

Package ‘bayesGARCH’

August 21, 2026

Version 2.2.0

Date 2026-08-19

Title Bayesian Estimation of the GARCH(1,1) Model with Student-t Innovations

Maintainer David Ardia <david.ardia.ch@gmail.com>

Imports coda

Suggests testthat (>= 3.0.0), mvtnorm

Description Provides the bayesGARCH() function which performs the Bayesian estimation of the GARCH(1,1) model with Student's t innovations as described in Ardia (2008) <doi:10.1007/978-3-540-78657-3>.

BugReports <https://github.com/ArdiaD/bayesGARCH/issues>

URL <https://github.com/ArdiaD/bayesGARCH>

License GPL (>= 2)

Encoding UTF-8

Config/testthat/edition 3

NeedsCompilation yes

Author David Ardia [aut, cre, cph, fnd] (ORCID: <<https://orcid.org/0000-0003-2823-782X>>)

Repository CRAN

Date/Publication 2026-08-21 15:50:02 UTC

Contents

bayesGARCH	2
dem2gbp	7
formSmpl	8
Index	9

bayesGARCH	<i>Bayesian Estimation of the GARCH(1,1) Model with Student-t Innovations</i>
------------	---

Description

Performs the Bayesian estimation of the GARCH(1,1) model with Student-t innovations.

Usage

```
bayesGARCH(y, mu.alpha = c(0,0), Sigma.alpha = 1000 * diag(1,2),
            mu.beta = 0, Sigma.beta = 1000,
            lambda = 0.01, delta = 2, control = list())
```

Arguments

y	vector of observations of size T . NA values are not allowed.
mu.alpha	hyper-parameter μ_α (prior mean) for the truncated Normal prior on parameter $\alpha := (\alpha_0 \ \alpha_1)'$. Default: a 2×1 vector of zeros.
Sigma.alpha	hyper-parameter Σ_α (prior covariance matrix) for the truncated Normal prior on parameter α . Default: a 2×2 diagonal matrix whose variances are set to 1'000, i.e., a diffuse prior. Note that the matrix must be symmetric positive definite.
mu.beta	hyper-parameter μ_β (prior mean) for the truncated Normal prior on parameter β . Default: zero.
Sigma.beta	hyper-parameter $\Sigma_\beta > 0$ (prior variance) for the truncated Normal prior on parameter β . Default: 1'000, i.e., a diffuse prior.
lambda	hyper-parameter $\lambda > 0$ for the translated Exponential distribution on parameter ν . Default: 0.01.
delta	hyper-parameter $\delta \geq 2$ for the translated Exponential distribution on parameter ν . Default: 2 (to ensure the existence of the conditional variance).
control	list of control parameters (See *Details*).

Details

The function bayesGARCH performs the Bayesian estimation of the GARCH(1,1) model with Student-t innovations. The underlying algorithm is based on Nakatsuma (1998, 2000) for generating the parameters of the GARCH(1,1) scedastic function $\alpha := (\alpha_0 \ \alpha_1)'$ and β and on Geweke (1993) and Deschamps (2006) for the generating the degrees of freedom parameter ν . Further details and examples can be found in Ardia (2008) and Ardia and Hoogerheide (2010). Finally, we refer to Ardia (2009) for an extension of the algorithm to Markov-switching GARCH models.

The series is rescaled internally to unit root mean square before sampling and the draws are mapped back, so that results do not depend on the units of y. Only α_0 carries that scale: under $y \rightarrow y/s$ one has $\alpha_0 \rightarrow \alpha_0/s^2$ while α_1 , β and ν are unchanged, and the Normal prior on α transforms exactly under that map, so the posterior explored is the same one. The factor is a power of two, which makes the rescaling and its inverse exact in binary floating point. Note that the prior itself is stated

in the units of y ; rescaling y without rescaling Sigma.alpha changes the posterior, because a prior that is diffuse on one scale is informative on another.

The control argument is a list that can supply any of the following components:

`n.chain` number of MCMC chain(s) to be generated. Default: `n.chain=1`. When `start.val` is a matrix holding one row per chain, `n.chain` may be omitted, in which case it is taken to be the number of rows.

`l.chain` length of each MCMC chain. Default: `l.chain=10000`.

`start.val` vector of starting values of chain(s). Default: `c(0.01, 0.1, 0.7, delta + 1/lambda)`, whose last entry is the prior mean of ν and therefore lies in the prior support for any δ , and whose first entry applies to the internally rescaled series (see **Details**), making it independent of the units of y . A value supplied by the user is read in the units of y . A matrix of size $n \times 4$ containing starting values in rows can also be provided. This will generate n chains starting at the different row values. A vector, or a matrix with a single row, is replicated across the `n.chain` chains.

Every row must lie in the support of the posterior: $\alpha_0, \alpha_1, \beta > 0$, $\nu > \delta$, and, when `addPriorConditions` is used, it must satisfy that condition as well. Rows that do not are rejected with an error naming the row and the condition it fails. See **Note**.

`addPriorConditions` function which allows the user to add constraints on the model parameters. Default: `NULL`, i.e. not additional constraints are imposed (see below). It must return a single non-missing `TRUE` or `FALSE` for every parameter vector it is given, both for the starting values and for the states proposed during sampling.

`refresh` frequency of reports. Default: `refresh=10` iterations.

`digits` number of printed digits in the reports. Default: `digits=4`.

Value

A list of class `mcmc.list` (R package **coda**), with one `mcmc` object per chain.

Each chain carries a `"move.rates"` attribute holding, for every parameter, the proportion of iterations at which its value changed. This is a diagnostic only: a high rate is not by itself evidence of good mixing, since the accepted moves are typically small. Use `effectiveSize` to judge how many independent draws the chain actually delivers.

Note

By using bayesGARCH you agree to the following rules:

- You must cite Ardia and Hoogerheide (2010) in working papers and published papers that use bayesGARCH. Use `citation("bayesGARCH")`.
- You must place the following URL in a footnote to help others find bayesGARCH: <https://CRAN.R-project.org/package=bayesGARCH>.
- You assume all risk for the use of bayesGARCH.

The GARCH(1,1) model with Student-t innovations may be written as follows:

$$y_t = \epsilon_t(\rho h_t)^{1/2}$$

for $t = 1, \dots, T$, where the conditional variance equation is defined as:

$$h_t := \alpha_0 + \alpha_1 y_{t-1}^2 + \beta h_{t-1}$$

where $\alpha_0 > 0, \alpha_1 \geq 0, \beta \geq 0$ to ensure a positive conditional variance. We set the initial variance and the initial observation to $h_0 := 0$ and $y_0 := 0$ for convenience, so that $h_1 = \alpha_0$. The parameter $\varrho := (\nu - 2)/\nu$ is a scaling factor which ensures the conditional variance of y_t to be h_t . Finally, ϵ_t follows a Student-t distribution with ν degrees of freedom.

The prior distributions on α is a bivariate truncated Normal distribution:

$$p(\alpha) \propto N_2(\alpha \mid \mu_\alpha, \Sigma_\alpha) I_{[\alpha > 0]}$$

where μ_α is the prior mean vector, Σ_α is the prior covariance matrix and $I_{[\bullet]}$ is the indicator function.

The prior distribution on β is a univariate truncated Normal distribution:

$$p(\beta) \propto N(\beta \mid \mu_\beta, \Sigma_\beta) I_{[\beta > 0]}$$

where μ_β is the prior mean and Σ_β is the prior variance.

The prior distribution on ν is a translated Exponential distribution:

$$p(\nu) = \lambda \exp[-\lambda(\nu - \delta)] I_{[\nu > \delta]}$$

where $\lambda > 0$ and $\delta \geq 2$. The prior mean for ν is $\delta + 1/\lambda$.

The joint prior on parameter $\psi := (\alpha, \beta, \nu)$ is obtained by assuming prior independence:

$$p(\psi) = p(\alpha)p(\beta)p(\nu).$$

The default hyperparameters $\mu_\alpha, \Sigma_\alpha, \mu_\beta, \Sigma_\beta$ and λ define a rather vague prior. The hyper-parameter $\delta \geq 2$ ensures the existence of the conditional variance. The k th conditional moment for ϵ_t is guaranteed by setting $\delta \geq k$.

The Bayesian estimation of the GARCH(1,1) model with Normal innovations is obtained as a special case by setting `lambda=100` and `delta=500`. In this case, the generated values for ν are centered around 500 which ensure approximate Normality for the innovations.

The function `addPriorConditions` allows to add prior conditions on the model parameters $\psi := (\alpha_0 \ \alpha_1 \ \beta \ \nu)'$. The function must return `TRUE` if the constraint holds and `FALSE` otherwise.

By default, the function is:

```
addPriorConditions <- function(psi)
{
  TRUE
}
```

and therefore does not add any other constraint than the positivity of the parameters which are obtained through the prior distribution for ψ .

You simply need to modify `addPriorConditions` in order to add constraints on the model parameters ψ . For instance, to impose the covariance-stationary conditions to hold, i.e. $\alpha_1 + \beta < 1$, just define the function `addPriorConditions` as follows:

```
addPriorConditions <- function(psi)
{
  psi[2] + psi[3] < 1
}
```

Note that adding prior constraints on the model parameters can diminish the acceptance rate and therefore lead to a very inefficient sampler. This would however indicate that the condition is not supported by the data.

The chain must *start* in the constrained region. Every step only ever accepts a move into it, so a chain started outside can never enter: the starting value would be returned unchanged at every iteration and the whole sample would consist of draws the posterior gives zero mass. This is easy to trigger by accident, since the advice below to start from the parameter values of a previous estimation step will supply an unconstrained estimate that need not satisfy the constraint. Starting values are therefore validated against `addPriorConditions` before any chain is generated.

The constraint enters the acceptance test of *each* Metropolis-Hastings step, so that the sampler leaves the constrained posterior $p(\psi \mid y)I_{[\psi \in A]}$ invariant. Up to version 2.1.10 the constraint was instead applied once to the whole block of parameters, accepting or rejecting the entire sweep on the final value of ψ . That scheme does not preserve the constrained posterior — the block is a composition of reversible steps, which is invariant but not itself reversible — and the chain converged to a different distribution. Results obtained with `addPriorConditions` under earlier versions should be re-computed; see the ‘NEWS’ file.

The estimation strategy implemented in bayesGARCH is fully automatic and does not require any tuning of the MCMC sampler. The cost per iteration is modest on current hardware, but the draws are strongly autocorrelated, so long chains are needed: on the `dem2gbp` series ($T = 1974$) the default `l.chain=10000` delivers an effective sample size of only a few hundred draws per parameter. Always inspect the output before using it — `effectiveSize` and `gelman.diag` (R package **coda**) apply directly to the value returned by bayesGARCH — and raise `l.chain` until the effective sample size is adequate for the quantity of interest. Estimating the model over several datasets on a daily basis can therefore take a significant amount of time. In this case, the algorithm can be easily parallelized, by running a single chain on several processors. Also, when the estimation is repeated over updated time series (i.e. time series with more recent observations), it is wise to start the algorithm using the posterior mean or median of the parameters obtained at the previous estimation step. The impact of the starting values (burn-in phase) is likely to be smaller and thus the convergence faster.

Finally, note that as any MH algorithm, the sampler can get stuck to a given value, so that the chain does not move anymore. However, the sampler uses Taylor-made candidate densities that are especially ‘constructed’ at each step, so it is almost impossible for this MCMC sampler to get stuck at a given value for many subsequent draws. A high proportion of accepted moves should not be read as good mixing, however: the accepted moves are typically small, and the resulting chains are strongly autocorrelated, as noted above. In the unlikely case that such ill behavior would occur, one could run the algorithm with different initial values or a different random seed. Rescaling the data is no longer necessary: it is done internally, as described in **Details**. Before version 2.2.0 a series whose scale was far from unity — log-returns in decimals rather than in percent was already enough — could have every α and β move rejected, leaving a chain that never moved and eventually failed in the ν update.

Author(s)

David Ardia <david.ardia.ch@gmail.com>

References

Ardia, D. (2009) Bayesian Estimation of a Markov-Switching Threshold Asymmetric GARCH Model with Student-t Innovations. *Econometrics Journal* **12**(1), pp. 105-126. doi:[10.1111/j.1368-423X.2008.00253.x](https://doi.org/10.1111/j.1368-423X.2008.00253.x)

Ardia, D., Hoogerheide, L.F. (2010) Bayesian Estimation of the GARCH(1,1) Model with Student-t Innovations. *R Journal* **2**(2), pp.41-47. doi:[10.32614/RJ2010014](https://doi.org/10.32614/RJ2010014)

Ardia, D. (2008) Financial Risk Management with Bayesian Estimation of GARCH Models. Lecture Notes in Economics and Mathematical Systems **612**. Springer-Verlag, Berlin, Germany. ISBN 978-3-540-78656-6, e-ISBN 978-3-540-78657-3, doi:[10.1007/9783540786573](https://doi.org/10.1007/9783540786573)

Deschamps, P.J. (2006) A Flexible Prior Distribution for Markov Switching Autoregressions with Student-t Errors. *Journal of Econometrics* **133**, pp.153-190.

Geweke, J.F. (1993) Bayesian Treatment of the Independent Student-t Linear Model. *Journal of Applied Econometrics* **8**, pp.19-40.

Nakatsuma, T. (2000) Bayesian Analysis of ARMA-GARCH Models: A Markov Chain Sampling Approach. *Journal of Econometrics* **95**(1), pp.57-69.

Nakatsuma, T. (1998) A Markov-Chain Sampling Algorithm for GARCH Models. *Studies in Non-linear Dynamics and Econometrics* **3**(2), pp.107-117.

See Also

[garchFit](#) (R package **fGarch**) for the classical Maximum Likelihood estimation of GARCH models.

Examples

```
## !!! INCREASE THE NUMBER OF MCMC ITERATIONS !!!

## LOAD DATA
data(dem2gbp)
y <- dem2gbp[1:750]

## RUN THE SAMPLER (2 chains)
MCMC <- bayesGARCH(y, control = list(n.chain = 2, l.chain = 200))

## MCMC ANALYSIS (using coda)
plot(MCMC)

## FORM THE POSTERIOR SAMPLE
smp1 <- formSmp1(MCMC, l.bi = 50)

## POSTERIOR STATISTICS
summary(smp1)
smp1 <- as.matrix(smp1)
pairs(smp1)
```

```
## GARCH(1,1) WITH NORMAL INNOVATIONS
MCMC <- bayesGARCH(y, lambda = 100, delta = 500,
                  control = list(n.chain = 2, l.chain = 200))

## GARCH(1,1) WITH NORMAL INNOVATIONS AND
## WITH COVARIANCE STATIONARITY CONDITION
addPriorConditions <- function(psi){psi[2] + psi[3] < 1}
MCMC <- bayesGARCH(y, lambda = 100, delta = 500,
                  control = list(n.chain = 2, l.chain = 200,
                                addPriorConditions = addPriorConditions))
```

dem2gbp

DEM/GBP exchange rate log-returns

Description

The vector dem2gbp contains daily observations of the Deutschmark vs British Pound foreign exchange rate log-returns. This data set has been promoted as an informal benchmark for GARCH time-series software validation. See McCullough and Renfro (1999), and Brooks, Burke, and Persand (2001) for details. The nominal returns are expressed in percent as in Bollerslev and Ghysels (1996). The sample period is from January 3, 1984, to December 31, 1991, for a total of 1974 observations.

Usage

```
data(dem2gbp)
```

Format

A vector of size 1974.

Source

Journal of Business and Economic Statistics

References

- Bollerslev T., Ghysels, E. (1996) Periodic Autoregressive Conditional Heteroscedasticity. *Journal of Business and Economic Statistics* **14**(2), pp.139–151.
- Brooks C., Burke S. P., Persand G. (2001) Benchmarks and the Accuracy of GARCH Model Estimation. *International Journal of Forecasting* **17**(1), pp.45–57.
- McCullough B. D., Renfro C. G. (1999) Benchmarks and Software Standards: A Case Study of GARCH Procedures. *Journal of Economic and Social Measurement* **25**(2), pp.59–71.

formSmpl	<i>Form the Posterior Sample</i>
----------	----------------------------------

Description

Form the joint posterior sampler from the MCMC output.

Usage

```
formSmpl(MCMC, l.bi = 0, batch.size = 1)
```

Arguments

MCMC	object of the class <code>mcmc.list</code> (R package coda) or a list of matrices or a matrix. When several chains are given they must share the same number of rows and columns and the same column names.
l.bi	length of the <i>burn-in</i> phase, a whole number smaller than the chain length.
batch.size	batching size used to diminish the autocorrelation within the chains, a whole number greater or equal than one.

Value

The joint posterior sample as an `mcmc` object (R package **coda**).

Note

Please cite the package in publications. Use `citation("bayesGARCH")`.

See Also

[bayesGARCH](#) for the Bayesian estimation of the GARCH(1,1) model with Student-t innovations.

Examples

```
## !!! INCREASE THE NUMBER OF MCMC ITERATIONS !!!

## LOAD DATA SET
data(dem2gbp)
y <- dem2gbp[1:750]

## RUN THE ESTIMATION
MCMC <- bayesGARCH(y, control = list(n.chain = 2, l.chain = 100))

## FORM THE SAMPLE FROM THE MCMC OUTPUT
smpl <- formSmpl(MCMC, l.bi = 50, batch.size = 2)

## POSTERIOR STATISTICS
summary(smpl)
```

Index

- * **Bayesian GARCH**

- bayesGARCH, [2](#)

- * **datasets**

- dem2gbp, [7](#)

- * **misc**

- formSmpl, [8](#)

- * **models**

- bayesGARCH, [2](#)

bayesGARCH, [2](#), [8](#)

dem2gbp, [7](#)

effectiveSize, [3](#), [5](#)

formSmpl, [8](#)

garchFit, [6](#)

gelman.diag, [5](#)