

# Package ‘tsDyn’

January 26, 2023

**Type** Package

**Title** Nonlinear Time Series Models with Regime Switching

**Version** 11.0.4

**Date** 2023-01-26

**Depends** R (>= 3.5.0)

**Imports** mnormt, mgcv, nnet, tseriesChaos, tseries, utils, vars, urca,  
forecast, MASS, Matrix, foreach, methods

**Suggests** sm, scatterplot3d, rgl, tidyverse, rugarch

**Maintainer** Matthieu Stigler <Matthieu.Stigler@gmail.com>

**Description** Implements nonlinear autoregressive (AR) time series models. For univariate series, a non-parametric approach is available through additive nonlinear AR. Parametric modeling and testing for regime switching dynamics is available when the transition is either direct (TAR: threshold AR) or smooth (STAR: smooth transition AR, LSTAR). For multivariate series, one can estimate a range of TVAR or threshold cointegration TVECM models with two or three regimes. Tests can be conducted for TVAR as well as for TVECM (Hansen and Seo 2002 and Seo 2006).

**License** GPL (>= 2)

**URL** <https://github.com/MatthieuStigler/tsDyn/wiki>

**BugReports** <https://github.com/MatthieuStigler/tsDyn/issues>

**RoxygenNote** 7.2.1

**LazyData** true

**NeedsCompilation** yes

**Author** Antonio Fabio Di Narzo [aut] (<<https://orcid.org/0000-0002-4033-5038>>),  
Jose Luis Aznarte [ctb] (<<https://orcid.org/0000-0002-1636-244X>>),  
Matthieu Stigler [aut, cre] (<<https://orcid.org/0000-0002-6802-4290>>)

**Repository** CRAN

**Date/Publication** 2023-01-26 15:00:02 UTC

**R topics documented:**

tsDyn-package	3
aar	4
accuracy_stat	5
addRegime	6
ar_mean	7
autopairs	8
autotriples	9
autotriples.rgl	10
availableModels	11
barry	12
BBCTest	12
charac_root	14
coefB	14
delta	16
delta.lin	18
fevd.nlVar	19
fitted.nlVar	20
getTh	21
GIRF	22
IIPUs	25
irf.linear	27
isLinear	30
KapShinTest	31
lags.select	32
LINEAR	33
lineVar	34
llar	36
logLik.nlVar	39
LSTAR	40
m.unrate	43
MakeThSpec	44
MAPE	45
mse	46
nlar-methods	47
NNET	49
plot methods	50
plot_ECT	51
predict.nlar	52
predict.TVAR	54
predict_rolling	55
rank.select	57
rank.test	59
regime	62
resample_vec	63
resVar	63
selectHyperParms	65

selectSETAR . . . . .	66
SETAR . . . . .	68
setar.sim . . . . .	70
setarTest . . . . .	72
setarTest_IIPUs_results . . . . .	75
sigmoid . . . . .	75
STAR . . . . .	76
toLatex.setar . . . . .	77
TVAR . . . . .	78
TVAR.LRtest . . . . .	80
TVAR.sim . . . . .	82
TVECM . . . . .	85
TVECM.HStest . . . . .	87
TVECM.SeoTest . . . . .	90
TVECM.sim . . . . .	92
UsUnemp . . . . .	95
VAR.sim . . . . .	95
VARrep . . . . .	97
VECM . . . . .	98
VECM_symbolic . . . . .	101
zeroyld . . . . .	102

<b>Index</b>	<b>104</b>
--------------	------------

---

tsDyn-package	<i>Getting started with the tsDyn package</i>
---------------	---

---

## Description

Getting started with the tsDyn package

## Details

This package provide some tools inspired by nonlinear dynamics for the analysis-modelling of observed time series.

For loading the package, type:

```
library(tsDyn)
```

A good place to start learning the package usage, is the vignette. It contains a more detailed guide on package contents, and an applied case study. At the R prompt, write:

```
vignette("tsDyn")
```

For a full list of functions exported by the package, type:

```
ls("package:tsDyn")
```

There is also an experimental GUI for built-in NLAR models. Call it with:

```
nlarDialog(timeSeries)
```

where `timeSeries` is an available time series object.

Each exported function has a corresponding man page (some man pages are in common to more functions). Display it by typing

```
help(functionName)
```

**Author(s)**

Antonio, Fabio Di Narzo

**See Also**

[availableModels](#) for listing all currently available NLAR models

[autopairs](#), [autotriples](#), [autotriples.rgl](#) for graphical exploratory functions

[llar](#), [delta](#), [delta.lin](#) for nonlinearity checking tools

---

 aar

---

*Additive nonlinear autoregressive model*


---

**Description**

Additive nonlinear autoregressive model.

**Usage**

```
aar(x, m, d=1, steps=d, series)
```

**Arguments**

x	time series
m, d, steps	embedding dimension, time delay, forecasting steps
series	time series name (optional)

**Details**

Nonparametric additive autoregressive model of the form:

$$x_{t+s} = \mu + \sum_{j=1}^m s_j(x_{t-(j-1)d})$$

where  $s_j$  are nonparametric univariate functions of lagged time series values. They are represented by cubic regression splines.  $s_j$  are estimated together with their level of smoothing using routines in the **mgcv** package (see references).

**Value**

An object of class `nlar`, subclass `aar`, i.e. a list with mostly internal structures for the fitted `gam` object.

**Author(s)**

Antonio, Fabio Di Narzo

## References

- Wood, mgcv:GAMs and Generalized Ridge Regression for R. R News 1(2):20-25 (2001)
- Wood and Augustin, GAMs with integrated model selection using penalized regression splines and applications to environmental modelling. Ecological Modelling 157:157-177 (2002)

## Examples

```
#fit an AAR model:
mod <- aar(log(lynx), m=3)
#Summary informations:
summary(mod)
#Diagnostic plots:
plot(mod)
```

---

accuracy_stat	<i>Forecasting accuracy measures.</i>
---------------	---------------------------------------

---

## Description

Compute forecasting accuracies. This is very similar to the [accuracy](#) method form **forecast**.

## Usage

```
accuracy_stat(object, ...)

## Default S3 method:
accuracy_stat(object, true, ...)

## S3 method for class 'pred_roll'
accuracy_stat(object, w, ...)
```

## Arguments

object	A data-frame, matrix, or object of class <code>pred_roll</code>
...	Not used currently.
true	If object is just a matrix or data-frame, true values to be compared to should be supplied
w	Optional. For objects of class <code>pred_roll</code> containing multiple variables, user can specify the way to aggregate the specific x-step-ahead into the 'all' category

## Details

The function works either for a simple data.frame or for objects `pred_roll`. For simple data.frames, the argument `true`, i.e. a data frame containing the true values, has to be provided. For `pred_roll` objects, the true values are contained in the object, so no need (nor possibility) to provide the true values.

**Value**

A data-frame containing the forecasting accuracy measures.

**Author(s)**

Matthieu Stigler

**Examples**

```
## univariate:
mod_ar <- linear(lynx[1:100], m=1)
mod_ar_pred <- predict_rolling(mod_ar, newdata=lynx[101:114])
accuracy_stat(object=mod_ar_pred$pred, true=mod_ar_pred$true)

## multivariate
data(barry)
mod_var <- lineVar(barry, lag=1)

mod_var_pred <- predict_rolling(object=mod_var, nroll=10, n.ahead=1:3)
accuracy_stat(object=mod_var_pred)
accuracy_stat(object=mod_var_pred, w=c(0.7, 0.2, 0.1))
```

---

addRegime

*addRegime test*

---

**Description**

addRegime test

**Usage**

```
addRegime(object, ...)
```

**Arguments**

object	fitted model object with at least 2 regimes
...	arguments to and from other methods

**Value**

A list containing the p-value of the F statistic and a boolean, true if there is some remaining nonlinearity and false otherwise.

**Author(s)**

J. L. Aznarte

**References**

TODO

**See Also**[star](#)**Examples**

```
##TODO
```

---

`ar_mean`*Long-term mean of an AR(p) process*

---

**Description**

Computes the long term mean of an AR process

**Usage**

```
ar_mean(object, ...)  
  
## S3 method for class 'linear'  
ar_mean(object, ...)  
  
## S3 method for class 'setar'  
ar_mean(object, ...)  
  
## S3 method for class 'lstar'  
ar_mean(object, ...)
```

**Arguments**

<code>object</code>	an object of class <a href="#">linear</a> , <a href="#">setar</a> or <a href="#">lstar</a>
<code>...</code>	unused argument

## Details

The function computes the long-term mean of an AR(p) process, or of the corresponding sub-regimes in SETAR or LSTAR model. There are three possible cases:

**No constant nor trend** The LT mean is 0

**constant** The LT mean is given by  $\text{const}/(1-\text{sum}(\text{AR coefs}))$

**Trend** The LT mean is not defined

## Examples

```
## estimate a (linear) AR, a SETAR and a LSTAR
lin_cst_l1 <- linear(lh, m = 1, include = "const")
set_cst_l1 <- setar(lh, m = 1, include = "const")
lst_cst_l1 <- lstar(lh, m = 1, include = "const", trace = FALSE)

ar_mean(lin_cst_l1)
ar_mean(set_cst_l1)
ar_mean(lst_cst_l1)
```

---

autopairs

*Bivariate time series plots*

---

## Description

Bivariate time series plots: scatterplots, directed lines and kernel density estimations using functions in the **sm** package.

## Usage

```
autopairs(
  x,
  lag = 1,
  h,
  type = c("levels", "persp", "image", "lines", "points", "regression")
)
```

## Arguments

x	time series
lag	time lag
h	kernel window (useful only for kernel estimations)
type	type of plot: contour levels, perspective plots, image, directed lines, points or points with superposed kernel regression



**Details**

Bivariate time series plots: scatterplots, directed lines and kernel density and regression functions estimations using functions in the package **sm**. In particular, for kernel density estimation [sm.density](#) is used, with smoothing parameter  $h$  defaulting to [hnorm](#). For kernel regression, [sm.regression](#) is used.

**Value**

None. Plots are produced on the default graphical device.

**Author(s)**

Wrappers to **sm** by Antonio, Fabio Di Narzo

**See Also**

For finer control on density estimation, consider using directly [sm.density](#) and, especially, [sm.ts.pdf](#) from package **sm**.

**Examples**

```
x <- log10(lynx)
autopairs(x, lag=2, type="lines")
```

---

autotriples

*Trivariate time series plots*

---

**Description**

Trivariate time series plots: kernel autoregression using functions in the **sm** package

**Usage**

```
autotriples(
  x,
  lags = 1:2,
  h,
  type = c("levels", "persp", "image", "lines", "points")
)
```

**Arguments**

x	time series
lags	vector of regressors lags
h	kernel window
type	type of plot: contour levels, perspective plots, image

**Details**

This function displays trivariate time series plots, i.e. kernel regression of  $x[t - \text{lags}[1]]$ ,  $x[t - \text{lags}[2]]$  against  $x[t]$  using functions in the package **sm**. In particular, **sm.regression** is used, with smoothing parameter defaulting to **hnorm(x)**.

**Value**

None. Plots are produced on the default graphical device.

**Author(s)**

Wrappers to **sm** by Antonio, Fabio Di Narzo

**See Also**

For finer control on kernel regression, consider using directly **sm.regression** and, especially, **sm.autoregression** in package **sm**.

**Examples**

```
autotriples(log(lynx))
autotriples(log(lynx), type="persp")
autotriples(log(lynx), type="image")
```

---

autotriples.rgl

*Interactive trivariate time series plots*

---

**Description**

Interactive trivariate time series plots

**Usage**

```
autotriples.rgl(x, lags = 1:2, type = c("lines", "points"))
```

**Arguments**

x	time series
lags	vector of regressors lags
type	type of plot: contour levels, perspective plots, image

**Details**

This function displays interactive trivariate time series plots  $x[t - \text{lags}[1]]$ ,  $x[t - \text{lags}[2]]$  against  $x[t]$  using the interactive **rgl** device.

**Value**

None. A plot is produced on the current rgl device.

**Author(s)**

Wrapper to 'sm' and GUI by Antonio, Fabio Di Narzo

**See Also**

[autotriples](#) for 3d visualization via **scatterplot3d** package and for kernel post-processing of the cloud for nonparametric autoregression functions estimates.

**Examples**

```
if(interactive())
  autotriples.rgl(log(lynx))
```

---

availableModels	<i>Available models</i>
-----------------	-------------------------

---

**Description**

Available built-in time series models

**Usage**

```
availableModels()
```

**Details**

Return the list of built-in available 'nlar' time series models

**Value**

A character vector containing built-in time series models. For help on a specific model, type: `help(modelName)`.

**Author(s)**

Antonio, Fabio Di Narzo

**Examples**

```
availableModels()
```

---

barry

*Time series of PPI used as example in Bierens and Martins (2010)*

---

**Description**

This data set contains the series used by *Bierens and Martins* for testing for PPI between Canada and US.

**Usage**

```
data(barry)
```

**Format**

A data frame with 324 monthly observations, ranging from 1973:M1 until 1999:M12.

dolcan	Exchange rate US/Can dollar.
cpiUSA	US Consumer Price Index.
cpiCAN	Canada Consumer Price Index.

**Author(s)**

Matthieu Stigler

**Source**

Bierens, H. and Martins, L. (2010), Time Varying Cointegration,

---

BBCTest

*Test of unit root against SETAR alternative*

---

**Description**

Test of unit root against a stationary three regime SETAR alternative

**Usage**

```
BBCTest(  
  x,  
  m,  
  series,  
  testStat = c("LR", "Wald", "LM"),  
  trim = 0.1,  
  grid = c("minPerc", "minObs")  
)
```

**Arguments**

x	time series
m	Number of lags under the alternative
series	time series name (optional)
testStat	Type of test statistic to use
trim	trimming parameter indicating the minimal percentage of observations in each regime
grid	Whether a minimal number of percentage or observations should be imposed. See details

**Details**

TODO

**Value**

A object of class "BBC2004Test" containing:

- The value of the sup Test
- The version of test used (either Wald, LM or LR).

**Author(s)**

Matthieu Stigler

**References**

Bec, Ben Salem and Carrasco (2004) Tests for Unit-Root versus Threshold Specification With an Application to the Purchasing Power Parity Relationship, Journal of Business and Economic Statistics; 22:4

**See Also**

[setarTest](#) for a test with stationarity as a null.

**Examples**

```
BBCTest(lynx, m=3, test="Wald", grid="minPerc")
```

---

charac_root	<i>Characteristic roots of the AR coefficients</i>
-------------	--

---

**Description**

Computes the (inverse) characteristic roots of the auto-regressive coefficients. To be stationary, the values should be outside the unit circle. The function here returns the modulus of the roots.

**Usage**

```
charac_root(object, ...)

## S3 method for class 'nlar'
charac_root(object, ...)
```

**Arguments**

object	object of class <code>nlar</code>
...	currently unused

**Details**

Computes the roots of the polynomial (1, -phi) using function `polyroot`

**Value**

a data.frame, with the modulus of the roots. For models with multiple regimes (setar, lstar, star), one column per regime.

**Examples**

```
mod.ar <- linear(lh, m = 5, include = "const")
mod.setar <- setar(lh, m = 5, include = "const")

charac_root(mod.ar)
charac_root(mod.setar)
```

---

coefB	<i>Extract cointegration parameters A, B and PI</i>
-------	---

---

**Description**

Extract parameters in VECM: adjustment coefficients A, cointegrating coefficients B , or the composite matrix PI

**Usage**

```
coefB(object, ...)

## S3 method for class 'VECM'
coefB(object, ...)

## S3 method for class 'ca.jo'
coefB(object, r = 1, normalize = TRUE, ...)

coefA(object, ...)

## S3 method for class 'VECM'
coefA(object, ...)

## S3 method for class 'ca.jo'
coefA(object, r = 1, normalize = TRUE, ...)

coefPI(object, ...)
```

**Arguments**

object	An object of class <a href="#">VECM</a> , <a href="#">ca.jo</a>
r	The cointegrating rank
normalize	Whether to normalize the A/B coefficients. See details
...	Further arguments passed to methods

**Details**

The functions extract the parameters from a VECM with  $K$  variables and rank  $r$ :

**A** Adjustment coefficients, of dim  $K \times r$

**B** Cointegrating coefficients, of dim  $K \times r$

**Pi** Matrix  $\Pi = AB'$ , of dim  $K \times K$

Coefficients are extracted from a VECM in package `tsDyn`, or from a VECM obtained in package `urca` from [ca.jo](#) or [cajorls](#).

Note that by default, the A and B coefficients returned are normalized (see below). This is the case for results obtained from [VECM/lineVar](#) and [cajorls](#), while for [ca.jo](#), the user has the choice (but `normalize=TRUE` by default), in which case the rank  $r$  is also to be specified. The normalization is the Phillips triangular representation, as suggested by Johansen (1995, p. 72), standardising the first  $r \times r$  coefficients to  $I_r$ :

$$\mathbf{B}_{norm} = B(c' B)^{-1} \text{ with } c = (I_r, 0_{K-r,r})'$$

$$\mathbf{A}_{norm} = AB' c$$

Finally, note that the function also apply to objects obtained from tests of class `ca.jo.test` (from [blrtest](#) etc...). Care should be taken however, since the normalization might override the restrictions imposed.

**Value**

A matrix containing the coefficients

**Author(s)**

Matthieu Stigler

**References**

Johansen, Soren, (1995), Likelihood-Based Inference in Cointegrated Vector Autoregressive Models, Oxford University Press

**Examples**

```
data(barry)
vecm <- VECM(barry, lag=1, estim="ML")
vecm_r2 <- VECM(barry, lag=1, estim="ML", r=2)

## extract coefficients:
coefA(vecm)
coefB(vecm)
coefPI(vecm)
coefB(vecm_r2)
coefPI(vecm_r2)

## Beta-Restricted VECM:
beta_vecm2 <- coefB(vecm_r2)
beta_vecm2[3,2] <- 0.02
vecm_r2_rest <- VECM(barry, lag=1, estim="ML", r=2, beta=beta_vecm2)
round(coefB(vecm_r2_rest),5)

## Package vars/urca
if(require(urca)){
  vecm_ur <- ca.jo(barry, K=2)
  coefB(vecm_ur)
  coefB(vecm_ur,r=2)
  coefB(cajorls(vecm_ur, r=2))
  all.equal(coefB(vecm), coefB(vecm_ur), check.attributes=FALSE)
  all.equal(coefB(vecm_r2), coefB(vecm_ur, r=2), check.attributes=FALSE)
}
```

---

delta

*delta test of conditional independence*

---

**Description**

delta statistic of conditional independence and associated bootstrap test



**Usage**

```
delta(x, m, d = 1, eps)

delta.test(
  x,
  m = 2:3,
  d = 1,
  eps = seq(0.5 * sd(x), 2 * sd(x), length = 4),
  B = 49
)
```

**Arguments**

x	time series
m	vector of embedding dimensions
d	time delay
eps	vector of length scales
B	number of bootstrap replications

**Details**

delta statistic of conditional independence and associated bootstrap test. For details, see Manzan(2003).

**Value**

delta returns the computed delta statistic. delta.test returns the bootstrap based 1-sided p-value.

**Warning**

Results are sensible to the choice of the window eps. So, try the test for a grid of m and eps values. Also, be aware of the curse of dimensionality: m can't be too high for relatively small time series. See references for further details.

**Author(s)**

Antonio, Fabio Di Narzo

**References**

Sebastiano Manzan, Essays in Nonlinear Economic Dynamics, Thela Thesis (2003)

**See Also**

BDS marginal independence test: [bds.test](#) in package **tseries**  
Teraesvirta's neural network test for nonlinearity: [terasvirta.test](#) in package **tseries**  
delta test for nonlinearity: [delta.lin.test](#)

## Examples

```
delta(log10(lynx), m=3, eps=sd(log10(lynx)))
```

---

delta.lin	<i>delta test of linearity</i>
-----------	--------------------------------

---

## Description

delta test of linearity based on conditional mutual information

## Usage

```
delta.lin(x, m, d = 1)

delta.lin.test(
  x,
  m = 2:3,
  d = 1,
  eps = seq(0.5 * sd(x), 2 * sd(x), length = 4),
  B = 49
)
```

## Arguments

x	time series
m	vector of embedding dimensions
d	time delay
eps	vector of length scales
B	number of bootstrap replications

## Details

delta test of linearity based on conditional mutual information

## Value

delta.lin returns the parametrically estimated delta statistic for the given time series (assuming linearity). delta.lin.test returns the bootstrap based 1-sided p-value. The test statistic is the difference between the parametric and nonparametric delta estimators.

## Author(s)

Antonio, Fabio Di Narzo

**References**

Sebastiano Manzan, *Essays in Nonlinear Economic Dynamics*, Thela Thesis (2003)

**Examples**

```
delta.lin(log10(lynx), m=3)
```

---

fevd.nlVar

*Forecast Error Variance Decomposition*


---

**Description**

Use the fevd function from package **vars** to compute the forecast error variance decomposition of a VAR(p) or VECM for n. ahead steps.

**Usage**

```
## S3 method for class 'nlVar'
fevd(x, n.ahead = 10, ...)
```

**Arguments**

x	Object of class 'VAR' generated by lineVar(), or an object of class 'VECM' generated by VECM()
n.ahead	Integer specifying the number of steps.
...	Currently not used.

**Details**

The function converts the VAR or VECM computed by package tsDyn into an object of class 'vec2var', on which then the fevd method is applied. For details, see package **vars**.

**Value**

A list with class attribute 'varfevd' of length K holding the forecast error variances as matrices.

**Author(s)**

Bernhard Pfaff

**References**

Hamilton, J. (1994), *Time Series Analysis*, Princeton University Press, Princeton.

Lutkepohl, H. (2006), *New Introduction to Multiple Time Series Analysis*, Springer, New York.

**See Also**

[plot](#) for the plot method. [lineVar](#), [VECM](#) for the models.

**Examples**

```
data(zeroyld)
mod_vecm <- VECM(zeroyld, lag = 2)
fevd(mod_vecm, n.ahead = 5)
```

---

fitted.nlVar

*fitted method for objects of class nlVar, i.e. VAR and VECM models.*

---

**Description**

Returns the fitted values of the model, either as computed in the model, or back to the original series level.

**Usage**

```
## S3 method for class 'nlVar'
fitted(object, level = c("model", "original"), ...)
```

**Arguments**

object	An object of class 'nlVar'; generated by <a href="#">VECM</a> or <a href="#">lineVar</a> .
level	How to return the fitted values. See below.
...	Currently not used.

**Details**

In case of a VAR in differences, in ADF specification, or a VECM, the fitted values are actually in differences. With the option `level="original"`, the function returns the series in the original level.

For VAR in levels, the two arguments are evidently the same and hence it is not taken into account, returning a warning.

**Value**

A matrix.

**Author(s)**

Matthieu Stigler

**Examples**

```
## estimate models
data(barry)

ve <- VECM(barry, lag=2)
va <- lineVar(barry, lag=1)
va_diff <- lineVar(barry, lag=1, I="diff")
va_ADF <- lineVar(barry, lag=1, I="ADF")

## get fitted values:
tail(fitted(ve))
tail(fitted(ve, level="original"))

tail(fitted(va))
tail(fitted(object=va, level="original"))

tail(fitted(va_diff))
tail(fitted(object=va_diff, level="original"))

tail(fitted(va_ADF))
tail(fitted(object=va_ADF, level="original"))
```

---

getTh	<i>Extract threshold(s) coefficient</i>
-------	---

---

**Description**

Extract threshold coefficient(s)

**Usage**

```
getTh(object, ...)
```

## Default S3 method:  
getTh(object, ...)

**Arguments**

object	object of class setar, summary.setar, nlVar
...	additional arguments to getTh

**Value**

Threshold value.

**Author(s)**

Matthieu Stigler

**Examples**

```
set<-setar(lynx, m=3)
getTh(set)
getTh(summary(set))
```

---

GIRF

*Generalized Impulse response Function (GIRF)*

---

**Description**

Generates a GIRF for multiple innovations and histories

**Usage**

```
GIRF(object, n.ahead, seed = NULL, ...)
```

```
## S3 method for class 'setar'
```

```
GIRF(
  object,
  n.ahead = 10,
  seed = NULL,
  n.hist = 20,
  n.shock = 20,
  R = 10,
  hist_li = NULL,
  shock_li = NULL,
  ...
)
```

```
## S3 method for class 'linear'
```

```
GIRF(
  object,
  n.ahead = 10,
  seed = NULL,
  n.hist = 20,
  n.shock = 20,
  R = 10,
  hist_li = NULL,
  shock_li = NULL,
  ...
)
```

```

## S3 method for class 'nlVar'
GIRF(
  object,
  n.ahead = 10,
  seed = NULL,
  n.hist = 20,
  n.shock = 20,
  R = 10,
  hist_li = NULL,
  shock_li = NULL,
  ...
)

## S3 method for class 'GIRF_df'
plot(
  x,
  plot_type = c("density", "line"),
  n.ahead = c(1, 5, 10),
  var = unique(x$var)[1],
  n_simu = c(1, 2),
  ...
)

```

### Arguments

object	An object of class <a href="#">linear</a> , <a href="#">setar</a> or <a href="#">nlVar</a> ( <a href="#">TVAR</a> , <a href="#">TVECM</a> )
n.ahead	The number of steps ahead to compute
seed	optional, the seed for the random numbers
n.hist	The number of past histories to consider. Should be high, ideally size of data (minus lags).
n.shock	The number of actual shocks to consider
R	the number of draws to use for the n.ahead innovations
hist_li	optional, a list of histories (each of same length as lags in the model). If not provided, n.hist histories will be randomly drawn for the original series.
shock_li	optional, a list of innovations. If not provided, n.shock shocks will be randomly drawn from the estimated residuals.
x	output of <a href="#">girf</a>
plot_type	plot: density (for each n.ahead), or line (for multiple n_simu)?
var	plot: which variable to plot?
n_simu	line plot: which simulation to plot?
...	Further arguments passed to specific methods.

### Details

In a nonlinear model, the Impulse response Function (IRF) is not time-, scale- and sign-invariant as in linear models. To address this, Koop et al (1996) introduced the Generalized Impulse response Function (GIRF):

$$GIRF(h, \delta, \omega_{t-1}) = E[y_{t+h} | \epsilon_t = \delta, \epsilon_{t+h} = 0, \omega_{t-1}] - E[y_{t+h} | \epsilon_t = 0, \epsilon_{t+h} = 0, \omega_{t-1}]$$

It is the difference between two conditional expectations, one containing the shock of interest, the second one averaging it out. The averaging-out is done by comparing against random innovations (unlike the IRF, which compares against innovation 0), The parameter R corresponds to the number of times this is done.

The GIRF as defined here depends on the particular shock, as well as on the history. Koop et al (1996) suggest to draw multiple combinations of histories and innovations. This is done with arguments `n.hist` and `n.shock` (or, alternatively, provide one of, or both, `hist_li` and `shock_li` as list of histories and shocks).

The output is a data-frame containing the two average paths for each combinations of shocks and histories.

### Value

A data-frame, with:

**n\_simu:** Id for the simulation (total number is `n.hist` times `n.shock`)

**hist, shock** History and shock used in the `n`th simulation

**n.ahead:** The forecasting horizon. Note the shocks happens at time 0

**var:** The variable (on which the shock happens, corresponds hence to the response argument in `irf`)

**sim\_1, sim\_2** The average (over R times) simulation with the specific shock (`sim_1`) or with random shocks (`sim_2`).

**girf** The difference between `sim_1` and `sim_2`

### Author(s)

Matthieu Stigler

### References

Koop, G, Pesaran, M. H. & Potter, S. M. (1996) Impulse response analysis in nonlinear multivariate models. *Journal of Econometrics*, 74, 119-147

### See Also

[irf.nlVar](#) for the IRF, for linear models, or in case of non-linear models, for each regime.



## Examples

```
## simulate a SETAR for the example. Higher regime more persistent (AR coef 0.5 instead of 0.2)
set <- setar.sim(B = c(0.5, 0.2, 0.5, 0.5), lag = 1, Thresh = 0.5, n = 500)
set_estim <- setar(set, m = 1)

## regime-specific IRF:
plot(irf(set_estim, regime = "L", boot = FALSE))
plot(irf(set_estim, regime = "H", boot = FALSE))

## GIRF
girf_out <- GIRF(set_estim, n.hist = 10, n.shock = 10) # smaller number for example only

## the GIRF shows a very fast convergence (the shock at n.ahead = 4 is already very close to 0)
plot(girf_out, n.ahead = 1:4)
## investigate a few specific GIRFS:
plot(girf_out, plot_type = "line", n_simu = 1:5)
```

---

IIPUs

*US monthly industrial production from Hansen (1999)*


---

## Description

This data, used as example in Hansen (1999), contains the US monthly industrial production.

## Usage

```
IIPUs
```

## Format

A monthly time series of class `ts` starting in January 1960 and ending in September 1997. Note that the series ends at 1997 and not 1998 as in the paper of Hansen, even if the data was taken from his site and the graph is exactly the same.

## Source

Hansen (1999) Testing for linearity, *Journal of Economic Surveys*, Volume 13, Number 5, December 1999, pp. 551-576(26) available at: <http://www.ssc.wisc.edu/~bhansen/papers/cv.htm>

## Examples

```
data(IIPUs)
end(IIPUs) #not same date as in the paper

## Figure 2 in paper (p. 555)
plot(IIPUs)
```

```

## Table 1 in paper (p. 556)
ar_1 <- linear(IIPUs, m=16)
coef(summary(ar_1))[, 1:2]
deviance(ar_1)

## Table 2 in paper (p. 559)
set_1 <- setar(IIPUs, m=16, thDelay=5, trim=0.1)
## tsDyn finds another threshold, with a better SSR:
getTh(set_1)
deviance(set_1)

## estimate with Hansen threshold:
set_1_han <- setar(IIPUs, m=16, thDelay=5, trim=0.1, th = 0.23)
deviance(set_1_han)
set_1_CO <- coef(summary(set_1_han))[, 1:2]
cbind(set_1_CO[1:17,], set_1_CO[18:34,])

## Table 4 in paper (p. 561)
set_2 <- setar(IIPUs, m=16, thDelay=5, trim=0.1, nthresh =2)
getTh(set_2)
deviance(set_2)

##most of the results agree, except constant in the low regime which has opposed signs!
set_2_CO <- coef(summary(set_2))[, 1:2]
cbind(set_2_CO[1:17,], set_1_CO[18:34,])

#this is obviously a error in Hansen, see:
XX<-embed(IIPUs, 17)
Y<-XX[,1]
X<-XX[,-1]
dummyDown<-ifelse(X[,6]<= -2.5, 1,0)
sum(dummyDown)
M<-cbind(1*dummyDown,X*dummyDown )
lm(Y~M-1)

## setar test (takes long to compute, even with small nboot)
## Not run:
test_1 <- setarTest(IIPUs, m=16, thDelay=5, nboot=10)
#because of the discrepancy. test1vs2 does not correspond, test 1vs3 conforms
test_1$Ftests ## compare with Table 2 (F12) and Table 4 (F13, F23)
summary(test_1)

setarTest(IIPUs, m=16, thDelay=5, nboot=10, test="2vs3")
#test 2vs3 is also different of the version in the article (27)

## End(Not run)

## results from the test is stored in: setarTest_IIPUs_results
data(setarTest_IIPUs_results)

## Table 5 and 6
test_1vs <- setarTest_IIPUs_results$test_1

```

```
test_1vs

## Table 7
test_2vs <- setarTest_IIPUs_results$test_2
test_2vs
plot(test_2vs)
```

---

irf.linear

*Impulse response function*

---

### Description

Compute the impulse response coefficients (IRF) of a VAR(p) (or transformed VECM to VAR(p)) for n.ahead steps. For TVECM and TVAR model, argument regime offers regime-specific IRF.

### Usage

```
## S3 method for class 'linear'
irf(
  x,
  impulse = NULL,
  response = NULL,
  n.ahead = 10,
  ortho = TRUE,
  cumulative = FALSE,
  boot = TRUE,
  ci = 0.95,
  runs = 100,
  seed = NULL,
  ...
)

## S3 method for class 'setar'
irf(
  x,
  impulse = NULL,
  response = NULL,
  n.ahead = 10,
  ortho = TRUE,
  cumulative = FALSE,
  boot = TRUE,
  ci = 0.95,
  runs = 100,
  seed = NULL,
  regime = c("L", "M", "H"),
  ...
)
```

```
## S3 method for class 'ar'
irf(
  x,
  impulse = NULL,
  response = NULL,
  n.ahead = 10,
  ortho = TRUE,
  cumulative = FALSE,
  boot = TRUE,
  ci = 0.95,
  runs = 100,
  seed = NULL,
  ...
)

## S3 method for class 'VAR'
irf(
  x,
  impulse = NULL,
  response = NULL,
  n.ahead = 10,
  ortho = TRUE,
  cumulative = FALSE,
  boot = TRUE,
  ci = 0.95,
  runs = 100,
  seed = NULL,
  ...
)

## S3 method for class 'VECM'
irf(
  x,
  impulse = NULL,
  response = NULL,
  n.ahead = 10,
  ortho = TRUE,
  cumulative = FALSE,
  boot = TRUE,
  ci = 0.95,
  runs = 100,
  seed = NULL,
  ...
)

## S3 method for class 'TVAR'
irf(
```

```

x,
impulse = NULL,
response = NULL,
n.ahead = 10,
ortho = TRUE,
cumulative = FALSE,
boot = TRUE,
ci = 0.95,
runs = 100,
seed = NULL,
regime = c("L", "M", "H"),
...
)

## S3 method for class 'TVECM'
irf(
  x,
  impulse = NULL,
  response = NULL,
  n.ahead = 10,
  ortho = TRUE,
  cumulative = FALSE,
  boot = TRUE,
  ci = 0.95,
  runs = 100,
  seed = NULL,
  regime = c("L", "M", "H"),
  ...
)

```

### Arguments

x	Object of class 'VAR'; generated by <code>lineVar()</code> , or object of class 'VECM'; generated by <code>VECM()</code> .
impulse, response	Not used!
n.ahead	Integer specifying the steps.
ortho	Logical, if TRUE (the default) the orthogonalised impulse response coefficients are computed .
cumulative	Logical, if TRUE the cumulated impulse response coefficients are computed. The default value is false.
boot, ci, runs, seed	Arguments for the bootstrap, see <a href="#">irf.varest</a>
...	Currently not used.
regime	For a setar model, which regime (L, M or H) to produce IRF for?

**Value**

A list of class 'varirf' see [irf.varest](#)

**Author(s)**

Matthieu Stigler

**See Also**

[plot](#) for the plot method. [lineVar](#), [VECM](#) for the models. Hamilton, J. (1994), *Time Series Analysis*, Princeton University Press, Princeton.

Lutkepohl, H. (2006), *New Introduction to Multiple Time Series Analysis*, Springer, New York.

**Examples**

```
data(barry)

## For VAR
mod_var <- lineVar(barry, lag = 2)
irf(mod_var, impulse = "dolcan", response = c("dolcan", "cpiUSA", "cpiCAN"), boot = FALSE)

## For VECM
mod_VECM <- VECM(barry, lag = 2, estim="ML", r=2)
irf(mod_VECM, impulse = "dolcan", response = c("dolcan", "cpiUSA", "cpiCAN"), boot = FALSE)
```

---

isLinear

*isLinear*

---

**Description**

Generic NLAR linearity test

**Usage**

```
isLinear(object, ...)
```

**Arguments**

object	fitted time series model
...	arguments to and from other methods

**Author(s)**

A. F. Di Narzo

---

KapShinTest	<i>Test of unit root against SETAR alternative with</i>
-------------	---

---

**Description**

Test of unit root against a stationary 3 regime SETAR alternative with random walk in the inner regime

**Usage**

```
KapShinTest(x, m=1, series, include = c("none", "const", "trend", "both"),
            c=3, delta=0.5, points=NULL, minObsMid=10,
            trick=c("for", "apply", "mapply"), trace=FALSE)
```

**Arguments**

<code>x</code>	time series
<code>m</code>	Number of lags under the alternative
<code>series</code>	time series name (optional)
<code>include</code>	Whether data should be raw, de-meanned or de-meanned and de-trended
<code>c</code>	Argument for the grid search. See details
<code>delta</code>	Argument for the grid search. See details
<code>points</code>	Points for the grid search. See details
<code>minObsMid</code>	Minimal number of observations in the inner regime
<code>trick</code>	type of internal function used
<code>trace</code>	should additional infos be printed? (logical)

**Details**

This function is currently spurious.

**Value**

A object of class `KapShin2006Test` containing:

<code>statistic</code>	The three (SupW, AvgW, ExpW) test statistics computed
<code>case</code>	Whether the data was transformed, corresponds to input argument <code>include</code>
<code>series</code>	The name of the series

**Author(s)**

Matthieu Stigler

**See Also**

[BBCTest](#) for a similar test. [setarTest](#) for a test with stationarity as a null.

**Examples**

```
KapShinTest(lynx, m=1, trace=FALSE, include="none", points=10)
```

---

lags.select

*Selection of the lag with Information criterion.*

---

**Description**

Selection of the cointegrating rank and the lags with Information criterion (AIC, BIC).

**Usage**

```
lags.select(
  data,
  lag.max = 10,
  include = c("const", "trend", "none", "both"),
  fitMeasure = c("SSR", "LL"),
  sameSample = TRUE
)
```

**Arguments**

data	multivariate time series.
lag.max	Maximum number of lags to investigate.
include	Type of deterministic regressors to include.
fitMeasure	Whether the AIC/BIC should be based on the full likelihood, or just the SSR. See explanations in <a href="#">logLik.VECM</a> .
sameSample	Logical. Whether the data should be shortened so that the AIC/BIC are estimated on the same sample. Default to TRUE.

**Details**

This function selects the lag according to AIC, BIC and Hannan-Quinn.

**Value**

An object of class [rank.select](#), with ‘print’ and ‘summary methods’, containing among other the matrices of AIC/BIC/HQ.

**Author(s)**

Matthieu Stigler



**See Also**

[rank.select](#), the underlying function, to estimate the rank also.

[VARselect](#) in package **vars**, does basically the same.

**Examples**

```
data(barry)

#
rk_sel <- lags.select(barry)
rk_sel
summary(rk_sel)
```

---

 LINEAR

*Linear AutoRegressive models*


---

**Description**

AR(m) model

**Usage**

```
linear(x, m, d=1, steps=d, series, include = c("const", "trend", "none", "both"),
       type=c("level", "diff", "ADF"), warn_root=TRUE)
```

**Arguments**

x	time series
m, d, steps	embedding dimension, time delay, forecasting steps
series	time series name (optional)
include	Type of deterministic regressors to include
type	Whether the variable is taken is level, difference or a mix (diff y= y-1, diff lags) as in the ADF test
warn_root	Whether to check (and warn) for roots outside the unit circle?

**Details**

AR(m) model:

$$x_{t+s} = \phi_0 + \phi_1 x_t + \phi_2 x_{t-d} + \dots + \phi_m x_{t-(m-1)d} + \epsilon_{t+s}$$

**Value**

A `nlar` object, linear subclass.

**Author(s)**

Antonio, Fabio Di Narzo

**See Also**

`nlar` for fitting this and other models to time series data

**Examples**

```
#fit an AR(2) model
mod.linear <- linear(log(lynx), m=2)
mod.linear
summary(mod.linear)
```

---

lineVar

*Multivariate linear models: VAR and VECM*

---

**Description**

Estimate either a VAR or a VECM.

**Usage**

```
lineVar(
  data,
  lag,
  r = 1,
  include = c("const", "trend", "none", "both"),
  model = c("VAR", "VECM"),
  I = c("level", "diff", "ADF"),
  beta = NULL,
  estim = c("2OLS", "ML"),
  LRinclude = c("none", "const", "trend", "both"),
  exogen = NULL
)
```

**Arguments**

<code>data</code>	multivariate time series (first row being first=oldest value)
<code>lag</code>	Number of lags to include in each regime
<code>r</code>	Number of cointegrating relationships
<code>include</code>	Type of deterministic regressors to include

model	Model to estimate. Either a VAR or a VECM
I	For VAR only: whether in the VAR the variables are to be taken in levels (original series) or in difference, or similarly to the univariate ADF case.
beta	for VECM only: user-specified cointegrating value. If NULL, will be estimated using the estimator specified in estim
estim	Type of estimator for the VECM: '2OLS' for the two-step approach or 'ML' for Johansen MLE
LRinclude	Possibility to include in the long-run relationship and the ECT a trend, a constant, etc. Can also be a matrix with exogenous regressors
exogen	Inclusion of exogenous variables (first row being first=oldest value). Is either of same size than data (then automatically cut) or than end-sample.

### Details

This function provides basic functionalities for VAR and VECM models. More comprehensive functions are in package **vars**. A few differences appear in the VECM estimation:

**Engle-Granger estimator** The Engle-Granger estimator is available

**Presentation** Results are printed in a different ways, using a matrix form

**lateX export** The matrix of coefficients can be exported to latex, with or without standard-values and significance stars

Two estimators are available: the Engle-Granger two-steps approach (2OLS) or the Johansen (ML). For the 2OLS, deterministic regressors (or external variables if LRinclude is of class numeric) can be added for the estimation of the cointegrating value and for the ECT. This is only working when the beta value is not pre-specified.

The argument beta is only for **VECM**, look at the specific help page for more details.

### Value

Fitted model data

### Author(s)

Matthieu Stigler

### See Also

**VECM** which is just a wrapper for `lineVar(..., model="VECM")`. Methods [predict.VAR](#), [VARrep](#), [regime](#), [irf](#) and [toLatex](#).

**TVAR** and **TVECM** for the corresponding threshold models. [linear](#) for the univariate AR model.

**Examples**

```

data(zeroyld)

#Fit a VAR
VAR <- lineVar(zeroyld, lag=1)
VAR
summary(VAR)

#compare results with package vars:
if(require(vars)) {
a<-VAR(zeroyld, p=1)
coef_vars <- t(sapply(coef(a), function(x) x[c(3,1,2),1]))
all.equal(coef(VAR),coef_vars, check.attributes=FALSE)
}

###VECM
VECM.EG <- lineVar(zeroyld, lag=2, model="VECM")
VECM.EG
summary(VECM.EG)

VECM.ML <- lineVar(zeroyld, lag=2, model="VECM", estim="ML")
VECM.ML
summary(VECM.ML)

###Check Johansen MLE
myVECM <- lineVar(zeroyld, lag=1, include="const", model="VECM", estim="ML")
summary(myVECM, digits=7)
#comparing with vars package
if(require(vars)){
a<-ca.jo(zeroyld, spec="trans")
summary(a)
#same answer also!
}

##export to Latex
toLatex(VECM.EG)
toLatex(summary(VECM.EG))
options("show.signif.stars"=FALSE)
toLatex(summary(VECM.EG), parenthese="Pvalue")
options("show.signif.stars"=TRUE)

```

**Description**

Casdagli test of nonlinearity via locally linear forecasts

**Usage**

```
llar(x, m, d = 1, steps = d, series, eps.min = sd(x)/2,
     eps.max = diff(range(x)), neps = 30, trace = 0)
```

```
llar.predict(x, m, d=1, steps=d, series, n.ahead=1,
            eps=stop("you must specify a window value"),
            onvoid=c("fail","enlarge"), r = 20, trace=1)
```

```
llar.fitted(x, m, d=1, steps=d, series, eps, trace=0)
```

**Arguments**

x	time series
m, d, steps	embedding dimension, time delay, forecasting steps
series	time series name (optional)
n.ahead	n. of steps ahead to forecast
eps.min, eps.max	min and max neighbourhood size
neps	number of neighbourhood levels along which iterate
eps	neighbourhood size
onvoid	what to do in case of an isolated point: stop or enlarge neighbourhood size by an r%
r	if an isolated point is found, enlarge neighbourhood window by r%
trace	tracing level: 0, 1 or more than 1 for llar, 0 or 1 for llar.forecast

**Details**

llar does the Casdagli test of non-linearity. Given the embedding state-space (of dimension  $m$  and time delay  $d$ ) obtained from time series `series`, for a sequence of distance values `eps`, the relative error made by forecasting time series values with a linear autoregressive model estimated on points closer than `eps` is computed. If minimum error is reached at relatively small length scales, a global linear model may be inappropriate (using current embedding parameters). This was suggested by Casdagli(1991) as a test for non-linearity.

llar.predict tries to extend the given time series by `n.ahead` points by iteratively fitting locally (in the embedding space of dimension  $m$  and time delay  $d$ ) a linear model. If the spatial neighbourhood window is too small, your time series last point would be probably isolated. You can ask to automatically enlarge the window `eps` by a factor of `r%` sequentially, until enough neighbours are found for fitting the linear model.

llar.fitted gives out-of-sample fitted values from locally linear models.

**Value**

llar gives an object of class 'llar'. I.e., a list of components:

RMSE	vector of relative errors
eps	vector of neighbourhood sizes (in the same order of RMSE)
frac	vector of fractions of the time series used for RMSE computation
avfound	vector of average number of neighbours for each point in the time series which can be plotted using the <code>plot</code> method, and transformed to a regular <code>data.frame</code> with the <code>as.data.frame</code> function.

Function `llar.forecast` gives the vector of `n` steps ahead locally linear iterated forecasts.

Function `llar.fitted` gives out-of-sample fitted values from locally linear models.

**Warning**

For long time series, this can be slow, especially for relatively big neighbourhood sizes.

**Note**

The C implementation was re-adapted from that in the TISEAN package ("ll-ar" routine, see references). However, here the euclidean norm is used, in place of the max-norm.

**Author(s)**

Antonio, Fabio Di Narzo

**References**

M. Casdagli, Chaos and deterministic versus stochastic nonlinear modelling, *J. Roy. Stat. Soc.* 54, 303 (1991)

Hegger, R., Kantz, H., Schreiber, T., Practical implementation of nonlinear time series methods: The TISEAN package; *CHAOS* 9, 413-435 (1999)

**Examples**

```
res <- llar(log(lynx), m=3, neps=7)
plot(res)
```

```
x.new <- llar.predict(log(lynx),n.ahead=20, m=3, eps=1, onvoid="enlarge", r=5)
lag.plot(x.new, labels=FALSE)
```

```
x.fitted <- llar.fitted(log(lynx), m=3, eps=1)
lag.plot(x.fitted, labels=FALSE)
```

---

logLik.nlVar	<i>Extract Log-Likelihood</i>
--------------	-------------------------------

---

**Description**

Log-Likelihood method for VAR and VECM models.

**Usage**

```
## S3 method for class 'nlVar'
logLik(object, ...)

## S3 method for class 'VECM'
logLik(object, r, ...)
```

**Arguments**

object	object of class VAR computed by <a href="#">lineVar</a> , or class VECM computed by <a href="#">VECM</a> .
...	additional arguments to logLik.
r	The cointegrating rank. By default the rank specified in the call to <a href="#">VECM</a> , but can be set differently by user.

**Details**

For a VAR, the Log-Likelihood is computed as in Luetkepohl (2006) equ. 3.4.5 (p. 89) and Juselius (2006) p. 56:

$$LL = -(TK/2) \log(2\pi) - (T/2) \log |\Sigma| - (1/2) \sum_{t=1}^T \left[ (y_t - A' x_t)' \Sigma^{-1} (y_t - A' x_t) \right]$$

Where  $\Sigma$  is the Variance matrix of residuals, and  $x_t$  is the matrix stacking the regressors (lags and deterministic).

However, we use a computationally simpler version:

$$LL = -(TK/2) \log(2\pi) - (T/2) \log |\Sigma| - (TK/2)$$

See Juselius (2006), p. 57.

(Note that Hamilton (1994) 11.1.10, p. 293 gives  $+(T/2) \log |\Sigma^{-1}|$ , which is the same as  $-(T/2) \log |\Sigma|$ ).

For VECM, the Log-Likelihood is computed in two different ways, depending on whether the VECM was estimated with ML (Johansen) or 2OLS (Engle and Granger).

When the model is estimated with ML, the LL is computed as in Hamilton (1994) 20.2.10 (p. 637):

$$LL = -(TK/2) \log(2\pi) - (TK/2) - (T/2) \log |\hat{\Sigma}_{UU}| - (T/2) \sum_{i=1}^r \log(1 - \hat{\lambda}_i)$$

Where  $\Sigma_{UU}$  is the variance matrix of residuals from the first auxiliary regression, i.e. regressing  $\Delta y_t$  on a constant and lags,  $\Delta y_{t-1}, \dots, \Delta y_{t-p}$ .  $\lambda_i$  are the eigenvalues from the  $\Sigma_{VV}^{-1} \Sigma_{VU} \Sigma_{UU}^{-1} \Sigma_{UV}$ , see 20.2.9 in Hamilton (1994).

When the model is estimated with 2OLS, the LL is computed as:

$$LL = \log |\Sigma|$$

Where  $\Sigma$  is the variance matrix of residuals from the the VECM model. There is hence no correspondence between the LL from the VECM computed with 2OLS or ML.

### Value

Log-Likelihood value.

### Author(s)

Matthieu Stigler

### References

Hamilton (1994) *Time Series Analysis*, Princeton University Press

Juselius (2006) *The Cointegrated VAR model: methodology and Applications*, Oxford Univesity Press

Luetkepohl (2006) *New Introduction to Multiple Time Series Analysis*, Springer

### Examples

```
data(zeroyld)

#Fit a VAR
VAR <- lineVar(zeroyld, lag=1)
logLik(VAR)

##Fit a VECM
vecm <- VECM(zeroyld, lag=1, r=1, estim="ML")
logLik(vecm)
```

### Description

Logistic Smooth Transition AutoRegressive model.



**Usage**

```
lstar(x, m, d=1, steps=d, series, mL, mH, mTh, thDelay,
      thVar, th, gamma, trace=TRUE, include = c("const", "trend", "none", "both"),
      control=list(), starting.control=list())
```

**Arguments**

x	time series
m, d, steps	embedding dimension, time delay, forecasting steps
series	time series name (optional)
mL	autoregressive order for 'low' regime (default: m). Must be <=m
mH	autoregressive order for 'high' regime (default: m). Must be <=m
thDelay	'time delay' for the threshold variable (as multiple of embedding time delay d)
mTh	coefficients for the lagged time series, to obtain the threshold variable
thVar	external threshold variable
th, gamma	starting values for coefficients in the LSTAR model. If missing, a grid search is performed
trace	should additional infos be printed? (logical)
include	Type of deterministic regressors to include
control	further arguments to be passed as control list to <a href="#">optim</a>
starting.control	further arguments for the grid search (dimension, bounds). See details below.

**Details**

$$x_{t+s} = (\phi_{1,0} + \phi_{1,1}x_t + \phi_{1,2}x_{t-d} + \dots + \phi_{1,mL}x_{t-(mL-1)d})G(z_t, th, \gamma) + (\phi_{2,0} + \phi_{2,1}x_t + \phi_{2,2}x_{t-d} + \dots + \phi_{2,mH}x_{t-(mH-1)d})$$

with  $z$  the threshold variable, and  $G$  the logistic function, computed as `plogis(q, location = th, scale = 1/gamma)`, so see [plogis](#) documentation for details on the logistic function formulation and parameters meanings. The threshold variable can alternatively be specified by:

$$\mathbf{mTh} \quad z[t] = x[t]mTh[1] + x[t-d]mTh[2] + \dots + x[t-(m-1)d]mTh[m]$$

$$\mathbf{thDelay} \quad z[t] = x[t - thDelay * d]$$

$$\mathbf{thVar} \quad z[t] = thVar[t]$$

Note that if starting values for  $\phi_{1,1}$  and  $\phi_{2,1}$  are provided, isn't necessary to specify  $mL$  and  $mH$ . Further, the user has to specify only one parameter between  $mTh$ ,  $thDelay$  and  $thVar$  for indicating the threshold variable.

Estimation of the transition parameters  $th$  and  $gamma$ , as well as the regression parameters  $\phi_{1,1}$  and  $\phi_{2,1}$ , is done using concentrated least squares, as suggested in *Leybourne et al. (1996)*.

Given  $th$  and  $gamma$ , the model is linear, so regression coefficients can be obtained as usual by OLS. So the nonlinear numerical search needs only to be done for  $th$  and  $gamma$ ; the regression parameters are then recovered by OLS again from the optimal  $th$  and  $gamma$ .

For the nonlinear estimation of the parameters *th* and *gamma*, the program uses the `optim` function, with optimization method BFGS using the analytical gradient. For the estimation of standard values, `optim` is re-run using the complete Least Squares objective function, and the standard errors are obtained by inverting the hessian. You can pass further arguments to `optim` directly with the control list argument. For instance, the option `maxit` maybe useful when there are convergence issues (see examples).

Starting parameters are obtained doing a simple two-dimensional grid-search over *th* and *gamma*. Parameters of the grid (interval for the values, dimension of the grid) can be passed to `starting.control`.

`nTh` The number of threshold values (*th*) in the grid. Defaults to 200

`nGamma` The number of smoothing values (*gamma*) in the grid. Defaults to 40

`trim` The minimal percentage of observations in each regime. Defaults to 10% (possible threshold values are between the 0.1 and 0.9 quantile)

`gammaInt` The lower and higher smoothing values of the grid. Defaults to `c(1,40)`

`thInt` The lower and higher threshold values of the grid. When not specified (default, i.e NA), the interval are the `trim` quantiles above.

### Value

An object of class `nlar`, subclass `lstar`, i.e. a list with fitted model informations.

### Author(s)

Antonio, Fabio Di Narzo

### References

Non-linear time series models in empirical finance, Philip Hans Franses and Dick van Dijk, Cambridge: Cambridge University Press (2000).

Non-Linear Time Series: A Dynamical Systems Approach, Tong, H., Oxford: Oxford University Press (1990).

Leybourne, S., Newbold, P., Vougas, D. (1998) Unit roots and smooth transitions, *Journal of Time Series Analysis*, 19: 83-97

### See Also

[plot.lstar](#) for details on plots produced for this model from the `plot` generic.

### Examples

```
#fit a LSTAR model. Note 'maxit': slow convergence
mod.lstar <- lstar(log10(lynx), m=2, mTh=c(0,1), control=list(maxit=3000))
mod.lstar
```

```
#fit a LSTAR model without a constant in both regimes.
mod.lstar2 <- lstar(log10(lynx), m=1, include="none")
mod.lstar2
```

```
#Note in example below that the initial grid search seems to be too narrow.
# Extend it, and evaluate more values (slow!):
controls <- list(gammaInt=c(1,2000), nGamma=50)
mod.lstar3 <- lstar(log10(lynx), m=1, include="none", starting.control=controls)
mod.lstar3

# a few methods for lstar:
summary(mod.lstar)
residuals(mod.lstar)
AIC(mod.lstar)
BIC(mod.lstar)
plot(mod.lstar)
predict(mod.lstar, n.ahead=5)
```

---

m.unrate

*Monthly US unemployment*

---

### Description

Monthly US civilian unemployment rate from 1948 through 2004. Used in the book financial time series for Tsay (2005, chapter 4)

### Usage

```
data(m.unrate)
```

### Format

zoo object

### Source

<https://faculty.chicagobooth.edu/ruey-s-tsay/research/analysis-of-financial-time-series-2nd-edition>

### References

Ruey Tsay (2005) Analysis of Financial Time Series, 2nd ed. (Wiley, ch. 4)

**Description**

This optional function allows the user to set different restrictions for the threshold grid search in function `selectSETAR`.

**Usage**

```
MakeThSpec(  
  ngrid = c("All", "Half", "Third", "Quarter"),  
  exact = NULL,  
  int = c("from", "to"),  
  around = "val",  
  ...  
)
```

**Arguments**

<code>ngrid</code>	The number of values to search for
<code>exact</code>	The user give an exact threshold value
<code>int</code>	The user gives an interval to search inside
<code>around</code>	The user gives an point to search around
<code>...</code>	currently unused

**Details**

This function is just to check the inputs for the specification of the grid search. If not provided, the search will be in the biggest interval (`ngrid = "All"`) between the minimum and maximum values. The user can reduce it by giving setting "Half" (only every two points is taken) and so on, or setting a number.

The search can also be made around a point, or between two points. When between a point, the argument `ngrid` is still used, whereas for `around`, a value of 30 is taken as default value if `ngrid` is not specified by user.

**Value**

The input values are given as output after checking for consistency (only one of `exact/int/around` should be given).

**Author(s)**

Matthieu Stigler

**See Also**[selectSETAR](#)**Examples**

```

sun<-(sqrt(sunspot.year+1)-1)*2
selectSETAR(sun, m=3, th=MakeThSpec(exact=10.40967),criterion="SSR", d=1, thDelay=0:2,
            plot=FALSE, nthresh=1)
#when pre-sepcified value does not correspond, function will search nearest value
selectSETAR(sun, m=3, th=MakeThSpec(exact=10.4),criterion="SSR", d=1, thDelay=0:2,
            plot=FALSE, nthresh=1)
#search around:
selectSETAR(sun, m=3, th=MakeThSpec(around=10.40967, ngrid=20),criterion="SSR", d=1, thDelay=0:2,
            plot=FALSE, nthresh=1)
#search in an interval
selectSETAR(sun, m=3, th=MakeThSpec(int=c(10, 11), ngrid=20),criterion="SSR", d=1, thDelay=0:2,
            plot=FALSE, nthresh=1)
#reduce size of the grid:
selectSETAR(sun, m=3, th=MakeThSpec(ngrid="Half"),criterion="SSR", d=1, thDelay=0:2,
            plot=FALSE, nthresh=1)

# 2 thresholds:
selectSETAR(sun, m=3, th=MakeThSpec(ngrid="Half"),criterion="SSR", d=1, thDelay=0:2,
            plot=FALSE, nthresh=2)

```

MAPE

*Mean Absolute Percent Error***Description**

Generic function to compute the Mean Absolute Percent Error of a fitted model.

**Usage**

```
MAPE(object, ...)
```

```
## Default S3 method:
```

```
MAPE(object, ...)
```

**Arguments**

```

object      object of class nlar.fit
...         additional arguments to MAPE

```

**Value**

Computed Mean Absolute Percent Error for the fitted model.

**Author(s)**

Antonio, Fabio Di Narzo

---

mse

*Mean Square Error*

---

**Description**

Generic function to compute the Mean Squared Error of a fitted model.

**Usage**

```
mse(object, ...)
```

```
## Default S3 method:
```

```
mse(object, ...)
```

**Arguments**

object            object of class `nlm`

...               additional arguments to `mse`

**Value**

Computed MSE for the fitted model.

**Author(s)**

Antonio, Fabio Di Narzo

---

nlar-methods	<i>NLAR methods</i>
--------------	---------------------

---

## Description

Generic ‘nlar’ methods. Method ‘nlar’ is described in a separate page: [nlar](#)

## Usage

```
## S3 method for class 'nlar'  
coef(object, ...)  
  
## S3 method for class 'nlar'  
fitted(object, ...)  
  
## S3 method for class 'nlar'  
residuals(object, initVal = TRUE, timeAttr = TRUE, ...)  
  
## S3 method for class 'nlar'  
deviance(object, ...)  
  
## S3 method for class 'nlar'  
mse(object, ...)  
  
## S3 method for class 'nlar'  
AIC(object, k = 2, ...)  
  
## S3 method for class 'nlar'  
BIC(object, ...)  
  
## S3 method for class 'nlar'  
MAPE(object, ...)  
  
## S3 method for class 'nlar'  
summary(object, ...)  
  
## S3 method for class 'nlar'  
plot(x, ask = interactive(), ...)  
  
## S3 method for class 'nlar'  
toLatex(object, digits, label, ...)
```

## Arguments

...	further arguments to be passed to and from other methods
initVal	Whether to return NA for initial values in residuals/predictions etc
timeAttr	Whether the time attributes should be returned in the output

k	numeric, the penalty per parameter to be used for AIC/BIC; the default k = 2 is the classical AIC
x, object	fitted 'nlar' object
ask	graphical option. See <a href="#">par</a>
digits	For print method, see <a href="#">printCoefmat</a> .
label	LaTeX label passed to the equation

### Details

**MAPE** Mean Absolute Percent Error

**mse** Mean Square Error

**plot** Diagnostic plots

### Author(s)

Antonio, Fabio Di Narzo

### See Also

[availableModels](#) for listing all currently available models.

### Examples

```
x <- log10(lynx)
mod.setar <- setar(x, m=2, thDelay=1, th=3.25)
mod.setar
AIC(mod.setar)
mse(mod.setar)
MAPE(mod.setar)
coef(mod.setar)
summary(mod.setar)

e <- residuals(mod.setar)
e <- e[!is.na(e)]
plot(e)
acf(e)

plot(x)
lines(fitted(mod.setar), lty=2)
legend(x=1910, y=3.9, lty=c(1,2), legend=c("observed", "fitted"))

plot(mod.setar)
```



---

 NNET

*Neural Network nonlinear autoregressive model*


---

**Description**

Neural Network nonlinear autoregressive model.

**Usage**

```
nnetTs(x, m, d = 1, steps = d, series, size,
control = list(trace = FALSE))
```

**Arguments**

x	time series
m, d, steps	embedding dimension, time delay, forecasting steps
series	time series name (optional)
size	number of hidden units in the neural network
control	control list to be passed to <code>nnet::nnet</code> optimizer

**Details**

Neural network model with 1 hidden layer and linear output:

$$x_{t+s} = \beta_0 + \sum_{j=1}^D \beta_j g(\gamma_{0j} + \sum_{i=1}^m \gamma_{ij} x_{t-(i-1)d})$$

Model is estimated using the `nnet` function in `nnet` package. Optimization is done via the BFGS method of `optim`. Note that for this model, no additional model-specific summary and plot methods are made available from this package.

**Value**

An object of class `nlar`, subclass `nnetTs`, i.e. a list with mostly `nnet::nnet` internal structures.

**Author(s)**

Antonio, Fabio Di Narzo

**References**

Non-linear time series models in empirical finance, Philip Hans Franses and Dick van Dijk, Cambridge: Cambridge University Press (2000).

Non-Linear Time Series: A Dynamical Systems Approach, Tong, H., Oxford: Oxford University Press (1990).

Chaos: A Statistical Perspective, Chan, K., Tong, H., New York: Springer-Verlag (2001).

## Examples

```
#fit a Neural Network model
mod.nnet <- nnetTs(log(lynx), m=2, size=3)
mod.nnet
```

---

plot methods

*Plotting methods for SETAR and LSTAR subclasses*

---

## Description

Plotting methods ‘setar’ and ‘lstar’ subclasses

## Usage

```
## S3 method for class 'setar'
plot(x, ask=interactive(), legend=FALSE, regSwStart, regSwStop, ...)
## S3 method for class 'lstar'
plot(x, ask=interactive(), legend=FALSE, regSwStart, regSwStop, ...)
```

## Arguments

x	fitted ‘setar’ or ‘lstar’ object
ask	graphical option. See <a href="#">par</a>
legend	Should a legend be plotted? (logical)
regSwStart, regSwStop	optional starting and stopping time indices for regime switching plot
...	further arguments to be passed to and from other methods

## Details

These plot methods produce a plot which gives to you an idea of the behaviour of the fitted model.

Firstly, if embedding dimension is, say,  $m$ ,  $m$  scatterplots are produced. On the x axis you have the lagged time series values. On the y axis the ‘response’ time series values. Observed points are represented with different colors-symbols depending on the level of the threshold variable. Specifically, for the setar model, black means ‘low regime’, red means ‘high regime’. For the lstar model, where the self-threshold variable is continuous, threshold values are grouped in 5 different zones with the same number of points in each. Note that if more than 300 points are to be plotted, they all share the same symbol, and regimes can be distinguished only by color. If you want, by specifying `legend=TRUE` a legend is added at the upper-left corner of each scatterplot. To each scatterplot, a dashed line is superposed, which links subsequent fitted values.

Finally, a new time series plot is produced, with lines segments coloured depending on the regime (colors meanings are the same of those in the preceding scatterplots). Optionally, you can specify a starting and ending time indices, for zooming on a particular segment of the time series.

**Author(s)**

Antonio, Fabio Di Narzo

**See Also**[setar](#), [lstar](#)[nlar-methods](#) for other generic available methods for this kind of objects.**Examples**

```
##
##See 'setar' examples
##
```

plot\_ECT

*Plot the Error Correct Term (ECT) response***Description**

This plot shows how variables in a (T)VECM respond to deviations from the long-term equilibrium

**Usage**

```
plot_ECT(x, add.legend = TRUE, legend.location = "topright", ...)
```

**Arguments**

x	object of class <a href="#">VECM</a> or <a href="#">TVECM</a>
add.legend	logical. Whether to add a legend?
legend.location	character. Location of the legend, see <a href="#">legend</a>
...	arguments passed to the initial plot call, see <a href="#">plot</a>

**Value**

a plot, and invisibly the underlying data.frame, containing the ECT and the response for each variable

**Examples**

```
data(zeroyld)
vec_l1 <- VECM(zeroyldMeta[, c("long.run", "short.run")], lag =1)
tvec_l1 <- TVECM(zeroyldMeta[, c("long.run", "short.run")], lag =1,
                plot = FALSE, trace = FALSE, th1 = list(exact = -1.263))

plot_ECT(vec_l1)
plot_ECT(tvec_l1, legend.location = "bottomright")
```

---

predict.nlar                      *Predict method for objects of class 'nlar'.*

---

### Description

Forecasting a non-linear model object of general class 'nlar', including 'setar' and 'star'.

### Usage

```
## S3 method for class 'nlar'
predict(
  object,
  newdata,
  n.ahead = 1,
  type = c("naive", "MC", "bootstrap", "block-bootstrap"),
  nboot = 100,
  ci = 0.95,
  block.size = 3,
  boot1Zero = TRUE,
  seed = NULL,
  ...
)
```

### Arguments

object	An object of class 'nlar'; generated by setar() or lstar().
newdata	Optional. A new data frame to predict from.
n.ahead	An integer specifying the number of forecast steps.
type	Type of forecasting method used. See details.
nboot	The number of replications for type MC or bootstrap.
ci	The forecast confidence interval (available only with types MC and bootstrap).
block.size	The block size when the block-bootstrap is used.
boot1Zero	Whether the first innovation for MC/bootstrap should be set to zero.
seed	optional, the seed for the random generation.
...	Further arguments passed to the internal 'oneStep' function. Mainly argument 'thVar' if an external threshold variable was provided

### Details

The forecasts are obtained recursively from the estimated model. Given that the models are non-linear, ignoring the residuals in the 2- and more steps ahead forecasts leads to biased forecasts (so-called naive). Different resampling methods, averaging n.boot times over future residuals, are available:

**naive** No residuals

**MC** Monte-Carlo method, where residuals are taken from a normal distribution, with a standard deviation equal to the residuals sd.

**bootstrap** Residuals are resampled from the empirical residuals from the model.

**block-bootstrap** Same as bootstrap, but residuals are resampled in block, with size `block.size`

The MC and bootstrap methods correspond to equations 3.90 and 3.91 of Franses and van Dijk (2000, p. 121). The bootstrap/MC is initiated either from the first forecast, `n.ahead=1` (set with `boot1zero` to TRUE), or from the second only.

When the forecast method is based on resampling, forecast intervals are available. These are obtained simply as empirical `ci` quantiles of the resampled forecasts (cf Method 2 in Franses and van Dijk, 2000, p. 122).

### Value

A 'ts' object, or, in the case of MC/bootstrap, a list containing the prediction (`pred`) and the forecast standard errors (`se`).

### Author(s)

Matthieu Stigler

### References

Non-linear time series models in empirical finance, Philip Hans Franses and Dick van Dijk, Cambridge: Cambridge University Press (2000).

### See Also

The model fitting functions [setar](#), [lstar](#).

A more sophisticated predict function, allowing to do sub-sample rolling predictions: [predict\\_rolling](#).

### Examples

```
x.train <- window(log10(lynx), end = 1924)
x.test <- window(log10(lynx), start = 1925)

### Use different forecasting methods:
mod.set <- setar(x.train, m=2, thDelay=0)
pred_setar_naive <- predict(mod.set, n.ahead=10)
pred_setar_boot <- predict(mod.set, n.ahead=10, type="bootstrap", n.boot=200)
pred_setar_Bboot <- predict(mod.set, n.ahead=10, type="block-bootstrap", n.boot=200)
pred_setar_MC <- predict(mod.set, n.ahead=10, type="bootstrap", n.boot=200)

## Plot to compare results:
pred_range <- range(pred_setar_naive, pred_setar_boot$pred, pred_setar_MC$pred, na.rm=TRUE)
plot(x.test, ylim=pred_range, main="Comparison of forecasts methods from same SETAR")
lines(pred_setar_naive, lty=2, col=2)
```

```

lines(pred_setar_boot$pred, lty=3, col=3)
lines(pred_setar_Bboot$pred, lty=4, col=4)
lines(pred_setar_MC$pred, lty=5, col=5)

legLabels <- c("Observed", "Naive F", "Bootstrap F", "Block-Bootstrap F", "MC F")
legend("bottomleft", leg=legLabels, lty=1:5, col=1:5)

```

---

predict.TVAR

*Predict method for objects of class 'VAR', 'VECM' or 'TVAR'*


---

### Description

Forecasting the **level** of a series estimated by 'VAR' / 'VECM' or 'TVAR'

### Usage

```

## S3 method for class 'TVAR'
predict(object, newdata, n.ahead = 5, newdataTrendStart, ...)

## S3 method for class 'VAR'
predict(object, newdata, n.ahead = 5, newdataTrendStart, exoPred = NULL, ...)

```

### Arguments

object	An object of class 'VAR', 'VECM' or 'TVAR'
newdata	Optional. A new data frame to predict from. This should contain lags of the level of the original series. See Details.
n.ahead	An integer specifying the number of forecast steps.
newdataTrendStart	If 'newdata' is provided by the user, and the estimated model includes a trend, this argument specifies where the trend should start
...	Arguments passed to the unexported 'VAR.gen' or 'TVAR.gen' function
exoPred	vector/matrix of predictions for the exogeneous variable(s) (with 'n.ahead' rows). Only for 'VAR'/'VECM', not for 'TVAR'.

### Details

The forecasts are obtained recursively, and are for the levels of the series.

When providing newdata, newdata has to be ordered chronologically, so that the first row/element is the earliest value.

For VECM, the forecasts are obtained by transforming the VECM to a VAR (using function [VARrep](#)). Note that a VECM(lag=p) corresponds to a VAR(lag=p+1), so that if the user provides newdata for a VECM(lag=p), newdata should actually contain p+1 rows.

**Value**

A matrix of predicted values.

**Author(s)**

Matthieu Stigler

**See Also**

[lineVar](#) and [VECM.VARrep](#)

A more sophisticated predict function, allowing to do sub-sample rolling predictions: [predict\\_rolling](#).

**Examples**

```
data(barry)
barry_in <- head(barry, -5)
barry_out <- tail(barry, 5)

mod_vecm <- VECM(barry_in, lag=2)
mod_var <- lineVar(barry_in, lag=3)
mod_tvar <- TVAR(barry_in, lag=3, nthresh=1, thDelay=1)

pred_vecm <- predict(mod_vecm)
pred_var <- predict(mod_var)
pred_tvar <- predict(mod_tvar)

## compare forecasts on a plot
n <- 30
plot(1:n, tail(barry[,1], n), type="l", xlim=c(0,n))
lines((n-5+1):n, pred_var[,1], lty=2, col=2)
lines((n-5+1):n, pred_vecm[,1], lty=2, col=3)
lines((n-5+1):n, pred_tvar[,1], lty=2, col=4)
legend("bottomright", lty=c(1,2,2,2), col=1:4, legend=c("true", "var", "vecm", "tvar"))

## example for newdata:
all.equal(predict(mod_vecm), predict(mod_vecm, newdata=barry[c(317, 318, 319),]))
```

---

predict\_rolling

*Rolling forecasts*

---

**Description**

Forecasts a VAR or VECM by discarding a part of the sample, and recursively generating a series of updated forecasts.

**Usage**

```
predict_rolling(object, ...)

## S3 method for class 'nlVar'
predict_rolling(object, nroll = 10, n.ahead = 1, refit.every, newdata, ...)
```

**Arguments**

object	A linear object of class 'nlVar'; generated by <a href="#">lineVar</a> or <a href="#">VECM</a> .
...	Currently not used.
nroll	The number of rolling forecasts
n.ahead	An integer specifying the number of forecast steps.
refit.every	Determines every how many periods the model is re-estimated.
newdata	In case the given model is already estimated on the sub-sample, the out-of-sample data can be provided with this argument. Note it should contain observations to predict the first values, that are also contained in the in-sample.

**Details**

This function allows to check the out-of sample forecasting accuracy by estimating the model on a sub-sample of the original, then making `nroll` forecasts of horizon `n.ahead`, each time by updating the sample. In other words, with a given model estimated on 100 observations, the function will estimate it on say 90 first obs (`nroll=10`), generate a say 1 step-ahead (`n.ahead=1`) from obs 90, then using true value 91 for to predict value at 92, etc, till the full sample is used.

Unlike with usual `predict()` methods, specifying `n.ahead=2` will not generate a 1 step-ahead and a 2 step-ahead forecasts, but only `nroll 2` step-ahead forecasts.

Note that while the forecasts are updated with new values, the model estimation is (by default) not updated. This can however be done with the argument `fit.every`, specifying at which frequency the model should be re-estimated. By setting it to 1 for example, each time a new observation is taken, the model is reestimated. This is similar to the [ugarchroll](#) in package **rugarch**.

**Value**

A matrix containing the forecasts.

**Author(s)**

Matthieu Stigler

**See Also**

[predict.nlar](#) for the standard predict function.



**Examples**

```

data(barry)

## model estimated on full sample:
mod_vec <- VECM(barry, lag=2)

## generate 10 1-step-ahead forecasts:
preds_roll <- predict_rolling(mod_vec, nroll=10)

## plot the results:
plot(window(barry[, "dolcan"], start=1998), type="l", ylab="barry: dolcan")
preds_roll_ts <- ts(preds_roll$pred, start=time(barry)[nrow(barry)-10], freq=12)
lines(preds_roll_ts[, "dolcan"], col=2, lty=2)
legend("bottomright", lty=c(1,2), col=1:2, leg=c("True", "Fitted"))
title("Comparison of true and rolling 1-ahead forecasts\n")

```

---

rank.select

*Selection of the cointegrating rank with Information criterion.*


---

**Description**

Selection of the cointegrating rank and the lags with Information criterion (AIC, BIC).

**Usage**

```

rank.select(
  data,
  lag.max = 10,
  r.max = ncol(data) - 1,
  include = c("const", "trend", "none", "both"),
  fitMeasure = c("SSR", "LL"),
  sameSample = TRUE,
  returnModels = FALSE
)

## S3 method for class 'rank.select'
print(x, ...)

## S3 method for class 'rank.select'
as.data.frame(x, ...)

## S3 method for class 'rank.select'
summary(object, ...)

```

**Arguments**

data	multivariate time series.
lag.max	Maximum number of lags to investigate.
r.max	Maximum rank to investigate.
include	Type of deterministic regressors to include. See <a href="#">VECM</a> or <a href="#">lineVar</a> .
fitMeasure	Whether the AIC/BIC should be based on the full likelihood, or just the SSR. See explanations in <a href="#">logLik.VECM</a> .
sameSample	Logical. Whether the data should be shortened so that the AIC/BIC are estimated on the same sample. Default to TRUE.
returnModels	Logical, default to FALSE. Whether the output should also contain the list of each model computed.
x	The output from rank.select for the print method.
...	Unused.
object	The output from rank.select for the summary method.

**Details**

This function estimates the AIC, BIC and Hannan-Quinn for each rank (up to `lags.max`) and lags (up to `lags.max`). This method has been shown to be useful to select simultaneously the rank and the lags, see references.

**Value**

An object of class 'rank.select', with 'print' and 'summary methods', containing among other the matrices of AIC/BIC/HQ, the Likelihood, and best ranks according to each criterion.

**Author(s)**

Matthieu Stigler

**References**

- Aznar A and Salvador M (2002). Selecting The Rank Of The Cointegration Space And The Form Of The Intercept Using An Information Criterion. *Econometric Theory*, \*18\*(04), pp. 926-947. <URL: [http://ideas.repec.org/a/cup/etheor/v18y2002i04p926-947\\_18.html](http://ideas.repec.org/a/cup/etheor/v18y2002i04p926-947_18.html)>.
- Cheng X and Phillips PCB (2009). Semiparametric cointegrating rank selection. *Econometrics Journal* , \*12\*(s1), pp. S83-S104. <URL: <http://ideas.repec.org/a/ect/emjrn/v12y2009is1ps83-s104.html>>.
- Gonzalo J and Pitarakis J (1998). Specification via model selection in vector error correction models. *Economics Letters*, \*60\*(3), pp. 321 - 328. ISSN 0165-1765, <URL: [http://dx.doi.org/DOI:10.1016/S0165-1765\(98\)00129-3](http://dx.doi.org/DOI:10.1016/S0165-1765(98)00129-3)>.
- Kapetanios G (2004). The Asymptotic Distribution Of The Cointegration Rank Estimator Under The Akaike Information Criterion. *Econometric Theory*, \*20\*(04), pp. 735-742. <URL: [http://ideas.repec.org/a/cup/etheor/v20y2004i04p735-742\\_20.html](http://ideas.repec.org/a/cup/etheor/v20y2004i04p735-742_20.html)>.

- Wang Z and Bessler DA (2005). A Monte Carlo Study On The Selection Of Cointegrating Rank Using Information Criteria. *Econometric Theory*, \*21\*(03), pp. 593-620. <URL: [http://ideas.repec.org/a/cup/etheor/v21y200620\\_05.html](http://ideas.repec.org/a/cup/etheor/v21y200620_05.html)>.

### See Also

[VECM](#) for estimating a VECM. [rank.test](#) (or [ca.jo](#) in package [urca](#)) for the classical Johansen cointegration test.

### Examples

```
data(barry)

#
rk_sel <- rank.select(barry)
rk_sel
summary(rk_sel)
```

---

rank.test	<i>Test of the cointegrating rank</i>
-----------	---------------------------------------

---

### Description

Maximum-likelihood test of the cointegrating rank.

### Usage

```
rank.test(vecm, type = c("eigen", "trace"), r_null, cval = 0.05)

## S3 method for class 'rank.test'
print(x, ...)

## S3 method for class 'rank.test'
summary(object, digits = max(1, getOption("digits") - 3), ...)
```

### Arguments

vecm	'VECM' object computed with the function <a href="#">VECM</a> .
type	Type of test, either 'trace' or 'eigenvalue'. See details below.
r_null	Rank to test specifically.
cval	Critical value level for the automatic test.
x	The output from <code>rank.test</code> for the print method.
...	Unused.
object	The output from <code>rank.test</code> for the summary method.
digits	The number of digits to use in <a href="#">format.pval</a>

## Details

This function computes the two maximum-likelihood tests for the cointegration rank from Johansen (1996). Tests are:

**trace** Test the hypothesis of rank ‘h’ against rank ‘K’, i.e. against the alternative that the system is stationary.

**eigenvalue** Test the hypothesis of rank ‘h’ against rank ‘h+1’.

The test works for five specifications of the deterministic terms as in Doornik et al (1998), to be specified in the previous call to [VECM](#):

**H\_q1** Unrestricted constant and trend: use `include="both"`

**H\_1** Unrestricted constant and restricted trend: use `include="const"` and `LRinclude="trend"`

**H\_lc** Unrestricted constant and no trend: use `include="const"`

**H\_c** Restricted constant and no trend: use `LRinclude="const"`

**H\_z** No constant nor trend: use `include="none"`

Two testing procedures can be used:

**Specific test** By specifying a value for ‘r\_null’. The ‘pval’ value returned gives the specific p-value.

**Automatic test** If no value is specified for ‘r\_null’, the function makes a simple automatic test: returns the rank (slot ‘r’) of the first test not rejected (level specified by arg `cval`) as recommended i.a. in Doornik et al (1998, p. 544).

A full table with both test statistics and their respective p-values is given in the summary method.

P-values are obtained from the gamma approximation from Doornik (1998, 1999). Small sample values adjusted for the sample size are also available in the summary method. Note that the ‘effective sample size’ for these values is different from output in `gretl` for example.

## Value

An object of class ‘rank.test’, with ‘print’ and ‘summary methods’.

## Comparison with `urca`

While `ca.jo` in package `urca` and `rank.test` both implement Johansen tests, there are a few differences:

- `rank.test` gives p-values, while `ca.jo` gives only critical values.
- `rank.test` allows for five different specifications of deterministic terms (see above), `ca.jo` for only three.
- `ca.jo` allows for seasonal and exogenous regressors, which is not available in `rank.test`.
- The lag is specified differently: `K` from `ca.jo` corresponds to `lag+1` in `rank.test`.

## Author(s)

Matthieu Stigler

## References

- Doornik, J. A. (1998) Approximations to the Asymptotic Distributions of Cointegration Tests, *Journal of Economic Surveys*, 12, 573-93
- Doornik, J. A. (1999) Erratum [Approximations to the Asymptotic Distribution of Cointegration Tests], *Journal of Economic Surveys*, 13, i
- Doornik, Hendry and Nielsen (1998) Inference in Cointegrating Models: UK M1 Revisited, *Journal of Economic Surveys*, 12, 533-72
- Johansen, S. (1996) Likelihood-based inference in cointegrated Vector Autoregressive Models, Oxford University Press

## See Also

[VECM](#) for estimating a VECM. [rank.select](#) to estimate the rank based on information criteria.  
[ca.jo](#) in package **urca** for another implementation of Johansen cointegration test (see section 'Comparison with urca' for more infos).

## Examples

```
data(barry)

## estimate the VECM with Johansen!
ve <- VECM(barry, lag=1, estim="ML")

## specific test:
ve_test_spec <- rank.test(ve, r_null=1)
ve_test_spec_tr <- rank.test(ve, r_null=1, type="trace")

ve_test_spec
ve_test_spec_tr

## No specific test: automatic method
ve_test_unspec <- rank.test(ve)
ve_test_unspec_tr <- rank.test(ve, type="trace")

ve_test_unspec
ve_test_unspec_tr

## summary method: output will be same for all types/test procedure:
summary(ve_test_unspec_tr)

## The function works for many specification of the VECM(), try:
rank.test(VECM(barry, lag=3, estim="ML"))
rank.test(VECM(barry, lag=3, include="both", estim="ML"))
rank.test(VECM(barry, lag=3, LRinclude="const", estim="ML"))

## Note that the tests are simple likelihood ratio, and hence can be obtained also manually:
-2*(logLik(ve, r=1)-logLik(ve, r=2)) # eigen test, 1 against 2
-2*(logLik(ve, r=1)-logLik(ve, r=3)) # eigen test, 1 against 3
```

---

 regime
 

---



---

*Extract a variable showing the regime*


---

**Description**

This function allows to extract the indicator variable specifying the regime in which the process is at time  $t$ .

**Usage**

```
regime(object, initVal = TRUE, timeAttr = TRUE, series = NULL, ...)
```

```
## S3 method for class 'lstar'
```

```
regime(object, initVal = TRUE, timeAttr = TRUE, series, discretize = TRUE, ...)
```

**Arguments**

object	object of class setar or nlVar
initVal	Logical. Whether the NA initial values should be returned. Default to TRUE.
timeAttr	Logical. Whether the time attributes should be returned. Default to TRUE.
series	Optional. A numeric vector to classify according to the model.
...	additional arguments to regime
discretize	logical (default TRUE) whether the series are discretized to 1,2, or whether regime probabilities are returned.

**Value**

Time series of same attributes as input to setar.

**Author(s)**

Matthieu Stigler

**Examples**

```
set<-setar(lynx, m=3)
regime(set)
regime(set, time=FALSE, initVal=FALSE)

plot(regime(set))
```

---

resample_vec	<i>Resampling schemes</i>
--------------	---------------------------

---

### Description

Bootstrap a vector according to multiple resampling schemes: resampling, block resampling, Wild bootstrap.

### Usage

```
resample_vec(
  x,
  boot.scheme = c("resample", "resample_block", "wild1", "wild2", "check"),
  seed = NULL,
  block.size = 2
)
```

### Arguments

x	A numeric vector
boot.scheme	The type of resampling scheme used, see Details
seed	the seed used, see <a href="#">set.seed</a>
block.size	for the resample_block scheme, the size of the blocks.

### Details

This function offers various bootstrap/resampling schemes:

**resample** Resampling with replacement

**resample\_block** Resampling contiguous observations (blocks) with replacement. Use argument `block.size`

**wild1** Wild bootstrap: do not resample, but add a  $N(0,1)$  distribution to each value

**wild12** Wild bootstrap: same, but add instead -1 or 1.

---

resVar	<i>Residual variance</i>
--------	--------------------------

---

### Description

Extracts the global and regime-dependent variance of the residuals

### Usage

```
resVar(x, adj=c("OLS", "ML"))
```

**Arguments**

x                    setar object  
 adj                 Degrees of freedom adjustment for the variance

**Details**

The degree of freedom adjustment in the formula for the variance is the number of parameters when adj="OLS" or zero when adj="ML".

**Value**

A vector containing:

Total                The residual variance of the full sample  
 L, (M), H            The residual variance of the lower (L), middle (if two thresholds) (M) and higher (H) regimes

**Author(s)**

Matthieu Stigler

**References**

Non-Linear Time Series: A Dynamical Systems Approach, Tong, H., Oxford: Oxford University Press (1990).

**Examples**

```
#Lynx model as in Tong (1980, p. 387)
mod.setar <- setar(log10(lynx), mL=7,mH=2, thDelay=1, th=3.116)
summary(mod.setar)
#coefficients are same for lower regime but differ for higer

resVar(mod.setar, adj="ML")
#variance or the residuals is same for lower regime but differ for higer regime and hence for total

#Lynx model as in Tong (1980, p. 405)
mod.setar2 <- setar(log10(lynx), mL=1,mM=7,mH=2, thDelay=1, nthresh=2,th=c(2.373, 3.154))
round(coefficients(mod.setar2),3)

resVar(mod.setar2, adj="ML")
```



---

selectHyperParms      *Automatic selection of model hyper-parameters*

---

### Description

Automatic selection of model hyper-parameters

### Usage

```
selectLSTAR(x, m, d=1, steps=d, mL = 1:m, mH = 1:m, thDelay=0:(m-1),
            fast=TRUE, trace=FALSE)
selectNNET(x, m, d=1, steps=d, size=1:(m+1), maxit=1e3, trace=FALSE)
```

### Arguments

x	time series
m, d, steps	embedding parameters. For their meanings, see help about <a href="#">nlar</a>
mL, mH	Vector of ‘low’ and ‘high’ regimes autoregressive orders
thDelay	Vector of ‘threshold delay’ values
size	Vector of numbers of hidden units in the nnet model
maxit	Max. number of iterations for each model estimation
fast	For LSTAR selection, whether a fast algorithm using starting values fro previous models should be used
trace	Logical. Whether informations from each model should be returned.

### Details

Functions for automatic selection of LSTAR and NNET models hyper parameters. An exhaustive search over all possible combinations of values of specified hyper-parameters is performed. Embedding parameters `m, d, steps` are kept fixed.

Selection criterion is the usual AIC.

For the LSTAR model, two methods are offered:

**fast=FALSE** Each model is run separately, each time using the full grid search for starting values.

**fast=TRUE** Only the first model is run with a full grid search, while the subsequent use the first model results for their starting values.

### Value

A data-frame, with columns giving hyper-parameter values and the computed AIC for each row (only the best 10s are returned)

### Author(s)

Antonio, Fabio Di Narzo

**Examples**

```
llynx <- log10(lynx)
selectLSTAR(llynx, m=2)
selectNNET(llynx, m=3, size=1:5)
```

---

selectSETAR

*Automatic selection of SETAR hyper-parameters*


---

**Description**

Automatic selection of SETAR hyper-parameters

**Usage**

```
selectSETAR(x, m, d=1, steps=d, series, mL, mH,mM, thDelay=0, mTh, thVar,
  th=MakeThSpec(), trace=TRUE, include = c("const", "trend", "none", "both"),
  common=c("none", "include", "lags", "both"), model=c("TAR", "MTAR"),
  ML=seq_len(mL), MH=seq_len(mH), MM=seq_len(mM), nthresh=1, trim=0.15,
  criterion = c("pooled-AIC", "AIC", "BIC", "SSR"), plot=TRUE,
  max.iter=2, type=c("level", "diff", "ADF"), same.lags=FALSE,
  restriction=c("none", "OuterSymAll", "OuterSymTh"), hpc=c("none", "foreach"))
```

**Arguments**

x	time series
m, d, steps	embedding parameters. For their meanings, see help about <a href="#">nlar</a>
series	time series name (optional)
mL, mH, mM	autoregressive order for ‘low’ (mL) ‘middle’ (mM, only useful if nthresh=2) and ‘high’ (mH) regime (default values: m). Must be <=m. Alternatively, you can specify ML
thDelay	Vector of possible ‘threshold delay’ values to check for
mTh	coefficients for the lagged time series, to obtain the threshold variable
thVar	external threshold variable
th	Different specifications of the grid search, to pre-specify a value or set the number of points to search. See <a href="#">MakeThSpec</a>
trace	should additional infos be printed? (logical)
include	Type of deterministic regressors to include
common	Indicates which elements are common to all regimes: no, only the include variables, the lags or both
model	Currently not implemented
ML, MM, MH	vector of lags for order for ‘low’ (ML) ‘middle’ (MM, only useful if nthresh=2) and ‘high’ (MH) regime. Max must be <=m
nthresh	Number of threshold of the model

trim	trimming parameter indicating the minimal percentage of observations in each regime. Default to 0.15
criterion	Model selection criterion
plot	Should a plot showing the criterion values be printed? (logical)
max.iter	Number of iterations for the algorithm
type	Whether the variable is taken is level, difference or a mix (diff y= y-1, diff lags) as in the ADF test
same.lags	Logical. When AIC or pooled-AIC is used and arg m is given, should it search for same number of lags in each regime (TRUE) or allow for different (FALSE) lags in each regime. Different lags involves more computation
restriction	Restriction on the threshold. OuterSymAll will take a symmetric threshold and symmetric coefficients for outer regimes. OuterSymTh currently unavailable
hpc	Possibility to run the bootstrap on parallel core. See details

### Details

Routine for automatic selection of SETAR models hyper parameters.

An exhaustive search over all possible combinations of values of specified hyper-parameters is performed. Thus the threshold delay, the number of lags in each regime and the threshold value are computed.

Embedding parameters  $d$ ,  $steps$  are kept fixed.

Possible criteria are the usual SSR, AIC and a pooled AIC formula:  $AIC(lowregimemodel) + AIC(highregimemodel)$ . The default criterion is the pooled AIC formula. SSR criterion can't be used to compare models with different lags.

When two thresholds ( $nthresh=2$ ) have to be computed, the search for the second is made conditional on results for first threshold as suggested in Gonzalo and Pittarakis (2002). Refinements can be obtained by using `max.iter` (first threshold being re-estimated based on the second one). If SSR is used, the number of lags in the inner regime is either the same if only `arg m` was given, otherwise it has to be pre-specified. Criterion AIC can be used to determine the number of lags in the inner regime, whereas pooled-AIC is currently not implemented for  $nthresh=2$ .

By default, all threshold values excluding the upper and lower trim of the threshold values are taken as potential threshold. `restriction` can be made with `arg th`. See function `MakeThSpec`.

With the argument `hpc`, the heavy grid search can be run on parallel cores, thus alleviating the time of computation. Preliminary results indicate however that the length of the series must be very considerable in order that the parallel code becomes advantageous. To use it, the user needs simply to choose a package (among `doMC`, `doMPI`, `doSNOW` or `doRedis`) and register the backend. See the vignette for more details.

### Value

An object of class `selectSETAR` (print and plot methods) with:

res	A data-frame, with columns giving hyper-parameter values and the computed AIC for each row (only the best 10/5s are returned)
res2	Same as <code>res</code> , returned if $nthresh=2$ otherwise set to NULL

bests                estimated hyper-parameters  
 th, firstBests, bests2th, ML, MM, MH  
                      estimated parameters, from first and conditional search  
 criterion, nthresh, same.lags  
                      returns args given by user  
 allTh                all threshold values and corresponding criterion from first search

### Author(s)

Antonio, Fabio Di Narzo and Stigler, Matthieu

### References

Gonzalo, J. & Pitarakis, J. (2002) Estimation and model selection based inference in single and multiple threshold models, *Journal of Econometrics*, 110, 319 - 352

### See Also

[selectLSTAR](#), [selectNNET](#), [MakeThSpec](#)

### Examples

```
llynx <- log10(llynx)
selectSETAR(llynx, m=2)
#Suggested model is the following:
setar(llynx, m=2, thDelay=1, th=3.4)
```

---

SETAR

*Self Threshold Autoregressive model*

---

### Description

Self Exciting Threshold AutoRegressive model.

### Usage

```
setar(x, m, d=1, steps=d, series, mL, mM, mH, thDelay=0, mTh, thVar, th, trace=FALSE,
      nested=FALSE, include = c( "const", "trend", "none", "both"),
      common=c("none", "include", "lags", "both"), model=c("TAR", "MTAR"), ML=seq_len(mL),
      MM=seq_len(mM), MH=seq_len(mH), nthresh=1, trim=0.15, type=c("level", "diff", "ADF"),
      restriction=c("none", "OuterSymAll", "OuterSymTh") )
```

**Arguments**

<code>x</code>	time series
<code>m, d, steps</code>	embedding dimension, time delay, forecasting steps
<code>series</code>	time series name (optional)
<code>mL, mM, mH</code>	autoregressive order for ‘low’ (mL) ‘middle’ (mM, only useful if nthresh=2) and ‘high’ (mH) regime (default values: m). Must be <=m. Alternatively, you can specify ML
<code>thDelay</code>	‘time delay’ for the threshold variable (as multiple of embedding time delay d)
<code>mTh</code>	coefficients for the lagged time series, to obtain the threshold variable
<code>thVar</code>	external threshold variable
<code>th</code>	threshold value (if missing, a search over a reasonable grid is tried)
<code>trace</code>	should additional infos be printed? (logical)
<code>include</code>	Type of deterministic regressors to include
<code>common</code>	Indicates which elements are common to all regimes: no, only the include variables, the lags or both
<code>ML, MM, MH</code>	vector of lags for order for ‘low’ (ML) ‘middle’ (MM, only useful if nthresh=2) and ‘high’ (MH) regime. Max must be <=m
<code>model</code>	Whether the threshold variable is taken in levels (TAR) or differences (MTAR)
<code>nthresh</code>	Number of threshold of the model
<code>trim</code>	trimming parameter indicating the minimal percentage of observations in each regime. Default to 0.15
<code>type</code>	Whether the variable is taken is level, difference or a mix (diff y= y-1, diff lags) as in the ADF test
<code>restriction</code>	Restriction on the threshold. OuterSymAll will take a symmetric threshold and symmetric coefficients for outer regimes. OuterSymTh currently unavailable
<code>nested</code>	Whether is this a nested call? (useful for correcting final model df)

**Details**

Self Exciting Threshold AutoRegressive model.

$$X_{t+s} = x_{t+s} = (\phi_{1,0} + \phi_{1,1}x_t + \phi_{1,2}x_{t-d} + \dots + \phi_{1,mL}x_{t-(mL-1)d})I(z_t \leq th) + (\phi_{2,0} + \phi_{2,1}x_t + \phi_{2,2}x_{t-d} + \dots + \phi_{2,mH}x_{t-(mH-1)d})I(z_t > th)$$

with  $z$  the threshold variable. The threshold variable can alternatively be specified by (in that order):

**thDelay**  $z[t] = x[t - thDelay*d]$

**mTh**  $z[t] = x[t] mTh[1] + x[t-d] mTh[2] + \dots + x[t-(m-1)d] mTh[m]$

**thVar**  $z[t] = thVar[t]$

For fixed  $th$  and threshold variable, the model is linear, so  $\phi_1$  and  $\phi_2$  estimation can be done directly by CLS (Conditional Least Squares). Standard errors for  $\phi_1$  and  $\phi_2$  coefficients provided by the summary method for this model are taken from the linear regression theory, and are to be considered asymptotical.

**Value**

An object of class `nlar`, subclass `setar`

**Author(s)**

Antonio, Fabio Di Narzo

**References**

Non-linear time series models in empirical finance, Philip Hans Franses and Dick van Dijk, Cambridge: Cambridge University Press (2000).

Non-Linear Time Series: A Dynamical Systems Approach, Tong, H., Oxford: Oxford University Press (1990).

**See Also**

[plot.setar](#) for details on plots produced for this model from the `plot` generic.

**Examples**

```
#fit a SETAR model, with threshold as suggested in Tong(1990, p 377)
mod.setar <- setar(log10(lynx), m=2, thDelay=1, th=3.25)
mod.setar
summary(mod.setar)

## example in Tsay (2005)
data(m.unrate)
setar(diff(m.unrate), ML=c(2,3,4,12), MH=c(2,4,12), th=0.1, include="none")
```

---

setar.sim

*Simulation and bootstrap of Threshold Autoregressive model (SETAR)*

---

**Description**

Simulate or bootstrap a Threshold AR (SETAR) The `setar.sim` function allows to simulate a SETAR model from scratch. The most important argument is the `B` argument, which should be a one row matrix, with first the constant/trend arguments, then the slope coefficients, and this for each regime (lower, middle, high). Other arguments such as `lag`, `nthresh` indicate the dimension of this matrix. As an example, a SETAR with 2 lags, 1 threshold, a constant, would have coefficient in the order: `c(const_L, phi_1_L, phi_2_L, const_H, phi_1_H, phi_2_H)` where L is for Lower regime, H for Higher.

`setar.boot` on the other side resample/bootstraps an existing `setar` output. It uses a recursive approach, reconstructing the series. Residuals from the original model are resampled using different bootstrap schemes, see [resample\\_vec](#).

**Usage**

```

setar.boot(
  setarObject,
  boot.scheme = c("resample", "resample_block", "wild1", "wild2", "check"),
  seed = NULL,
  ...
)

linear.boot(
  linearObject,
  boot.scheme = c("resample", "resample_block", "wild1", "wild2", "check"),
  seed = NULL,
  ...
)

setar.sim(
  B,
  n = 200,
  lag = 1,
  include = c("const", "trend", "none", "both"),
  nthresh = 1,
  Thresh,
  starting = NULL,
  innov = rnorm(n),
  ...
)

linear.sim(
  B,
  n = 200,
  lag = 1,
  include = c("const", "trend", "none", "both"),
  starting = NULL,
  innov = rnorm(n),
  ...
)

```

**Arguments**

setarObject	Bootstrap: the <a href="#">setar</a> object to resample data from.
boot.scheme	Bootstrap: which resampling scheme to use for the residuals. See <a href="#">resample_vec</a> .
seed	Bootstrap: seed used in the resampling
...	additional arguments for the unexported <code>setar.gen</code> .
linearObject	Bootstrap: the <a href="#">linear</a> object to resample data from.
B	Simulation: vector of coefficients to simulate from.
n	Simulation: Number of observations to simulate.

Thresh, nthresh, lag, include  
 Simulation: parameters for the SETAR to simulate. See [setar](#) for their description.

starting  
 Simulation: Starting values (same length as lag)

innov  
 Simulation: time series of innovations/residuals.

**Value**

a list with the simulated/bootstrapped data and the parameter matrix used.

**Author(s)**

Matthieu Stigler

**See Also**

[SETAR](#) to estimate a SETAR, [arima.sim](#) to simulate an ARMA.

**Examples**

```
##Simulation of a TAR with 1 threshold
TvarMat <- c(2.9,-0.4,-0.1,-1.5, 0.2,0.3)
sim<-setar.sim(B=TvarMat,lag=2, type="simul", nthresh=1, Thresh=2, starting=c(2.8,2.2))
mean(iffelse(sim>2,1,0)) #approximation of values over the threshold

#check the result
selectSETAR(sim, m=2)

##Bootstrap a TAR with two threshold (three regimes)
sun <- (sqrt(sunspot.year+1)-1)*2
sun_est <- setar(sun, nthresh=2, m=2)
sun_est_boot <- setar.boot(sun_est)
head(sun_est_boot)

##Check the bootstrap: with no resampling, is it the same series?
sun_est_boot <- setar.boot(sun_est, boot.scheme = "check")
all.equal(as.numeric(sun), sun_est_boot)
```

---

setarTest

*Test of linearity against threshold (SETAR)*

---

**Description**

Test of linearity against threshold of Hansen (1999) with bootstrap distribution



**Usage**

```

setarTest(
  x,
  m,
  thDelay = 0,
  trim = 0.1,
  include = c("const", "trend", "none", "both"),
  nboot = 10,
  test = c("1vs", "2vs3"),
  hpc = c("none", "foreach"),
  boot.scheme = c("resample", "resample_block", "wild1", "wild2", "check"),
  seed = NULL
)

```

**Arguments**

x	time series
m, thDelay	lag and 'time delay' for the threshold variable
trim	trimming parameter indicating the minimal percentage of observations in each regime
include	Type of deterministic regressors to include: none, a constant, a trend, or constant and trend (both).
nboot	number of bootstrap replications
test	whether to test AR against SETAR, or SETAR(1 reg) against SETAR(2 reg)
hpc	Possibility to run the bootstrap on parallel core. See details in
boot.scheme	Type of resampling scheme to use for the residuals. See <a href="#">resample_vec</a> .
seed	Seed used in the bootstrap resampling <a href="#">TVECM.HStest</a>

**Details**

Estimation of the first threshold parameter is made with CLS, a conditional search with one iteration is made for the second threshold. The Ftest comparing the residual sum of squares (SSR) of each model is computed.

$$F_{ij} = T((S_i - S_j)/S_j)$$

where  $S_i$  is the SSR of the model with i regimes (and so i-1 thresholds).

Three test are available. The both first can be seen as linearity test, whereas the third can be seen as a specification test: once the 1vs2 or/and 1vs3 rejected the linearity and henceforth accepted the presence of a threshold, is a model with one or two thresholds preferable?

Test **1vs2**: Linear AR versus 1 threshold TAR

Test **1vs3**: Linear AR versus 2 thresholds TAR

Test **2vs3**: 1 threshold TAR versus 2 thresholds TAR

The two first tests are computed together and available with test="1vs". The third test is available with test="2vs3".

The homoskedastic bootstrap distribution is based on resampling the residuals from H0 model (ar for test 1vs, and setar(1) for test 2vs3), estimating the threshold parameter and then computing the Ftest, so it involves many computations and is pretty slow.

### Value

A object of class "Hansen99Test" containing:

SSRs	The residual Sum of squares of model AR, 1 threshold TAR and 2 thresholds TAR
Ftests	The Ftest statistic for the test
PvalBoot	The bootstrap p-values for the test selected
CriticalValBoot	The critical values for the test selected
Ftestboot	All the F-test computed
firstBests, secBests	The thresholds for the original series, obtained from search for 1 thresh (firstBests) and conditional search for 2 thresh (secBests)
nboot, m	The number of bootstrap replications (nboot), the lags used (m)

### Author(s)

Matthieu Stigler

### References

Hansen (1999) Testing for linearity, Journal of Economic Surveys, Volume 13, Number 5, December 1999, pp. 551-576(26) available at: <http://www.ssc.wisc.edu/~bhansen/papers/cv.htm>

### See Also

[TVAR.LRtest](#) for the multivariate version. [SETAR](#) for estimation of the model.

### Examples

```
#Data used by Hansen
sun <- (sqrt(sunspot.year + 1) - 1) * 2

#Test 1vs2 and 1vs3
#setarTest(sun, m=11, thDelay=0:1, nboot=5, trim=0.1, test="1vs")
```

---

`setarTest_IIPUs_results`*Results from the setarTest, applied on Hansen (1999) data*

---

**Description**

Saved objects

**Usage**`setarTest_IIPUs_results`**Format**

A list containing output from two tests

**See Also**

Example in [IIPUs](#)

**Examples**

```
library(tsDyn)
data(IIPUs)
B <- 1000
## Not run:
test_1 <- setarTest(IIPUs, m=16, thDelay=5, nboot=B)
test_2 <- setarTest(IIPUs, m=16, thDelay=5, nboot=B, test = "2vs3")

## End(Not run)
```

---

`sigmoid`*sigmoid functions*

---

**Description**

Some sigmoid functions. See R sources for their definition

**Usage**`sigmoid(x)``dsigmoid(x)`**Arguments**

x                    numeric vector

**Author(s)**

J. L. Aznarte

STAR

*STAR model***Description**

STAR model fitting with automatic selection of the number of regimes based on LM tests.

**Usage**

```
star(x, m=2, noRegimes, d = 1, steps = d, series, rob = FALSE,
     mTh, thDelay, thVar, sig=0.05, trace=TRUE, control=list(), ...)
```

**Arguments**

x	time series
m, d, steps	embedding dimension, time delay, forecasting steps
noRegimes	max number of regimes
series	time series name (optional)
rob	perform robust test (not implemented)
thDelay	'time delay' for the threshold variable (as multiple of embedding time delay d)
mTh	coefficients for the lagged time series, to obtain the threshold variable
thVar	external threshold variable
sig	significance level for the tests to select the number of regimes.
control	further arguments to be passed as control list to <code>optim</code>
trace	should additional infos be printed out?
...	currently unused

**Details**

The function `star` implements the iterative building strategy described in [1] to identify and estimate Smooth Transition Autoregressive models.

[1] T. Terasvirta, "Specification, estimation and evaluation of smooth transition autoregressive models", *J. Am. Stat. Assoc.* 89 (1994): 208-218.

**Value**

`star` returns an object of class `nlar`, subclass `star`, i.e. a list with informations about the fitted model.

**Author(s)**

J. L. Aznarte M.

**See Also**

[addRegime](#)

**Examples**

```
mod.star <- star(log10(lynx), mTh=c(0,1), control=list(maxit=3000))
mod.star

addRegime(mod.star)
```

---

toLatex.setar

*Latex representation of fitted setar models*

---

**Description**

Produce LaTeX output of the SETAR model.

**Usage**

```
## S3 method for class 'setar'
toLatex(object, digits = 3, label, ...)
```

**Arguments**

object	fitted setar model (using <a href="#">nlar</a> )
digits	options to be passed to <a href="#">format</a> for formatting numbers
label	LaTeX label passed to the equation
...	Not used

**Author(s)**

Antonio, Fabio Di Narzo

**See Also**

[setar](#), [nlar-methods](#)

**Examples**

```
mod.setar <- setar(log10(lynx), m=2, thDelay=1, th=3.25)
toLatex(mod.setar)
```

TVAR

*Multivariate Threshold Vector Autoregressive model***Description**

Estimate a multivariate Threshold VAR (TVAR), either using lags as transition variable (default), or specifying an external variable.

**Usage**

```
TVAR(
  data,
  lag,
  include = c("const", "trend", "none", "both"),
  model = c("TAR", "MTAR"),
  commonInter = FALSE,
  nthresh = 1,
  thDelay = 1,
  mTh = 1,
  thVar,
  trim = 0.1,
  ngrid,
  gamma = NULL,
  around,
  plot = FALSE,
  dummyToBothRegimes = TRUE,
  trace = TRUE,
  trick = "for",
  max.iter = 2
)
```

**Arguments**

data	time series
lag	Number of lags to include in each regime
include	Type of deterministic regressors to include
model	Whether the transition variable is taken in levels (TAR) or difference (MTAR)
commonInter	Whether the deterministic regressors are regime specific (commonInter=FALSE) or not.
nthresh	Number of thresholds
thDelay	'time delay' for the threshold variable (as multiple of embedding time delay d) PLEASE NOTE that the notation is currently different to univariate models in tsDyn. The left side variable is taken at time t, and not t+1 as in univariate cases.
mTh	combination of variables with same lag order for the transition variable. Either a single value (indicating which variable to take) or a combination

<code>thVar</code>	Optional. External transition variable.
<code>trim</code>	trimming parameter indicating the minimal percentage of observations in each regime
<code>ngrid</code>	number of elements of the grid, especially for <code>nthresh=3</code>
<code>gamma</code>	prespecified threshold values
<code>around</code>	The grid search is restricted to <i>ngrid</i> values around this point. Especially useful for <code>nthresh=3</code> .
<code>plot</code>	Whether a plot showing the results of the grid search should be printed
<code>dummyToBothRegimes</code>	Whether the dummy in the one threshold model is applied to each regime or not.
<code>trace</code>	should additional infos be printed out?
<code>trick</code>	type of R function called: <code>for</code> or <code>mapply</code>
<code>max.iter</code>	Number of iterations for the algorithm

### Details

For fixed `th` and threshold variable, the model is linear, so estimation can be done directly by CLS (Conditional Least Squares). The search of the parameters values is made upon a grid of potential values. So it is pretty slow.

`nthresh=1`: estimation of one threshold model (two regimes) upon a grid of *ngrid* values (default to ALL) possible thresholds and delays values.

`nthresh=2`: estimation of two thresholds model (three regimes) Conditional on the threshold found in model where `nthresh=1`, the second threshold is searched. When both are found, a second grid search is made with 30 values around each threshold.

`nthresh=3`: DOES NOT estimate a 3 thresholds model, but a 2 thresholds model with a whole grid over the thresholds parameters (so is really slow) with a given delay, is there rather to check the consistency of the method `nthresh=2`

### Value

An object of class TVAR, with standard methods.

### Author(s)

Matthieu Stigler

### References

Lo and Zivot (2001) "Threshold Cointegration and Nonlinear Adjustment to the Law of One Price," *Macroeconomic Dynamics*, Cambridge University Press, vol. 5(4), pages 533-76, September.

### See Also

[lineVar](#) for the linear VAR/VECM, [TVAR.LRtest](#) to test for TVAR, [TVAR.sim](#) to simulate/bootstrap a TVAR.

**Examples**

```

data(zeroyld)

tv <- TVAR(zeroyld, lag=2, nthresh=2, thDelay=1, trim=0.1, mTh=1, plot=FALSE)

print(tv)
summary(tv)

# a few useful methods:
plot(tv)
predict(tv)
c(AIC(tv), BIC(tv), logLik(tv))

```

---

TVAR.LRtest	<i>Test of linearity</i>
-------------	--------------------------

---

**Description**

Multivariate extension of the linearity against threshold test from Hansen (1999) with bootstrap distribution

**Usage**

```

TVAR.LRtest(
  data,
  lag = 1,
  trend = TRUE,
  series,
  thDelay = 1:m,
  mTh = 1,
  thVar,
  nboot = 10,
  plot = FALSE,
  trim = 0.1,
  test = c("1vs", "2vs3"),
  model = c("TAR", "MTAR"),
  hpc = c("none", "foreach"),
  trace = FALSE,
  check = FALSE
)

```

**Arguments**

data	multivariate time series
lag	Number of lags to include in each regime
trend	whether a trend should be added



series	name of the series
thDelay	'time delay' for the threshold variable (as multiple of embedding time delay d) PLEASE NOTE that the notation is currently different to univariate models in tsDyn. The left side variable is taken at time t, and not t+1 as in univariate cases.
mTh	combination of variables with same lag order for the transition variable. Either a single value (indicating which variable to take) or a combination
thVar	external transition variable
nboot	Number of bootstrap replications
plot	Whether a plot showing the results of the grid search should be printed
trim	trimming parameter indicating the minimal percentage of observations in each regime
test	Type of usual and alternative hypothesis. See details
model	Whether the threshold variable is taken in level (TAR) or difference (MTAR)
hpc	Possibility to run the bootstrap on parallel core. See details in <a href="#">TVECM.HStest</a>
trace	should additional infos be printed? (logical)
check	Possibility to check the function by no sampling: the test value should be the same as in the original data

## Details

This test is just the multivariate extension proposed by Lo and Zivot of the linearity test of Hansen (1999). As in univariate case, estimation of the first threshold parameter is made with CLS, for the second threshold a conditional search with one iteration is made. Instead of a Ftest comparing the SSR for the univariate case, a Likelihood Ratio (LR) test comparing the covariance matrix of each model is computed.

$$LR_{ij} = T(\ln(\det \hat{\Sigma}_i) - \ln(\det \hat{\Sigma}_j))$$

where  $\hat{\Sigma}_i$  is the estimated covariance matrix of the model with i regimes (and so i-1 thresholds).

Three test are available. The both first can be seen as linearity test, whereas the third can be seen as a specification test: once the 1vs2 or/and 1vs3 rejected the linearity and henceforth accepted the presence of a threshold, is a model with one or two thresholds preferable?

Test **1vs2**: Linear VAR versus 1 threshold TVAR

Test **1vs3**: Linear VAR versus 2 threshold2 TVAR

Test **2vs3**: 1 threshold TAR versus 2 threshold2 TAR

The both first are computed together and available with test="1vs". The third test is available with test="2vs3".

The homoskedastik bootstrap distribution is based on resampling the residuals from H0 model, estimating the threshold parameter and then computing the Ftest, so it involves many computations and is pretty slow.

**Value**

A list containing:

- The values of each LR test
- The bootstrap Pvalues and critical values for the test selected

**Author(s)**

Matthieu Stigler

**References**

Hansen (1999) Testing for linearity, *Journal of Economic Surveys*, Volume 13, Number 5, December 1999, pp. 551-576(26) available at: <http://www.ssc.wisc.edu/~bhansen/papers/cv.htm>

Lo and Zivot (2001) "Threshold Cointegration and Nonlinear Adjustment to the Law of One Price," *Macroeconomic Dynamics*, Cambridge University Press, vol. 5(4), pages 533-76, September.

**See Also**

[setarTest](#) for the univariate version. [OlsTVAR](#) for estimation of the model.

**Examples**

```
data(zeroyld)
data<-zeroyld
```

```
TVAR.LRtest(data, lag=2, mTh=1, thDelay=1:2, nboot=3, plot=FALSE, trim=0.1, test="1vs")
```

---

TVAR.sim

*Simulation of a multivariate Threshold Autoregressive model (TVAR)*

---

**Description**

Simulate a multivariate Threshold VAR (TVAR)

**Usage**

```
TVAR.sim(  
  B,  
  Thresh,  
  nthresh = 1,  
  n = 200,  
  lag = 1,  
  include = c("const", "trend", "none", "both"),  
  thDelay = 1,  
  mTh = 1,
```

```

    starting = NULL,
    innov = rmnorm(n, varcov = varcov),
    varcov = diag(1, nrow(B)),
    show.parMat = FALSE,
    ...
)

TVAR.boot(
  TVARobject,
  innov,
  seed,
  boot.scheme = c("resample", "wild1", "wild2", "check"),
  ...
)

```

### Arguments

B	Matrix of coefficients to simulate
Thresh	The threshold value(s). Vector of length nthresh
nthresh	number of threshold (see details)
n	Number of observations to create when type="simul"
lag	Number of lags to include in each regime
include	Type of deterministic regressors to include. NOT WORKING PROPERLY CURRENTLY if not const
thDelay	'time delay' for the threshold variable (as multiple of embedding time delay d) PLEASE NOTE that the notation is currently different to univariate models in tsDyn. The left side variable is taken at time t, and not t+1 as in univariate cases.
mTh	combination of variables with same lag order for the transition variable. Either a single value (indicating which variable to take) or a combination
starting	Starting values (matrix of dimension lag x k). If not given, set to zero.
innov	Innovations used for simulation. Should be matrix of dim n x k. By default multivariate normal. For the bootstrap case TVAR.boot, residuals are resampled if argument is missing.
varcov	Variance-covariance matrix for the innovations. By default identity matrix.
show.parMat	Logical. Should the parameter matrix be shown? Useful to understand how to give right input
...	Further arguments passed to the underlying (un-exported) TVAR.gen function
TVARobject	Object of class TVAR generated by function <a href="#">TVAR</a>
seed	Optional. Seed for the random resampling function.
boot.scheme	The bootstrap scheme.

## Details

This function offers the possibility to generate series following a TVAR.

By giving a matrix of coefficients, one can only simulate a VAR (nthresh=0) or TVAR (nthresh=1 or 2). One can have a specification with constant (default), trend, both or none (see arg include). Order in parameters is include/lags (VECM) and include/lags/include/lags for TVECM, hence, a matrix for a TVECM with 3 regimes, a const and a 2 lags would have 2 lines and 2\*(1+4) columns. The innovations can be given by the user (a matrix of dim n x k, here n does not include the starting values!), by default it uses a multivariate normal distribution, with covariance matrix specified by varcov. The starting values (of dim lags x k) can be given. The user should take care for their choice, since it is not sure that the simulated values will cross the threshold even once.

The matrix 'B' has to be in the form: constant, trend, lags, then repeated if many regimes. In case of uncertainty, using `who.parMat=TRUE` will print the matrix as interpreted by the function, helping the user to feed the right input.

For the bootstrap, the function resamples data from a given TVAR model generated by [TVAR](#), returning the resampled data. A residual recursive bootstrap is used, where one uses either a simple resampling, or the Wild bootstrap, either with a normal distribution (wild1) or inverting the sign randomly (wild2)

## Value

A matrix with the simulated/bootstrapped series.

## Author(s)

Matthieu Stigler

## See Also

[TVAR](#) to estimate the TVAR. Similar [TVECM.sim](#) and [TVECM.boot](#) for TVECM, [VAR.sim](#) and [VAR.boot](#) for VAR models estimated with [lineVar](#).

## Examples

```
## TVAR.sim: Simulation of a TVAR with 1 threshold
B <- rbind(c(0.11928245, 1.00880447, -0.009974585, -0.089316, 0.95425564, 0.02592617),
          c(0.25283578, 0.09182279, 0.914763741, -0.0530613, 0.02248586, 0.94309347))
colnames(B) <- paste(rep(c("Const", "Lag_1_var1", "Lag_1_var2"), 2), c("Low", "High"), sep="_")
sim <- TVAR.sim(B=B,nthresh=1,n=500, mTh=1, Thresh=5, starting=matrix(c(5.2, 5.5), nrow=1))

#estimate the new serie
TVAR(sim, lag=1, dummyToBothRegimes=TRUE)
```

```
## TVAR.boot: Bootstrap a TVAR with two threshold (three regimes)
data(zeroyld)
serie <- zeroyld
mod <- TVAR(data=serie,lag=1, nthresh=1)
TVAR.boot(mod)
```

TVECM

*Threshold Vector Error Correction model (VECM)***Description**

Estimate a Threshold Vector Error Correction model (VECM)

**Usage**

```
TVECM(
  data,
  lag = 1,
  nthresh = 1,
  trim = 0.05,
  ngridBeta = 50,
  ngridTh = 50,
  plot = TRUE,
  th1 = list(exact = NULL, int = c("from", "to"), around = "val"),
  th2 = list(exact = NULL, int = c("from", "to"), around = "val"),
  beta = list(exact = NULL, int = c("from", "to"), around = c("val", "by")),
  restr = c("none", "equal", "signOp"),
  common = c("All", "only_ECT"),
  include = c("const", "trend", "none", "both"),
  dummyToBothRegimes = TRUE,
  beta0 = 0,
  methodMapply = FALSE,
  trace = TRUE
)
```

**Arguments**

data	time series
lag	Number of lags to include in each regime
nthresh	number of threshold (see details)
trim	trimming parameter indicating the minimal percentage of observations in each regime
ngridBeta	number of elements to search for the cointegrating value
ngridTh	number of elements to search for the threshold value
plot	Whether the grid with the SSR of each threshold should be plotted.

th1	different possibilities to pre-specify an exact value, an interval or a central point for the search of the threshold (or first threshold if nthresh=2)
th2	different possibilities to pre-specify an exact value or a central point for the search of the second threshold (used only if nthresh=2)
beta	different possibilities to pre-specify an exact value, an interval or a central point for the search of the cointegrating value
restr	Currently not available
common	Whether the regime-specific dynamics are only for the ECT or for the ECT and the lags
include	Type of deterministic regressors to include
dummyToBothRegimes	Whether the dummy in the one threshold model is applied to each regime or not.
beta0	Additional regressors to include in the cointegrating relation
methodMapply	only for programming. Is to make the choice between a for loop or mapply implementation
trace	should additional infos be printed? (logical)

## Details

For fixed threshold and cointegrating vector, the model is linear, so estimation of the regression parameters can be done directly by CLS (Conditional Least Squares). The search of the threshold and cointegrating parameters values which minimize the residual sum of squares (SSR) is made on a grid of potential values. For specification of the grids, see below.

The function can estimate one as well as two thresholds:

**nthresh=1:** estimation of one threshold model (two regimes) upon a grid of *ngridTh* values (default to ALL) possible thresholds and delays values.

**nthresh=2:** estimation of two thresholds model (three regimes). Conditional on the threshold found in model where nthresh=1, the second threshold is searched. When both are found, a second grid search is made with 30 values around each threshold.

The model can be either with a threshold effect on all variables ("All") or only on the error correction term (ECT) (argument "only ECT"). In the second case, the value for the middle threshold is taken a null, as in Balke and Fomby (1997).

The grid for the threshold parameters can be set in different ways, through the argument *th1*, *th2* and *beta*:

**exact:** Pre-specified value (for beta: cointegrating vector will be  $c(1, -\text{beta})$ )

**int:** Specify an interval (of length *ngridTh*) in which to search.

**around:** Specify to take *ngridTh* points around the value given.

The default is to do an interval search. Interval bounds for the threshold interval are simply the *trim* and  $1-\text{trim}$  percents of the sorted error correction term. For the cointegrating parameter, bounds of the interval are obtained from the (OLS) confidence interval of the linear cointegration case. It is often found however that this interval is too tight. It is hence recommended to inspect the plot of the grid search.

**Value**

Fitted model data

**Author(s)**

Matthieu Stigler

**References**

Hansen, B. and Seo, B. (2002), Testing for two-regime threshold cointegration in vector error-correction models, *Journal of Econometrics*, 110, pages 293 - 318

Seo, M. H. (2009) Estimation of non linear error-correction models, Working paper

**See Also**

[VECM](#) for the linear VECM, [TVAR](#) for the threshold VAR, [TVECM.SeoTest](#) to test for TVECM, [TVECM.sim](#) to simulate/bootstrap a TVECM.

**Examples**

```
data(zeroyld)

##Estimate a TVECM (we use here minimal grid, it should be usually much bigger!)

tvec <- TVECM(zeroyld, nthresh=2, lag=1, ngridBeta=20, ngridTh=30, plot=TRUE, trim=0.05, common="All")

print(tvec)
summary(tvec)

#Obtain diverse infos:
AIC(tvec)
BIC(tvec)

res.tvec<-residuals(tvec)

#export the equations as Latex:
toLatex(tvec)
```

---

TVECM.HStest

*Test of linear cointegration vs threshold cointegration*

---

**Description**

Tests the null of linear cointegration against threshold cointegration following Hansen and Seo (2002). Fixed regressor and residual bootstrap are available.

**Usage**

```
TVECM.HStest(data, lag=1, ngridTh=300, trim=0.05,
             nboot=100, fixed.beta=NULL, intercept=TRUE,
             boot.type=c("FixedReg", "ResBoot"),
             hpc=c("none", "foreach"))
```

**Arguments**

<code>data</code>	Time series
<code>lag</code>	Number of lags to include in each regime
<code>ngridTh</code>	Number of threshold grid points to evaluate the optimal threshold.
<code>trim</code>	Trimming parameter indicating the minimal percentage of observations in each regime
<code>nboot</code>	Number of bootstrap replications
<code>fixed.beta</code>	Numeric. User pre-specified cointegrating value, as in <a href="#">VECM</a> (i.e. cointegrating vector will be $c(1, -\text{beta.fixed})$ ). When NULL (default), the value is estimated from the linear VECM.
<code>intercept</code>	Logical. Whether an intercept has to be included in the VECM
<code>boot.type</code>	Character. Type of bootstrap simulation (only if <code>nboot&gt;0</code> )
<code>hpc</code>	Possibility to run the bootstrap on parallel core. See details

**Details**

This test follows the implementation done by Hansen and Seo (2002). The cointegrating value is estimated from the linear VECM. Then, conditional on this value, the LM test is run for a range of different threshold values. The maximum of those LM test values is reported.

Two bootstrap are available: a fixed regressor, as well as a usual residual bootstrap (using the function [TVECM.sim](#)).

Available methods are `print()`, `summary()` and `plot()`.

With the argument `hpc`, the burdensome bootstrap replication can be run on parallel cores, thus alleviating the time of computation. The user needs simply to choose a package (among `doMC`, `doMPI`, `doSNOW` or `doRedis`) and register the backend. See the vignette for more details.

**Value**

A list containing diverse values:

<code>stat</code>	The sup-LM statistic.
<code>values</code>	The whole LM values.
<code>PvalBoot</code>	The bootstrap p-value
<code>CriticalValBoot</code>	The bootstrap critical values
<code>allBoots</code>	The boot sup-LM values
<code>args</code>	Some user given args ( <code>nboot</code> , <code>boot.type</code> )



## Reproducibility

Comparison with original paper is made difficult as values of the test are not shown in the paper, only their critical values, which depend on random bootstrap.

Comparison is done with the GAUSS code available on the page of Bruce Hansen. Running `tar_ci`, we have the same sup-LM value when `lags=1` and `lags=2`, a higher value with `lag=3`. When the test is run with pre-specified beta values, we have different results, sometimes higher but also smaller sup-LM value.

## Author(s)

Matthieu Stigler

## References

Hansen, B. and Seo, B. (2002), Testing for two-regime threshold cointegration in vector error-correction models, *Journal of Econometrics*, 110, pages 293 - 318

## See Also

[zeroyld](#): data used in the original paper of Hansen and Seo.

[TVECM.SeoTest](#): a similar test, but with null hypothesis of no-cointegration.

[TVECM](#) for estimating a TVECM, [TVECM.sim](#) for simulating/bootstrap a TVECM,

## Examples

```
#Use original data from paper:
data(zeroyld)
dataPaper<-zeroyld
# Test: nboot, number of bootstrap replications, should be high
## Not run:
test1<-TVECM.HStest(dataPaper, lag=1, intercept=TRUE, nboot=1000)

## End(Not run)

#we use here for the example a much smaller number of bootstrap:
test1<-TVECM.HStest(dataPaper, lag=1, intercept=TRUE, nboot=10)

test1
summary(test1)
plot(test1)

#can have only specific plots:
plot(test1, which="LM values")
plot(test1, which="Density")

## Run the function in parallel:
## Not run:
#we show here the use with package doMC
library(doMC)
registerDoMC(2) #Number of cores
```

```
test1<-TVECM.HStest(dataPaper, lag=1, intercept=TRUE, nboot=1000, hpc="foreach")
## End(Not run)
```

---

TVECM.SeoTest

*No cointegration vs threshold cointegration test*


---

### Description

Test the null of no cointegration against threshold cointegration with bootstrap distribution of Seo (2006)

### Usage

```
TVECM.SeoTest(
  data,
  lag,
  beta,
  trim = 0.1,
  nboot,
  plot = FALSE,
  hpc = c("none", "foreach"),
  check = FALSE
)
```

### Arguments

data	time series
lag	Number of lags to include in each regime
beta	Pre-specified cointegrating value (i.e. cointegrating vector will be $c(1, -\beta)$ )
trim	trimming parameter indicating the minimal percentage of observations in each regime
nboot	Number of bootstrap replications
plot	Whether a grid with the SSR of each threshold should be printed
hpc	Possibility to run the bootstrap on parallel core. See details in <a href="#">TVECM.HStest</a>
check	Possibility to check the function by no sampling: the test value should be the same as in the original data

### Details

For this test, the cointegrating value has to be specified by the user.

The model used is one where the threshold effect concerns only the cointegrating vector, and only in the outer regimes.

Due to the presence of parameters unidentified under the null hypothesis, the test employed is a Sup-Wald test, that means that for each combination of the thresholds, a Wald Test is computed and the supremum of all tests is taken. For each bootstrap replication, this approach is taken, so that the test is really slow.

### Value

A list containing diverse informations:

Estimated threshold parameters and usual slope parameters.

Value of the test.

Critical and Pvalue from bootstrap distribution.

### Author(s)

Matthieu Stigler

### References

Seo, Myunghwan, 2006. "Bootstrap testing for the null of no cointegration in a threshold vector error correction model," *Journal of Econometrics*, vol. 127(1), pages 129-150, September.

### See Also

[TVECM](#) for estimating a TVECM, [TVECM.sim](#) for simulating/bootstrap a TVECM,

### Examples

```
# As the function takes long long time to be executed, we show in in don't run environment
## Not run:
data(zeroyld)

#can be useful to check whether the bootstrap is working:
#without sampling, results of boot should be same as original
#this is indeed not always the case duye to floating point algorithm
TVECM.SeoTest(zeroyld,lag=2, beta=1, trim=0.1,nboot=2, plot=FALSE,check=TRUE)

#then run the function:
TVECM.SeoTest(zeroyld,lag=2, beta=1, trim=0.1,nboot=100, plot=FALSE,check=FALSE)

## End(Not run)
```

**Description**

Estimate or bootstraps a multivariate Threshold VAR

**Usage**

```
TVECM.sim(
  B,
  n = 200,
  lag = 1,
  include = c("const", "trend", "none", "both"),
  beta,
  nthresh = 1,
  Thresh,
  starting = NULL,
  innov = rmnorm(n, varcov = diag(1, nrow(B))),
  show.parMat = FALSE,
  returnStarting = FALSE,
  ...
)

VECM.sim(
  B,
  n = 200,
  lag = 1,
  include = c("const", "trend", "none", "both"),
  beta,
  starting = NULL,
  innov = rmnorm(n, varcov = diag(1, nrow(B))),
  show.parMat = FALSE,
  returnStarting = FALSE,
  ...
)

VECM.boot(
  object,
  boot.scheme = c("resample", "resample_block", "wild1", "wild2", "check"),
  seed = NULL,
  ...
)

TVECM.boot(
  object,
  boot.scheme = c("resample", "resample_block", "wild1", "wild2", "check"),
```

```

    seed = NULL,
    ...
)

```

### Arguments

B	Simulation: Matrix of coefficients to simulate
n	Simulation: Number of observations to simulate.
beta	The cointegrating value
Thresh, nthresh, lag, include	Simulation: parameters for the VECM/TVECM to simulate. See <a href="#">TVECM</a> for their description.
starting	Simulation: Starting values (same length as lag = 1)
innov	Simulation: time series of innovations/residuals.
show.parMat	Logical. Whether to show how the parameter matrix B is interpreted.
returnStarting	Logical. Whether to return the starting values.
...	additional arguments for the unexported <code>TVECM.gen</code> .
object	Object computed by function <a href="#">TVECM</a> or linear <a href="#">VECM</a>
boot.scheme	Bootstrap: which resampling scheme to use for the residuals. See <a href="#">resample_vec</a> .
seed	Bootstrap: seed used in the resampling

### Details

This function offers the possibility to generate series following a VECM/TVECM from two approaches: bootstrap or simulation. `VECM.sim` is just a wrapper for [TVECM.sim](#).

When the argument `matrix` is given, one can only simulate a VECM (`nthresh=0`) or TVECM (`nthresh=1` or `2`). One can have a specification with constant ("`const`"), "`trend`", "`both`" or "`none`" (see argument `include`). Order for the parameters is `ECT/include/lags` for VECM and `ECT1/include1/lags1/ECT2/include2/lags2` for TVECM. To be sure that once is using it correctly, setting `show.parMat = TRUE` will show the matrix of parameters together with their values and names.

The argument `beta` is the cointegrating value on the right side of the long-run relationship, and hence the function use the vector  $(1, -\beta)$ . The `innov` argument specifies the innovations. It should be given as a matrix of dim  $n \times k$ , (here  $n$  does not include the starting values!), by default it uses a multivariate normal distribution, with covariance matrix specified by `varcov`.

The starting values (of dim  $lags \times k$ ) can be given through argument `starting`. The user should take care for their choice, since it is not sure that the simulated values will cross the threshold even once. Notice that only one cointegrating value is allowed. User interested in simulating a VECM with more cointegrating values should do use the VAR representation and use [TVAR.sim](#).

The second possibility is to bootstrap series. This is done on a object generated by [TVECM](#) (or [VECM](#)). A simple residual bootstrap is done, or one can simulate a series with the same parameter matrix and with normal distributed residuals (with variance pre-specified), corresponding to Monte-carlo simulations.

One can alternatively give only the series, and then the function will call internally [TVECM](#).

**Value**

A matrix with the simulated/bootstrapped series.

**Author(s)**

Matthieu Stigler

**See Also**

[VECM](#) or [TVECM](#) to estimate the VECM or TVECM. Similar [TVAR.sim](#) and [TVAR.boot](#) for [TVAR](#), [VAR.sim](#) and [VAR.boot](#) for VAR models estimated with [lineVar](#) models.

**Examples**

```
###reproduce example in Enders (2004, 2 edition) p. 350,
# (similar example in Enders (2010, 3 edition) 301-302).

if(require(mnormt)){
#see that the full "VAR" coefficient matrix is:
A <- matrix(c(-0.2, 0.2, 0.2, -0.2), byrow=TRUE, ncol=2)

# but this is not the input of VECM.sim. You should decompose into the a and b matrix:
a<-matrix(c(-0.2, 0.2), ncol=1)
b<-matrix(c(1,-1), nrow=1)

# so that:
a*%b

# The a matrix is the input under argument B, while the b matrix is under argument beta:
# (the other zeros in B are for the not-specified lags)
innov<-rmnorm(100, varcov=diag(2))
Bvecm <- rbind(c(-0.2, 0,0), c(0.2, 0,0))
vecm1 <- VECM.sim(B=Bvecm, beta=1,n=100, lag=1,include="none", innov=innov)
ECT <- vecm1[,1]-vecm1[,2]

#add an intercept as in panel B
Bvecm2 <- rbind(c(-0.2, 0.1,0,0), c(0.2,0.4, 0,0))
vecm2 <- VECM.sim(B=Bvecm2, n=100,beta=1, lag=1,include="const", innov=innov)

par(mfrow=c(2,1))
plot(vecm1[,1], type="l", main="Panel a: no drift or intercept", ylab="", xlab="")
lines(vecm1[,2], lty=2)
plot(vecm2[,1], type="l", main="Panel b: drift terms (0.1)", ylab="", xlab="")
lines(vecm2[,2], lty=2)
}
##Bootstrap a TVAR with 1 threshold (two regimes)
data(zeroyld)
TVECMobject <- TVECM(zeroyld, nthresh=1, lag=1, ngridBeta=20, ngridTh=20, plot=FALSE, trace = FALSE)
TVECM.boot(TVECMobject)
```

```
##Check the bootstrap: do we get original series, when not resampling residuals?
TVECM.boot.check <- TVECM.boot(TVECMobject, boot.scheme = "check")
all.equal(as.data.frame(TVECM.boot.check), zeroYld)
```

---

UsUnemp

*US unemployment series used in Caner and Hansen (2001)*


---

### Description

This data, used as example in *Caner and Hansen (2001)*, contains the monthly US adult male unemployment from 1956 to 1999.

### Usage

```
data(UsUnemp)
```

### Format

A monthly time series of class `ts` starting in January 1956 and ending in August 1999.

### Source

Caner and Hansen, Threshold autoregression with a unit root *Econometrica*, 2001, 69, 1555-1596 available at: <http://www.ssc.wisc.edu/~bhansen/papers/cv.htm>

---

VAR.sim

*Simulate or bootstrap a VAR model*


---

### Description

Allow to either simulate from scratch (by providing coefficients) or bootstrap from an estimated VAR model,

### Usage

```
VAR.sim(
  B,
  n = 200,
  lag = 1,
  include = c("const", "trend", "none", "both"),
  starting = NULL,
  innov = rmnorm(n, varcov = varcov),
  varcov = diag(1, nrow(B)),
  show.parMat = FALSE,
  returnStarting = FALSE,
```

```

    ...
  )

VAR.boot(
  VARobject,
  boot.scheme = c("resample", "wild1", "wild2", "check"),
  seed,
  ...
)

```

### Arguments

B	Matrix of coefficients.
n	Number of observations to simulate
lag	Number of lags of the VAR to simulate
include	Type of deterministic regressors to include in the VAR to simulate
starting	Starting values (matrix of dimension lag x k) for the VAR to simulate. If not given, set to zero.
innov	Innovations used for in the VAR to simulate. Should be matrix of dim n x k. By default multivariate normal.
varcov	Variance-covariance matrix for the innovations. By default identity matrix.
show.parMat	Logical. Should the parameter matrix be shown? Useful to understand how to give right input
returnStarting	Whether starting values are returned. Default to FALSE
...	Further arguments passed to the underlying (un-exported) VAR.gen function
VARobject	Object of class VAR generated by function <a href="#">lineVar</a>
boot.scheme	The bootstrap scheme. See details.
seed	Optional. Seed for the random resampling function.

### Details

For the bootstrap, the function resamples data from a given VAR model generated by [lineVar](#), returning the resampled data. A residual recursive bootstrap is used, where one uses either a simple resampling, or the Wild bootstrap, either with a normal distribution (wild1) or inverting the sign randomly (wild2)

### Value

A matrix with the resampled series.

### Author(s)

Matthieu Stigler



**See Also**

[lineVar](#) to estimate the VAR. Similar [TVECM.sim](#) and [TVECM.boot](#) for [TVECM](#), [TVAR.sim](#) and [TVAR.boot](#) for [TVAR](#) models.

**Examples**

```
## VAR.sim: simulate VAR as in Enders 2004, p 268
B1<-matrix(c(0.7, 0.2, 0.2, 0.7), 2)
var1 <- VAR.sim(B=B1, n=100, include="none")
ts.plot(var1, type="l", col=c(1,2))

B2<-rbind(c(0.5, 0.5, 0.5), c(0, 0.5, 0.5))
varcov<-matrix(c(1,0.2, 0.3, 1),2)
var2 <- VAR.sim(B=B2, n=100, include="const", varcov=varcov)
ts.plot(var2, type="l", col=c(1,2))

## VAR.boot: Bootstrap a VAR
data(zeroyld)
mod <- lineVar(data=zeroyld,lag=1)
VAR.boot(mod)
```

---

VARrep

*VAR representation*


---

**Description**

Show the VAR representation of a VECM

**Usage**

```
VARrep(object, ...)

## S3 method for class 'VECM'
VARrep(object, ...)

## S3 method for class 'VAR'
VARrep(object, ...)
```

**Arguments**

object	An object of class ‘VECM’ created by <a href="#">VECM</a> , or of class ‘VAR’ created by <a href="#">lineVar</a>
...	Currently not used

**Value**

A matrix containing the parameters of the VECM under their VAR representation.

**Author(s)**

Matthieu Stigler

**References**

Hamilton (1994) *Time Series Analysis*, Princeton University Press

**Examples**

```
data(barry)

# VECM model:
mod_vecm <- VECM(barry, lag=2, estim="ML")
VARrep(mod_vecm)

# VAR model:
mod_var <- lineVar(barry, lag=2, I="diff")
VARrep(mod_var)
```

---

VECM

*Estimation of Vector error correction model (VECM)*

---

**Description**

Estimate a VECM by either Engle-Granger (2OLS) or Johansen (MLE) method.

**Usage**

```
VECM(
  data,
  lag,
  r = 1,
  include = c("const", "trend", "none", "both"),
  beta = NULL,
  estim = c("2OLS", "ML"),
  LRinclude = c("none", "const", "trend", "both"),
  exogen = NULL
)
```

**Arguments**

data	multivariate time series (first row being first=oldest value)
lag	Number of lags (in the VECM representation, see Details)
r	Number of cointegrating relationships
include	Type of deterministic regressors to include
beta	for VECM only: user-specified cointegrating values, the cointegrating vector will be taken as: (1, -beta) If NULL, will be estimated using the estimator specified in estim
estim	Type of estimator: 2OLS for the two-step approach or ML for Johansen MLE
LRinclude	Type of deterministic regressors to include in the long-term relationship. Can also be a matrix with exogenous regressors (2OLS only).
exogen	Inclusion of exogenous variables (first row being first=oldest value). Is either of same size than data (then automatically cut) or than end-sample.

**Details**

This function is just a wrapper for the `lineVar`, with `model="VECM"`.

More comprehensive functions for VECM are in package `vars`. Differences with that package are:

**Engle-Granger estimator** The Engle-Granger estimator is available

**Presentation** Results are printed in a different ways, using a matrix form

**lateX export** The matrix of coefficients can be exported to latex, with or without standard-values and significance stars

**Prediction** The `predict` method contains a `newdata` argument allowing to compute rolling forecasts.

Two estimators are available: the Engle-Granger two step approach (2OLS) or the Johansen (ML). For the 2OLS, deterministic regressors (or external variables if `LRinclude` is of class numeric) can be added for the estimation of the cointegrating value and for the ECT. This is only working when the beta value is not pre-specified.

The arg beta is the cointegrating value, the cointegrating vector will be taken as: (1, -beta).

Note that the lag specification corresponds to the lags in the VECM representation, not in the VAR (as is done in package `vars` or software GRETL). Basically, a VAR with 2 lags corresponds here to a VECM with 1 lag. The lag can be set to 0, although some methods (`irf`, `fevd`) won't work for this case.

#\*The arg beta allows to specify constrained cointegrating values, leading to  $ECT = \beta' X_{t-1}$ . It should be specified as a  $K \times r$  matrix. In case of  $r = 1$ , can also be specified as a vector. Note that the vector should be normalised, with the first value to 1, and the next values showing the opposite sign in the long-run relationship  $-\beta$ . In case the vector has  $K - 1$  values, this is what `lineVar` is doing, setting  $(1, -\beta)$ . Note finally one should provide values for all the coefficients (eventually except for special case of  $r=1$  and  $k-1$ ), if you want to provide only part of the parameters, and let the others be estimated, look at the functions in package `urca`.

The eigenvector matrix  $\beta$  is normalised using the Phillips triangular representation, see Hamilton (1994, p. 576) and Juselius (2006, p. 216), see `coefA` for more details.

**Value**

An object of class VECM (and higher classes VAR and n1Var) with methods:

**Usual methods:** Print, summary, residuals, fitted, vcov

**Fit criteria:** AIC, BIC, [MAPE](#), [mse](#), [logLik](#) (the latter only for models estimated with MLE)

**Prediction:** [predict](#) and [predict\\_rolling](#)

**coef extraction:** Extract cointegrating/adjustment coefficients, [coefA](#), [coefB](#) [coefPI](#)

**VAR/VECM methods:** Impulse response function ([irf.VECM](#)) and forecast error variance decomposition ([fevd](#))

**LaTeX:** `toLatex`

**Author(s)**

Matthieu Stigler

**References**

Hamilton (1994) Time Series Analysis, Princeton University Press

Juselius (2006) The Cointegrated VAR model, Oxford University Press

**See Also**

[coefA](#), [coefB](#) and [coefPI](#) to extract the relevant parameter matrices.

[lineVar](#) [TVAR](#) and [TVECM](#) for the corresponding threshold models. [linear](#) for the univariate AR model.

**Examples**

```
data(zeroyld)
data<-zeroyld

#Fit a VECM with Engle-Granger 2OLS estimator:
vecm.eg<-VECM(zeroyld, lag=2)

#Fit a VECM with Johansen MLE estimator:
vecm.jo<-VECM(zeroyld, lag=2, estim="ML")

#compare results with package vars:
if(require(vars)) {
  data(finland)
  #check long coint values
  all.equal(VECM(finland, lag=2, estim="ML", r=2)$model.specific$beta,
            cajorls(ca.jo(finland, K=3, spec="transitory"), r=2) $beta, check.attributes=FALSE)
  # check OLS parameters
  all.equal(t(coefficients(VECM(finland, lag=2, estim="ML", r=2))),
            coefficients(cajorls(ca.jo(finland, K=3, spec="transitory"), r=2)$rlm), check.attributes=FALSE)
}
```

```
##export to Latex
toLatex(vecm.eg)
toLatex(summary(vecm.eg))
options("show.signif.stars"=FALSE)
toLatex(summary(vecm.eg), parenthese="Pvalue")
options("show.signif.stars"=TRUE)
```

---

VECM\_symbolic

*Virtual VECM model*


---

### Description

Pedagogical tool to create a symbolic VECM model, i.e. just for representation purpose.

### Usage

```
VECM_symbolic(
  alpha,
  beta,
  lags,
  inc,
  include = c("none", "const", "trend", "both")
)
```

### Arguments

alpha	Matrix of alpha speed adjustment coefficients.
beta	Matrix of alpha, cointegrating coefficients.
lags	Matrix containing the lags coefficients.
inc	Matrix containing the include (see following arg.) coefficients.
include	Character indicating the type of deterministic term included, if any.

### Value

An object of class ‘VECM’, without however any data.

### Examples

```
a<-matrix(c(-0.4, 0.1), ncol=1)
b<-matrix(c(1, -2), ncol=2)
```

```
# VECM_symb(alpha=a, beta=t(b))
d<- VECM_symbolic(alpha=a, beta=t(b))
VARrep(d)
d<- VECM_symbolic(alpha=a, beta=t(b), lags=matrix(0, ncol=2, nrow=2))
VARrep(d)
LagMat <- matrix(c(0.1, 0.3, 0.1, 0.2), ncol=2, nrow=2)
incMat <- matrix(c(0.5, 0.1), ncol=1)
d3<- VECM_symbolic(alpha=a, beta=t(b), lags=LagMat, inc=incMat, include="const")
VARrep(d3)
```

---

zeroyld

*zeroyld time series*


---

### Description

U.S. Term Structure Data, 1951-1991. Dataset used by Hansen and Seo (2002). The data contains the 12 month short rate and 120 month long rate.

### Usage

```
zeroyld
zeroyldMeta
```

### Format

zeroyld contains two variables, while zeryldMeta contains also Year and Month columns.  
zeroyld is a data frame with 482 observations and 2 variables:

short.run	numeric	Short term, 12 month
long.run	numeric	Long term, 120 month

### Source

Hansen, B. and Seo, B. (2002), Testing for two-regime threshold cointegration in vector error-correction models, *Journal of Econometrics*, 110, pages 293 - 318

The data can be downloaded from: [http://www.ssc.wisc.edu/~bhansen/progs/joe\\_02r.zip](http://www.ssc.wisc.edu/~bhansen/progs/joe_02r.zip).

The authors themselves took the data from the webpage of Huston McCulloch: <https://www.asc.ohio-state.edu/mcculloch.2/ts/mcckwon/mccull.htm>

### See Also

[TVECM.HStest](#): Hansen and Seo test.

[TVECM](#) for estimating a TVECM.

**Examples**

```
data(zeroyldMeta)
plot(zeroyldMeta$Date, zeroyldMeta$short.run, type = "l", xlab = "Date", ylab = "Rate")
lines(zeroyldMeta$Date, zeroyldMeta$long.run, lty = 2)
legend("topleft", lty = c(1, 2), legend = c("Short rate: 12 months", "Long rate: 120 months"))
```

# Index

- \* **Testing for linearity**
    - UsUnemp, 95
  - \* **VAR**
    - VARrep, 97
  - \* **VECM**
    - rank.select, 57
    - VARrep, 97
    - VECM\_symbolic, 101
  - \* **bootstrap**
    - TVAR.sim, 82
    - VAR.sim, 95
  - \* **cointegration**
    - rank.select, 57
    - VARrep, 97
    - VECM\_symbolic, 101
  - \* **datasets**
    - barry, 12
    - IIPUs, 25
    - m.unrate, 43
    - setarTest\_IIPUs\_results, 75
    - UsUnemp, 95
    - zeroYld, 102
  - \* **predict**
    - predict\_rolling, 55
  - \* **regression**
    - fevd.nlVar, 19
    - fitted.nlVar, 20
    - irf.linear, 27
    - predict.nlar, 52
    - predict.TVAR, 54
  - \* **ts**
    - aar, 4
    - accuracy\_stat, 5
    - addRegime, 6
    - autopairs, 8
    - autotriples, 9
    - autotriples.rgl, 10
    - availableModels, 11
    - delta, 16
    - delta.lin, 18
    - getTh, 21
    - isLinear, 30
    - lags.select, 32
    - LINEAR, 33
    - lineVar, 34
    - llar, 36
    - logLik.nlVar, 39
    - LSTAR, 40
    - MAPE, 45
    - mse, 46
    - nlar-methods, 47
    - NNET, 49
    - plot methods, 50
    - rank.select, 57
    - rank.test, 59
    - regime, 62
    - resVar, 63
    - selectHyperParms, 65
    - selectSETAR, 66
    - SETAR, 68
    - setar.sim, 70
    - setarTest, 72
    - sigmoid, 75
    - STAR, 76
    - toLatex.setar, 77
    - tsDyn-package, 3
    - TVAR, 78
    - TVAR.LRtest, 80
    - TVAR.sim, 82
    - TVECM, 85
    - TVECM.HStest, 87
    - TVECM.SeoTest, 90
    - TVECM.sim, 92
    - VAR.sim, 95
    - VARrep, 97
    - VECM, 98
    - VECM\_symbolic, 101
- AAR (aar), 4



- aar, 4
- accuracy, 5
- accuracy\_stat, 5
- addRegime, 6, 77
- AIC.nlar (nlar-methods), 47
- ar\_mean, 7
- arima.sim, 72
- as.data.frame, 38
- as.data.frame.llar (llar), 36
- as.data.frame.rank.select (rank.select), 57
- autopairs, 4, 8
- autotriples, 4, 9, 11
- autotriples.rgl, 4, 10
- availableModels, 4, 11, 48
  
- barry, 12
- BBCTest, 12, 32
- bds.test, 17
- BIC.nlar (nlar-methods), 47
- blrtest, 15
  
- ca.jo, 15, 59–61
- cajorls, 15
- charac\_root, 14
- coef.nlar (nlar-methods), 47
- coefA, 99, 100
- coefA (coefB), 14
- coefB, 14, 100
- coefPI, 100
- coefPI (coefB), 14
  
- d2sigmoid (sigmoid), 75
- delta, 4, 16
- delta.lin, 4, 18
- delta.lin.test, 17
- deviance.nlar (nlar-methods), 47
- dsigmoid (sigmoid), 75
  
- fevd, 19, 100
- fevd.nlVar, 19
- fitted (fitted.nlVar), 20
- fitted.nlar (nlar-methods), 47
- fitted.nlVar, 20
- format, 77
- format.pval, 59
  
- gam, 4
- getTh, 21
  
- GIRF, 22
  
- hnorm, 9, 10
  
- IIPUs, 25, 75
- irf, 35
- irf.ar (irf.linear), 27
- irf.linear, 27
- irf.nlVar, 24
- irf.nlVar (irf.linear), 27
- irf.setar (irf.linear), 27
- irf.TVAR (irf.linear), 27
- irf.TVECM (irf.linear), 27
- irf.VAR (irf.linear), 27
- irf.varest, 29, 30
- irf.VECM, 100
- irf.VECM (irf.linear), 27
- isLinear, 30
  
- KapShinTest, 31
  
- lags.select, 32
- legend, 51
- LINEAR, 33
- linear, 7, 23, 35, 71, 100
- linear (LINEAR), 33
- linear.boot (setar.sim), 70
- linear.sim (setar.sim), 70
- lineVar, 15, 20, 30, 34, 39, 55, 56, 58, 79, 84, 94, 96, 97, 99, 100
- llar, 4, 36
- logLik, 100
- logLik.nlVar, 39
- logLik.VAR (logLik.nlVar), 39
- logLik.VECM, 32, 58
- logLik.VECM (logLik.nlVar), 39
- LSTAR, 40
- lstar, 7, 51, 53
- lstar (LSTAR), 40
  
- m.unrate, 43
- MakeThSpec, 44, 66–68
- makeThSpec (MakeThSpec), 44
- MAPE, 45, 100
- MAPE.nlar (nlar-methods), 47
- mse, 46, 100
- mse.nlar (nlar-methods), 47
  
- nlar, 14, 34, 47, 65, 66, 77
- nlar-methods, 47, 51

- NNET, 49
- nnet, 49
- nnetTs (NNET), 49
- OlsTVAR, 82
- OlsTVAR (TVAR), 78
- optim, 41, 42, 49, 76
- par, 48, 50
- plogis, 41
- plot, 20, 30, 51
- plot methods, 50
- plot-methods (plot methods), 50
- plot.aar (aar), 4
- plot.GIRF\_df (GIRF), 22
- plot.llar (llar), 36
- plot.lstar, 42
- plot.lstar (plot methods), 50
- plot.nlar (nlar-methods), 47
- plot.setar, 70
- plot.setar (plot methods), 50
- plot\_ECT, 51
- polyroot, 14
- predict (predict.nlar), 52
- predict.nlar, 52, 56
- predict.TVAR, 54
- predict.VAR, 35
- predict.VAR (predict.TVAR), 54
- predict.VECM (predict.TVAR), 54
- predict\_rolling, 53, 55, 55, 100
- print.aar (aar), 4
- print.linear (LINEAR), 33
- print.llar (llar), 36
- print.rank.select (rank.select), 57
- print.rank.test (rank.test), 59
- print.summary.linear (LINEAR), 33
- printCoefmat, 48
- rank.select, 32, 33, 57, 61
- rank.test, 59, 59
- regime, 35, 62
- resample\_vec, 63, 70, 71, 73, 93
- residuals.nlar (nlar-methods), 47
- resVar, 63
- rgl, 10
- selectHyperParms, 65
- selectLSTAR, 68
- selectLSTAR (selectHyperParms), 65
- selectNNET, 68
- selectNNET (selectHyperParms), 65
- selectSETAR, 44, 45, 66
- selectSetar (selectSETAR), 66
- selectsetar (selectSETAR), 66
- set.seed, 63
- SETAR, 68, 72, 74
- setar, 7, 23, 51, 53, 71, 72, 77
- setar (SETAR), 68
- setar.boot (setar.sim), 70
- setar.sim, 70
- setarTest, 13, 32, 72, 82
- setartest (setarTest), 72
- setarTest\_IIPUs\_results, 75
- sigmoid, 75
- sm, 10
- sm.autoregression, 10
- sm.density, 9
- sm.regression, 9, 10
- sm.ts.pdf, 9
- STAR, 76
- star, 7
- star (STAR), 76
- summary.aar (aar), 4
- summary.linear (LINEAR), 33
- summary.nlar (nlar-methods), 47
- summary.rank.select (rank.select), 57
- summary.rank.test (rank.test), 59
- summary.setar (SETAR), 68
- terasvirta.test, 17
- toLatex, 35
- toLatex.nlar (nlar-methods), 47
- toLatex.setar, 77
- tsDyn (tsDyn-package), 3
- tsDyn-package, 3
- TVAR, 23, 35, 78, 83, 84, 87, 94, 97, 100
- TVAR.boot, 94, 97
- TVAR.boot (TVAR.sim), 82
- TVAR.LRtest, 74, 79, 80
- TVAR.sim, 79, 82, 93, 94, 97
- TVECM, 23, 35, 51, 84, 85, 89, 91, 93, 94, 97, 100, 102
- TVECM.boot, 84, 97
- TVECM.boot (TVECM.sim), 92
- TVECM.HStest, 73, 81, 87, 90, 102
- TVECM.HStest (TVECM.HStest), 87
- TVECM.SeoTest, 87, 89, 90
- TVECM.sim, 84, 87–89, 91, 92, 93, 97

ugarchroll, [56](#)  
UsUnemp, [95](#)

VAR.boot, [84](#), [94](#)  
VAR.boot (VAR.sim), [95](#)  
VAR.sim, [84](#), [94](#), [95](#)  
VARrep, [35](#), [54](#), [55](#), [97](#)  
VARselect, [33](#)  
VECM, [15](#), [20](#), [30](#), [35](#), [39](#), [51](#), [55](#), [56](#), [58–61](#), [87](#),  
[88](#), [93](#), [94](#), [97](#), [98](#)  
VECM.boot (TVECM.sim), [92](#)  
VECM.sim (TVECM.sim), [92](#)  
VECM\_symbolic, [101](#)

zeroyld, [89](#), [102](#)  
zeroyldMeta (zeroyld), [102](#)