Package 'BTSR'

June 18, 2025

Type Package

Title Bounded Time Series Regression

Version 1.0.0

Date 2025-06-18

Maintainer Taiane Schaedler Prass <taianeprass@gmail.com>

Description Simulate, estimate and forecast a wide range of regression based dynamic models for bounded time series, covering the most commonly applied models in the literature. The main calculations are done in FORTRAN, which translates into very fast algorithms.

License GPL (>= 3)

Depends R (>= 4.0)

Imports Rdpack

RdMacros Rdpack

Copyright see file COPYRIGHTS

Encoding UTF-8

NeedsCompilation yes

RoxygenNote 7.3.2

Author Taiane Schaedler Prass [aut, cre, com] (ORCID: <https://orcid.org/0000-0003-3136-909X>), Guilherme Pumi [ctb, aut] (ORCID: <https://orcid.org/0000-0002-6256-3170>), Fábio Mariano Bayer [ctb] (ORCID: <https://orcid.org/0000-0002-1464-0805>), Jack Joseph Dongarra [ctb] (LINPACK subroutines (dtrsl, dpofa, ddot)), Cleve Moler [ctb] (LINPACK subroutines (dtrsl, dpofa, ddot)), Gilbert Wright Stewart [ctb] (LINPACK subroutines (dtrsl, dpofa, ddot)), Ciyou Zhu [ctb] (L-BFGS-B algorithm subroutines), Richard H. Byrd [ctb] (L-BFGS-B algorithm subroutines), Jorge Nocedal [ctb] (L-BFGS-B algorithm subroutines), Jose Luis Morales [ctb] (L-BFGS-B algorithm subroutines), Peihuang Lu-Chen [ctb] (L-BFGS-B algorithm subroutines), John Burkardt [ctb] (Trigamma function (FORTRAN90 version)),

Alan Miller [ctb] (FORTRAN90 version of NSWC special function psi and
amendments to minim subroutine),
D.E. Shaw [ctb] (Original minim subroutine),
Robert W.M. Wedderburn [ctb] (Amendments to minim subroutine)

Repository CRAN

Date/Publication 2025-06-18 15:10:02 UTC

Contents

`btsr-package` *Bounded Time Series Regression*

Description

The BTSR package provides a unified framework for simulating, fitting, and forecasting bounded time series regression models. It supports a wide range of models, including i.i.d., regression, ARMA-like, and ARFIMA-like models, with a focus on bounded time series data.

Key features of the BTSR package include

- Simulation of bounded time series data using various models.
- Estimation of model parameters using efficient algorithms.

- Forecasting future values based on fitted models.
- Support for both short-memory and long-memory models.
- Flexible link functions and error scales.

Mathematical Notation

The BTSR package is based on the following mathematical framework

- Y_t: The bounded random variable at time t, with $Y_t \in (a, b)$.
- $\mathcal{F}_t$: The σ-field generated by information up to time t.
- μ_t, ν_t: Parameters of the conditional distribution of Y_t.
- ϑ_t: A transformation of ν_t (e.g., $\vartheta_t = \nu_t^2$).
- η_{1t}, η_{2t}: Linear predictors for μ_t and ϑ_t, respectively.
- ϕ, θ: Autoregressive (AR) and moving average (MA) coefficients.
- d: Fractional differencing parameter, controlling long-memory behavior.
- e_{1t}, e_{2t}: Error terms for each part of the model (see the model definition for details).

The BTSR Structure

Let $\{Y_t\}_{t\in\mathbb{Z}}$ be a stochastic process for which $Y_t \in (a, b)$ with probability 1 (a and b not necessarily finite), for all $t \in \mathbb{Z}$, and let $\mathcal{F}_t$ denote the σ-field generated by the information observed up to time t. The general structure of a BTSR model is as follows

$$
\begin{aligned}
Y_t|\mathcal{F}_{t-1} &\sim f(\cdot|\mu_t, \nu_t), \quad \vartheta_t = g_2(\nu_t) \\
\eta_{1t} = g_{11}(\mu_t) &= \alpha_1 + \boldsymbol{X}'_{1t}\boldsymbol{\beta}_1 + \sum_{i=1}^{p_1} \phi_{1i}[g_{12}(Y_{t-i}) - I_{X_1}\boldsymbol{X}'_{1(t-i)}\boldsymbol{\beta}_1] + \xi_t, \quad \text{(part 1)} \\
\eta_{2t} = g_{21}(\vartheta_t) &= \alpha_2 + \boldsymbol{X}'_{2t}\boldsymbol{\beta}_2 + \sum_{i=1}^{p_2} \phi_{2i}[g_{22}(\vartheta_{t-i}) - I_{X_2}\boldsymbol{X}'_{2(t-i)}\boldsymbol{\beta}_2] + \sum_{k=1}^{\infty} c_{2k}e_{2,t-k}, \quad \text{(part 2)}
\end{aligned}
$$

with ξ_t depending on the model, controlled by the argument `model`,

$$
\xi_t = \begin{cases} h(T^{t-1}(U_0)), & \text{if } \texttt{model = "BARC"}, \\ \sum_{k=1}^{\infty} c_{1k}e_{1,t-k}, & \text{otherwise,} \end{cases}
$$

$e_{1,t}$ depending on the `error.scale` adopted

$$
e_{1,t} = g_{13}(Y_t, \mu_t) = \begin{cases} Y_t - \mu_t, & \text{if } \texttt{error.scale = 0} \text{ (data scale),} \\ g_{11}(Y_t) - g_{11}(\mu_t), & \text{if } \texttt{error.scale = 1} \text{ (predictive scale)} \end{cases}
$$

and $e_{2,t} = g_{23}(e_{1,t})$, where

- I_{X_1}, I_{X_2} are indicator functions, which control whether the regressors should be included in the AR part of the equation. These functions are controlled by the argument `xregar` as follows: `xregar = FALSE` corresponds to $I_X = 0$ and `xregar = TRUE` corresponds to $I_X = 1$. If `xregar = FALSE`, the corresponding argument `xreg` is not included in the AR part of the model.

- g_{13} is a function of Y_t and μ_t, defined by the error scale adopted.
- g_2 and g_{ij}, $i, j \in \{1, 2\}$, and g_{23} are link functions defined in the argument `linkg`. Notice that the links g_{ij} are only used in the AR part of the model and, typically, $g_{i1} = g_{i2}$ for each $i \in \{1, 2\}$. The link g_2 might depend on the distribution adopted. Finally, g_{23} is a link function that transforms the error term e_{1t}.
- h is the link function for BARC models, defined in the argument `linkh`.
- $\{c_{ik}\}_{k\geq 1}$ are the coefficients obtained through the relation

$$\theta_i(z) = \sum_{k=0}^{q_i} \theta_{ik} z^k \quad \text{and} \quad (1-L)^{-d_i}\theta_i(z) = \sum_{k=0}^{\infty} c_{ik} z^k, \quad i \in \{1, 2\}.$$

  In particular, if $d_i = 0$, then $c_{ik} = \theta_{ik}$, for $k = 1, \ldots, q_i$.

Author(s)

Taiane Schaedler Prass <taianeprass@gmail.com>, Guilherme Pumi <guipumi@gmail.com>

References

Bayer FM, Bayer DM, Pumi G (2017). "Kumaraswamy autoregressive moving average models for double bounded environmental data." *Journal of Hydrology*, **555**, 385–396. doi:10.1016/j.jhydrol.2017.10.006.

Pumi G, Valk M, Bisognin C, Bayer FM, Prass TS (2019). "Beta autoregressive fractionally integrated moving average models." *Journal of Statistical Planning and Inference*, **200**, 196–212. doi:10.1016/j.jspi.2018.10.001.

Pumi G, Prass TS, Souza RR (2021). "A dynamic model for double bounded time series with chaotic driven conditional averages." *Scandinavian Journal of Statistics*, **48**(1), 68–86. doi:10.1111/sjos.12439.

Pumi G, Prass TS, Taufemback CG (2024). "Unit-Weibull autoregressive moving average models." *TEST*, **33**, 204–229. doi:10.1007/s11749023008938.

Pumi G, Prass TS, Taufemback CG (2024). "Publisher Correction: Unit-Weibull autoregressive moving average models." *TEST*, **33**, 358–359. doi:10.1007/s11749023009057.

Pumi G, Matsuoka DH, Prass TS (2025). "A GARMA Framework for Unit-Bounded Time Series Based on the Unit-Lindley Distribution with Application to Renewable Energy Data." doi:10.48550/arXiv.2504.07351.

Pumi G, Matsuoka DH, Prass TS, Palm BG (2025). "A Matsuoka-Based GARMA Model for Hydrological Forecasting: Theory, Estimation, and Applications." doi:10.48550/arXiv.2502.18645.

Prass TS, Pumi G, Taufemback CG, Carlos JH (2025). "Positive time series regression models: theoretical and computational aspects." *Computational Statistics*, **40**, 1185–1215. doi:10.1007/s0018002401531z.

See Also

For detailed examples and usage instructions, see the documentation for individual functions

- btsr.sim: Simulate bounded time series data.

- btsr.extract: Extract components of a BTSR model, for a given set of parameters
- btsr.fit: Fit a BTSR model to data.
- predict: Forecast future values using a fitted model.
- arguments: Shared documentation for arguments

Examples

```
#----------------------------
# Quickstart examples.
#----------------------------

# Example 1: Simulate i.i.d. samples
set.seed(1234)
y1 <- btsr.sim(model = "BETA", n = 1000, coefs = list(alpha = 0.2, nu = 20))
hist(y1)

# Example 2: Simulate ARMA-like model with fixed nu
y2 <- btsr.sim(
  model = "BARMA", n = 100, link = "logit",
  coefs = list(alpha = 0.2, phi = 0.5, theta = 0.3, nu = 20)
)
plot(y2, type = "l")
```

`arguments.coefs` *Shared documentation for coefficients*

Description

This is the common documentation for arguments related to the coefficients in BTSR models.

Arguments

`ignore.start`	optional; logical value indicating whether the argument `start` should be ignored (fit only). If starting values are not provided, the function uses the default values and `ignore.start` is ignored. In case starting values are provided and `ignore.start = TRUE`, those starting values are ignored and recalculated. The default is `ignore.start = FALSE`. Partial starting values are not allowed.
`start`	optional; a list with the starting values for the non-fixed coefficients of the model (fit only). The default is `start = NULL`, in which case the function coefs.start is used internally to obtain starting values for the parameters. For details on the expected format and the arguments that can be passed through `coefs`, see the Section Model coefficients.
`coefs`	a list with the coefficients of the model (simulation and extraction only). The default is `coefs = NULL`. For details on the expected format and the arguments that can be passed through `coefs`, see the Section Model coefficients.

`lags` optional; a list with the lags (integer values) that the entries in `coefs` or `start` correspond to (extract and fit only). The default is `lags = NULL`, in which the `lags` are computed from the `fixed.lags` argument (if provided). When components are missing or empty in both, `lags` and `fixed.lags`, the default behavior is to include all lags based on `nreg = ncol(xreg)`, `p`, and `q`. For details, see the Section Model coefficients.

`fixed.values` optional; a list with the values of the coefficients that are fixed (extract and fit only). The default is `fixed.values = NULL`. See the Section Model coefficients.

`fixed.lags` optional; a list with the lags (integer values) that the fixed values in `fixed.values` correspond to (extract and fit only). The default is `fixed.lags = NULL`. For missing components, fixed values will are set based on `lags`.

`lower` optional; list with the lower bounds for the parameters (fit only). Default is `lower = NULL`. The default is to assume that the parameters have no lower bound except for `nu`, for which de default is 0. Only the bounds for bounded parameters need to be specified. The format of `lower` and the arguments that can be passed through this list are the same as the ones for `start`.

`upper` optional; list with the upper bounds for the parameters (fit only). Default is `upper = NULL`. The default is to assume that the parameters have no upper bound. Only the bounds for bounded parameters need to be specified. The format of `lower` and the arguments that can be passed through this list are the same as the ones for `start`.

Model coefficients

`start`, `coefs`, `fixed.values`, `lags` and `fixed.lags` can be specified in one of two ways

- **Legacy structure:** a list with optional components `alpha`, `beta`, `phi`, `theta`, `d`, `u0` (BARC only) and required argument `nu` (except for one-parameter models such as ULARMA and MARMA).
- **New structure:** a list with elements `part1` and `part2`, each being a list with with optional components `alpha`, `beta`, `phi`, `theta`, `d` and `u0` (BARC only).

The optional arguments in this lists are

- `alpha`: a numeric value corresponding to the intercept. For i.i.d. corresponds to the mean of the distribution.
- `beta`: a vector of coefficients corresponding to the regressors in `xreg`.
- `phi`: a vector of autoregressive coefficients.
- `theta`: for BARC models, this is the parameter for the map function (see BARC.functions for details). For any other model, this is a vector of moving average coefficients corresponding to the MA order.
- `d`: a numeric value corresponding to the long memory parameter.
- `u0`: a numeric value in the interval $(0, 1)$, corresponding to the value of the random variable U_0. See BARC.functions for details.
- `nu`: distribution related parameter, usually the dispersion.

The following rules apply for these lists and their arguments.

Simulation:

- Passing `coefs` as an empty list will result in an error message.
- `start` and `fixed.values` (consequently, `fixed.lags`) are not used.
- If `xreg` is provided but `coefs` does not include a `beta` argument, an error message is issued.
- `phi` must be a vector of length p (the AR order), meaning all coefficients must be provided, including zeros.
- `theta` (non-BARC models) must be a vector of length q (the MA order), meaning all coefficients must be provided, including zeros.

Extraction:

- One dimensional parameters (e.g. `alpha`) that do not appear in `coefs` are assume to be fixed.
- An error message will be issued if both `coefs` and `fixed.values` are both empty.
- If ν is not constant over time and `nu` is missing in both `coefs` and `fixed.values`, an error message is issued (except for one-parameter models such as ULARMA and MARMA). Ignored if the new format is used.

Fitting:

- One dimensional parameters (e.g. `alpha`) cannot appear in both `start` and `fixed.values`.
- `coefs` is not used.

Extraction and fitting:

- Coefficients may include both fixed lags (with values in `fixed.values`) and non-fixed lags (with values in `coefs` or `start`).
- `lags` and `fixed.lags` are complementary. Either suffices, or mix them (e.g., `lags` for some parameters, `fixed.lags` for others).
- For one dimensional parameters, the lag is obviously always 1 and can be suppressed when the parameter added to the `fixed.values` list.
- For extraction, if `coefs = NULL`, one dimensional parameters that do not appear in `fixed.values` are assumed to be non-fixed. The same goes for fitting when `start = NULL` or `ignore.start = TRUE`.
- If `coefs`/`start` is provided, one dimensional parameters that do not appear in this list are assumed to be fixed.
- By default, if a given vector has fixed lags and the corresponding entry in `fixed.values` is empty, the fixed values are set as zero.
- If parameter values are provided in `coefs`, `start` or `fixed.values` and the size of the vector is not the same as the dimension of the parameters, either `lags` or `fixed.lags` must also be provided.

arguments.configs *Shared documentation for configuration related parameteres*

Description

This is the common documentation for arguments related the configurations for fitting models and printing reports.

Arguments

control	a list with configurations to be passed to the optimization subroutines (fit only). Default is control = NULL. Missing arguments will receive default values. For details, see fit.control.
report	logical; indicates whether the summary from the fitted model should be be printed (fit only). Default is report = TRUE, in which case info is automatically set to TRUE.
complete	logical; if FALSE returns only yt, else returns additional components (simulation only). Default is complete = FALSE.
debug	logical, if TRUE the output from FORTRAN is return (for debugging purposes). Default is debug = FALSE.
...	further arguments passed to the internal functions. See, for instance, summary.btsr for details.

arguments.link *Shared documentation for link functions*

Description

This is the common documentation for arguments related link functions in BTSR models.

Arguments

error.scale	either 0 or 1; the scale for the error term. Default is error.scale = 1 (predictive scale).
linkg	link functions. Can be specified as a character, two-character vector or a named list. The corresponding text strings for currently available links are listed in link.btsr. Default values depend on the model. For some models default values override user specifications. See the Section Link defaults for details.
linkh	a character indicating which link must be associated to the chaotic process. See the Section 'The BTSR structure' in btsr-package for details and link.btsr for valid links. Default is linkh = "linear".

`configs.linkg` a list with two elements, `ctt` and `power`, which define the constant a and the exponent b in the link function $g(x) = ax^b$. Each element can be specified as a numeric value, a vector of size 2 or a named list. The default is `configs.linkg = NULL`. See the Section Link defaults for details.

`configs.linkh` a list with extra configurations for the link h. For now, only used if `linkh = "linear"` or `"polynomial"`. Default is `configs.linkh = list(ctt = 1, power = 1)`.

Link defaults

`linkh` and `configs.linkh` only apply to BARC models.

`linkg` can be specified in one of two ways

- **Legacy structure:** a character or two-character vector. If only one string is provided, the same link name is used for `g11` and `g12`. Internally, this structure is automatically converted to the new format with `g2 = g21 = g22 = g23 = "linear"`.
- **New structure:** a named list with optional elements (order is irrelevant) `g11`, `g12`, `g2`, `g21`, `g22` and `g23`. These links apply, respectively, to μ_t, Y_t (in the AR recursion or part 1), ν_t, $\vartheta_t = g_2(\nu_t)$, ϑ_t (in the AR recursion of part 2) and e_{1t} (to build the error term in part 2).

For models that do not have the ν parameter, the links `g2`, `g21`, `g22` and `g23` are set to `"linear"` for compatibility with Fortran subroutines.

Missing entries in the `linkg` list follow these rules

- If either `g11` or `g12` is missing (but not both), internally it is set `g12 = g11`.
- If both `g11` and `g12` are missing, use the default values for the particular model (see below).
- If `phi = NULL` for part 1, `g12` is not required, hence set to `"linear"` and ignored in Fortran.
- If `phi = NULL` for part 2, `g22` is not required, hence set to `"linear"` and ignored in Fortran.
- If either `g21` or `g22` is missing (but not both), internally it is set `g22 = g21`.
- If both `g21` and `g22` are missing, use the default values for the particular model (see below).

Default `linkg` values are model-dependent (based on the string provided with `model`):

- For all models where ν is constant over time:
  internally, `g2`, `g21`, and `g22` are forced to `"linear"`, with $a = 1$.
  Overrides any user specifications.
- iid samples:
  Overrides any user specifications.
  `linkg = "linear"` (with $a = 1$). Internally converted to

```
linkg = list(g11 = "linear", g12 = "linear", g2 = "linear"
             g21 = "linear", g22 = "linear", g23 = "linear")
```

- BARFIMA, KARFIMA, ULARFIMA, UWARFIMA:
  `linkg = "logit"`. Internally converted to

```
linkg = list(g11 = "logit", g12 = "logit", g2 = "linear"
             g21 = "linear", g22 = "linear", g23 = "linear")
```

- GARFIMA:
  `linkg = "log"`. Internally converted to

  ```
  linkg = list(g11 = "log", g12 = "log", g2 = "linear"
               g21 = "linear", g22 = "linear", g23 = "linear")
  ```

- MARFIMA:
  `linkg = "cloglog"`. Internally converted to

  ```
  linkg = list(g11 = "cloglog", g12 = "cloglog", g2 = "linear"
               g21 = "linear", g22 = "linear", g23 = "linear")
  ```

- BARFIMAV, GARFIMAV, KARFIMAV, UWARFIMAV:

  `g11` and `g12` have the same default values as the particular model where ν is constant over time.

  `g2 = "default"`, meaning that `g2` is set as the the default link for the model.

  - For BARFIMAV `"default" = SIP` with $a = b = 1$.
  - For GARFIMAV `"default" = SIP` with $a = 0$ and $b = 1$.
  - For remaining models `"default" = "linear"` with $a = 1$.

  `g21` depends on the model.

  - For BARFIMAV `g21 = "logit"`
  - For any other model `g21 = "log"`.

  For `g22`, the default is to assume `g22 = g21`.

  Finally, `g23 = "polynomial`, with $a = 1$ and $b = 2$ (set in `configs.link`)

- Particular cases (e.g., BREG, BREGV) inherit defaults from parent models (except iid samples).

`configs.linkg` if provided, it must be provided as a list with optional elements, `ctt` and `power`, which define the constant a and the exponent b in the link function $g(x) = ax^b$. Each element in this list can be specified in one of two ways

- **Legacy structure:** a numeric value (applied uniformly across all linear links) or a numeric vector of length 2, which will be associated to `g11` and `g12`.
- **New structure:** a named list with optional elements (order is irrelevant) `g11`, `g12`, `g2`, `g21`, `g22` and `g23`.

For now, the arguments `ctt` and `power` are only used when the link function is `"linear"` or `"polynomial"`. If `NULL`, default is to assume that `ctt` and `power` are both equal to 1 for all links.

See Also

BTSR.model.defaults: function to print default settings for a specified model

`arguments.loglik` *Shared documentation for log-likelihood*

Description

This is the common documentation for arguments related the log-likelihood functions, score vector and information matrix for BTSR models.

Arguments

`m`	a non-negative integer indicating the starting time for the sum of the partial log-likelihood, given by $\ell = \sum_{t=m+1}^{n} \ell_t$ (extract and fit only). Default is `m = 0`. For details, see the Section The log-likelihood.
`llk`	logical; indicates whether the value of the log-likelihood function should be returned (extract and fit only). Default is `llk = TRUE`.
`sco`	logical; indicates whether the score vector should be returned (extract and fit only). Default is `sco = FALSE`.
`info`	logical; indicates whether the information matrix should be returned (extract and fit only). Default is `info = FALSE`. For the fitting function, `info` is automatically set to `TRUE` when `report = TRUE`.
`extra`	logical, if `TRUE` the matrices and vectors used to calculate the score vector and the information matrix are returned (extract and fit only). Default is `extra = FALSE`. Ignored by BARC models.

The log-likelihood

Let $\boldsymbol{\gamma} = (\boldsymbol{\rho}', \boldsymbol{\lambda}')'$ be the vector of unknown parameters in the model where

- $\boldsymbol{\rho}$ is the vector of unknown parameters in part 1
- $\boldsymbol{\lambda}$ is the vector of unknown parameters in part 2.

The **log-likelihood function**, conditioned on a set of initial conditions $\mathcal{F}_m$ is given by

$$\ell(\boldsymbol{\gamma}) = \sum_{t=m+1}^{n} \ell_t = \sum_{t=m+1}^{n} \log\big(f(Y_t \mid \mathcal{F}_{t-1}, \boldsymbol{\gamma})\big).$$

For simplicity of notation assume $m = 0$. The **score vector** $U(\boldsymbol{\gamma}) = \big(U_{\boldsymbol{\rho}}(\boldsymbol{\gamma})', U_{\boldsymbol{\lambda}}(\boldsymbol{\gamma})'\big)'$ can be written as

$$U_{\boldsymbol{\rho}}(\boldsymbol{\gamma}) = D_{\boldsymbol{\rho}}' T_1 \boldsymbol{h}_1 + M_{\boldsymbol{\rho}}' T_2 \boldsymbol{h}_2 \qquad \text{and} \qquad U_{\boldsymbol{\lambda}}(\boldsymbol{\gamma}) = D_{\boldsymbol{\lambda}}' T_2 \boldsymbol{h}_2,$$

where

- $D_{\boldsymbol{\rho}}$, $D_{\boldsymbol{\lambda}}$ and $M_{\boldsymbol{\rho}}$ are the matrices for which the (i, j)th elements are given, respectively, by

$$[D_{\boldsymbol{\rho}}]_{i,j} = \frac{\partial \eta_{1i}}{\partial \rho_j}, \quad [D_{\boldsymbol{\lambda}}]_{i,j} = \frac{\partial \eta_{2i}}{\partial \lambda_j} \quad \text{and} \quad [M_{\boldsymbol{\rho}}]_{i,j} = \frac{\partial \eta_{2i}}{\partial \rho_j},$$

- T_1 and T_2 are diagonal matrices given by

$$T_1 = \text{diag}\left\{\frac{\partial \mu_1}{\partial \eta_{1t}}, \ldots, \frac{\partial \mu_n}{\partial \eta_{1n}}\right\}, \quad T_2 = \text{diag}\left\{\frac{\partial \nu_1}{\partial \eta_{2t}}, \ldots, \frac{\partial \nu_n}{\partial \eta_{2n}}\right\},$$

- $\boldsymbol{h}_1$ and $\boldsymbol{h}_2$ are the vectors defined by

$$\boldsymbol{h}_1 = \left(\frac{\partial \ell_1}{\partial \mu_1}, \cdots, \frac{\partial \ell_n}{\partial \mu_n}\right)' \quad \text{and} \quad \boldsymbol{h}_2 = \left(\frac{\partial \ell_1}{\partial \nu_1}, \cdots, \frac{\partial \ell_n}{\partial \nu_n}\right)'.$$

For the models implemented so far, $\partial \eta_{1t}/\partial \lambda_j = 0$ so that we don't need a matrix for these derivatives.

The **conditional Fisher information matrix** for $\boldsymbol{\gamma}$ is given by

$$K_n(\boldsymbol{\gamma}) = \begin{pmatrix} K_{\boldsymbol{\rho},\boldsymbol{\rho}} & K_{\boldsymbol{\rho},\boldsymbol{\lambda}} \\ K_{\boldsymbol{\lambda},\boldsymbol{\rho}} & K_{\boldsymbol{\lambda},\boldsymbol{\lambda}} \end{pmatrix}$$

with

$$\begin{aligned} K_{\boldsymbol{\rho},\boldsymbol{\rho}} &= D_{\boldsymbol{\rho}}' T_1 E_\mu T_1 D_{\boldsymbol{\rho}} + M_{\boldsymbol{\rho}}' T_2 E_{\mu\nu} T_1 D_{\boldsymbol{\rho}} + D_{\boldsymbol{\rho}}' T_1 E_{\mu\nu} T_2 M_{\boldsymbol{\rho}} + M_{\boldsymbol{\rho}}' T_2 E_\nu T_2 M_{\boldsymbol{\rho}} \\ K_{\boldsymbol{\rho},\boldsymbol{\lambda}} &= K_{\boldsymbol{\lambda},\boldsymbol{\rho}}' = D_{\boldsymbol{\rho}}' T_1 E_{\mu\nu} T_2 D_{\boldsymbol{\lambda}} + M_{\boldsymbol{\rho}}' T_2 E_\nu T_2 D_{\boldsymbol{\lambda}}, \\ K_{\boldsymbol{\lambda},\boldsymbol{\lambda}} &= D_{\boldsymbol{\lambda}}' T_2 E_\nu T_2 D_{\boldsymbol{\lambda}} \end{aligned}$$

where E_μ, $E_{\mu\nu}$ and E_ν are diagonal matrices for which the (t,t)th element is given by

$$[E_\mu]_{t,t} = -\mathbb{E}\left(\frac{\partial^2 \ell_t}{\partial \mu_t^2}\middle| \mathcal{F}_{t-1}\right), \quad [E_{\mu\nu}]_{t,t} = -\mathbb{E}\left(\frac{\partial^2 \ell_t}{\partial \mu_t \partial \nu_t}\middle| \mathcal{F}_{t-1}\right) \quad \text{and} \quad [E_\nu]_{t,t} = -\mathbb{E}\left(\frac{\partial^2 \ell_t}{\partial \nu_t^2}\middle| \mathcal{F}_{t-1}\right).$$

`arguments.map` *Available map functions in BTSR package*

Description

This documentation describes the `map` argument in BARC models and the map functions implemented in the BTSR package.

Arguments

`map`	a non-negative integer from 1 to 5 corresponding to the map function. Default is `map = 4`. See the Section The map function.

The map function

The map function $T : [0,1] \to [0,1]$ in BARC models is a dynamical system, i.e., a function, potentially depending on a r-dimensional vector of parameters θ. As for today, for all implemented maps, $r = 1$.

Available choices are

- map = 1, $\theta = k$, for k integer greater or equal to 2.

$$T(u) = (ku) (\text{mod } 1)$$

- map = 2, $0 \leq \theta \leq 1$

$$T(u) = \frac{u}{\theta} I(u < \theta) + \theta \frac{(u-\theta)}{(1-\theta)} I(u \geq \theta)$$

- map = 3 (logistic map), $0 \leq \theta \leq 4$,

$$T(u) = \theta(1-\theta)$$

- map = 4 (Manneville-Pomeau map), $0 < \theta < 1$

$$T(u) = (u + u^{1+\theta}) (\text{mod } 1)$$

- map = 5 (Lasota-Mackey's map),

$$T(u) = \frac{u}{(1-u)} I(u \leq 0.5) + (2u-1) I(u > 0.5)$$

`arguments.model` *Available models in BTSR package*

Description

The BTSR package supports a variety of models, including

- i.i.d structure,
- regression models,
- short- and long-memory time series models
- chaotic processes.

This documentation describes

- the `model` argument and available model strings,
- default configurations for specific models,
- how to reproduce models from literature.

Arguments

`model`	character string (case-insensitive) indicating the model to be fitted to the data. Must be one of the options listed in the Section Supported Models.

Supported Models

Internally, all models are handled by the same function and all models can be obtained from the more general case `"*ARFIMAV"`. When a particular model (e.g. `"BREG"` or `"BARMA"`) is invoked some default values are assumed.

The following table summarizes the available distributions and the corresponding string to generate each model type. The character `V` at the end of the string indicates that ν is time-varying.

Distribution	i.i.d. sample	Regression	Short Memory	Long Memory	Chaotic
Beta	BETA	BREG	BARMA	BARFIMA	BARC
		BREGV	BARMAV	BARFIMAV	
Gamma	GAMMA	GREG	GARMA	GARFIMA	
		GREGV	GARMAV	GARFIMAV	
Kumaraswamy	KUMA	KREG	KARMA	KARFIMA	
		KREGV	KARMAV	KARFIMAV	
Matsuoka	MATSU	MREG	MARMA	MARFIMA	
Unit-Lindley	UL	ULREG	ULARMA	ULARFIMA	
Unit-Weibull	UW	UWREG	UWARMA	UWARFIMA	
		UWREGV	UWARMAV	UWARFIMAV	

Default values

All models are special cases of the general `"*ARFIMAV"` structure. When a specific model is selected via `model = "NAME"`, the package automatically applies these default configurations (any parameter that does not appear in the equations below is ignored)

i.i.d samples (e.g., BETA, GAMMA,...)

$$\eta_{1t} = \alpha_1 = \mu, \quad \eta_{2t} = \alpha_2 = \nu.$$

Fixed

```
p <- q <- d <- 0
xreg <- NULL
linkg <- list(g11 = "linear", g2 = "linear",
              g21 = "linear", g23 = "linear")
```

Regression models with ν_t constant over time (e.g., BREG, GREG,...)

$$\eta_{1t} = g_{11}(\mu_t) = \alpha_1 + \boldsymbol{X}'_{1t}\boldsymbol{\beta}_1, \quad \eta_{2t} = \alpha_2 = \nu.$$

Fixed

```
p <- q <- d <- 0
xreg <- list(part1 = "user's regressors", part2 = NULL)
linkg <- list(g11 = "user's choice", g12 = "linear",
              g2 = "linear", g21 = "linear", g23 = "linear")
```

Regression models with ν_t varying on time (e.g. BREGV, GREGV)

$$\eta_{1t} = g_{11}(\mu_t) = \alpha_1 + \boldsymbol{X}'_{1t}\boldsymbol{\beta}_1, \quad \eta_{2t} = g_{21}(g_2(\nu_t)) = \alpha_2 + \boldsymbol{X}'_{2t}\boldsymbol{\beta}_2.$$

Fixed

```
p <- q <- d <- 0
linkg <- list(g11 = "user's choice", g12 = "linear",
              g2 = "user's choice", g21 = "user's choice",
              g22 = "linear", g23 = "linear")
```

Short-memory models with ν constant over time (ARMA-like) (e.g. BARMA, GARMA,...)

$$\eta_{1t} = g_{11}(\mu_t) = \alpha_1 + \boldsymbol{X}'_{1t}\boldsymbol{\beta}_1 + \sum_{i=1}^{p_1} \phi_{1i}\big(g_{12}(Y_{t-i}) - I_{X_1}\boldsymbol{X}'_{1(t-i)}\boldsymbol{\beta}_1\big) + \sum_{k=1}^{q_1} \theta_{1k} e_{1,t-k},$$

$$\eta_{2t} = \alpha_2 = \nu.$$

Fixed

```
d <- 0
xreg <- list(part1 = "user's regressors", part2 = NULL)
linkg <- list(g11 = "user's choice", g12 = "user's choice",
              g2 = "linear", g21 = "linear", g23 = "linear")
```

Short-memory models with ν_t varying on time (e.g. BARMAV, GARMAV,...)

$$\eta_{1t} = g_{11}(\mu_t) = \alpha_1 + \boldsymbol{X}'_{1t}\boldsymbol{\beta}_1 + \sum_{i=1}^{p_1} \phi_{1i}\big(g_{12}(Y_{t-i}) - I_{X_1}\boldsymbol{X}'_{1(t-i)}\boldsymbol{\beta}_1\big) + \sum_{k=1}^{q_1} \theta_{1k} r_{t-k},$$

$$\vartheta_t = g2(\nu_t)$$

$$\eta_{2t} = g_{21}(\vartheta_t) = \alpha_2 + \boldsymbol{X}'_{2t}\boldsymbol{\beta}_2 + \sum_{i=1}^{p_2} \phi_{2i}\big(g_{22}(\vartheta_{t-i}) - I_{X_2}\boldsymbol{X}'_{2(t-i)}\boldsymbol{\beta}_2\big) + \sum_{k=1}^{q_2} \theta_{2k} g_{23}(e_{1,t-k}).$$

Fixed

```
d <- 0
```

Long-memory models with ν constant over time (ARFIMA-like models) (e.g. BARFIMA, GARFIMA,...)

$$\eta_{1t} = g_{11}(\mu_t) = \alpha_1 + \boldsymbol{X}'_{1t}\boldsymbol{\beta}_1 + \sum_{i=1}^{p_1} \phi_{1i}\big(g_{12}(Y_{t-i}) - I_{X_1}\boldsymbol{X}'_{1(t-i)}\boldsymbol{\beta}_1\big) + \sum_{k=1}^{\infty} c_{1k} r_{t-k},$$

$$\eta_{2t} = \alpha_2 = \nu.$$

Fixed

```
p <- c("user's p", 0)
q <- c("user's q", 0)
d <- c("user's d", 0)
xreg <- list(part1 = "user's regressors", part2 = NULL)
linkg <- list(g11 = "user's choice", g12 = "user's choice",
              g2 = "linear", g21 = "linear", g23 = "linear")
```

Reproducing Models from the Literature

This section summarizes how to replicate well-known time series models from the literature using the BTSR package. For each model type, we provide the necessary parameter settings and references to the original publications. These configurations act as templates, helping users correctly apply the package to reproduce results or extend established models.

Key arguments (e.g., `error.scale`, `xregar`, `y.lower`, `y.upper`, `rho`) should be set to match the specifications in the referenced articles. While we focus on the `btsr.*` functions (see BTSR.functions), all models can also be implemented using the corresponding parent model functions (for details, see BTSR.parent.models).

i.i.d. samples: The arguments `error.scale` and `xregar` are ignored.

- Beta distribution with parameters `shape1` and `shape2` compatible with the one from rbeta:

  ```
  model = "BETA"
  alpha = shape1/(shape1 + shape2)
  nu = shape1 + shape2
  ```

- Gamma distribution with parameters `shape` and `scale` compatible with the one from rgamma:

  ```
  model = "GAMMA"
  alpha = shape*scale
  nu = shape
  ```

- Kumaraswamy distribution with shape parameters `shape1` and `shape2` (respectively denoted by a and b in Kumaraswamy 1980):

  ```
  model = "KUMA"
  alpha = (y.lower - y.upper)*(1 - (1-rho)^1/shape2)*1/shape1 + y.lower
  nu = shape1
  ```

  Warning: Choose μ, ν and ρ carefully since $|\log(1-\rho)| >> |\log(1-\mu^{\nu})|$ may cause numerical instability.

- Matsuoka distribution with shape parameter `shape` (Matsuoka et al. 2024):

  ```
  model = "MATSU"
  alpha = (shape/(shape+1))^(3/2)
  ```

- Unit-Lindley distribution with parameter `theta` (Mazucheli et al. 2018):

  ```
  model = "UL"
  alpha = 1/(1 + theta)
  ```

- Unit-Weibull distribution with parameter `mu`, `beta` and `tau` from (Mazucheli et al. 2019):

```
model = "UW"
alpha = mu
nu = beta
rho = tau
```

Regression models: the argument `error.scale` and all entries but `g11` in `linkg` are ignored

- Beta regression (Ferrari and Cribari-Neto 2004): `model = "BREG"`
- Kumaraswamy regression (Mitnik and Baek 2013): `model = "KREG"`.
- Unit-Lindley regression (Mazucheli et al. 2018): `model = "ULREG"`.
- Unit-Weibull regression (Mazucheli et al. 2019): `model = "UWREG"`.

ARMA-like models

- BARMA model (Rocha and Cribari-Neto 2009; Rocha and Cribari-Neto 2017):

```
model = "BARMA"
error.scale = 1
xregar = TRUE
```

- KARMA model (Bayer et al. 2017):

```
model = "KARMA"
error.scale = 1
xregar = TRUE
y.lower = 0
y.upper = 1
rho = 0.5
```

- GARMA model (Prass et al. 2025):

```
model = "GARMA"
error.scale = 0
```

- MARMA model (Pumi et al. 2025):

```
model = "MARMA"
error.scale = 1
xregar = TRUE
```

- ULARMA model (Pumi et al. 2025):

```
model = "ULARMA"
error.scale = 1
xregar = TRUE
```

ARFIMA-like models

- BARFIMA model (Pumi et al. 2019):

```
model = "BARFIMA"
error.scale = 1
xregar = TRUE
d = TRUE (for fitting)
```

Chaotic models

- BARC model (Pumi et al. 2021): set `model = "BARC"` and `error.scale = 1`.

References

Bayer FM, Bayer DM, Pumi G (2017). "Kumaraswamy autoregressive moving average models for double bounded environmental data." *Journal of Hydrology*, **555**, 385–396. doi:10.1016/j.jhydrol.2017.10.006.

Ferrari SLP, Cribari-Neto F (2004). "Beta Regression for Modelling Rates and Proportions." *Journal of Applied Statistics*, **31**(7), 799–815. doi:10.1080/0266476042000214501.

Kumaraswamy P (1980). "A generalized probability density function for double-bounded random processes." *Journal of Hydrology*, **46**(1-2), 79–88. doi:10.1016/00221694(80)900360.

Matsuoka DH, Pumi G, Torrent HS, Valk M (2024). "A three-step approach to production frontier estimation and the Matsuoka's distribution." doi:10.48550/arXiv.2311.06086.

Mazucheli J, Menezes AFB, Fernandes LB, de Oliveira RP, Ghitany ME (2019). "The unit-Weibull distribution as an alternative to the Kumaraswamy distribution for the modeling of quantiles conditional on covariates." *Journal of Applied Statistics*. doi:10.1080/02664763.2019.1657813.

Mazucheli J, Menezes AJB, Chakraborty S (2018). "On the one parameter unit-Lindley distribution and its associated regression model for proportion data." *Journal of Applied Statistics*. doi:10.1080/02664763.2018.1511774.

Mitnik PA, Baek S (2013). "The Kumaraswamy distribution: median-dispersion re-parameterizations for regression modeling and simulation-based estimation." *Statistical Papers*, **54**, 177–192. doi:10.1007/s003620110417y.

Prass TS, Pumi G, Taufemback CG, Carlos JH (2025). "Positive time series regression models: theoretical and computational aspects." *Computational Statistics*, **40**, 1185–1215. doi:10.1007/s0018002401531z.

Pumi G, Matsuoka DH, Prass TS (2025). "A GARMA Framework for Unit-Bounded Time Series Based on the Unit-Lindley Distribution with Application to Renewable Energy Data." doi:10.48550/arXiv.2504.07351.

Pumi G, Matsuoka DH, Prass TS, Palm BG (2025). "A Matsuoka-Based GARMA Model for Hydrological Forecasting: Theory, Estimation, and Applications." doi:10.48550/arXiv.2502.18645.

Pumi G, Prass TS, Souza RR (2021). "A dynamic model for double bounded time series with chaotic driven conditional averages." *Scandinavian Journal of Statistics*, **48**(1), 68–86. doi:10.1111/sjos.12439.

Pumi G, Valk M, Bisognin C, Bayer FM, Prass TS (2019). "Beta autoregressive fractionally integrated moving average models." *Journal of Statistical Planning and Inference*, **200**, 196–212. doi:10.1016/j.jspi.2018.10.001.

Rocha AV, Cribari-Neto F (2009). "Beta autoregressive moving average models." *Test*, **18**, 529–545. doi:10.1007/s117490080112z.

Rocha AV, Cribari-Neto F (2017). "Erratum to: Beta autoregressive moving average models." *Test*, **26**, 451–459. doi:10.1007/s1174901705284.

See Also

BTSR.models, BTSR.model.defaults, get.defaults

`arguments.order` *Shared documentation for models order*

Description

This is the common documentation for arguments related to order of polynomials and truncation points for infinite sums, presented in BTSR models.

Arguments

`inf`	a length 1 or 2 integer vector given the truncation point for infinite sums. Default is `inf = 1000`. See the Section Model Order for details.
`p`	optional; a length 1 or 2 integer vector given the order of the AR polynomial (extract and fit only). Default is `p = NULL`. See the Section Model Order for details.
`q`	optional; a length 1 or 2 integer vector given the order of the MA polynomial (extract and fit only). Default is `q = NULL`. See the Section Model Order for details.
`d`	a length 1 or 2 logical vector indicating whether the long memory parameter d should be included in the model either as a fixed or non-fixed parameter (fit only). If `d = FALSE`, internally the value of the parameter `d` is fixed as 0. In this case, if `start` or `fixed.values` include `d`, the value provided by the user is ignored. If ν is time-varying and a single value is provided it is assumed that $d_1 = d_2 = d$.

Model Order

The coefficients $\{c_{ik}\}_{k\geq 0}$ are defined through the relation (see the section 'The BTSR Structure' in btsr-package)

$$c_i(z) := (1-L)^{-d_i}\theta_i(z) = \sum_{k=0}^{\infty} c_{ik} z^k, \quad i \in \{1, 2\}.$$

where $\theta_i(z) = \sum_{k=0}^{q_i} \theta_{ik} z^k$ is the moving average characteristic polynomial, with order q_i. For practical purposes, the following approximation is used

$$c_i(z) \approx \sum_{k=0}^{K_i} c_{ik} z^k,$$

for some K_i sufficiently large.

`inf` corresponds to the truncation point for all infinite sums using the coefficients $\{c_{ik}\}_{k\geq 0}$, $i \in \{1, 2\}$, including samples generation and derivatives calculation. It can be provided as either a single integer (legacy format) or a length 2 integer vector (new format) specifying the trunction points for `part1`/`part2`. If ν is time-varying and a single value is provided the same value is used for both parts. When $d = 0$, Fortran automatically sets `inf` to q (MA order).

By default `p` and `q` are set to `NULL`, in which case their values are computed internally, based on the size of the argument `phi` and `theta`, respectively, in the lists of coefficients (or staring values), fixed lags, and fixed values. For fitting purposes, if `p` (analogously, `q`) and `start` are both `NULL`, an error message is issued. These parameters can be provided as either a single integer (legacy format) or a length 2 integer vector (new format) specifying orders for `part1`/`part2`. If ν is time-varying and a single value of `p` (analogously, `q`) is provided it is assumed that $p_1 = p_2 = p$ (analogously, $q_1 = q_2 = q$).

`arguments.regressors` *Shared documentation for regressors*

Description

This is the common documentation for arguments related to the regressors.

Arguments

`xreg`	optional; external regressors. Can be specified as a vector, a matrix or a list. Default is `xreg = NULL`. For details, see the Section Regressors format.
`xnew`	optional; `nnew` new observations of the external regressors (extract and fit only). Follows the same format is the same as `xreg`. Default is `xnew = NULL`.
`xreg.start`	optional; initial value for the regressors (to initialize recursion). Can be specified as a vector or a list. Default is `xreg.start = NULL`, in which case, the average of the first p values (AR order) is used. Only relevant if `xreg` is provided, `xregar = TRUE` and $p > 0$. For details, see the Section Regressors format.
`xregar`	a length 1 or 2 logical vector indicating whether `xreg` should be included in the AR recursion for each part of the model. Default is `xregar = TRUE`. Only relevant if $p > 0$. If a single value is provided and ν is time-varying, the same option is assumed for both parts of the model. See the Section 'The BTSR structure' in btsr-package for details.

Regressors format

In-sample (`xreg`) and out-of-sample values (`xnew`) for regressors can be provided in two formats

- **Legacy structure:** a vector or matrix. Internally `xreg` is converted to `xreg = list(part1 = xreg, part2 = NULL)`. The same applies to `xnew`
- **New structure:** a list with elements `part1` (regressors for first model component) and `part2` (regressors for second model component), each being a vector or matrix.

`xreg.start` can be provided in two formats

- **Legacy structure:** a vector with initial values for each regressor. Internally `xreg.start` is converted to `xreg.start = list(part1 = xreg.start, part2 = NULL)`.
- **New structure:** a list with elements `part1` and `part2`, each a vector of initial values for the respective regressors.

The following rules apply to `xreg`, `xnew` and `xreg.start`

- if `model` corresponds to a case where ν is constant over time (e.g., `model = "BARMA"`), `part2` is ignored.
- For simulation, regressors must include `n + burn` observations.
- For model fitting, parameter initialization, or component extraction, the number of regressor observations must match the length of the observed time series `yt`.
- When `xreg = NULL` or `nnew = 0`, `xnew` is ignored. If `nnew > 0` and the number of regressors in `xnew` does not match `xreg` an error message is issued.
- If starting values for `xreg` are not provided and $p_i > 0$ for the ith part of the model, the default behavior is to assume

$$\boldsymbol{X}_t = \frac{1}{p_i}\sum_{k=1}^{p_i}\boldsymbol{X}_k, \quad \text{for } t < 1.$$

`arguments.series` *Shared documentation for the time series*

Description

This is the common documentation for arguments related to the observed/simulated time series and its conditional distribution.

Arguments

`n`	the sample size of the output time series `yt` after burn-in (simulation only). Default is `n = 1`.
`nnew`	optional; the number of out-of sample predicted values required (extract and fit only). Default is `nnew = 0`.
`burn`	the length of the 'burn-in' period (simulation only). Default is `burn = 0`. The first `burn` values of the time series are discarded.
`yt`	numeric vector with the observed time series (extract and fit only). Missing values (NA's) are not allowed.
`y.start`	optional; an initial value for Y_t (to initialize recursions when $t < 1$). Default is `y.start = NULL`, in which case, the recursion assumes that $Y_t = g_{12}^{-1}(0)$, for $t < 1$. Only relevant if $p > 0$.
`rho`	the quantile being considered in the conditional distribution of Y_t (only present in Kumaraswamy and Unit-Weibull based models). It can be any positive number between 0 and 1. Default is `rho = 0.5`, which corresponds to the median.

y.lower	the lower limit for the Kumaraswamy density support. Default is y.lower = 0.
y.upper	the upper limit for the Kumaraswamy density support. Default is y.upper = 1.
vt.start	optional; an initial value for ϑ_t (to initialize recursions when $t < 1$). Default is vt.start = NULL, in which case, the recursion assumes that $\vartheta_t = g_{22}^{-1}(0)$, for $t < 1$. Only relevant if ν is time-varying and $p_2 > 0$.
e2.start	optional; an initial value for $g_{23}(e_{1t})$ (to initialize recursions when $t < 1$). Default is e2.start = NULL, in which case, the recursion assumes that $e_{1t} = g_{23}^{-1}(0)$, for $t < 1$. Only relevant if ν is time-varying and $q_2 > 0$ or $d_2 > 0$.

BARC.functions *Functions to simulate, extract components and fit BARC models*

Description

These functions can be used to simulate, extract components and fit any model of the class barc. A model with class barc is a special case of a model with class btsr. See the Section 'The BTSR structure' in btsr-package for more details on the general structure. See also 'Details' below.

Usage

```
BARC.sim(n = 1, burn = 0, y.start = NULL, xreg = NULL,
  xreg.start = NULL, xregar = TRUE, coefs = NULL, map = 4,
  error.scale = 0, linkg = "linear", configs.linkg = NULL,
  linkh = "linear", configs.linkh = list(ctt = 1, power = 1),
  complete = FALSE, debug = FALSE)

BARC.extract(yt, y.start = NULL, xreg = NULL, xreg.start = NULL,
  xnew = NULL, xregar = TRUE, nnew = 0, p = NULL, coefs = NULL,
  lags = NULL, fixed.values = NULL, fixed.lags = NULL, error.scale = 0,
  map = 4, linkg = "linear", configs.linkg = NULL, linkh = "linear",
  configs.linkh = list(ctt = 1, power = 1), llk = TRUE, sco = FALSE,
  info = FALSE, debug = FALSE)

BARC.fit(yt, y.start = NULL, xreg = NULL, xreg.start = NULL,
  xregar = TRUE, xnew = NULL, nnew = 0, p = NULL,
  ignore.start = FALSE, start = NULL, lags = NULL, fixed.values = NULL,
  fixed.lags = NULL, lower = NULL, upper = NULL, map = 4,
  linkg = "linear", configs.linkg = NULL, linkh = "linear",
  configs.linkh = list(ctt = 1, power = 1), sco = FALSE, info = FALSE,
  error.scale = 0, control = NULL, report = TRUE, debug = FALSE, ...)
```

Arguments

n	a strictly positive integer. The sample size of yt (after burn-in). Default is n = 1.
burn	a non-negative integer. length of "burn-in" period. Default is burn = 0.

`y.start` optionally, an initial value for Y_t (to be used in the recursions). Default is `y.start = NULL`, in which case, the recursion assumes that $Y_t = g_{12}^{-1}(0)$, for $t < 1$. Only relevant if $p > 0$.

`xreg` optionally, a vector or matrix of external regressors. Default is `xreg = NULL`. For simulation purposes, the length of `xreg` must be equal to `n + burn`. For extraction or fitting purposes, the length of `xreg` must be the same as the length of the observed time series Y_t.

`xreg.start` optionally, a vector of initial value for $\boldsymbol{X}_t$ (to be used in the recursions). Default is `xreg.start = NULL`, in which case, the average of the first p values is used, that is, the recursion assumes that $\boldsymbol{X}_t = p^{-1}\sum_{k=1}^{p}\boldsymbol{X}_k$, for $t < 1$. Only relevant if `xregar = TRUE` and $p > 0$.

`xregar` logical; indicates whether `xreg` should be included in the AR recursion of the model. Default is `xregar = TRUE`. Only relevant if `xreg` is included and $p > 0$. See the Section 'The BTSR structure' in btsr-package for details.

`coefs` a list with the coefficients of the model. An empty list will result in an error. The arguments that can be passed through this list are

- `alpha`: optionally, a numeric value corresponding to the intercept. If the argument is missing, it will be treated as zero.
- `beta`: optionally, a vector of coefficients corresponding to the regressors in `xreg`. For simulation purposes, if `xreg` is provided but `coefs` does not have a `beta` argument, an error message is issued. The extracting function also verify the `fixed.values` list before issuing an error message.
- `phi`: optionally, for the simulation function this must be a vector of size, p corresponding to the autoregressive coefficients (including the ones that are zero), where p is the AR order. For the extraction and fitting functions, this is a vector with the non-fixed values in the vector of autoregressive coefficients.
- `theta` the parameter (or vector of parameters) corresponding to the map function. If `map = 5` this value is ignored. For simulation, purposes, the default is `map = 4` and `theta = 0.5`. **Note:** Do not confuse `theta` from a BARC model with the moving average term in the general BTSR class of models
- `nu` the dispersion parameter. If missing, an error message is issued.
- `u0` a numeric value in the interval $(0, 1)$, corresponding to the value of the random variable U_0. For simulation purposes, the default is `u0 = pi/4`.

For simulation purposes, an empty list will result in an error message. For extraction purposes, an error message will be issued if both `coefs` and `fixed.values` are empty. The argument `coefs` is not used when fitting a model. Missing parameters are treated as zero.

`map` a non-negative integer from 1 to 5 corresponding to the map function. Default is `map = 4`. See the Section The map function.

`error.scale` the scale for the error term. Default is `error.scale = 0` (data scale).

`linkg` character or a two character vector indicating which links must be used in the model. See the Section 'The BTSR structure' in btsr-package for details and link.btsr for valid links. If only one value is provided, the same link is used for μ_t and for Y_t in the AR part of the model. Default is `linkg = "linear"`.

`configs.linkg` a list with two elements, `ctt` and `power`, which define the constant a and the exponent b in the link function $g(x) = ax^b$. Each element can be a single numeric value (applied uniformly across all linear links), a numeric vector of length 2, or a named list with entries `g11` and `g12`. This argument is only used when the link function is `"linear"` or `"polynomial"`. The default is `configs.linkg = NULL`, in which case the function internally assumes `configs.linkg = list(ctt = list(g11 = 1, g12 = 1), power = list(g11 = 1, g12 = 1))`.

`linkh` a character indicating which link must be associated to the chaotic process. See the Section 'The BTSR structure' in btsr-package for details and link.btsr for valid links. Default is `linkh = "linear"`.

`configs.linkh` a list with extra configurations for the link h. For now, only used if `linkh = "linear"` or `"polynomial"`. Default is `configs.linkh = list(ctt = 1, power = 1)`.

`complete` logical; if FALSE returns only `yt`, else returns additional components. Default is `complete = FALSE`.

`debug` logical, if `TRUE` the output from Fortran is return (for debugging purposes). Default is `debug = FALSE`.

`yt` a numeric vector with the observed time series. If missing, an error message is issued.

`xnew` a vector or matrix, with `nnew` observations of the regressors observed/predicted values corresponding to the period of out-of-sample forecast. Default is `xreg = NULL`. If `xreg = NULL` or `nnew = 0`, `xnew` is ignored. If `nnew > 0` and the number of regressors in `xnew` does not match `xreg` an error message is issued.

`nnew` optionally, the number of out-of sample predicted values required. Default is `nnew = 0`.

`p` optionally, a non-negative integer. The order of the AR polynomial. Default is `p = NULL`, in which case the value of `p` is computed internally, based on the size of the argument `phi` in the lists of coefficients (or staring values), fixed lags, and fixed values. For fitting purposes, if `p` and `start` are both `NULL`, an error message is issued.

`lags` optionally, a list with the lags that the values in `coefs` correspond to. The names of the entries in this list must match the ones in `coefs` (or `start`). For one dimensional coefficients, the `lag` is obviously always 1 and can be suppressed. The default is `lags = NULL`, in which the `lags` are computed from the `fixed.lags` argument (if provided). If both, `lags` and `fixed.lags` are missing, it is assumed that all lags must be used. The arguments `lags` and `fixed.lags` are complementary. Either suffices, or mix them (e.g., `lags` for some parameters, `fixed.lags` for others).

`fixed.values` optionally, a list with the values of the coefficients that are fixed. The default is `fixed.lags = NULL`. By default, if a given vector (such as the vector of AR coefficients) has fixed values and the corresponding entry in this list is empty, the fixed values are set as zero. The names of the entries in this list must match the ones in `coefs` (or `start`).

`fixed.lags` optionally, a list with the lags that the fixed values in `fixed.values` correspond to. The names of the entries in this list must match the ones in `fixed.values`.

For one dimensional coefficients, the `lag` is obviously always 1 and can be suppressed. If an empty list is provided and the model has fixed lags, the argument `lags` is used as reference.

`llk` logical; indicates whether the value of the log-likelihood function should be returned. Default is `llk = TRUE`.

`sco` logical; indicates whether the score vector should be returned. Default is `sco = FALSE`. For now, the score vector is computed using numerical derivatives.

`info` logical; indicates whether the information matrix should be returned. Default is `info = FALSE`. For the fitting function, `info` is automatically set to `TRUE` when `report = TRUE`. For now, the information matrix is computed using numerical derivatives.

`ignore.start` logical; indicates whether the argument `start` should be ignored. If starting values are not provided, the function uses the default values and `ignore.start` is ignored. In case starting values are provided and `ignore.start = TRUE`, those starting values are ignored and recalculated. The default is `ignore.start = FALSE`.

`start` a list with the starting values for the non-fixed coefficients of the model. The default is `start = NULL`, in which case the function coefs.start is used internally to obtain starting values for the parameters.

`lower` optionally, list with the lower bounds for the parameters. The names of the entries in these lists must match the ones in `start`. Default is `lower = NULL`. The default is to assume that the parameters have no lower bound except for `nu`, for which de default is 0. Only the bounds for bounded parameters need to be specified.

`upper` optionally, list with the upper bounds for the parameters. The names of the entries in these lists must match the ones in `start`. Default is `upper = NULL`. The default is to assume that the parameters have no upper bound. Only the bounds for bounded parameters need to be specified.

`control` a list with configurations to be passed to the optimization subroutines. Default is `control = NULL`. Missing arguments will receive default values. For details, see fit.control.

`report` logical; indicates whether the summary from the fitted model should be be printed. Default is `report = TRUE`, in which case `info` is automatically set to `TRUE`.

`...` further arguments passed to the internal functions. See, for instance, summary.btsr for details.

Details

Sim, Extract and Fit functions:

The function `BARC.sim` generates a random sample from a BARC(p) model.

The function `BARC.extract` allows the user to extract the components Y_t, μ_t, $\eta_t = g(\mu_t)$, e_t, $T^t(U_0)$, the log-likelihood, the score vector and the information matrix associated to a given set of parameters. This function can be used by any user to create an objective function that can be passed to optimization algorithms not available in the BTSR Package.

The function `BARC.fit` fits a BARC model to a given univariate time series. For now, available optimization algorithms are `"L-BFGS-B"` and `"Nelder-Mead"`. Both methods accept bounds for the parameters. For `"Nelder-Mead"`, bounds are set via parameter transformation.

Particular cases:

Neither the beta regression or an i.i.d. sample from a beta distribution can be obtained as special cases of the BARC model since the term $h(T(U_0))$ is always present.

The model from Pumi et al. (2021) is obtained by setting `xregar = TRUE` (so that the regressors are included in the AR part of the model) and using the same link for Y_t and μ_t.

Value

By default, the function `BARC.sim` returns the simulated time series `yt`. If `complete = TRUE`, it returns a list with the following components

- `model`: string with the text `"BARC"`
- `yt`: the simulated time series Y_t
- `mut`: the conditional mean μ_t
- `etat`: the linear predictor $\eta_t = g_{11}(\mu_t)$
- `u0`: the starting values of U_0
- `Ts`: the chaotic process $T^t(U_0)$
- `error`: the error term e_{1t}
- `out.Fortran`: the output from FORTRAN (if requested).

The function `BARC.extract` returns a list with the following components.

- `model`: string with the text `"BARC"`
- `yt`: the observed time series Y_t
- `TS`: the chaotic process $T^t(U_0)$.
- `mut`: the conditional mean μ_t
- `etat`: the linear predictor $\eta_t = g_{11}(\mu_t)$
- `error`: the error term e_{1t}
- `forecast`: the out-of-sample forecast (if requested)
- `xnew`: the out-of-sample values of `xreg` provided by the user (only present if the model includes regressors and forecast is requested)
- `sll`: the sum of the conditional log-likelihood (if requested)
- `score`: the score vector (if requested)
- `info.Matrix.`: the score vector (if requested)
- `out.Fortran`: FORTRAN output (if requested)

The function `BARC.fit` returns a list with the following components.

- `model`: string with the text `"BARC"`
- `call`: string with a complete description of the model, including the AR and MA order.

- n: the sample size used for estimation.
- series: the observed time series Y_t
- gyt: a vector or a matrix with the transformed time series $g_{11}(Y_t)$ and $g_{12}(Y_t)$. Only returns a matrix if the links g_{11} and g_{12} are not the same.
- xreg: a vector or matrix of regressors $\boldsymbol{X}_t$ (if included in the model).
- control: a list of control parameters.
- convergence: An integer code. 0 indicates successful completion. The error codes depend on the algorithm used.
- message: A character string giving any additional information returned by the optimizer (if any), or NULL.
- counts: an integer giving the number of function evaluations.
- start: the starting values used by the algorithm.
- coefficients: The best set of parameters found.
- fitted.values: the conditional time series μ_t and the chaotic process $T^t(U_0)$, which corresponds to the in-sample forecast, also denoted fitted values.
- etat: the linear predictor $\eta_{1t} = g_{11}(\mu_t)$
- error: the error term e_{1t}
- residual: the observed values Y_t minus the fitted values μ_t. The same as the error term if error.scale = 0.
- forecast: a matrix with the out-of-sample forecast (if requested) for μ_t and η_{1t}
- xnew: the observations of the regressors observed/predicted values corresponding to the period of out-of-sample forecast. Only included if xreg is not NULL and nnew > 0.
- sll: the sum of the conditional log-likelihood (if requested)
- score: the score vector (if requested)
- info.Matrix: the information matrix (if requested)
- link: the codes for the link functions (for summary purposes)
- configs: a list with the configurations passed to FORTRAN to fit the model. This information is used by the prediction function.
- out.Fortran: FORTRAN output (if requested).

The map function

The map function $T : [0,1] \to [0,1]$ in BARC models is a dynamical system, i.e., a function, potentially depending on a r-dimensional vector of parameters θ. As for today, for all implemented maps, $r = 1$.

Available choices are

- map = 1, $\theta = k$, for k integer greater or equal to 2.

$$T(u) = (ku) (\text{mod } 1)$$

- map = 2, $0 \leq \theta \leq 1$

$$T(u) = \frac{u}{\theta} I(u < \theta) + \theta \frac{(u-\theta)}{(1-\theta)} I(u \geq \theta)$$

- map = 3 (logistic map), $0 \leq \theta \leq 4$,

$$T(u) = \theta(1-\theta)$$

- map = 4 (Manneville-Pomeau map), $0 < \theta < 1$

$$T(u) = (u + u^{1+\theta})(\text{mod } 1)$$

- map = 5 (Lasota-Mackey's map),

$$T(u) = \frac{u}{(1-u)} I(u \leq 0.5) + (2u-1) I(u > 0.5)$$

References

Pumi G, Prass TS, Souza RR (2021). "A dynamic model for double bounded time series with chaotic driven conditional averages." *Scandinavian Journal of Statistics*, **48**(1), 68–86. doi:10.1111/sjos.12439.

See Also

BTSR.functions: sim, extract and fit functions for BTSR models

BTSR.parent.models: sim, extract and fit functions for parent models

get.defaults: Retrieve default arguments for BTSR package functions

Examples

```
##############################################################################
#
#   Example of usage of BARC.sim, BARC.extract and BARC.fit
#
##############################################################################

#-------------------------------------------------------------
# Generating a sample from a BARC model
#-------------------------------------------------------------
set.seed(1234)
m1 <- BARC.sim(
  coefs = list(nu = 15, theta = 0.85, u0 = pi / 4),
  linkg = "linear",
  linkh = "linear",
  configs.linkh = list(ctt = 0.6),
  n = 100,
  complete = TRUE
)

plot.ts(m1$yt)
lines(m1$mut, col = "red")

#-------------------------------------------------------------
#  Extracting the conditional time series given yt and
```

```
#  a set of parameters
#-------------------------------------------------------------

e1 <- BARC.extract(
  yt = m1$yt,
  map = 4,
  coefs = list(nu = 15, theta = 0.85),
  fixed.values = list(u0 = pi / 4),
  linkg = "linear",
  linkh = "linear",
  configs.linkh = list(ctt = 0.6),
  llk = TRUE,
  sco = TRUE,
  info = TRUE
)

#-----------------------------------------------------
# comparing the simulated and the extracted values
#-----------------------------------------------------
cbind(head(m1$mut), head(e1$mut))

#-------------------------------------------------------------
# the log-likelihood, score vector and information matrix
# score vector and information matrix are obtained
# numerically.
#-------------------------------------------------------------
e1$sll
e1$score
e1$info.Matrix

#-------------------------------------------------------------
#  Fitting a BARC model. Assuming only alpha fixed.
#-------------------------------------------------------------
f1 <- BARC.fit(
  yt = m1$yt,
  map = 4,
  configs.linkh = list(ctt = 0.6),
  start = list(nu = 10, theta = 0.6, u0 = 0.5),
  lower = list(nu = 0, theta = 0, u0 = 0),
  upper = list(theta = 1, u0 = 1),
  fixed.values = list(alpha = 0),
  control = list(iprint = -1, method = "Nelder-Mead")
)

coefficients(f1)

plot.ts(m1$yt)
lines(f1$fitted.values[, "mut"], col = "red")

#-------------------------------------------------------------
#  Out-of-sample forecast
#-------------------------------------------------------------
pred <- predict(f1, nnew = 5)
```

```
pred$forecast
```

`BTSR.functions` *Generic functions to simulate, extract components and fit BTSR models*

Description

These generic functions can be used to simulate, extract components and fit any model of the class `btsr`. See 'Details' below.

The package handles function arguments in two compatible formats

- **Legacy structure** (pre-1.0.0). Used for models with fixed or no ν parameter. Automatically converted to the new format when processed.
- **New structure** (1.0.0+). Required for models with time-varying ν parameter.

All functions accept both formats seamlessly, ensuring backward compatibility. The internal processing automatically standardizes to the new structure.

Usage

```
btsr.sim(model, ...)

btsr.extract(model, ...)

btsr.fit(model, ...)
```

Arguments

`model`	character string (case-insensitive) indicating the model to be fitted to the data. Must be one of the options listed in the Section Supported Models.
`...`	further arguments passed to the internal functions. See, for instance, summary.btsr for details.

Details

A detailed description of the general structure (mathematical formulation) of BTSR models, associated to the `btsr` class, is presented in Section 'The BTSR Structure' of btsr-package. Particular models are discussed in arguments.model.

All functions are compatible with the new format for the arguments, introduced in version 1.0.0. and the previous format.

- The function `btsr.sim` is used to generate random samples from any BTSR models.

- The function `btsr.extract` allows the user to extract all conditional time series, the log-likelihood, and the vectors and matrices used to calculate the score vector and the information matrix associated to a given set of parameters. This function can be used by any user to create an objective function that can be passed to optimization functions not available in BTSR Package. At this point, there is no other use for which this function was intended.
- The function `btsr.fit` fits a BTSR model to a given univariate time series. For now, available optimization algorithms are `"L-BFGS-B"` and `"Nelder-Mead"`. Both methods accept bounds for the parameters. For `"Nelder-Mead"`, bounds are set via parameter transformation.

For compatibility with previous versions of the package, all functions associated to parent models (e.g. BARFIMA) are still available (see BTSR.parent.models). Also, analogous functions are available for parent models with time varying ν (e.g. BARFIMAV). The list of arguments and default values for these specific functions can be accessed using the function get.defaults.

Particular models (e.g. BETA, BARMA) share the same arguments as the parent model, however, some arguments can have different default values (see the documentation for shared arguments for details). Information on the parent model can be obtained using the function BTSR.model.defaults.

Value

By default, the *simulation function* return the simulated time series `yt`. If `complete = TRUE`, it returns a list with the following components

- `model`: string with the name of the model.
- `yt`: the simulated time series Y_t
- `mut`: the conditional mean μ_t
- `etat`: the linear predictor(s)
  For all models where ν is constant over time, returns $\eta_{1t} = g_{11}(\mu_t)$
  For models with time varying ν, returns a matrix whose columns are $\eta_{1t} = g_{11}(\mu_t)$ and $\eta_{2t} = g_{21}(\vartheta_t)$.
- `nut`: the conditional precision ν_t (only for models with time varying ν)
- `varthetat`: the transformed time series $\vartheta_t = g_2(\nu_t)$. (only for models with time varying ν)
  For BARFIMAV models, if `g2 = "default"`, then `varthetat` is the conditional dispersion given by $\vartheta_t = (1 + \nu_t)^{-1}$.
- `error`: the error term e_{1t}
- `out.Fortran`: the output from Fortran (if requested).

The *extraction function* returns a list with the following components

- `model`: string with the name of the model.
- `yt`: the simulated time series Y_t
- `mut`: the conditional mean μ_t
- `etat`: the linear predictor(s)
  For models with ν fixed, returns $\eta_{1t} = g_{11}(\mu_t)$
  For models with time varying ν, returns a matrix whose columns are $\eta_{1t} = g_{11}(\mu_t)$ and $\eta_{2t} = g_{21}(\vartheta_t)$.

- nut: the conditional precision ν_t (only for models with time varying ν)
- varthetat: the transformed time series $\vartheta_t = g_2(\nu_t)$. (only for models with time varying ν) For BARFIMAV models, if g2 = "default", then varthetat is the conditional dispersion given by $\vartheta_t = (1 + \nu_t)^{-1}$.
- error: the error term e_{1t} (depends on the argument error.scale)
- forecast: the out-of-sample forecast. (if requested)
  If ν_t is fixed: a vector with the predicted values for μ_t and η_{1t}
  If ν_t is time varying: a matrix the predicted values for μ_t and η_{1t}, ν_t, ϑ_t and η_{2t}.
- xnew: the observations of the regressors observed/predicted values corresponding to the period of out-of-sample forecast. Only included if xreg is not NULL and nnew > 0.
- sll: the sum of the conditional log-likelihood (if requested)
- score: the score vector (if requested)
- info.Matrix.: the score vector (if requested)
- D, T, E, h: additional matrices and vectors used to calculate the score vector and the information matrix. (if requested)
- out.Fortran: FORTRAN output (if requested)

The *fitting function* returns a list with the following components.

- model: character; the same as the input argument.
- call: string with a complete description of the model, including the AR and MA order.
- n: the sample size used for estimation.
- series: the observed time series Y_t
- gyt: a vector or a matrix with the transformed time series $g_{11}(Y_t)$ and $g_{12}(Y_t)$. Only returns a matrix if the links g_{11} and g_{12} are not the same.
- xreg: a vector or matrix of regressors $\boldsymbol{X}_t$ (if included in the model).
- control: a list of control parameters.
- convergence: An integer code. 0 indicates successful completion. The error codes depend on the algorithm used.
- message: A character string giving any additional information returned by the optimizer (if any), or NULL.
- counts: an integer giving the number of function evaluations.
- start: the starting values used by the algorithm.
- coefficients: The best set of parameters found.
- fitted.values: in-sample forecast.
  If ν_t is fixed: a vector with the in-sample value of μ_t.
  If ν_t is time varying: a matrix with the in-sample values of μ_t, ν_t and ϑ_t.
- etat: the linear predictor(s)
  For models with ν fixed, returns $\eta_{1t} = g_{11}(\mu_t)$
  For models with time varying ν, returns a matrix whose columns are $\eta_{1t} = g_{11}(\mu_t)$ and $\eta_{2t} = g_{21}(\vartheta_t)$.
- error: the error term e_{1t} (depends on the argument error.scale)

- `residual`: the observed values Y_t minus the fitted values μ_t. The same as the `error` term if `error.scale = 0`.
- `forecast`: the out-of-sample forecast. (if requested)
  If ν_t is fixed: a vector with the predicted values for μ_t and η_{1t}
  If ν_t is time varying: a matrix the predicted values for μ_t and η_{1t}, ν_t, ϑ_t and η_{2t}.
- `xnew`: the observations of the regressors observed/predicted values corresponding to the period of out-of-sample forecast. Only included if `xreg` is not `NULL` and `nnew > 0`.
- `sll`: the sum of the conditional log-likelihood (if requested)
- `score`: the score vector (if requested)
- `info.Matrix`: the information matrix (if requested)
- `link`: the codes for the link functions (for summary purposes)
- `configs`: a list with the configurations passed to FORTRAN to fit the model. This information is used by the prediction function.
- `out.Fortran`: FORTRAN output (if requested).

See Also

BARC.functions: sim, extract and fit functions for BARC models

BTSR.parent.models: sim, extract and fit functions for parent models

get.defaults: Retrieve default arguments for BTSR package functions

Examples

```
#############################################################################
#
#   Examples of usage of btsr.sim
#
#############################################################################
#---------------------------------------------------------------------------
# Generating a Beta model were both mu and nu do not vary with time
# yt ~ Beta(a,b), a = mu*nu, b = (1-mu)*nu
#---------------------------------------------------------------------------

# CASE 1: using the legacy format for coefs
set.seed(1234)
y1 <- btsr.sim(
  model = "BETA", n = 1000,
  coefs = list(alpha = 0.2, nu = 20)
)
hist(y1)

# CASE 2: using the new layout for coefs
set.seed(1234)
y2 <- btsr.sim(
  model = "BETA", n = 1000,
  coefs = list(part1 = list(alpha = 0.2), part2 = list(alpha = 20))
)
hist(y2)
```

```
# CASE 3: function for the parent model plus legacy format for coefs.
# - requires setting linkg = "linear", otherwhise the default "logit"
#   link is used.
set.seed(1234)
y3 <- BARFIMA.sim(
  linkg = "linear", n = 1000,
  coefs = list(alpha = 0.2, nu = 20)
)
hist(y3)

# CASE 4: function for the parent model plus new format for coefs.
# - requires setting linkg = "linear", otherwhise the default "logit"
#   link is used.
set.seed(1234)
y4 <- BARFIMA.sim(
  n = 1000, linkg = "linear",
  coefs = list(part1 = list(alpha = 0.2), part2 = list(alpha = 20))
)
hist(y4)

# comparing the results:
range(abs(y2 - y1))
range(abs(y3 - y1))
range(abs(y3 - y4))

#-------------------------------------------------------------------------
# Generating a sample from a Beta regression model
#-------------------------------------------------------------------------
burn <- 100
n <- 500
N <- n + burn
covar <- cbind(sin(2 * pi * (1:N) / 50), 1:N)

set.seed(1234)
y1 <- btsr.sim(
  model = "BREG", linkg = "logit",
  n = n, burn = burn, xreg = covar,
  coefs = list(alpha = -1.3, beta = c(0.6, 0.002), nu = 30),
  complete = TRUE
)

# The regressors: X1 = sin(2*pi*t/50) and X2 = t
plot.ts(
  covar,
  main = "Regressors" ~ X[1][t] == sin(2 * pi * t / 50) ~ "and" ~ X[2][t] == t
)

# Conditional time series:
plot.ts(y1$etat, main = "Linear predictor" ~ eta[t] == g[11](mu[t]))
plot.ts(y1$mut, main = "Conditional mean" ~ mu[t])
plot.ts(y1$yt, main = "Time series" ~ Y[t])
```

```
#############################################################################
#
#   Examples of usage of btsr.extract
#
#############################################################################
#---------------------------------------------------------------------------
# Generating a sample from a BARMAX(1,1) model (BARMA with covariates)
#---------------------------------------------------------------------------
burn <- 100
n <- 500
N <- n + burn
covar <- cbind(sin(2 * pi * (1:N) / 50), 1:N)

set.seed(1234)
m1 <- btsr.sim(
  model = "BARMA", linkg = "logit",
  n = n, burn = burn, xreg = covar,
  coefs = list(
    alpha = 0, phi = -0.65, theta = -0.25,
    beta = c(0.6, -0.002), nu = 20
  ),
  error.scale = 1, complete = TRUE
)

# Extracting components assuming that all coefficients are non-fixed
e1 <- btsr.extract(
  model = "BARMA", yt = m1$yt,
  xreg = covar[(burn + 1):N, ], linkg = "logit",
  coefs = list(
    alpha = 0, phi = -0.65, theta = -0.25,
    beta = c(0.6, -0.002), nu = 20
  ),
  llk = TRUE, sco = TRUE, info = TRUE
)

# Extracting components assuming that all coefficients are fixed
#  - no need to provide fixed.lags in this case.
e2 <- btsr.extract(
  model = "BARMA", yt = m1$yt,
  xreg = covar[(burn + 1):N, ], linkg = "logit",
  fixed.values = list(
    alpha = 0, phi = -0.65, theta = -0.25,
    beta = c(0.6, -0.002), nu = 20
  ),
  llk = TRUE, sco = TRUE, info = TRUE
)

# Extracting components assuming that some coefficients are fixed
#  - e3 and e4 give the same result
#  - e3 uses legacy format for all arguments
#  - e4 uses the new format for all arguments (not optimal here)
e3 <- btsr.extract(
  model = "BARMA", yt = m1$yt,
```

```
  xreg = covar[(burn + 1):N, ], linkg = "logit",
  coefs = list(
    phi = -0.65, theta = -0.25,
    beta = c(0.6, -0.002)
  ),
  fixed.values = list(alpha = 0, nu = 20),
  llk = TRUE, sco = TRUE, info = TRUE
)

e4 <- btsr.extract(
  model = "BARMA", yt = m1$yt,
  xreg = list(part1 = covar[(burn + 1):N, ]),
  linkg = list(g11 = "logit"),
  coefs = list(part1 = list(
    phi = -0.65, theta = -0.25,
    beta = c(0.6, -0.002)
  )),
  fixed.values = list(
    part1 = list(alpha = 0),
    part2 = list(alpha = 20)
  ),
  llk = TRUE, sco = TRUE, info = TRUE
)

#---------------------------------------------------
# comparing the simulated and the extracted values
#---------------------------------------------------
cbind(head(m1$mut), head(e1$mut), head(e2$mut), head(e3$mut), head(e4$mut))

#---------------------------------------------------
# comparing the log-likelihood values obtained (must be the all equal)
#---------------------------------------------------
c(e1$sll, e2$sll, e3$sll, e4$sll)

#---------------------------------------------------
# comparing the score vectors:
#---------------------------------------------------
# - e1 must have 6 values: dl/dro values and dl/dlambda values
# - e2 must be empty
# - e3 and e4 must have only the values corresponding
#    to the non-fixed coefficient. The value sof
#    dl/dlambda are the same as in e1, but dl/drho are
#    differente since the mixed derivatives are not present.
#---------------------------------------------------
round(e1$score, 4)
e2$score
e3$score
e4$score

#---------------------------------------------------
# comparing the information matrices.
#---------------------------------------------------
# - e1 must be a 6x6 matrix
```

```
# - e2 must be empty
# - e3 and e4 must have only the value corresponding
#    to the non-fixed coefficient
#-------------------------------------------------
round(e1$info.Matrix, 4)
e2$info.Matrix
e3$info.Matrix
e4$info.Matrix

#-----------------------------------------------------------------------
# Generating a sample from a BARFIMAVX(1,d1,1)x(1,0,1) with d1 = 0.35
# (BARFIMAV with covariates)
# Here using the nre format for coefficients and xreg is required.
#-----------------------------------------------------------------------
burn <- 100
n <- 500
N <- n + burn
covar1 <- cbind(sin(2 * pi * (1:N) / 50), 1:N)
covar2 <- sin(2 * pi * (1:N) / 25)

set.seed(1234)
m1 <- btsr.sim(
  model = "BARFIMAV",
  linkg = list(g11 = "logit", g2 = "default", g21 = "logit"),
  n = n, burn = burn, xreg = list(part1 = covar1, part2 = covar2),
  coefs = list(
    part1 = list(
      alpha = 0, d = 0.35, phi = -0.65,
      theta = -0.25, beta = c(0.6, -0.002)
    ),
    part2 = list(
      alpha = -3, phi = 0.25,
      theta = -0.2, beta = -0.15
    )
  ),
  error.scale = 1, complete = TRUE, vt.start = 0.02
)


# Extracting components assuming that all coefficients are non-fixed
e1 <- btsr.extract(
  model = "BARFIMAV", yt = m1$yt,
  xreg = list(part1 = covar1[(burn + 1):N, ], part2 = covar2[(burn + 1):N]),
  linkg = list(g11 = "logit", g2 = "default", g21 = "logit"),
  coefs = list(
    part1 = list(
      alpha = 0, d = 0.35, phi = -0.65,
      theta = -0.25, beta = c(0.6, -0.002)
    ),
    part2 = list(
      alpha = -3, phi = 0.25,
      theta = -0.2, beta = -0.15
    )
```

```
  ),
  vt.start = 0.02,
  llk = TRUE, sco = TRUE, info = TRUE, extra = TRUE, debug = TRUE
)

# score vector
e1$score

# information matrix
e1$info.Matrix

#############################################################################
#
#   Examples of usage of btsr.fit
#
#############################################################################
#---------------------------------------------------------------------------
#  Generating a sample from a BARMAVX(1,0)x(0,1) (BARMAV with covariates)
#---------------------------------------------------------------------------
burn <- 100
n <- 500
N <- n + burn
covar1 <- cbind(sin(2 * pi * (1:N) / 50), 1:N)
covar2 <- sin(2 * pi * (1:N) / 25)

set.seed(1234)
m1 <- btsr.sim(
  model = "BARMAV",
  linkg = list(g11 = "logit", g2 = "default", g21 = "logit"),
  n = n, burn = burn, xreg = list(part1 = covar1),
  coefs = list(
    part1 = list(
      alpha = 0, phi = -0.3,
      beta = c(0.6, -0.002)
    ),
    part2 = list(alpha = -2.5, theta = -0.4)
  ),
  error.scale = 1, complete = TRUE
)

# fitting the model
f1 <- btsr.fit(
  model = "BARMAV", yt = m1$yt, report = FALSE, info = TRUE,
  xreg = list(part1 = covar1[(burn + 1):N, ]),
  linkg = list(g11 = "logit", g2 = "default", g21 = "logit"),
  p = c(1, 0), q = c(0, 1)
)

# fitting the model using the name string for the parent model
#   - same result
f2 <- btsr.fit(
  model = "BARFIMAV", yt = m1$yt, report = FALSE, info = TRUE,
  xreg = list(part1 = covar1[(burn + 1):N, ]),
```

```
  linkg = list(g11 = "logit", g2 = "default", g21 = "logit"),
  p = c(1, 0), q = c(0, 1), d = FALSE
)

summary(f1, full.report = TRUE) # default
summary(f2, full.report = TRUE)

summary(f1, full.report = FALSE) # simplified output
summary(f2, full.report = FALSE)
```

`BTSR.model.defaults` *Print Model Default Settings*

Description

Displays the default settings for a specified model in the BTSR package, including link functions and their configurations.

Usage

```
BTSR.model.defaults(model)
```

Arguments

`model`	Character string specifying the model name (e.g., "KREGV", "MARMA").

Value

Invisibly returns a list of data frames containing:

- `basic_info` - Model name and parent model (if different)
- `link_functions` - Link functions with their `ctt` and `power` parameters (for `"polynomial"` links)

Examples

```
## Not run:
# Print settings for KREGV model
BTSR.model.defaults("KREGV")

# Print settings for MARMA model
BTSR.model.defaults("MARMA")

## End(Not run)
```

BTSR.models *Table of available model*

Description

This function returns the table of available models.

Usage

```
BTSR.models(do.plot = interactive())
```

Arguments

do.plot	logical; if TRUE returns a plot with the output, otherwise prints the results in the console.

Value

NULL (invisibly). The function is called for its side effects (printing/plotting).

BTSR.parent.models *BTSR models with ν constant over time*

Description

Function to simulate, extract components, and fit BTSR parent models

- ν constant over time:
  BARFIMA, GARFIMA, KARFIMA, MARFIMA, ULARFIMA, and UWARFIMA
- ν varying over time:
  BARFIMAV, GARFIMAV, KARFIMAV and UWARFIMAV

all of which are special cases of the general BTSR structure. See the Section 'The BTSR Structure' in btsr-package for details. These functions are maintained for backward compatibility.

All models share core arguments with

- BARFIMA.sim() for simulation
- BARFIMA.extract() for extraction
- BARFIMA.fit() for fitting.

Usage

```
BARFIMA.sim(n = 1, burn = 0, y.start = NULL, xreg = NULL,
  xreg.start = NULL, xregar = TRUE, coefs = NULL, error.scale = 1,
  linkg = "logit", configs.linkg = NULL, inf = 1000, complete = FALSE,
  debug = FALSE)

BARFIMAV.sim(vt.start = NULL, e2.start = NULL, linkg = list(g11 =
  "logit", g2 = "default", g21 = "log"), ...)

GARFIMA.sim(linkg = "log", ...)

GARFIMAV.sim(vt.start = NULL, e2.start = NULL, linkg = list(g11 = "log",
  g2 = "default", g21 = "log"), ...)

KARFIMA.sim(rho = 0.5, y.lower = 0, y.upper = 1, ...)

KARFIMAV.sim(rho = 0.5, y.lower = 0, y.upper = 1, vt.start = NULL,
  e2.start = NULL, linkg = list(g11 = "logit", g2 = "default", g21 =
  "logit"), ...)

MARFIMA.sim(linkg = "cloglog", ...)

ULARFIMA.sim(...)

UWARFIMA.sim(rho = 0.5, ...)

UWARFIMAV.sim(rho = 0.5, vt.start = NULL, e2.start = NULL,
  linkg = list(g11 = "logit", g2 = "default", g21 = "log"), ...)

BARFIMA.extract(yt, xreg = NULL, nnew = 0, xnew = NULL, y.start = NULL,
  xreg.start = NULL, p = NULL, q = NULL, coefs = NULL, lags = NULL,
  fixed.values = NULL, fixed.lags = NULL, xregar = TRUE,
  error.scale = 1, inf = 1000, linkg = "logit", configs.linkg = NULL,
  m = 0, llk = TRUE, sco = FALSE, info = FALSE, extra = FALSE,
  debug = FALSE)

BARFIMAV.extract(vt.start = NULL, e2.start = NULL, linkg = list(g11 =
  "logit", g2 = "default", g21 = "log"), ...)

GARFIMA.extract(linkg = "log", ...)

GARFIMAV.extract(vt.start = NULL, e2.start = NULL, linkg = list(g11 =
  "log", g2 = "default", g21 = "log"), ...)

KARFIMA.extract(rho = 0.5, y.lower = 0, y.upper = 1, ...)

KARFIMAV.extract(rho = 0.5, y.lower = 0, y.upper = 1, vt.start = NULL,
  e2.start = NULL, linkg = list(g11 = "logit", g2 = "default", g21 =
```

```
  "logit"), ...)

MARFIMA.extract(linkg = "cloglog", ...)

ULARFIMA.extract(...)

UWARFIMA.extract(rho = 0.5, ...)

UWARFIMAV.extract(rho = 0.5, vt.start = NULL, e2.start = NULL,
  linkg = list(g11 = "logit", g2 = "default", g21 = "log"), ...)

BARFIMA.fit(yt, xreg = NULL, nnew = 0, xnew = NULL, y.start = NULL,
  xreg.start = NULL, p = NULL, d = FALSE, q = NULL, xregar = TRUE,
  inf = 1000, start = NULL, ignore.start = FALSE, lags = NULL,
  fixed.values = NULL, fixed.lags = NULL, lower = NULL, upper = NULL,
  error.scale = 1, linkg = "logit", configs.linkg = NULL, m = 0,
  llk = TRUE, sco = FALSE, info = FALSE, extra = FALSE,
  control = NULL, report = TRUE, debug = FALSE, ...)

BARFIMAV.fit(vt.start = NULL, e2.start = NULL, linkg = list(g11 =
  "logit", g2 = "default", g21 = "log"), ...)

GARFIMA.fit(linkg = "log", ...)

GARFIMAV.fit(vt.start = NULL, e2.start = NULL, linkg = list(g11 = "log",
  g2 = "default", g21 = "log"), ...)

KARFIMA.fit(rho = 0.5, y.lower = 0, y.upper = 1, ...)

KARFIMAV.fit(rho = 0.5, y.lower = 0, y.upper = 1, vt.start = NULL,
  e2.start = NULL, linkg = list(g11 = "logit", g2 = "default", g21 =
  "logit"), ...)

MARFIMA.fit(linkg = "cloglog", ...)

ULARFIMA.fit(...)

UWARFIMA.fit(rho = 0.5, ...)

UWARFIMAV.fit(rho = 0.5, vt.start = NULL, e2.start = NULL,
  linkg = list(g11 = "logit", g2 = "default", g21 = "log"), ...)
```

Arguments

n	the sample size of the output time series yt after burn-in (simulation only). Default is n = 1.
burn	the length of the 'burn-in' period (simulation only). Default is burn = 0. The first burn values of the time series are discarded.

`y.start` optional; an initial value for Y_t (to initialize recursions when $t < 1$). Default is `y.start = NULL`, in which case, the recursion assumes that $Y_t = g_{12}^{-1}(0)$, for $t < 1$. Only relevant if $p > 0$.

`xreg` optional; external regressors. Can be specified as a vector, a matrix or a list. Default is `xreg = NULL`. For details, see the Section Regressors format.

`xreg.start` optional; initial value for the regressors (to initialize recursion). Can be specified as a vector or a list. Default is `xreg.start = NULL`, in which case, the average of the first p values (AR order) is used. Only relevant if `xreg` is provided, `xregar = TRUE` and $p > 0$. For details, see the Section Regressors format.

`xregar` a length 1 or 2 logical vector indicating whether `xreg` should be included in the AR recursion for each part of the model. Default is `xregar = TRUE`. Only relevant if $p > 0$. If a single value is provided and ν is time-varying, the same option is assumed for both parts of the model. See the Section 'The BTSR structure' in btsr-package for details.

`coefs` a list with the coefficients of the model (simulation and extraction only). The default is `coefs = NULL`. For details on the expected format and the arguments that can be passed through `coefs`, see the Section Model coefficients.

`error.scale` either 0 or 1; the scale for the error term. Default is `error.scale = 1` (predictive scale).

`linkg` link functions. Can be specified as a character, two-character vector or a named list. The corresponding text strings for currently available links are listed in link.btsr. Default values depend on the model. For some models default values override user specifications. See the Section Link defaults for details.

`configs.linkg` a list with two elements, `ctt` and `power`, which define the constant a and the exponent b in the link function $g(x) = ax^b$. Each element can be specified as a numeric value, a vector of size 2 or a named list. The default is `configs.linkg = NULL`. See the Section Link defaults for details.

`inf` a length 1 or 2 integer vector given the truncation point for infinite sums. Default is `inf = 1000`. See the Section Model Order for details.

`complete` logical; if FALSE returns only `yt`, else returns additional components (simulation only). Default is `complete = FALSE`.

`debug` logical, if `TRUE` the output from FORTRAN is return (for debugging purposes). Default is `debug = FALSE`.

`vt.start` optional; an initial value for ϑ_t (to initialize recursions when $t < 1$). Default is `vt.start = NULL`, in which case, the recursion assumes that $\vartheta_t = g_{22}^{-1}(0)$, for $t < 1$. Only relevant if ν is time-varying and $p_2 > 0$.

`e2.start` optional; an initial value for $g_{23}(e_{1t})$ (to initialize recursions when $t < 1$). Default is `e2.start = NULL`, in which case, the recursion assumes that $e_{1t} = g_{23}^{-1}(0)$, for $t < 1$. Only relevant if ν is time-varying and $q_2 > 0$ or $d_2 > 0$.

`...` further arguments passed to the internal functions. See, for instance, summary.btsr for details.

`rho` the quantile being considered in the conditional distribution of Y_t (only present in Kumaraswamy and Unit-Weibull based models). It can be any positive number between 0 and 1. Default is `rho = 0.5`, which corresponds to the median.

y.lower	the lower limit for the Kumaraswamy density support. Default is y.lower = 0.
y.upper	the upper limit for the Kumaraswamy density support. Default is y.upper = 1.
yt	numeric vector with the observed time series (extract and fit only). Missing values (NA's) are not allowed.
nnew	optional; the number of out-of sample predicted values required (extract and fit only). Default is nnew = 0.
xnew	optional; nnew new observations of the external regressors (extract and fit only). Follows the same format is the same as xreg. Default is xnew = NULL.
p	optional; a length 1 or 2 integer vector given the order of the AR polynomial (extract and fit only). Default is p = NULL. See the Section Model Order for details.
q	optional; a length 1 or 2 integer vector given the order of the MA polynomial (extract and fit only). Default is q = NULL. See the Section Model Order for details.
lags	optional; a list with the lags (integer values) that the entries in coefs or start correspond to (extract and fit only). The default is lags = NULL, in which the lags are computed from the fixed.lags argument (if provided). When components are missing or empty in both, lags and fixed.lags, the default behavior is to include all lags based on nreg = ncol(xreg), p, and q. For details, see the Section Model coefficients.
fixed.values	optional; a list with the values of the coefficients that are fixed (extract and fit only). The default is fixed.values = NULL. See the Section Model coefficients.
fixed.lags	optional; a list with the lags (integer values) that the fixed values in fixed.values correspond to (extract and fit only). The default is fixed.lags = NULL. For missing components, fixed values will are set based on lags.
m	a non-negative integer indicating the starting time for the sum of the partial log-likelihood, given by $\ell = \sum_{t=m+1}^{n} \ell_t$ (extract and fit only). Default is m = 0. For details, see the Section The log-likelihood.
llk	logical; indicates whether the value of the log-likelihood function should be returned (extract and fit only). Default is llk = TRUE.
sco	logical; indicates whether the score vector should be returned (extract and fit only). Default is sco = FALSE.
info	logical; indicates whether the information matrix should be returned (extract and fit only). Default is info = FALSE. For the fitting function, info is automatically set to TRUE when report = TRUE.
extra	logical, if TRUE the matrices and vectors used to calculate the score vector and the information matrix are returned (extract and fit only). Default is extra = FALSE. Ignored by BARC models.
d	a length 1 or 2 logical vector indicating whether the long memory parameter d should be included in the model either as a fixed or non-fixed parameter (fit only). If d = FALSE, internally the value of the parameter d is fixed as 0. In this case, if start or fixed.values include d, the value provided by the user is ignored. If ν is time-varying and a single value is provided it is assumed that $d_1 = d_2 = d$.

`start`	optional; a list with the starting values for the non-fixed coefficients of the model (fit only). The default is `start = NULL`, in which case the function coefs.start is used internally to obtain starting values for the parameters. For details on the expected format and the arguments that can be passed through `coefs`, see the Section Model coefficients.
`ignore.start`	optional; logical value indicating whether the argument `start` should be ignored (fit only). If starting values are not provided, the function uses the default values and `ignore.start` is ignored. In case starting values are provided and `ignore.start = TRUE`, those starting values are ignored and recalculated. The default is `ignore.start = FALSE`. Partial starting values are not allowed.
`lower`	optional; list with the lower bounds for the parameters (fit only). Default is `lower = NULL`. The default is to assume that the parameters have no lower bound except for `nu`, for which de default is 0. Only the bounds for bounded parameters need to be specified. The format of `lower` and the arguments that can be passed through this list are the same as the ones for `start`.
`upper`	optional; list with the upper bounds for the parameters (fit only). Default is `upper = NULL`. The default is to assume that the parameters have no upper bound. Only the bounds for bounded parameters need to be specified. The format of `lower` and the arguments that can be passed through this list are the same as the ones for `start`.
`control`	a list with configurations to be passed to the optimization subroutines (fit only). Default is `control = NULL`. Missing arguments will receive default values. For details, see fit.control.
`report`	logical; indicates whether the summary from the fitted model should be be printed (fit only). Default is `report = TRUE`, in which case `info` is automatically set to `TRUE`.

Details

All functions implemented in the current version of the package are compatible with the new format for the arguments, introduced in version 1.0.0. and the previous format.

- The *simulatio functions* (e.g. `BARFIMA.sim`) are used to generate random samples from the corresponding model.
- The *extraction functions* (e.g. `BARFIMA.extract`) allow the user to extract all conditional time series, the log-likelihood, and the vectors and matrices used to calculate the score vector and the information matrix associated to a given set of parameters. This function can be used by any user to create an objective function that can be passed to optimization functions not available in BTSR Package. At this point, there is no other use for which this function was intended.
- The *fitting functions* (e.g. `BARFIMA.fit`) fit the corresponding model to a given univariate time series. For now, available optimization algorithms are `"L-BFGS-B"` and `"Nelder-Mead"`. Both methods accept bounds for the parameters. For `"Nelder-Mead"`, bounds are set via parameter transformation.

Value

These functions return the same ouptuts as btsr.sim, btsr.extract and btsr.fit.

See Also

BTSR.functions, BARC.functions, link.btsr, get.defaults

Examples

```
##############################################################################
#
#   Examples of usage of "MODEL.sim" type of functions
#
##############################################################################
#-----------------------------------------------------------------------------
# iid samples
#-----------------------------------------------------------------------------
# BARFIMA: yt ~ Beta(a,b), a = mu*nu, b = (1-mu)*nu
# CASE 1: using coefs as in the previous version of the package
set.seed(1234)
yb1 <- BARFIMA.sim(
  linkg = "linear", n = 1000,
  coefs = list(alpha = 0.5, nu = 20)
)
hist(yb1)

# CASE 2: using the new layout
set.seed(1234)
yb2 <- BARFIMA.sim(
  n = 1000,
  linkg = list(g11 = "linear", g2 = "linear", g21 = "linear"),
  coefs = list(part1 = list(alpha = 0.5), part2 = list(alpha = 20))
)
hist(yb2)

# comparing the results
range(abs(yb2 - yb1))

# samples from other models in the package
yg <- GARFIMA.sim(
  linkg = "linear", n = 1000,
  coefs = list(alpha = 0.5, nu = 20)
)
yk <- KARFIMA.sim(
  linkg = "linear", n = 1000,
  coefs = list(alpha = 0.5, nu = 20)
)
ym <- MARFIMA.sim(
  linkg = "linear", n = 1000,
  coefs = list(alpha = 0.5)
)
yul <- ULARFIMA.sim(
  linkg = "linear", n = 1000,
  coefs = list(alpha = 0.5)
)
yuw <- UWARFIMA.sim(
```

```
  linkg = "linear", n = 1000,
  coefs = list(alpha = 0.5, nu = 20)
)

# comparing the different distributions
par(mfrow = c(2, 3))
hist(yb1, xlim = c(0, 1))
hist(yk, xlim = c(0, 1))
hist(yg, xlim = c(0, 1))
hist(ym, xlim = c(0, 1))
hist(yul, xlim = c(0, 1))
hist(yuw, xlim = c(0, 1))

#------------------------------------------------------------------------
#  BARFIMA(1,d,1) with d = 0.25 and no regressors
#------------------------------------------------------------------------

# CASE 1: using coefs as in the previous version of the package
set.seed(1234)
y1 <- BARFIMA.sim(
  n = 1000,
  linkg = "logit",
  coefs = list(alpha = 0.2, phi = 0.2, theta = 0.4, d = 0.25, nu = 20)
)

# CASE 2: using the new layout
set.seed(1234)
y2 <- BARFIMA.sim(
  n = 1000,
  linkg = list(g11 = "logit", g2 = "linear", g21 = "linear"),
  coefs = list(
    part1 = list(alpha = 0.2, phi = 0.2, theta = 0.4, d = 0.25),
    part2 = list(alpha = 20)
  )
)

# comparing the results
range(abs(y1 - y2))

##########################################################################
#
#   Examples of usage of "MODEL.extract" type of functions
#
##########################################################################

#------------------------------------------------------------------------
#  code to simulate and extract components of a BARMA(1,1) model
#------------------------------------------------------------------------
burn <- 100
n <- 500
N <- n + burn
covar <- cbind(sin(2 * pi * (1:N) / 50), 1:N)
```

```
set.seed(1234)
m1 <- BARFIMA.sim(
  linkg = "logit", n = n, burn = burn, xreg = covar,
  coefs = list(
    alpha = 0, phi = -0.65, theta = -0.25,
    beta = c(0.6, -0.002), nu = 20
  ), complete = TRUE
)

# Extracting assuming that all coefficients are non-fixed
e1 <- BARFIMA.extract(
  yt = m1$yt, xreg = covar[(burn + 1):N, ], linkg = "logit",
  coefs = list(
    alpha = 0, phi = -0.65, theta = -0.25,
    beta = c(0.6, -0.002), nu = 20
  ),
  llk = TRUE, sco = TRUE, info = TRUE
)

# Extracting assuming that all coefficients are fixed
e2 <- BARFIMA.extract(
  yt = m1$yt, xreg = covar[(burn + 1):N, ], linkg = "logit",
  fixed.values = list(
    alpha = 0, phi = -0.65, theta = -0.25,
    beta = c(0.6, -0.002), nu = 20
  ),
  llk = TRUE, sco = TRUE, info = TRUE
)

# Extracting using a mix of fixed and non-fixed parameters
e3 <- BARFIMA.extract(
  yt = m1$yt, xreg = covar[(burn + 1):N, ], linkg = "logit",
  coefs = list(
    phi = -0.65, theta = -0.25,
    beta = c(0.6)
  ),
  fixed.values = list(alpha = 0, nu = 20, beta = -0.002),
  fixed.lags = list(beta = 2),
  llk = TRUE, sco = TRUE, info = TRUE
)

# comparing the simulated and the extracted values of mut
cbind(head(m1$mut), head(e1$mut), head(e2$mut), head(e3$mut))

# comparing the log-likelihood values obtained (must be the all equal)
c(e1$sll, e2$sll, e3$sll)

# comparing the score vectors:
# - e1 must have 6 values: dl/dro values and dl/dlambda values
# - e2 must be empty (all parameters are fixed)
# - e3 must have only the values corresponding to the non-fixed coefficients.
round(e1$score, 4)
e2$score
```

```
round(e3$score, 4)

# comparing the information matrices.
# - e1 must be a 6x6 matrix
# - e2 must be empty
# - e3 must have only the value corresponding to the non-fixed coefficient
round(e1$info.Matrix, 4)
e2$info.Matrix
round(e3$info.Matrix, 4)

###########################################################################
#
#   Examples of usage of "MODEL.fit" type of functions
#
###########################################################################

#---------------------------------------------------------------------------
#  code to simulate and fit a BARMA(1,1) model
#---------------------------------------------------------------------------
burn <- 100
n <- 500
N <- n + burn
covar <- cbind(sin(2 * pi * (1:N) / 50), 1:N)

set.seed(1234)
m1 <- BARFIMA.sim(
  linkg = "logit", n = n, burn = burn, xreg = covar,
  coefs = list(
    alpha = 0, phi = -0.65, theta = -0.25,
    beta = c(0.6, -0.002), nu = 20
  ),
  complete = TRUE
)

plot.ts(m1$yt)

# Fit a model assuming that all coefficients are non-fixed
# for a more simplified summary, set full.report = FALSE
f1 <- BARFIMA.fit(
  yt = m1$yt, xreg = covar[(burn + 1):N, ],
  linkg = "logit", p = 1, q = 1, report = TRUE
)

# the fitted coefficients (full model, including d)
coefficients(f1)

# if you try to use `robust` or `outer` without setting `extra = TRUE`, the
# code issues a message and uses the information matrix
summary(f1, robust = TRUE)
summary(f1, outer = TRUE)

# Fit a model assuming alpha and d are fixed
f2 <- BARFIMA.fit(
```

```
  yt = m1$yt, xreg = covar[(burn + 1):N, ], linkg = "logit",
  p = 1, q = 1, fixed.values = list(alpha = 0, d = 0)
)
# Alternatively, set `d = FALSE`
f2 <- BARFIMA.fit(
  yt = m1$yt, xreg = covar[(burn + 1):N, ], linkg = "logit",
  p = 1, q = 1, fixed.values = list(alpha = 0), d = FALSE
)

# comparing the results
true <- c(
  alpha = 0, beta = c(0.6, -0.002),
  phi = -0.65, theta = -0.25,
  d = 0, nu = 20
)
cf1 <- coefficients(f1)
cf2 <- c(NA, coefficients(f2)[1:4], NA, coefficients(f2)[5])
round(cbind(true, cf1, cf2), 3)
```

`coefs.start` *Initial values for coefficients*

Description

Generates initial values for coefficients in BTSR models.

Usage

```
coefs.start(model, yt, y.start = NULL, y.lower = 0, y.upper = 1,
  xreg = NULL, p = 0, q = 0, d = FALSE, map = .default.map.barc,
  lags = NULL, fixed.values = NULL, fixed.lags = NULL,
  linkg = "linear", configs.linkg = NULL)
```

Arguments

`model`	character string (case-insensitive) indicating the model to be fitted to the data. Must be one of the options listed in the Section Supported Models.
`yt`	numeric vector with the observed time series. Missing values (NA's) are not allowed.
`y.start`	optional; an initial value for Y_t (to initialize recursions when $t < 1$). Default is `y.start = NULL`, in which case, the recursion assumes that $Y_t = g_{12}^{-1}(0)$, for $t < 1$. Only relevant if $p > 0$.
`y.lower`	the lower limit for the Kumaraswamy density support. Default is `y.lower = 0`.
`y.upper`	the upper limit for the Kumaraswamy density support. Default is `y.upper = 1`.
`xreg`	optional; external regressors. Can be specified as a vector, a matrix or a list. Default is `xreg = NULL`. For details, see the Section Regressors format.

`p` the AR order. Default is `p = 0`. Can be provided as either a single integer (legacy format) or a length 2 integer vector (new format) specifying orders for `part1`/`part2`. If ν is time-varying and a single value is provided it is assumed that $p_1 = p_2 = p$.

`q` the MA order. Default is `q = 0`. Can be provided as either a single integer (legacy format) or a length 2 integer vector (new format) specifying orders for `part1`/`part2`. If ν is time-varying and a single value is provided it is assumed that $q_1 = q_2 = q$.

`d` a length 1 (legacy format) or 2 (new format) logical vector indicating whether the long memory parameter d is presented in the model (either as a fixed or non-fixed parameter). In the new format, if only one value is provided the code assumes that the same option is valid for both parts of the model. Default is `d = FALSE`.

`map` a non-negative integer from 1 to 5 corresponding to the map function. Default is `map = 4`. See the Section The map function.

`lags` optional; specification of which lags to include in the model. For one dimensional coefficients, the lag is obviously always 1 and can be suppressed. Can be specified in one of two ways

- a list with components `beta`, `phi`, and `theta` (legacy format) specifying which lags to include for each parameter type.
- a list with elements `part1` and `part2` (new format), each being a list with components `beta`, `phi`, and `theta` specifying which lags to include for each parameter type.

Default is `lags = NULL`, in which the `lags` are computed from the `fixed.lags` argument (if provided). When components are missing or empty in both, `lags` and `fixed.lags`, the default behavior is to include all lags based on `nreg = ncol(xreg)`, `p`, and `q`. The arguments `lags` and `fixed.lags` are complementary. Either suffices, or mix them (e.g., `lags` for some parameters, `fixed.lags` for others).

`fixed.values` optional; specification of fixed parameter values. Can be specified in one of two ways

- a list with optional components `alpha`, `beta`, `phi`, `theta`, `d` and `nu` (legacy format) containing fixed values for each parameter type.
- a list with elements `part1` and `part2` (new format), each being a list with optional components `alpha`, `beta`, `phi`, `theta` and `d` containing fixed values for each parameter type.

If fixed values are provided and there exists more than one possible lag, either `lags` or `fixed.lags` must also be provided. The default is `fixed.lags = NULL`. By default, if a given vector (such as the vector of AR coefficients) has fixed values and the corresponding entry in this list is empty, the fixed values are set as zero.

`fixed.lags` optional; specification of which lags should be fixed. For one dimensional coefficients, the lag is obviously always 1 and can be suppressed. Can be specified in one of two ways

- a list with components `beta`, `phi`, and `theta` (legacy format) specifying which lags should be fixed.

- a list with elements `part1` and `part2` (new format), each being a list with components `beta`, `phi`, and `theta` specifying which lags should be fixed.

For missing components, fixed values will are set based on `lags`.

`linkg` specification of link functions. Can be specified in one of two ways

- A character or two-character vector (legacy format). If only one string is provided, the same link name is used for μ_t (g_{11}) and for Y_t (g_{12}).
- A named list (new format) with elements `g11`, `g12`, `g2`, `g21`, and `g22` (order does not matter). Particular models (see 'Particular Models' in BTSR.functions) have default values for some links. Missing links follow these rules
  - If either `g11` or `g12` is missing (but not both), assumes `g12 = g11`
  - If `phi = NULL` for part 1, `g12` is not required
  - If `phi = NULL` for part 2, `g22` is not required
  - If either `g21` or `g22` is missing (but not both), assumes `g22 = g21`

  Special case: `g2 = "default"` uses the model's default link. The default depends on the model.

Default is `linkg = "linear"`, which is equivalent (done internally) to set all links as `"linear"` in the new format. See link.btsr for valid links. For details, see the Section 'The BTSR structure' in btsr-package.

`configs.linkg` a list with two elements, `ctt` and `power`, which define the constant a and the exponent b in the link function $g(x) = ax^b$. Each element can be specified as a numeric value, a vector of size 2 or a named list. The default is `configs.linkg = NULL`. See the Section Link defaults for details.

Details

Parameter initialization is done as follows.

1. Legacy flat lists are converted to nested `part1`/`part2` format. Link functions and density bounds are validated.
2. **Part 1:** μ_t related parameters.
   A linear model is used to estimate α, $\boldsymbol{\beta}$ and $\boldsymbol{\phi}$ by setting

$$\boldsymbol{Y} = \begin{pmatrix} Y_1 \\ \vdots \\ Y_n \end{pmatrix} \quad \text{and} \quad D = \begin{pmatrix} 1 & X_{11} & \cdots & X_{1s} & g_{12}(Y_0) & \cdots & g_{12}(Y_{1-p}) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & X_{n1} & \cdots & X_{ns} & g_{12}(Y_{n-1}) & \cdots & g_{12}(Y_{n-p}) \end{pmatrix}$$

   where s is the number of regressors and p is the AR order, and solving the linear regression problem $\boldsymbol{Y} = D\boldsymbol{\gamma} + \boldsymbol{\epsilon}$ via `lm.fit`.
   MA coefficients $\boldsymbol{\theta}$ are initialized to zero (though small non-zero values may help with optimization stability)
   The long-memory parameter d starts at 0.01 when estimated
   For BARC models:
   - Map parameters use:

map	1	2	3	4	5
theta	3	0.5	3.5	0.5	NA

- u_0 defaults to $\pi/4$ when not fixed

3. **Part 2:** ν_t related parameters.

   If presented and not time varying, ν is initialized as follows:

   - $\nu = 5$, for the Kumaraswamy and the Unit-Weibull distributions,
   - $\nu = \frac{1}{n}\sum_{t=1}^{n} \frac{\hat{\mu}_t(1-\hat{\mu}_t)}{\sigma^2} - 1$, for the Beta distribution,
   - $\nu = \frac{1}{n}\sum_{t=1}^{n} \frac{\hat{\mu}_t^2}{\sigma^2}$, for the Gamma distribution

   where $(\hat{\mu}_1, \cdots, \hat{\mu}_n)' = D\hat{\gamma}$ are the fitted values from the regression model and σ^2 is the estimated variance of the residuals.

   If ν is time varying,

   - set α as $g_{12}(g_2(\nu))$, with ν estimated as in the case where the parameter does not vary on time.
   - set $\boldsymbol{\beta}$, $\boldsymbol{\phi}$ and $\boldsymbol{\theta}$ to zero.
   - The long-memory parameter d starts at 0.01 when estimated.

Value

For models where ν_t is not time-varying, returns a list (legacy format) with starting values for the parameters of the selected model. Possible outputs are

`alpha`	the intercept.
`beta`	the coefficients for the regressors.
`phi`	the AR coefficients.
`theta`	for BARC models, the parameter associate to the map function. For any other model, the MA coefficients.
`d`	the long memory parameter.
`nu`	distribution related parameter (usually, the precision).
`u0`	for BARC models, the initial value of the iterated map.

For models where ν_t is time-varying, returns a list whose elements are `part1` and `part2`. Each element is a list with starting values for the parameters corresponding to each part o the selected model. Possible outputs for each part are the same as for the legacy format.

Examples

```
mu <- 0.5
nu <- 20
```

```
yt <- rbeta(100, shape1 = mu * nu, shape2 = (1 - mu) * nu)
# using the general model BARFIMA
coefs.start(model = "BARFIMA", yt = yt, linkg = "linear")
# same output as the specific model BETA
coefs.start(model = "BETA", yt = yt, linkg = "linear")

yt <- rgamma(100, shape = nu, scale = mu / nu)
coefs.start(model = "GARFIMA", yt = yt, linkg = "linear")
```

`fit.control` *Default control list*

Description

Sets default values for constants used by the optimization functions in FORTRAN.

Usage

```
fit.control(control = list())
```

Arguments

`control`	a list with configurations to be passed to the optimization subroutines. Missing arguments will receive default values. See 'Details'.

Details

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

`method` The optimization method. Current available options are `"L-BFGS-B"` and `"Nelder-Mead"`. Default is `"L-BFGS-B"`.

`maxit` The maximum number of iterations. Defaults is `1000`.

`iprint` The frequency of reports if `control$trace` is positive. Defaults is -1 (no report).

- For `"L-BFGS-B"` method:
  - `iprint` < `0` no output is generated;
  - `iprint` = `0` print only one line at the last iteration;
  - `0 < iprint < 99` print also f and |proj g| every iprint iterations;
  - `iprint` = `99` print details of every iteration except n-vectors;
  - `iprint` = `100` print also the changes of active set and final x;
  - `iprint` > `100` print details of every iteration including x and g;
- For `"Nelder-Mead"` method:
  - `iprint` < `0` No printing
  - `iprint` = `0` Printing of parameter values and the function value after initial evidence of convergence.

- `iprint > 0` As for `iprint = 0` plus progress reports after every `iprint` evaluations, plus printing for the initial simplex.

`factr` controls the convergence of the `"L-BFGS-B"` method. Convergence occurs when the reduction in the objective is within this factor of the machine tolerance. The iteration will stop when

$$\frac{(f^k - f^{k+1})}{\max\{|f^k|, |f^{k+1}|, 1\}} \leq factr \times epsmch$$

where $epsmch$ is the machine precision, which is automatically generated by the code. Typical values for `factr`: 1.e+12 for low accuracy; 1.e+7 for moderate accuracy; 1.e+1 for extremely high accuracy. Default is `1e7`, that is a tolerance of about `1e-8`.

`pgtol` helps control the convergence of the `"L-BFGS-B"` method. It is a tolerance on the projected gradient in the current search direction. the iteration will stop when

$$\max\{|\text{proj } g_i|, i = 1, ..., n\} \leq pgtol$$

where proj g_i is the ith component of the projected gradient. Default is `1e-12`.

`stopcr` The criterion applied to the standard deviation of the values of objective function at the points of the simplex, for `"Nelder-Mead"` method.

Value

Returns a list with all arguments in 'Details'.

Examples

```
BTSR::fit.control()
```

`get.defaults` *Retrieve Default Arguments for BTSR Package Functions*

Description

Extracts and displays the default argument values for any function in the BTSR package, including both exported and non-exported functions.

Usage

```
get.defaults(fun)
```

Arguments

`fun`	Character string specifying the function name to examine (case-sensitive).

Value

Invisibly returns a data frame with two columns:

- `Argument`: Name of the function parameter
- `Default`: Default value or `"(no default)"` string if no default exists

The function primarily prints a formatted table of the results to the console.

Examples

```
## Not run:
# View defaults for BTSR.fit function
get.defaults("BARFIMA.fit")

# Capture the results for later use
defaults <- get.defaults("BARFIMA.fit")

## End(Not run)
```

`link.btsr` *Create a Link for BTSR models*

Description

Given the name of a link, this function returns a link function, an inverse link function, the derivative $d\eta/d\mu$ and the derivative $d\mu/d\eta$.

Usage

```
link.btsr(link = "linear")
```

Arguments

`link`	character; one of `"linear"`,`"polynomial"`, `"logit"`, `"log"`, `"loglog"`, `"cloglog"`, `"SIP"`. See 'Details' below. Default is `link = "linear"`.

Details

The available links are:

`linear:` $g(x) = ax$, for a real.

`polynomial:` $g(x) = ax^b$, for a and b real.

`logit:` $g(x) = \log((x-l)/(u-x))$

`log:` $g(x) = \log(x-l)$

`loglog:` $g(x) = \log(-\log((x-l)/(u-l)))$

`cloglog:` $g(x) = \log(-\log(1-(x-l)/(u-l)))$

`SIP` **(Shifted Inverse Power):** $g(x) = 1/(a+x)^b$, with $a \in \{0, 1\}$ and b real.

Here l and u denote the lower and upper bounds of x. The linear link is a particular case of the polynomial link. It was kept for compatibility with older versions of the package.

The parameters a, b, l, and u are configured using the `configs.linkg` list, which can include the following elements

- `ctt`: A constant multiplier for the link function (default: 1).
- `power`: The power parameter for polynomial and SIP links (default: 1).
- `lower`: The lower bound for `mu` (default: 0).
- `upper`: The upper bound for `mu` (default: 1).

This list must be passed to the functions created by `link.btsr`, when invoking them.

Value

An object of class `"link-btsr"`, a list with components

`linkfun`	Link function `function(mu)` $g(\mu)$
`linkinv`	Inverse link function `function(eta)` $g^{-1}(eta)$
`linkdif`	Derivative `function(mu)` $d\eta/d\mu$
`mu.eta`	Derivative `function(eta)` $d\mu/d\eta$
`name`	a name to be used for the link

Examples

```
#---------------------------------------------
#  0 < y < 1 and linear link
#---------------------------------------------
mylink <- link.btsr("linear")
y <- 0.8
a <- 3.4
gy <- a * y

# comparing the expected result with the output of the function:
mylink$linkfun(mu = y, configs.linkg = list(ctt = a))
gy

mylink$linkinv(eta = gy, configs.linkg = list(ctt = a))
y

mylink$diflink(mu = y, configs.linkg = list(ctt = a))
a

mylink$mu.eta(eta = gy, configs.linkg = list(ctt = a))
1 / a

# same function, different ctt:
mylink$linkfun(mu = y, configs.linkg = list(ctt = a + 1))
```

```
#-------------------------------------------
# For linear link bounds have no effect
#-------------------------------------------
mylink <- link.btsr("linear")
y <- 0.8
a <- 3.4
gy <- a * y

mylink$linkfun(mu = y, configs.linkg = list(ctt = a, lower = 1, upper = 2))
mylink$linkfun(mu = y, configs.linkg = list(ctt = a)) # same result
gy

mylink$linkinv(eta = gy, configs.linkg = list(ctt = a, lower = 1, upper = 2))
y

mylink$diflink(mu = y, configs.linkg = list(ctt = a, lower = 1, upper = 2))
a

mylink$mu.eta(eta = gy, configs.linkg = list(ctt = a, lower = 1, upper = 2))
1 / a

#-------------------------------------------
# 0 < y < 1 and logit link
#-------------------------------------------
mylink <- link.btsr("logit")
y <- 0.8
gy <- log(y / (1 - y))
ginv <- 1 / (1 + exp(-gy))

mylink$linkfun(mu = y)
gy

mylink$linkinv(eta = gy)
ginv

mylink$diflink(mu = y)
1 / (y - y**2)

mylink$mu.eta(eta = gy)
ginv - ginv**2

#-------------------------------------------
# 1 < y < 2 and logit link
#-------------------------------------------
mylink <- link.btsr("logit")
a <- 1
b <- 2
y <- 1.8
gy <- log((y - a) / (b - y))
ginv <- b / (1 + exp(-gy)) + a / (1 + exp(gy))

mylink$linkfun(mu = y, configs.linkg = list(lower = 1, upper = 2))
gy
```

```
mylink$linkinv(eta = gy, configs.linkg = list(lower = 1, upper = 2))
ginv

mylink$diflink(mu = y, configs.linkg = list(lower = 1, upper = 2))
(b - a) / ((a + b) * y - y**2 - a * b)

mylink$mu.eta(eta = gy, configs.linkg = list(lower = 1, upper = 2))
((a + b) * ginv - ginv**2 - a * b) / (b - a)
```

`predict.btsr` *Predict method for BTSR*

Description

Predicted values based on btsr object.

Usage

```
## S3 method for class 'btsr'
predict(object, newdata, nnew = 0, debug = FALSE, ...)
```

Arguments

`object`	Object of class inheriting from `"btsr"`
`newdata`	A matrix with new values for the regressors. If omitted and `"xreg"` is present in the model, the fitted values are returned. If the model does not include regressors, the functions will use the value of `nnew`.
`nnew`	number of out-of-sample forecasts required. If `newdata` is provided, `nnew` is ignored.
`debug`	logical, if `TRUE` the output from Fortran is return (for debugging purposes). Default is `debug = FALSE`.
`...`	further arguments passed to or from other methods.

Details

`predict.btsr` produces predicted values, obtained by evaluating the regression function in the frame `newdata`.

If `newdata` is omitted the predictions are based on the data used for the fit.

For now, prediction intervals are not provided.

Value

If `nnew = 0`, returns a list with in-sample predictions (`fitted.values`, `etat` and `error`), otherwise, returns a list with the following arguments

- `fitted.values`: in-sample forecast.
  If ν_t is fixed: a vector with the in-sample value of μ_t.
  If ν_t is time varying: a matrix with the in-sample values of μ_t, ν_t and ϑ_t.
- `etat`: the linear predictor(s)
  For models with ν fixed, returns $\eta_{1t} = g_{11}(\mu_t)$
  For models with time varying ν, returns a matrix whose columns are $\eta_{1t} = g_{11}(\mu_t)$ and $\eta_{2t} = g_{21}(\vartheta_t)$.
- `error`: the error term e_{1t} (depends on the argument `error.scale`)
- `residual`: The (in-sample) residuals, that is, the observed values Y_t minus the fitted values μ_t. The same as the `error` term if `error.scale = 0`.
- `forecast`: the out-of-sample forecast.
  If ν_t is fixed: a vector with the predicted values for μ_t and η_{1t}
  If ν_t is time varying: a matrix the predicted values for μ_t and η_{1t}, ν_t, ϑ_t and η_{2t}.
  For BARC models also returs a column with predicted values for the iterated map.
- `TS`: only for `"BARC"` models. The iterated map.
- `xnew`: out-of-sample values for `xreg` (if presented). These are the values passed through `newdata`.

Examples

```
#------------------------------------------------------------
# Generating a Beta model were mut does not vary with time
# yt ~ Beta(a,b), a = mu*nu, b = (1-mu)*nu
#------------------------------------------------------------

y <- btsr.sim(
  model = "BARFIMA", linkg = "linear",
  n = 100, coefs = list(alpha = 0.2, nu = 20)
)

# fitting the model
f <- btsr.fit(
  model = "BARFIMA", yt = y, report = TRUE,
  start = list(alpha = 0.5, nu = 10),
  linkg = "linear", d = FALSE
)

pred <- predict(f, nnew = 5)
pred$forecast
```

print.btsr *Print Method of class BTSR*

Description

Print method for objects of class `btsr`.

Usage

```
## S3 method for class 'btsr'
print(x, digits = max(3L, getOption("digits") - 3L), ...)
```

Arguments

`x`	object of class `btsr`.
`digits`	minimal number of significant digits, see print.default.
`...`	further arguments to be passed to or from other methods. They are ignored in this function

Details

Users are not encouraged to call these internal functions directly. Internal functions for package BTSR.

Value

Invisibly returns its argument, `x`.

summary *Summary Method of class BTSR*

Description

`summary` method for class `"btsr"`.

Usage

```
## S3 method for class 'btsr'
summary(object, robust = FALSE, outer = FALSE,
  full.report = TRUE, ...)

## S3 method for class 'summary.btsr'
print(x, digits = max(3L, getOption("digits") - 3L),
  signif.stars = getOption("show.signif.stars"), ...)
```

Arguments

object	object of class "btsr".
robust	logical. If TRUE the robust covariance matrix is computed
outer	logical. If TRUE uses the outer product of the gradient to compute the covariance matrix. If robust = TRUE, outer is used as a second option (in case of error computing the robust version)
full.report	logical. If TRUE prints a more detailed report.
...	further arguments passed to or from other methods.
x	an object of class "summary.btsr", usually, a result of a call to summary.btsr.
digits	minimal number of significant digits, see print.default.
signif.stars	logical. If TRUE, 'significance stars' are printed for each coefficient.

Details

print.summary.btsr tries to be smart about formatting the coefficients, standard errors, etc. and additionally provides 'significance stars'.

Value

The function summary.btsr computes and returns a list of summary statistics of the fitted model given in object. Returns a list of class summary.btsr, which contains the following components

- model: the corresponding model.
- call: the matched call.
- residuals: The (in-sample) residuals, that is, the observed values Y_t minus the fitted values μ_t. The same as the error term if error.scale = 0.
- coefficients: a $k \times 4$ matrix with columns for the estimated coefficient, its standard error, z-statistic and corresponding (two-sided) p-value.
- sigma.res: the square root of the estimated variance of the error term in residuals

$$\hat{\sigma}^2 = \frac{1}{n-k}\sum_{i=1}^{n-k} e_i^2,$$

  where e_i is the i-th residual.
- df: degrees of freedom, a 2-vector $(k, n-k)$, the first being the number of estimated coefficients.
- vcov: a $k \times k$ matrix of (unscaled) covariances. The inverse ov the information matrix.
- loglik: the sum of the log-likelihood values (L)
- aic: the AIC value. $AIC = -2L + 2k$.
- bic: the BIC value. $BIC = -2L + k\log(n)$.
- hqc: the HQC value. $HQC = -2L + k\log(\log(n))$.

Index