteroskedastic errors. Underlying data is simulated with $\boldsymbol{\beta}^{\text{true}} = (0.1, 0.5)^\top$, $\sigma_\epsilon^{\text{true}} = 0.01$, $n = 1000$.

4.2. The Bayesian normal linear model with SV errors

Instead of homoskedastic errors, we now specify the error covariance matrix in Equation 5 to be $\boldsymbol{\Sigma} \equiv \text{diag}(e^{h_1}, \ldots, e^{h_n})$, thus introducing nonlinear dependence between the observations due to the AR(1)-nature of $\boldsymbol{h}$. Instead of cooking up an entire new sampler, we adapt the code from above utilizing the **stochvol** package.[3] To do so, we simply replace the sampling

[3]Since **stochvol** version 3.0, regression is a built-in feature of the function `svsample_fast_cpp` through its argument `designmatrix`.

step of σ_ϵ^2 from an Inverse-Gamma distribution with a sampling step of $\boldsymbol{\theta}$ and $\boldsymbol{h}$ through a call to `svsample_fast_cpp`.[4] This function is a minimal-overhead version of the regular `svsample`. It provides the full sampling functionality of the original version but has slightly different default values, a simplified return value structure, and it does not perform costly input checks. Thus, it executes faster and is more suited for repeated calls. The drawback is that it needs to be used with proper care.[5] Note that the current draws of the variables need to be passed to the function through `startpara` and `startlatent`.

Here is how it goes:

- Simulate some data:

```
R> mu.true <- log(sigma.true^2)
R> phi.true <- 0.97
R> vv.true <- 0.3
R> simresid <- svsim(n, mu = mu.true, phi = phi.true, sigma = vv.true)
R> y <- X %*% beta.true + simresid$y
```

- Specify configuration parameters and prior values:

```
R> draws <- 50000
R> burnin <- 1000
R> thinning <- 10
R> priors <- specify_priors(
+    mu = sv_normal(-10, 2),
+    phi = sv_beta(20, 1.5),
+    sigma2 = sv_gamma(0.5, 0.5)
+  )
```

- Assign some storage space for holding the draws and set initial values:

```
R> draws2 <- matrix(NA_real_, nrow = floor(draws / thinning),
+    ncol = 3 + n + p)
R> colnames(draws2) <- c("mu", "phi", "sigma",
+    paste("beta", 0:(p-1), sep = "_"), paste("h", 1:n, sep = "_"))
R> betadraw <- c(0, 0)
R> paradraw <- list(mu = -10, phi = 0.9, sigma = 0.2)
R> latentdraw <- rep(-10, n)
R> paranames <- names(paradraw)
```

- Run the main sampler, i.e., iteratively draw
  - the latent volatilities/parameters by conditioning on the regression parameters and calling `svsample_fast_cpp`,
  - the regression parameters by conditioning on the latent volatilities and calling `rmvnorm`:

[4] In earlier version of the **stochvol** package, this function was called `svsample2`.

[5] Erroneous or incompatible input values most likely result in run-time errors of compiled code, often implying a segmentation fault and the consecutive abnormal termination of R. This can render debugging tedious.

```
R> for (i in -(burnin-1):draws) {
+    ytilde <- y - X %*% betadraw
+    svdraw <- svsample_fast_cpp(ytilde, startpara = paradraw,
+      startlatent = latentdraw, priorspec = priors)
+    paradraw <- svdraw$para
+    latentdraw <- drop(svdraw$latent)
+    normalizer <- as.numeric(exp(-latentdraw / 2))
+    Xnew <- X * normalizer
+    ynew <- y * normalizer
+    Sigma <- solve(crossprod(Xnew) + B0inv)
+    mu <- Sigma %*% (crossprod(Xnew, ynew) + B0inv %*% b0)
+    betadraw <- as.numeric(mvtnorm::rmvnorm(1, mu, Sigma))
+    if (i > 0 && i %% thinning == 0) {
+      draws2[i/thinning, 1:3] <- drop(paradraw)[paranames]
+      draws2[i/thinning, 4:5] <- betadraw
+      draws2[i/thinning, 6:(n+5)] <- latentdraw
+    }
+  }
```

- Finally, visualize and summarize (some) posterior draws:

```
R> plot(coda::mcmc(draws2[, 4:7]))
```

```
R> colMeans(draws2[, 4:8])
```

```
R> colMeans(draws2selection)
```

```
    beta_0     beta_1        h_1        h_2        h_3
 0.1000300  0.4935052 -8.2097012 -8.4570505 -8.6609265
```

It should be noted that even though `svsample_fast_cpp` is considerably faster than `svsample`, the cost of interpreting this function in each MCMC iteration is still rather high (which lies in the nature of R as an interpreted language). Thus, as of package version `0.8-0`, a single-step `update` is also made available at the C/C++ level for samplers coded there. For details, please consult the `NEWS` file in the package source; for an application case using this approach, see `?`.

5. Illustrative predictive exercise

In the following, we compare the performance of the Bayesian normal linear model with homoskedastic errors from Section 4.1 with the Bayesian normal linear model with SV errors from Section 4.2 using the `exrates` dataset introduced in Section 3.1. As a benchmark, we also include the Bayesian normal linear model with GARCH(1,1) errors given through Equation 5 with

$$\begin{aligned} \mathbf{\Sigma} &= \text{diag}(\sigma_1^2, \ldots, \sigma_n^2), \\ \sigma_t^2 &= \alpha_0 + \alpha_1 \tilde{y}_{t-1}^2 + \alpha_2 \sigma_{t-1}^2, \end{aligned}$$

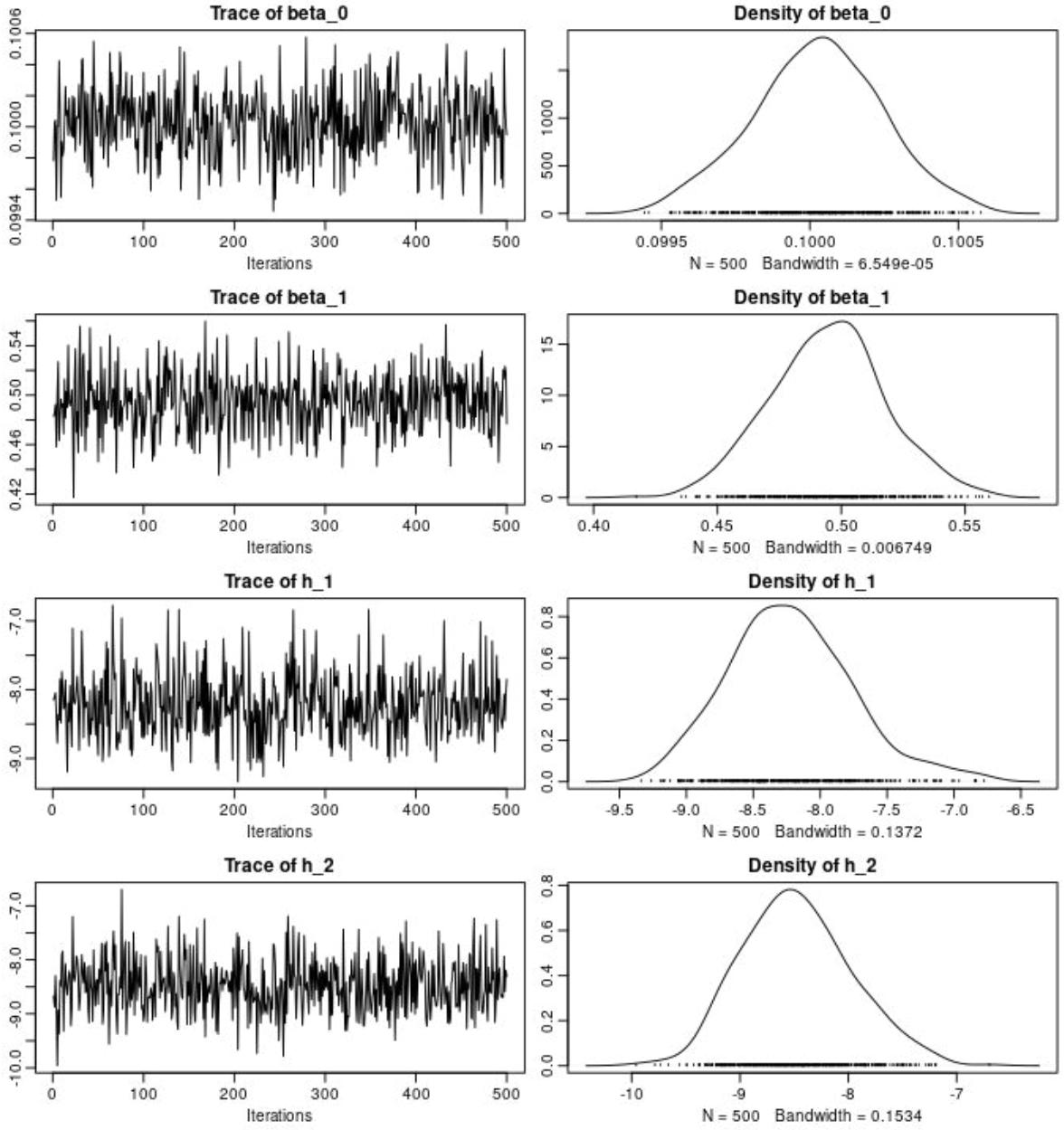

Figure 10: Trace plots and kernel density estimates in the regression model with heteroskedastic errors. Data is simulated with $\boldsymbol{\beta}^{\text{true}} = (0.1, 0.5)^\top$, $h_1^{\text{true}} = -8.28$, $h_2^{\text{true}} = -8.50$, $n = 1000$.

where time index $t = 1, \ldots, n$. In the second equation, $\tilde{y}_{t-1}$ denotes the past "residual", i.e., the $(t-1)$th element of $\tilde{\boldsymbol{y}} = \boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}$.

5.1. Model setup

We again use the daily price of 1 EUR in USD from January 3, 2000 to April 4, 2012, denoted by $\boldsymbol{p} = (p_1, p_2, \ldots, p_n)^\top$. This time however, instead of modeling log returns, we investigate

the development of log levels by regression. Let $\boldsymbol{y}$ contain the logarithm of all observations except the very first one, and let $\boldsymbol{X}$ denote the design matrix containing ones in the first column and lagged log prices in the second, i.e.,

$$\boldsymbol{y} = \begin{pmatrix} \log p_2 \\ \log p_3 \\ \vdots \\ \log p_n \end{pmatrix}, \quad \boldsymbol{X} = \begin{pmatrix} 1 & \log p_1 \\ 1 & \log p_2 \\ \vdots & \vdots \\ 1 & \log p_{n-1} \end{pmatrix}.$$

Note that this specification simply corresponds to an AR(1) model for the log prices with three different error specifications: homoskedastic, SV, GARCH(1,1). It is a generalization of directly dealing with demeaned log returns where β_0 is a priori assumed to be equal to the mean of the log returns and β_1 is held constant at 1.

```
R> x <- log(exrates$USD[-length(exrates$USD)])
R> y <- log(exrates$USD[-1])
R> X <- matrix(c(rep(1, length(x)), x), nrow = length(x))
R> par(mfrow=c(1,1), mar = c(2.9, 2.9, 2.7, .5), mgp = c(1.8,.6,0), tcl = -.4)
R> plot(x,y, xlab=expression(log(p[t])), ylab=expression(log(p[t+1])),
+      main="Scatterplot of lagged daily log prices of 1 EUR in USD",
+      col="#00000022", pch=16, cex=1)
R> abline(0,1)
```

Irrespectively of the error specification, we expect the posterior distribution of $\boldsymbol{\beta}$ to spread around $(0, 1)^\top$ which corresponds to a random walk. A scatterplot of $\log p_t$ against $\log p_{t+1}$, displayed in Figure 11, confirms this.

5.2. Posterior inference

To obtain draws from the posterior distributions for the models with homoskedastic/SV errors, samplers developed in Section 4 are used. For the GARCH-errors, a simple random walk Metropolis-Hastings sampler is employed. We run each of the three samplers for 100 000 iterations and discard the first 10 000 draws as burn-in. Starting values and tuning parameters are set using maximum likelihood estimates obtained through the R package **fGarch** (**?**).

Aiming for comparable results with minimal prior impact on predictive performance, the hyperparameters are chosen to yield vague priors: $\boldsymbol{b}_0 = (0, 0)^\top$, $\boldsymbol{B}_0 = \mathrm{diag}(10^{10}, 10^{10})$, $c_0 = C_0 = 0.001$, $b_\mu = 0$, $B_\mu = 10^4$, $a_0 = 1$, $b_0 = 1$, $B_{\sigma_\eta} = 1$. For the GARCH(1,1) parameter vector $\boldsymbol{\alpha} = (\alpha_0, \alpha_1, \alpha_2)^\top$, we pick independent flat priors on $\mathbb{R}^+$ for the components; the initial variance σ_0^2 is fixed to the empirical residual variance and $\tilde{y}_0$ is assumed to be zero. Due to the length of the dataset (and its obvious heteroskedasticity), none of these specific choices seem to particularly influence the following analysis. Note, however, that for shorter series or series with less pronounced heteroskedasticity, sensible prior choices are of great importance as the likelihood carries less information in these cases.

The samplers yield slightly different posteriors for $\boldsymbol{\beta}$, visualized in Figure 12. In the top panels, the estimated marginal posterior densities $p(\beta_0|\boldsymbol{y})$ and $p(\beta_1|\boldsymbol{y})$ are displayed; the bottom panel depicts a scatterplot of draws from the joint posterior $\boldsymbol{\beta}|\boldsymbol{y}$.

To assess the model fit, mean standardized residuals are depicted in Figure 13. The model with homoskedastic errors shows deficiencies in terms of pronounced dependence amongst the

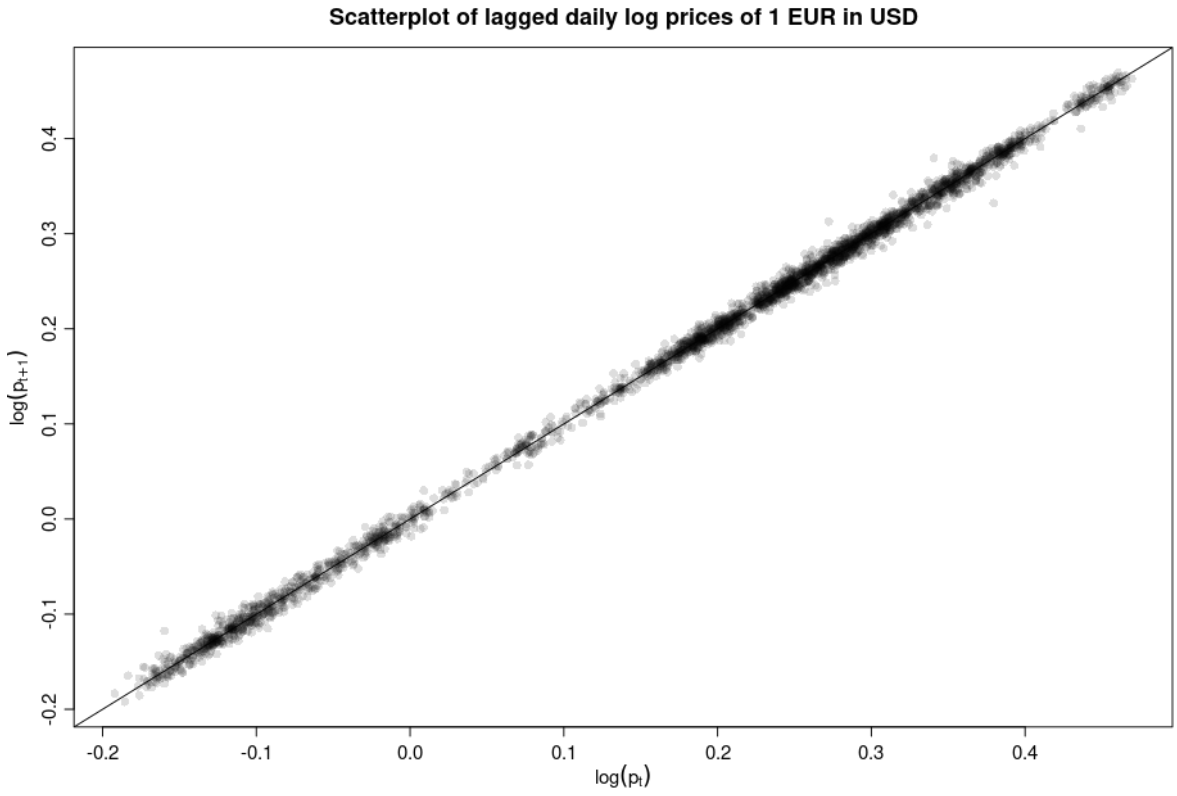

Figure 11: Scatterplot of daily log prices at time t against daily log prices at time $t+1$. The solid line indicates the identity function $f(x) = x$.

residuals. This can clearly be seen in the top left panel, where mean standardized residuals are plotted against time. The middle left panel shows the same plot for the model with GARCH errors where this effect is greatly diminished. The bottom left panel pictures that plot for the model with SV errors; here, this effect practically vanishes. Moreover, in the model with homoskedastic errors, the normality assumption about the unconditional error distribution is clearly violated. This can be seen by inspecting the quantile-quantile plot in the top right panel, where observed residuals exhibit much heavier tails than one would expect from a normal distribution. The model with GARCH errors provides a better fit, however, heavy tails are still visible. Standardized residuals from the model with SV errors appear to be approximately normal with only few potential outliers.

5.3. Measuring predictive performance and model fit

Within a Bayesian framework, a natural way of assessing the predictive performance of a given model is through its *predictive density* (sometimes also referred to as *posterior predictive distribution*). It is given through

$$p(y_{t+1}|\boldsymbol{y}^o_{[1:t]}) = \int_{\boldsymbol{K}} p(y_{t+1}|\boldsymbol{y}^o_{[1:t]}, \boldsymbol{\kappa}) \times p(\boldsymbol{\kappa}|\boldsymbol{y}^o_{[1:t]})\, \mathrm{d}\boldsymbol{\kappa}, \tag{6}$$

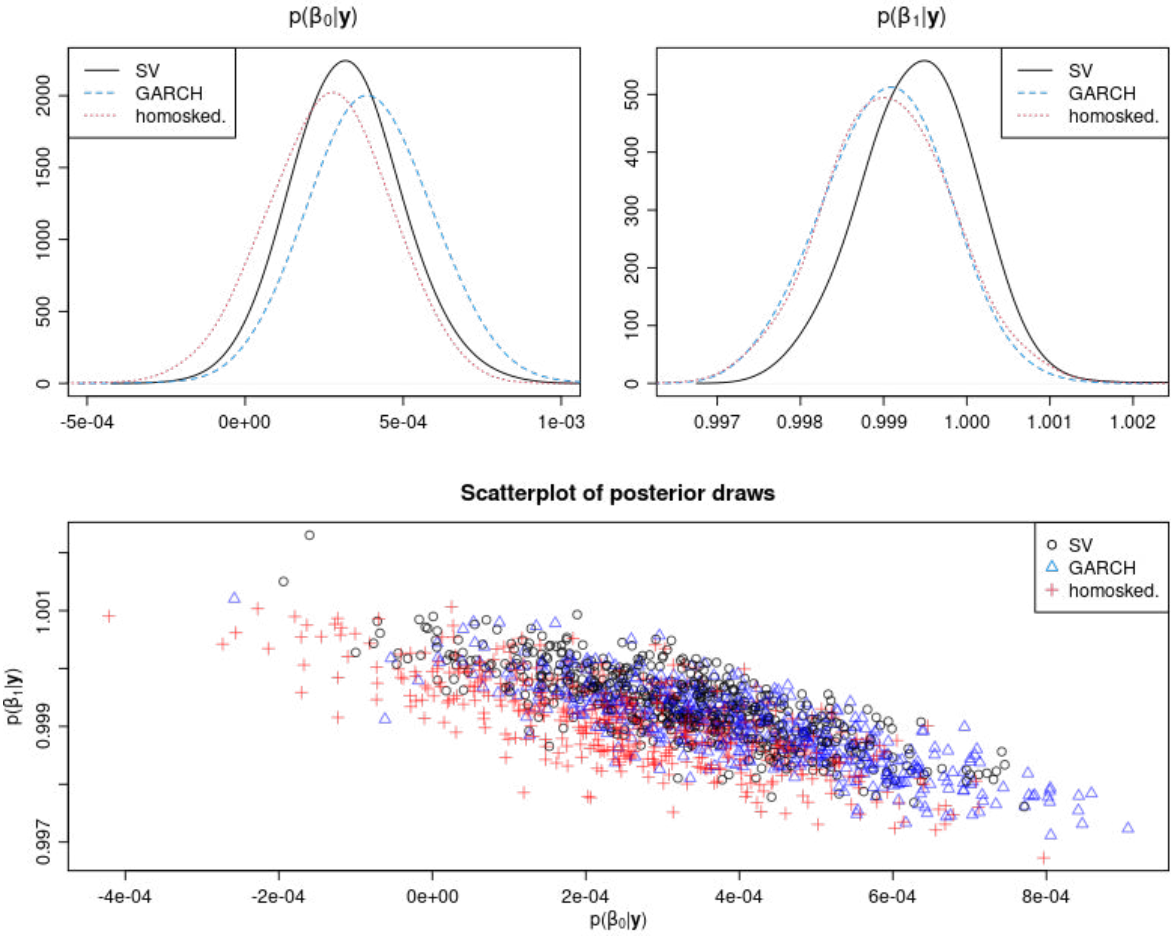

Figure 12: Visualization of the posterior distributions $\boldsymbol{\beta}|\boldsymbol{y}$ for the model with SV regression residuals, the model with GARCH(1,1) regression residuals, and the model with homoskedastic regression residuals. Top panels: Kernel density estimates of the univariate posterior marginal distributions. Bottom panel: Bivariate scatterplot of posterior draws.

where $\boldsymbol{\kappa}$ denotes the vector of all unobservables, i.e., parameters and possible latent variables. Note that for the model with homoskedastic errors, $\boldsymbol{\kappa} = (\boldsymbol{\beta}, \sigma_\epsilon)^\top$; for the model with SV errors, $\boldsymbol{\kappa} = (\boldsymbol{\beta}, \boldsymbol{\theta}, \boldsymbol{h})^\top$; in the GARCH(1,1) case, $\boldsymbol{\kappa} = (\boldsymbol{\beta}, \boldsymbol{\alpha}, \sigma_0^2, \tilde{y}_0^2)^\top$. By using the superscript o in $\boldsymbol{y}^o_{[1:t]}$, we follow **?** and denote *ex post* realizations (observations) for the set of points in time $\{1, 2, \ldots, t\}$ of the *ex ante* random values $\boldsymbol{y}_{[1:t]} = (y_1, y_2, \ldots, y_t)^\top$. Integration is carried out over $\boldsymbol{K}$ which simply stands for the space of all possible values for $\boldsymbol{\kappa}$. Equation 6 can be viewed as the integral of the likelihood function over the joint posterior distribution of the unobservables $\boldsymbol{\kappa}$. Thus, it can be interpreted as the predictive density for a future value y_{t+1} after integrating out the uncertainty about $\boldsymbol{\kappa}$, conditional on the history $\boldsymbol{y}^o_{[1:t]}$.

In the SV errors case, Equation 6 is a $(n+p+3)$-dimensional integral which cannot be solved analytically. Nevertheless, it may be evaluated at an arbitrary point x through Monte Carlo

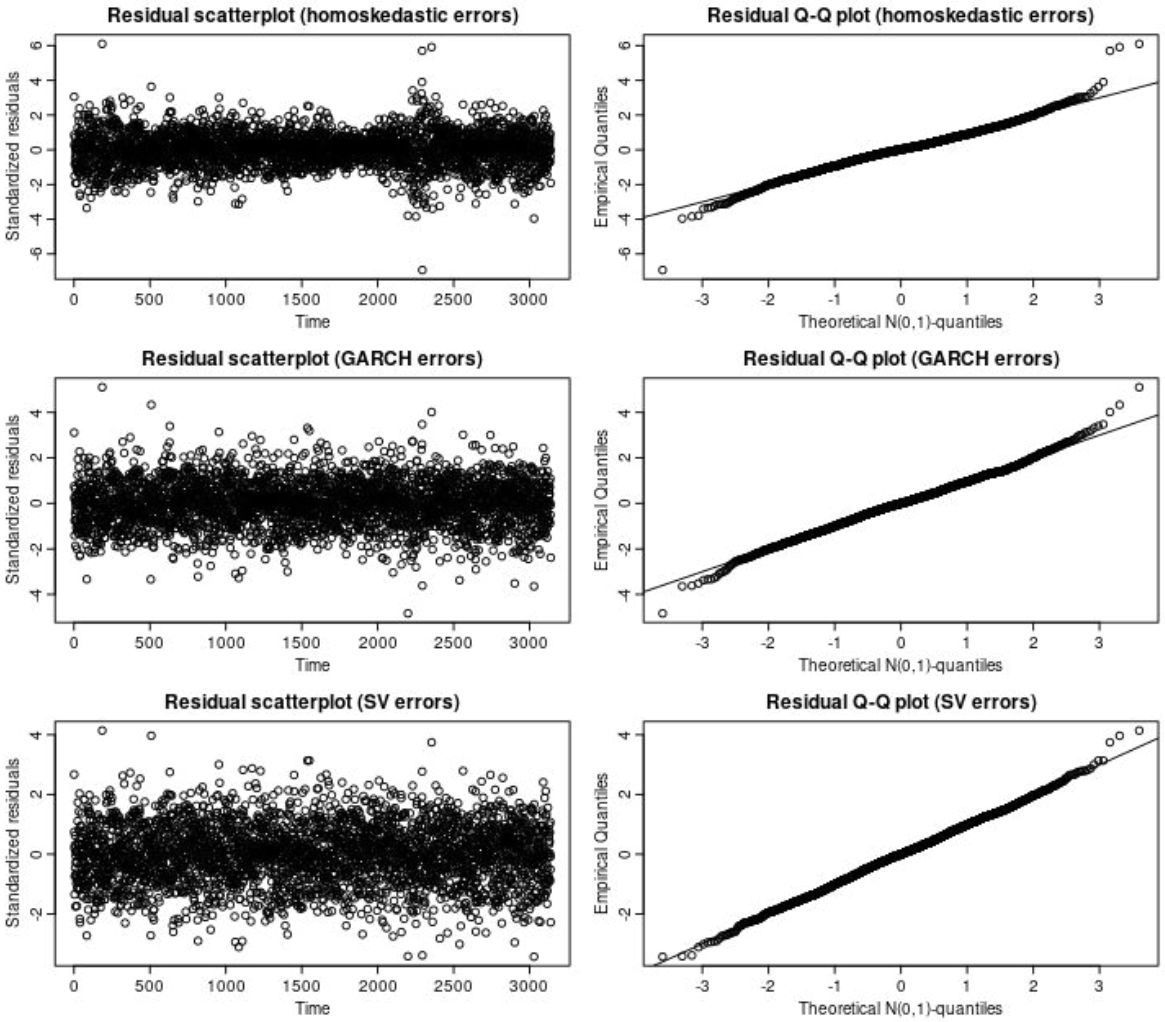

Figure 13: Visualization of mean standardized residuals. Top left panel shows a scatterplot against time for the model with homoskedastic errors, middle left panel shows this plot for the model with GARCH errors, and bottom left panel shows this plot for the model with SV errors. Quantile-Quantile plots of empirical quantiles against expected quantiles from a $\mathcal{N}(0,1)$-distribution are displayed on the panels on the right-hand side.

integration,

$$p(x|\boldsymbol{y}^o_{[1:t]}) \approx \frac{1}{M}\sum_{m=1}^{M} p(x|\boldsymbol{y}^o_{[1:t]}, \boldsymbol{\kappa}^{(m)}_{[1:t]}), \tag{7}$$

where $\boldsymbol{\kappa}^{(m)}_{[1:t]}$ stands for the m^{th} draw from the respective posterior distribution up to time t. If Equation 7 is evaluated at $x = y^o_{t+1}$, we refer to it as the (one-step-ahead) *predictive likelihood* at time $t+1$, denoted PL_{t+1}. Moreover, draws from the posterior predictive distribution can be obtained by simulating values $y^{(m)}_{t+1}$ from the distribution given through the density $p(y_{t+1}|\boldsymbol{y}^o_{[1:t]}, \boldsymbol{\kappa}^{(m)}_{[1:t]})$, the summands of Equation 7.

For the model at hand, the predictive density and likelihood can thus be computed through the following

Algorithm 1 (Predictive density and likelihood evaluation at time $t+1$)

1. *Reduce the dataset to a training set* $\boldsymbol{y}^o_{[1:t]} = (y^o_1, \ldots, y^o_t)^\top$.
2. *Run the posterior sampler using data from the training set only to obtain* M *posterior draws* $\boldsymbol{\kappa}^{(m)}_{[1:t]}$, $m = 1, \ldots, M$.

(3a.) Needed for the SV model only: Simulate M *values from the conditional distribution* $h_{t+1,[1:t]}|\boldsymbol{y}^o_{[1:t]}, \boldsymbol{\kappa}_{[1:t]}$ *by drawing* $h^{(m)}_{t+1,[1:t]}$ *from a normal distribution with mean* $\mu^{(m)}_{[1:t]} + \phi^{(m)}_{[1:t]}(h^{(m)}_{t,[1:t]} - \mu^{(m)}_{[1:t]})$ *and standard deviation* $\sigma^{(m)}_{\eta,[1:t]}$ *for* $m = 1, \ldots, M$.

(3b.) Needed for the GARCH model only: Obtain M *draws from the conditional distribution* $\sigma_{t+1,[1:t]}|\boldsymbol{y}^o_{[1:t]}, \boldsymbol{\kappa}_{[1:t]}$ *by computing* $\sigma^{(m)}_{t+1,[1:t]} = \sqrt{\alpha^{(m)}_{0,[1:t]} + \alpha^{(m)}_{1,[1:t]}\,(\tilde{y}^o_t)^2 + \alpha^{(m)}_{2,[1:t]}\left(\sigma^{(m)}_{t,[1:t]}\right)^2}$ *for* $m = 1, \ldots, M$.

4a. To obtain PL_{t+1}*, average over* M *densities of normal distributions with mean* $(1, y^o_t) \times \boldsymbol{\beta}^{(m)}_{[1:t]}$ *and standard deviation* $\exp\{h^{(m)}_{t+1,[1:t]}/2\}$ *(SV model),* $\sigma^{(m)}_{t+1,[1:t]}$ *(GARCH model), or* $\sigma^{(m)}_{\epsilon,[1:t]}$ *(homoskedastic model), each evaluated at* y^o_{t+1}*, for* $m = 1, \ldots, M$.

4b. To obtain M *draws from the predictive distribution, simulate from a normal distribution with mean* $(1, y^o_t) \times \boldsymbol{\beta}^{(m)}_{[1:t]}$ *and standard deviation* $\exp\{h^{(m)}_{t+1,[1:t]}/2\}$ *(SV model),* $\sigma^{(m)}_{t+1,[1:t]}$ *(GARCH model), or* $\sigma^{(m)}_{\epsilon,[1:t]}$ *(homoskedastic model) for* $m = 1, \ldots, M$.

It is worth pointing out that log predictive likelihoods also carry an intrinsic connection to the log *marginal likelihood*, defined through

$$\log ML = \log p(\boldsymbol{y}^o) = \log \int_{\boldsymbol{K}} p(\boldsymbol{y}^o|\boldsymbol{\kappa}) \times p(\boldsymbol{\kappa})\,\mathrm{d}\boldsymbol{\kappa}.$$

This real number corresponds to the logarithm of the normalizing constant which appears in the denominator of Bayes' law and is often used for evaluating model evidence. It can straightforwardly be decomposed into the sum of the logarithms of the one-step-ahead predictive likelihoods:

$$\log ML = \log p(\boldsymbol{y}^o) = \log \prod_{t=1}^{n} p(y^o_t|\boldsymbol{y}^o_{[1:t-1]}) = \sum_{t=1}^{n} \log PL_t.$$

Thus, Algorithm 1 provides a conceptually simple way of computing the marginal likelihood. However, these computations are quite costly in terms of CPU time, as they require an individual model fit for each of the n points in time. On the other hand, due to the embarrassingly parallel nature of the task and because of today's comparably easy access to distributed computing environments, this burden becomes manageable. E.g., the computations for the stochastic volatility analysis in this paper have been conducted in less than one hour, using

25 IBM dx360M3 nodes within a cluster of workstations. Implementation in R was achieved through the packages **parallel** (**?**) and **snow** (**?**).

Cumulative sums of log PL_t also allow for model comparison through cumulative log predictive Bayes factors. Letting $PL_t(A)$ denote the predictive likelihood of model A at time t, and $PL_t(B)$ the corresponding value of model B, the cumulative log predictive Bayes factor at time u (and starting point s) in favor of model A over model B is straightforwardly given through

$$\log\left[\frac{p_A(\boldsymbol{y}^o_{[s+1:u]}|\boldsymbol{y}^o_{[1:s]})}{p_B(\boldsymbol{y}^o_{[s+1:u]}|\boldsymbol{y}^o_{[1:s]})}\right] = \sum_{t=s+1}^{u} \log\left[\frac{PL_t(A)}{PL_t(B)}\right] = \sum_{t=s+1}^{u} [\log PL_t(A) - \log PL_t(B)]. \quad (8)$$

When the cumulative log predictive Bayes factor is positive at a given point in time, there is evidence in favor of model A, and vice versa. In this context, information up to time s is regarded as prior information, while out-of-sample predictive evaluation starts at time $s+1$. Note that the usual (overall) log Bayes factor is a special case of Equation 8 for $s = 0$ and $u = n$.

5.4. Results

In order to reduce prior influence, the first 1000 days are used as a training set only and the evaluation of the predictive distribution starts at $t = 1001$, corresponding to December 4, 2003. The homoskedastic model is compared to the model with SV residuals in Figure 14. In the top panel, the observed series along with the 98% one-day-ahead predictive intervals are displayed. The bottom panel shows the log one-day-ahead predictive likelihood. According to these values, SV errors can handle the inflated volatility during that time substantially better. In the course of 2009, the width of the intervals as well as the predictive likelihoods consolidate again. Figure 14 also contains a close-up of the one-year time span from September 2008 to August 2009. Both credible intervals are similar at the beginning and at the end of this interval. However, there is a substantial difference in early 2009, where SV intervals become around twice as large compared to the corresponding homoskedastic analogs.

A visually barely distinguishable picture emerges when GARCH(1,1) errors are employed instead of SV errors, thus no separate figure is included in this article.

The top panel of Figure 15 visualizes the performance of the three models; it depicts observed regression residuals against their (one-day-ahead) predicted distributions. For the purpose of this image, observed regression residuals are simply defined as the deviation of the data from the median of the predicted distribution. It stands out that predictive quantiles arising from the models with heteroskedastic errors exhibit much more flexibility to adapt to the "current state of the world", while the simpler homoskedastic model barely exhibits this feature.

Generally speaking, there is little difference in predicted residuals until the beginning of 2007. However, during the pre-crisis era (less volatility) and during the actual crisis (more volatility), the models catering for heteroskedasticity perform substantially better. It is interesting to see that predictive intervals for the models with SV errors and GARCH errors are very similar.

In the bottom panel of Figure 15, the cumulative sums of the log predictive Bayes factors are displayed. The last points plotted equal to 176.52 in the SV case and 166.37 in the GARCH case; this provides overwhelming overall evidence in favor of a model catering for heteroskedasticity and "decisive" (**?**) respectively "very strong" (**?**) evidence in favor of SV over vanilla GARCH(1,1) with a final cumulative predictive Bayes factor around 25 000 : 1.

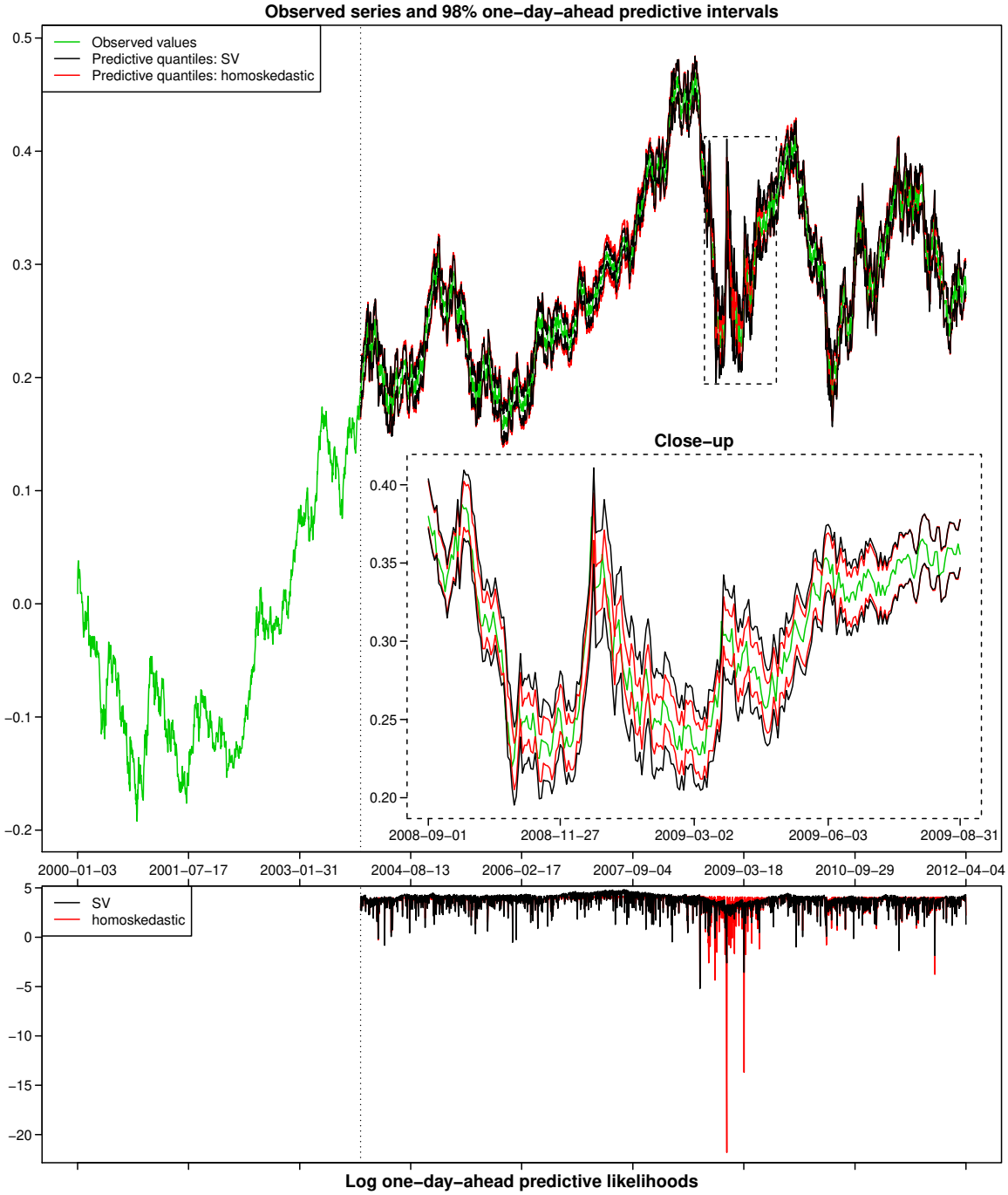

Figure 14: Top panel: Observed series (green) and symmetrical 98% one-day-ahead predictive intervals for the model with homoskedastic errors (red) and the model with SV errors (black). The display also contains a close-up, showing only the period from September 2008 until August 2009. This time span is chosen to include the most noticeable ramifications of the financial crisis. During that phase, predictive performance of the model with homoskedastic errors deteriorates substantially, while SV errors can capture the inflated volatility much better. Bottom panel: Log one-day-ahead predictive likelihoods for both models.

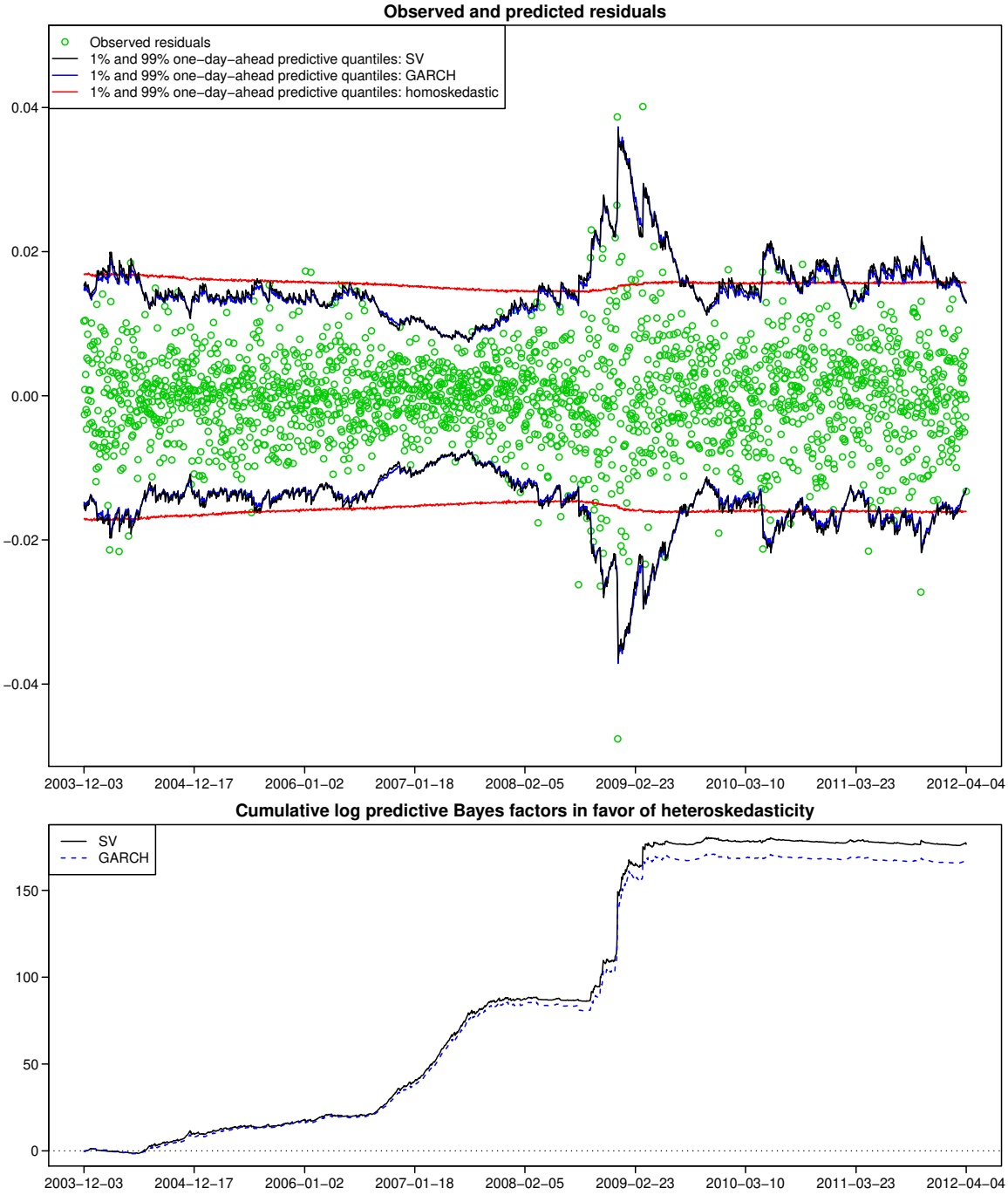

Figure 15: Top panel: Observed residuals with respect to the median of the one-day-ahead predictive distribution along with 1% and 99% quantiles of the respective predictive distributions. It can clearly be seen that the variances of the predictive distributions in the GARCH and SV models adjust to heteroskedasticity, while the model with homoskedastic errors is much more restrictive. Bottom panel: Cumulative log predictive Bayes factors in favor of the model with SV residuals. Values greater than zero mean that the model with GARCH/SV residuals performs better out of sample up to this point in time.

It is worth noting that for other exchange rates contained in `exrates`, a qualitatively similar picture emerges. For an overview of cumulative log predictive likelihoods, see Table 1.

Currency	SV	GARCH	homoskedastic
AUD	7938	7890	7554
CAD	7851	7844	7714
CHF	9411	9337	8303
CZK	9046	8995	8659
DKK[6]	1473	1366	1178
GBP	8568	8549	8218
HKD	7907	7897	7728
IDR	7697	7662	7269
JPY	7607	7586	7280
KRW	7766	7749	7188
MXN	7536	7507	7055
MYR	8082	8064	7928
NOK	8648	8616	8331
NZD	7619	7587	7440
PHP	7890	7862	7654
PLN	8126	8094	7727
RON	9011	8880	8255
RUB	8664	8617	8146
SEK	9110	9101	8648
SGD	8540	8529	8308
THB	7867	7844	7692
USD	7878	7868	7701
USD [AR(0)][7]	7876	7865	7699

Table 1: Final cumulative predictive log likelihoods for AR(1) models with different error assumptions, applied to the logarithm of several EUR exchange rates. SV is strongly favored in all cases.

As pointed out above, the analysis of log returns can be viewed upon as a special case of the analysis of log levels where $\beta_1 \equiv 1$ is fixed a priori. When doing so, evidence in favor of heteroskedasticity is again striking. Once more, the model with SV scores highest; its demeaned predictive quantiles are almost indistinguishable from the "varying β_1 case". Predictive quantiles of the model with GARCH residuals again resemble those of the model with SV residuals very closely. The sum of log predictive likelihoods for $t \in \{1001, 1002, \ldots, n\}$ is given in the last line of Table 1. The values are slightly but consistently lower than for the models where β_1 is estimated from the data. The display of predictive intervals for the log returns is practically identical to Figure 15 and thus omitted here.

Concluding this somewhat anecdotal prediction exercise, we would like to note that for a thorough and more universal statement concerning the real-world applicability and predictive accuracy of **stochvol**, further studies with different datasets and a richer variety of competing models, potentially including realized volatility measures, are called for. Such a voluminous undertaking is, however, beyond the scope of this paper.

[6] For exchange rates of the Danish krone with respect to the euro, we analyze `1000*log(exrates$DKK)`, i.e., per mille (‰) log prices. This way, we avoid convergence issues when obtaining starting values for the MCMC sampler from **fGarch** that appear otherwise because the krone is pegged very closely to the euro.

[7] These results refer to the analysis of log returns by means of a intercept-only model, i.e., AR(0).

6. Conclusion

The aim of this article was to introduce the reader to the functionality of **stochvol**. This R package provides a fully Bayesian simulation-based approach for statistical inference in stochastic volatility models. The typical application workflow of **stochvol** was illustrated through the analysis of exchange rate data contained in the package's `exrates` dataset. Furthermore, it was shown how the package can be used as a "plug-in" tool for other MCMC samplers. This was illustrated by estimating a Bayesian linear model with SV errors.

In the predictive example, both log levels of exchange rates from EUR to USD and log returns thereof were analyzed. For this dataset, out-of-sample analysis through cumulative predictive Bayes factors clearly showed that modeling regression residuals heteroskedastically substantially improves predictive performance, especially in turbulent times. A direct comparison of SV and vanilla GARCH(1,1) indicated that the former performs better in terms of predictive accuracy.

Acknowledgments

The author would like to thank Sylvia Frühwirth-Schnatter, Hedibert Freitas Lopes, Karin Dobernig, and two anonymous referees for helpful comments and suggestions.

Affiliation:

Gregor Kastner
Institute for Statistics and Mathematics
Department of Finance, Accounting and Statistics
WU Vienna University of Economics and Business
Welthandelsplatz 1, Building D4, Level 4
1020 Vienna, Austria
Telephone: +43/1/31336-5593
Fax: +43/1/31336-90-5593
E-mail: gregor.kastner@wu.ac.at
URL: http://statmath.wu.ac.at/~kastner/