Package: sstvars (via r-universe)

September 27, 2024

Type Package

Title Toolkit for Reduced Form and Structural Smooth Transition Vector Autoregressive Models

Version 1.0.1

Maintainer Savi Virolainen <savi.virolainen@helsinki.fi>

Description Maximum likelihood estimation of smooth transition vector autoregressive models with various types of transition weight functions, conditional distributions, and identification methods. Constrained estimation with various types of constraints is available. Residual based model diagnostics, forecasting, simulations, and calculation of impulse response functions, generalized impulse response functions, and generalized forecast error variance decompositions. See Heather Anderson, Farshid Vahid (1998) <doi:10.1016/S0304-4076(97)00076-6>, Helmut Lütkepohl, Aleksei Netšunajev (2017) <doi:10.1016/j.jedc.2017.09.001>, Markku Lanne, Savi Virolainen (2024) <doi:10.48550/arXiv.2403.14216>, Savi Virolainen (2024) <doi:10.48550/arXiv.2404.19707>.

Depends R (>= 4.0.0)

URL https://github.com/saviviro/sstvars

BugReports https://github.com/saviviro/sstvars/issues

SystemRequirements BLAS, LAPACK

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 7.2.3

LinkingTo Rcpp, RcppArmadillo

Imports Rcpp (>= 1.0.0), RcppArmadillo (>= 0.12.0.0.0), parallel (>= 4.0.0), pbapply (>= 1.7-0), stats (>= 4.0.0), graphics (>= 4.0.0)

Suggests knitr, rmarkdown

VignetteBuilder knitr

NeedsCompilation yes

Author Savi Virolainen [aut, cre]
(<https://orcid.org/0000-0002-5075-6821>)

Repository CRAN

Date/Publication 2024-05-29 06:50:02 UTC

Contents

`sstvars-package` *sstvars: toolkit for reduced form and structural smooth transition vector autoregressive models*

Description

`sstvars` is a package for reduced form and structural smooth transition vector autoregressive models. The package implements various transition weight functions, conditional distributions, identification methods, and parameter restrictions. The model parameters are estimated with the method of maximum likelihood by running multiple rounds of a two-phase estimation procedure in which a genetic algorithm is used to find starting values for a gradient based method. For evaluating the adequacy of the estimated models, `sstvars` utilizes residuals based diagnostics and provides functions for graphical diagnostics and for calculating formal diagnostic tests. `sstvars` also accommodates the estimation of linear impulse response functions, nonlinear generalized impulse response functions, and generalized forecast error variance decompositions. Further functionality includes hypothesis testing, plotting the profile log-likelihood functions about the estimate, simulation from STVAR processes, and forecasting, for example.

The vignette is a good place to start, and see also the readme file.

Author(s)

you <savi.virolainen@helsinki.fi>

See Also

Useful links:

- https://github.com/saviviro/sstvars
- Report bugs at https://github.com/saviviro/sstvars/issues

`acidata` *A monthly U.S. data covering the period from 1961I to 2022III (735 observations) and consisting four variables. First, The Actuaries Climate Index (ACI), which is a measure of the frequency of severe weather and the extend changes in sea levels. Second, the monthly GDP growth rate constructed by the Federal Reserve Bank of Chicago from a collapsed dynamic factor analysis of a panel of 500 monthly measures of real economic activity and quarterly real GDP growth. Third, the monthly growth rate of the consumer price index (CPI). Third, an interest rate variable, which is the effective federal funds rate that is replaced by the the Wu and Xia (2016) shadow rate during zero-lower-bound periods. The Wu and Xia (2016) shadow rate is not bounded by the zero lower bound and also quantifies unconventional monetary policy measures, while it closely follows the federal funds rate when the zero lower bound does not bind.*

Description

A monthly U.S. data covering the period from 1961I to 2022III (735 observations) and consisting four variables. First, The Actuaries Climate Index (ACI), which is a measure of the frequency of severe weather and the extend changes in sea levels. Second, the monthly GDP growth rate constructed by the Federal Reserve Bank of Chicago from a collapsed dynamic factor analysis of a panel of 500 monthly measures of real economic activity and quarterly real GDP growth. Third, the monthly growth rate of the consumer price index (CPI). Third, an interest rate variable, which is the effective federal funds rate that is replaced by the the Wu and Xia (2016) shadow rate during zero-lower-bound periods. The Wu and Xia (2016) shadow rate is not bounded by the zero lower bound and also quantifies unconventional monetary policy measures, while it closely follows the federal funds rate when the zero lower bound does not bind.

Usage

```
acidata
```

Format

A numeric matrix of class `'ts'` with 735 rows and 4 columns with one time series in each column:

First column (GDP): The cyclical component of the log of real GDP, https://fred.stlouisfed.org/series/GDPC1.

Second column (GDPDEF): The log-difference of GDP implicit price deflator, https://fred.stlouisfed.org/series/GDPDEF.

Third column (RATE): The Federal funds rate from 1954Q3 to 2008Q2 and after that the Wu and Xia (2016) shadow rate, https://fred.stlouisfed.org/series/FEDFUNDS, https://www.atlantafed.org/cqer/research/wu-xia-shadow-federal-funds-rate.

Source

The Federal Reserve Bank of St. Louis database and the Federal Reserve Bank of Atlanta's website

References

- American Academy of Actuaries, Canadian Institute of Actuaries, Casualty Actuarial Society, and Society of Actuaries, 2023. Actuaries Climate Index. https://actuariesclimateindex.org.
- Federal Reserve Bank of Chicago, 2023. Monthly GDP Growth Rate Data. https://www.chicagofed.org/publications/bbki/index.
- Wu J. and Xia F. 2016. Measuring the macroeconomic impact of monetary policy at the zero lower bound. *Journal of Money, Credit and Banking*, 48(2-3): 253-291.

alt_stvar *Construct a STVAR model based on results from an arbitrary estimation round of* fitSTVAR

Description

alt_stvar constructs a STVAR model based on results from an arbitrary estimation round of fitSTVAR

Usage

```
alt_stvar(stvar, which_largest = 1, which_round, calc_std_errors = FALSE)
```

Arguments

stvar	object of class "stvar"
which_largest	based on estimation round with which largest log-likelihood should the model be constructed? An integer value in 1,...,nrounds. For example, which_largest=2 would take the second largest log-likelihood and construct the model based on the corresponding estimates.
which_round	based on which estimation round should the model be constructed? An integer value in 1,...,nrounds. If specified, then which_largest is ignored.
calc_std_errors	should approximate standard errors be calculated?

Details

It's sometimes useful to examine other estimates than the one with the highest log-likelihood. This function is wrapper around STVAR that picks the correct estimates from an object returned by fitSTVAR.

Value

Returns an S3 object of class 'stvar' defining a smooth transition VAR model. The returned list contains the following components (some of which may be NULL depending on the use case):

data	The input time series data.
model	A list describing the model structure.
params	The parameters of the model.
std_errors	Approximate standard errors of the parameters, if calculated.
transition_weights	The transition weights of the model.
regime_cmeans	Conditional means of the regimes, if data is provided.
total_cmeans	Total conditional means of the model, if data is provided.
total_ccovs	Total conditional covariances of the model, if data is provided.

`uncond_moments`	A list of unconditional moments including regime autocovariances, variances, and means.
`residuals_raw`	Raw residuals, if data is provided.
`residuals_std`	Standardized residuals, if data is provided.
`structural_shocks`	Recovered structural shocks, if applicable.
`loglik`	Log-likelihood of the model, if data is provided.
`IC`	The values of the information criteria (AIC, HQIC, BIC) for the model, if data is provided.
`all_estimates`	The parameter estimates from all estimation rounds, if applicable.
`all_logliks`	The log-likelihood of the estimates from all estimation rounds, if applicable.
`which_converged`	Indicators of which estimation rounds converged, if applicable.
`which_round`	Indicators of which round of optimization each estimate belongs to, if applicable.

References

- Anderson H., Vahid F. 1998. Testing multiple equation systems for common nonlinear components. *Journal of Econometrics*, **84**:1, 1-36.
- Hubrich K., Teräsvirta. T. 2013. Thresholds and Smooth Transitions in Vector Autoregressive Models. *CREATES Research Paper 2013-18, Aarhus University.*
- Lanne M., Virolainen S. 2024. A Gaussian smooth transition vector autoregressive model: An application to the macroeconomic effects of severe weather shocks. Unpublished working paper, available as arXiv:2403.14216.
- Kheifets I.L., Saikkonen P.J. 2020. Stationarity and ergodicity of Vector STAR models. *Econometric Reviews*, **39**:4, 407-414.
- Lütkepohl H., Netšunajev A. 2017. Structural vector autoregressions with smooth transition in variances. *Journal of Economic Dynamics & Control*, **84**, 43-57.
- Tsay R. 1998. Testing and Modeling Multivariate Threshold Models. *Journal of the American Statistical Association*, **93**:443, 1188-1202.
- Virolainen S. 2024. Identification by non-Gaussianity in structural threshold and smooth transition vector autoregressive models. Unpublished working paper, available as arXiv:2404.19707.

See Also

STVAR

Examples

```
## These are long-running examples that take approximately 10 seconds to run.

# Estimate a Gaussian STVAR p=1, M=2 model with exponential weight function and
# the first lag of the second variable as the switching variables. Run only two
# estimation rounds:
```

```
fit12 <- fitSTVAR(gdpdef, p=1, M=2, weight_function="exponential", weightfun_pars=c(2, 1),
 nrounds=2, seeds=c(1, 7))
fit12$loglik # Log-likelihood of the estimated model

# Print the log-likelihood obtained from each estimation round:
fit12$all_logliks

# Construct the model based on the second largest log-likelihood found in the
# estimation procedure:
fit12_alt <- alt_stvar(fit12, which_largest=2, calc_std_errors=FALSE)
fit12_alt$loglik # Log-likelihood of the alternative solution

# Construct a model based on a specific estimation round, the second round:
fit12_alt2 <- alt_stvar(fit12, which_round=2, calc_std_errors=FALSE)
fit12_alt2$loglik # Log-likelihood of the alternative solution
```

---

bound_JSR *Calculate upper bound for the joint spectral radius of the "companion form AR matrices" of the regimes*

Description

bound_JSR calculates an bounds for the joint spectral radius of the "companion form AR matrices" matrices of the regimes to assess the validity of the stationarity condition.

Usage

```
bound_JSR(
  stvar,
  epsilon = 0.01,
  adaptive_eps = FALSE,
  ncores = 2,
  print_progress = TRUE
)
```

Arguments

stvar	object of class "stvar"
epsilon	a strictly positive real number that approximately defines the goal of length of the interval between the lower and upper bounds. A smaller epsilon value results in a narrower interval, thus providing better accuracy for the bounds, but at the cost of increased computational effort. Note that the bounds are always wider than epsilon and it is not obvious what epsilon should be chosen obtain bounds of specific tightness.
adaptive_eps	logical: if TRUE, starts with a large epsilon and then decreases it gradually whenever the progress of the algorithm requires, until the value given in the argument

	epsilon is reached. Usually speeds up the algorithm substantially but is an unconventional approach, and there is no guarantee that the algorithm converges appropriately towards bounds with the tightness given by the argument epsilon.
ncores	the number of cores to be used in parallel computing.
print_progress	logical: should the progress of the algorithm be printed?

Details

A sufficient condition for ergodic stationarity of the STVAR processes implemented in sstvars is that the joint spectral radius of the "companion form AR matrices" of the regimes is smaller than one (Kheifets and Saikkonen, 2020). This function calculates an upper (and lower) bound for the JSR and is implemented to assess the validity of this condition in practice. If the bound is smaller than one, the model is deemed ergodic stationary.

Implements the branch-and-bound method by Gripenberg (1996) in the conventional form (adaptive_eps=FALSE) and in a form incorporating "adaptive tightness" (adaptive_eps=FALSE). The latter approach is unconventional and does not guarantee appropriate convergence of the bounds close to the desired tightness given in the argument epsilon, but it usually substantially speeds up the algorithm. When print_progress==TRUE, the tightest bounds found so-far are printed in each iteration of the algorithm, so you can also just terminate the algorithm when the bounds are tight enough for your purposes. Consider also adjusting the argument epsilon, in particular when adaptive_eps=FALSE, as larger epsilon does not just make the bounds less tight but also speeds up the algorithm significantly. See Chang and Blondel (2013) for a discussion on variuous methods for bounding the JSR.

Value

Returns lower and upper bounds for the joint spectral radius of the "companion form AR matrices" of the regimes.

References

- C-T Chang and V.D. Blondel. 2013 . An experimental study of approximation algorithms for the joint spectral radius. *Numerical algorithms*, **64**, 181-202.
- Gripenberg, G. 1996. Computing the joint spectral radius. *Linear Algebra and its Applications*, 234, 43–60.
- I.L. Kheifets, P.J. Saikkonen. 2020. Stationarity and ergodicity of Vector STAR models. *Econometric Reviews*, **39**:4, 407-414.
- Virolainen S. 2024. Identification by non-Gaussianity in structural threshold and smooth transition vector autoregressive models. Unpublished working paper, available in ArXiv.

See Also

bound_jsr_G

Examples

```
## Below examples take approximately 5 seconds to run.

# Gaussian STVAR p=1, M=2 model with weighted relative stationary densities
```

```
# of the regimes as the transition weight function:
theta_122relg <- c(0.734054, 0.225598, 0.705744, 0.187897, 0.259626, -0.000863,
  -0.3124, 0.505251, 0.298483, 0.030096, -0.176925, 0.838898, 0.310863, 0.007512,
  0.018244, 0.949533, -0.016941, 0.121403, 0.573269)
mod122 <- STVAR(data=gdpdef, p=1, M=2, params=theta_122relg)

# Absolute values of the eigenvalues of the "companion form AR matrices":
summary(mod122)$abs_boldA_eigens
# It is a necessary (but not sufficient!) condition for ergodic stationary that
# the spectral radius of the "companion form AR matrices" are smaller than one
# for all of the regimes. A sufficient (but not necessary) condition for
# ergodic stationary is that the joint spectral radius of the companion form
# AR matrices" of the regimes is smaller than one. Therefore, we calculate
# bounds for the joint spectral radius.

## Bounds by Gripenberg's (1996) branch-and-bound method:
# Since the largest modulus of the companion form AR matrices is not very close
# to one, we likely won't need very thight bounds to verify the JSR is smaller
# than one. Thus, using a small epsilon would make the algorithm unnecessarily slow,
# so we use the (still quite small) epsilon=0.01:
bound_JSR(mod122, epsilon=0.01, adaptive_eps=FALSE)
# The upper bound is smaller than one, so the model is ergodic stationary.

# If we want tighter bounds, we can set smaller epsilon, e.g., epsilon=0.001:
bound_JSR(mod122, epsilon=0.001, adaptive_eps=FALSE)

# Using adaptive_eps=TRUE usually speeds up the algorithm when the model
# is large, but with the small model here, the speed-difference is small:
bound_JSR(mod122, epsilon=0.001, adaptive_eps=TRUE)
```

---

bound_jsr_G *Calculate upper bound for the joint spectral radius of a set of matrices*

Description

bound_jsr_G calculates lower and upper bounds for the joint spectral radious of a set of square matrices, typically the "bold A" matrices, using the algorithm by Gripenberg (1996).

Usage

```
bound_jsr_G(
  S,
  epsilon = 0.01,
  adaptive_eps = FALSE,
  ncores = 2,
  print_progress = TRUE
)
```

Arguments

S	the set of matrices the bounds should be calculated for in an array, in STVAR applications, all $((dp)x(dp))$ "bold A" (companion form) matrices in a 3D array, so that [, , m] gives the matrix the regime m.
epsilon	a strictly positive real number that approximately defines the goal of length of the interval between the lower and upper bounds. A smaller epsilon value results in a narrower interval, thus providing better accuracy for the bounds, but at the cost of increased computational effort. Note that the bounds are always wider than epsilon and it is not obvious what epsilon should be chosen obtain bounds of specific tightness.
adaptive_eps	logical: if TRUE, starts with a large epsilon and then decreases it gradually whenever the progress of the algorithm requires, until the value given in the argument epsilon is reached. Usually speeds up the algorithm substantially but is an unconventional approach, and there is no guarantee that the algorithm converges appropriately towards bounds with the tightness given by the argument epsilon.
ncores	the number of cores to be used in parallel computing.
print_progress	logical: should the progress of the algorithm be printed?

Details

The upper and lower bounds are calculated using the Gripenberg's (1996) branch-and-bound method, which is also discussed in Chang and Blondel (2013). This function can be generally used for approximating the JSR of a set of square matrices, but the main intention is STVAR applications (for models created with sstvars, the function bound_JSR should be preferred). Specifically, Kheifets and Saikkonen (2020) show that if the joint spectral radius of the companion form AR matrices of the regimes is smaller than one, the STVAR process is ergodic stationary. Virolainen (2024) shows the same result for his parametrization of of threshold and smooth transition vector autoregressive models. Therefore, if the upper bound is smaller than one, the process is stationary ergodic. However, as the condition is not necessary but sufficient and also because the bound might be too conservative, upper bound larger than one does not imply that the process is not ergodic stationary. You can try higher accuracy, and if the bound is still larger than one, the result does not tell whether the process is ergodic stationary or not.

Note that with high precision (small epsilon), the computational effort required are substantial and the estimation may take long, even though the function takes use of parallel computing. This is because with small epsilon the the number of candidate solutions in each iteration may grow exponentially and a large number of iterations may be required. For this reason, adaptive_eps=TRUE can be considered for large matrices, in which case the algorithm starts with a large epsilon, and then decreases it when new candidate solutions are not found, until the epsilon given by the argument epsilon is reached.

Value

Returns an upper bound for the joint spectral radius of the "companion form AR matrices" of the regimes.

References

- C-T Chang and V.D. Blondel. 2013 . An experimental study of approximation algorithms for the joint spectral radius. *Numerical algorithms*, **64**, 181-202.
- Gripenberg, G. 1996. Computing the joint spectral radius. *Linear Algebra and its Applications*, 234, 43–60.
- I.L. Kheifets, P.J. Saikkonen. 2020. Stationarity and ergodicity of Vector STAR models. *Econometric Reviews*, **39**:4, 407-414.
- Virolainen S. 2024. Identification by non-Gaussianity in structural threshold and smooth transition vector autoregressive models. Unpublished working paper, available in ArXiv.

See Also

bound_JSR

Examples

```
## Below examples take approximately 5 seconds to run.

# A set of two (5x5) square matrices:
set.seed(1); S1 <- array(rnorm(20*20*2), dim=c(5, 5, 2))

# Bound the joint spectral radius of the set of matrices S1, with the
# approximate tightness epsilon=0.01:
bound_jsr_G(S1, epsilon=0.01, adaptive_eps=FALSE)

# Obtain bounds faster with adaptive_eps=TRUE:
bound_jsr_G(S1, epsilon=0.01, adaptive_eps=TRUE)
# Note that the upper bound is not the same as with adaptive_eps=FALSE.

# A set of three (3x3) square matrices:
set.seed(2); S2 <- array(rnorm(3*3*3), dim=c(3, 3, 3))

# Bound the joint spectral radius of the set of matrices S2:
bound_jsr_G(S2, epsilon=0.01, adaptive_eps=FALSE)

# Larger epsilon terminates the iteration earlier and results in wider bounds:
bound_jsr_G(S2, epsilon=0.05, adaptive_eps=FALSE)

# A set of eight (2x2) square matrices:
set.seed(3); S3 <- array(rnorm(2*2*8), dim=c(2, 2, 8))

# Bound the joint spectral radius of the set of matrices S3:
bound_jsr_G(S3, epsilon=0.01, adaptive_eps=FALSE)
```

calc_gradient *Calculate gradient or Hessian matrix*

Description

calc_gradient or calc_hessian calculates the gradient or Hessian matrix of the given function at the given point using central difference numerical approximation. get_gradient or get_hessian calculates the gradient or Hessian matrix of the log-likelihood function at the parameter estimates of a class 'stvar' object. get_soc returns eigenvalues of the Hessian matrix, and get_foc is the same as get_gradient but named conveniently.

Usage

```
calc_gradient(x, fn, h = 6e-06, ...)

calc_hessian(x, fn, h = 6e-06, ...)

get_gradient(stvar)

get_hessian(stvar)

get_foc(stvar)

get_soc(stvar)
```

Arguments

x	a numeric vector specifying the point where the gradient or Hessian should be calculated.
fn	a function that takes in argument x as the **first** argument.
h	difference used to approximate the derivatives.
...	other arguments passed to fn
stvar	object of class "stvar"

Details

In particular, the functions get_foc and get_soc can be used to check whether the found estimates denote a (local) maximum point, a saddle point, or something else. Note that profile log-likelihood functions can be conveniently plotted with the function profile_logliks.

Value

Gradient functions return numerical approximation of the gradient and Hessian functions return numerical approximation of the Hessian. get_soc returns eigenvalues of the Hessian matrix.

Warning

No argument checks!

Examples

```
# Create a simple function:
foo <- function(x) x^2 + x

# Calculate the gradient at x=1 and x=-0.5:
calc_gradient(x=1, fn=foo)
calc_gradient(x=-0.5, fn=foo)

# Create a more complicated function
foo <- function(x, a, b) a*x[1]^2 - b*x[2]^2

# Calculate the gradient at x=c(1, 2) with parameter values a=0.3 and b=0.1:
calc_gradient(x=c(1, 2), fn=foo, a=0.3, b=0.1)

# Create a linear Gaussian VAR p=1 model:
theta_112 <- c(0.649526, 0.066507, 0.288526, 0.021767, -0.144024, 0.897103,
 0.601786, -0.002945, 0.067224)
mod112 <- STVAR(data=gdpdef, p=1, M=1, params=theta_112)

# Calculate the gradient of the log-likelihood function about the parameter values:
get_foc(mod112)

# Calculate the eigenvalues of the Hessian matrix of the log-likelihood function
# about the parameter values:
get_soc(mod112)
```

---

check_params *Check whether the parameter vector is in the parameter space and throw error if not*

Description

check_params checks whether the parameter vector is in the parameter space.

Usage

```
check_params(
  data,
  p,
  M,
  d,
  params,
 weight_function = c("relative_dens", "logistic", "mlogit", "exponential", "threshold",
    "exogenous"),
  weightfun_pars = NULL,
```

```
  cond_dist = c("Gaussian", "Student", "ind_Student"),
  parametrization = c("intercept", "mean"),
  identification = c("reduced_form", "recursive", "heteroskedasticity",
    "non-Gaussianity"),
  AR_constraints = NULL,
  mean_constraints = NULL,
  weight_constraints = NULL,
  B_constraints = NULL,
  transition_weights,
  stab_tol = 0.001,
  posdef_tol = 1e-08,
  distpar_tol = 1e-08,
  weightpar_tol = 1e-08
)
```

Arguments

`data`	a matrix or class `'ts'` object with `d>1` columns. Each column is taken to represent a univariate time series. Missing values are not supported.
`p`	a positive integer specifying the autoregressive order
`M`	a positive integer specifying the number of regimes
`d`	the number of time series in the system, i.e., the dimension
`params`	a real valued vector specifying the parameter values. Should have the form $\theta = (\phi_{1,0}, ..., \phi_{M,0}, \varphi_1, ..., \varphi_M, \sigma, \alpha, \nu)$, where (see exceptions below):

- $\phi_{m,0}$ = the $(d \times 1)$ intercept (or mean) vector of the mth regime.
- $\varphi_m = (vec(A_{m,1}), ..., vec(A_{m,p}))$ $(pd^2 \times 1)$.
- **if** `cond_dist="Gaussian"` **or** `"Student"`**:** $\sigma = (vech(\Omega_1), ..., vech(\Omega_M))$ $(Md(d+1)/2 \times 1)$.

  if `cond_dist="ind_Student"`**:** $\sigma = (vec(B_1), ..., vec(B_M)$ $(Md^2 \times 1)$.
- α = the $(a \times 1)$ vector containing the transition weight parameters (see below).
- **if** `cond_dist = "Gaussian"`**):** Omit ν from the parameter vector.

  if `cond_dist="Student"`**:** $\nu > 2$ is the single degrees of freedom parameter.

  if `cond_dist="ind_Student"`**:** $\nu = (\nu_1, ..., \nu_M)$ $(M \times 1)$, $nu_m > 2$.

For models with...

`weight_function="relative_dens"`**:** $\alpha = (\alpha_1, ..., \alpha_{M-1})$ $(M-1 \times 1)$, where α_m $(1 \times 1), m = 1, ..., M-1$ are the transition weight parameters.

`weight_function="logistic"`**:** $\alpha = (c, \gamma)$ (2×1), where $c \in \mathbb{R}$ is the location parameter and $\gamma > 0$ is the scale parameter.

`weight_function="mlogit"`**:** $\alpha = (\gamma_1, ..., \gamma_M)$ $((M-1)k \times 1)$, where γ_m $(k \times 1)$, $m = 1, ..., M-1$ contains the multinomial logit-regression coefficients of the mth regime. Specifically, for switching variables with indices in $I \subset \{1, ..., d\}$, and with $\tilde{p} \in \{1, ..., p\}$ lags included, γ_m contains the coefficients for the vector $z_{t-1} = (1, \tilde{z}_{\min\{I\}}, ..., \tilde{z}_{\max\{I\}})$, where

$\tilde{z}_i = (y_{it-1}, ..., y_{it-\tilde{p}})$, $i \in I$. So $k = 1 + |I|\tilde{p}$ where $|I|$ denotes the number of elements in I.

`weight_function="exponential"`: $\alpha = (c, \gamma)$ (2×1), where $c \in \mathbb{R}$ is the location parameter and $\gamma > 0$ is the scale parameter.

`weight_function="threshold"`: $\alpha = (r_1, ..., r_{M-1})$ $(M - 1 \times 1)$, where $r_1, ..., r_{M-1}$ are the threshold values.

`weight_function="exogenous"`: Omit α from the parameter vector.

AR_constraints: Replace $\varphi_1, ..., \varphi_M$ with ψ as described in the argument `AR_constraints`.

mean_constraints: Replace $\phi_{1,0}, ..., \phi_{M,0}$ with $(\mu_1, ..., \mu_g)$ where μ_i, $(d \times 1)$ is the mean parameter for group i and g is the number of groups.

weight_constraints: If linear constraints are imposed, replace α with ξ as described in the argument `weigh_constraints`. If weight functions parameters are imposed to be fixed values, simply drop α from the parameter vector.

`identification="heteroskedasticity"`: $\sigma = (vec(W), \lambda_2, ..., \lambda_M)$, where W $(d \times d)$ and λ_m $(d \times 1)$, $m = 2, ..., M$, satisfy $\Omega_1 = WW'$ and $\Omega_m = W\Lambda_m W'$, $\Lambda_m = diag(\lambda_{m1}, ..., \lambda_{md})$, $\lambda_{mi} > 0$, $m = 2, ..., M$, $i = 1, ..., d$.

B_constraints (only for structural models identified by heteroskedasticity): Replace $vec(W)$ with $v\tilde{e}c(W)$ that stacks the columns of the matrix W in to vector so that the elements that are constrained to zero are not included.

Above, $\phi_{m,0}$ is the intercept parameter, $A_{m,i}$ denotes the ith coefficient matrix of the mth regime, Ω_m denotes the positive definite error term covariance matrix of the mth regime, and B_m is the invertible $(d \times d)$ impact matrix of the mth regime. ν_m is the degrees of freedom parameter of the mth regime. If `parametrization=="mean"`, just replace each $\phi_{m,0}$ with regimewise mean μ_m. $vec()$ is vectorization operator that stacks columns of a given matrix into a vector. $vech()$ stacks columns of a given matrix from the principal diagonal downwards (including elements on the diagonal) into a vector. $Bvec()$ is a vectorization operator that stacks the columns of a given impact matrix B_m into a vector so that the elements that are constrained to zero by the argument `B_constraints` are excluded.

`weight_function`

What type of transition weights $\alpha_{m,t}$ should be used?

`"relative_dens"`: $\alpha_{m,t} = \frac{\alpha_m f_{m,dp}(y_{t-1},...,y_{t-p+1})}{\sum_{n=1}^{M} \alpha_n f_{n,dp}(y_{t-1},...,y_{t-p+1})}$, where $\alpha_m \in (0, 1)$ are weight parameters that satisfy $\sum_{m=1}^{M} \alpha_m = 1$ and $f_{m,dp}(\cdot)$ is the dp-dimensional stationary density of the mth regime corresponding to p consecutive observations. Available for Gaussian conditional distribution only.

`"logistic"`: $M = 2$, $\alpha_{1,t} = 1 - \alpha_{2,t}$, and $\alpha_{2,t} = [1 + \exp\{-\gamma(y_{it-j} - c)\}]^{-1}$, where y_{it-j} is the lag j observation of the ith variable, c is a location parameter, and $\gamma > 0$ is a scale parameter.

`"mlogit"`: $\alpha_{m,t} = \frac{\exp\{\gamma_m' z_{t-1}\}}{\sum_{n=1}^{M} \exp\{\gamma_n' z_{t-1}\}}$, where γ_m are coefficient vectors, $\gamma_M = 0$, and z_{t-1} $(k \times 1)$ is the vector containing a constant and the (lagged) switching variables.

`"exponential"`: $M = 2$, $\alpha_{1,t} = 1 - \alpha_{2,t}$, and $\alpha_{2,t} = 1 - \exp\{-\gamma(y_{it-j} - c)\}$, where y_{it-j} is the lag j observation of the ith variable, c is a location parameter, and $\gamma > 0$ is a scale parameter.

`"threshold"`: $\alpha_{m,t} = 1$ if $r_{m-1} < y_{it-j} \leq r_m$ and 0 otherwise, where $-\infty \equiv r_0 < r_1 < \cdots < r_{M-1} < r_M \equiv \infty$ are thresholds y_{it-j} is the lag j observation of the ith variable.

`"exogenous"`: Exogenous nonrandom transition weights, specify the weight series in `weightfun_pars`.

See the vignette for more details about the weight functions.

`weightfun_pars` **If** `weight_function == "relative_dens"`: Not used.

If `weight_function %in% c("logistic", "exponential", "threshold")`: a numeric vector with the switching variable $i \in \{1, ..., d\}$ in the first and the lag $j \in \{1, ..., p\}$ in the second element.

If `weight_function == "mlogit"`: a list of two elements:

The first element `$vars`: a numeric vector containing the variables that should used as switching variables in the weight function in an increasing order, i.e., a vector with unique elements in $\{1, ..., d\}$.

The second element `$lags`: an integer in $\{1, ..., p\}$ specifying the number of lags to be used in the weight function.

If `weight_function == "exogenous"`: a size (`nrow(data) - p` x `M`) matrix containing the exogenous transition weights as `[t, m]` for time t and regime m. Each row needs to sum to one and only weakly positive values are allowed.

`cond_dist` specifies the conditional distribution of the model as `"Gaussian"`, `"Student"`, or `"ind_Student"`, where the latest is the Student's t distribution with independent components.

`parametrization`

`"intercept"` or `"mean"` determining whether the model is parametrized with intercept parameters $\phi_{m,0}$ or regime means μ_m, m=1,...,M.

`identification` is it reduced form model or an identified structural model; if the latter, how is it identified (see the vignette or the references for details)?

`"reduced_form"`: Reduced form model.

`"recursive"`: The usual lower-triangular recursive identification of the shocks via their impact responses.

`"heteroskedasticity"`: Identification by conditional heteroskedasticity, which imposes constant relative impact responses for each shock.

`"non-Gaussianity"`: Identification by non-Gaussianity; requires mutually independent non-Gaussian shocks, thus, currently available only with the conditional distribution `"ind_Student"`.

`AR_constraints` a size (Mpd^2xq) constraint matrix C specifying linear constraints to the autoregressive parameters. The constraints are of the form $(\varphi_1, ..., \varphi_M) = C\psi$, where $\varphi_m = (vec(A_{m,1}), ..., vec(A_{m,p}))$ (pd^2x1), $m = 1, ..., M$, contains the coefficient matrices and ψ $(qx1)$ contains the related parameters. For example, to restrict the AR-parameters to be the identical across the regimes, set $C =$ `[I:...:I]'` (Mpd^2xpd^2) where `I = diag(p*d^2)`.

`mean_constraints`
: Restrict the mean parameters of some regimes to be identical? Provide a list of numeric vectors such that each numeric vector contains the regimes that should share the common mean parameters. For instance, if `M=3`, the argument `list(1, 2:3)` restricts the mean parameters of the second and third regime to be identical but the first regime has freely estimated (unconditional) mean. Ignore or set to `NULL` if mean parameters should not be restricted to be the same among any regimes. This constraint is available only for mean parametrized models; that is, when `parametrization="mean"`.

`weight_constraints`
: a list of two elements, R in the first element and r in the second element, specifying linear constraints on the transition weight parameters α. The constraints are of the form $\alpha = R\xi + r$, where R is a known $(a \times l)$ constraint matrix of full column rank (a is the dimension of α), r is a known $(a \times 1)$ constant, and ξ is an unknown $(l \times 1)$ parameter. **Alternatively**, set $R = 0$ in order to constrain the the weight parameter to the constant r (in this case, α is dropped from the constrained parameter vector).

`B_constraints`
: a $(d \times d)$ matrix with its entries imposing constraints on the impact matrix B_t: `NA` indicating that the element is unconstrained, a positive value indicating strict positive sign constraint, a negative value indicating strict negative sign constraint, and zero indicating that the element is constrained to zero. Currently only available for models with `identification="heteroskedasticity"` or `"non-Gaussianity"` due to the (in)availability of appropriate parametrizations that allow such constraints to be imposed.

`transition_weights`
: (optional; only for models with `cond_dist="ind_Student"` or `identification="non-Gaussianity"`) A $T \times M$ matrix containing the transition weights. If `cond_dist="ind_Student"` checks that the impact matrix $\sum_{m=1}^{M} \alpha_{m,t}^{1/2} B_m$ is invertible for all $t = 1, ..., T$.

`stab_tol`
: numerical tolerance for stability of condition of the regimes: if the "bold A" matrix of any regime has eigenvalues larger that `1 - stat_tol` the parameter is considered to be outside the parameter space. Note that if tolerance is too small, numerical evaluation of the log-likelihood might fail and cause error.

`posdef_tol`
: numerical tolerance for positive definiteness of the error term covariance matrices: if the error term covariance matrix of any regime has eigenvalues smaller than this, the parameter is considered to be outside the parameter space. Note that if the tolerance is too small, numerical evaluation of the log-likelihood might fail and cause error.

`distpar_tol`
: the parameter vector is considered to be outside the parameter space if the degrees of freedom parameters is not larger than `2 + distpar_tol` (applies only if `cond_dist="Student"`).

`weightpar_tol`
: numerical tolerance for weight parameters being in the parameter space. Values closer to to the border of the parameter space than this are considered to be "outside" the parameter space.

Value

Throws an informative error if there is something wrong with the parameter vector.

References

- Kheifets I.L., Saikkonen P.J. 2020. Stationarity and ergodicity of Vector STAR models. *Econometric Reviews*, **39**:4, 407-414.
- Lütkepohl H. 2005. New Introduction to Multiple Time Series Analysis, *Springer*.
- Lanne M., Virolainen S. 2024. A Gaussian smooth transition vector autoregressive model: An application to the macroeconomic effects of severe weather shocks. Unpublished working paper, available as arXiv:2403.14216.
- Virolainen S. 2024. Identification by non-Gaussianity in structural threshold and smooth transition vector autoregressive models. Unpublished working paper, available as arXiv:2404.19707.

@keywords internal

Examples

```
# There examples will cause an informative error
params112_notpd <- c(6.5e-01, 7.0e-01, 2.9e-01, 2.0e-02, -1.4e-01,
 9.0e-01, 6.0e-01, -1.0e-02, 1.0e-07)
try(check_params(p=1, M=1, d=2, params=params112_notpd))

params112_notstat <- c(6.5e-01, 7.0e-01, 10.9e-01, 2.0e-02, -1.4e-01,
 9.0e-01, 6.0e-01, -1.0e-02, 1.0e-07)
try(check_params(p=1, M=1, d=2, params=params112_notstat))

params112_wronglength <- c(6.5e-01, 7.0e-01, 2.9e-01, 2.0e-02, -1.4e-01,
 9.0e-01, 6.0e-01, -1.0e-02)
try(check_params(p=1, M=1, d=2, params=params112_wronglength))
```

---

diagnostic_plot *Residual diagnostic plot for a STVAR model*

Description

diagnostic_plot plots a multivariate residual diagnostic plot for either autocorrelation, conditional heteroskedasticity, or distribution, or simply draws the residual time series.

Usage

```
diagnostic_plot(
  stvar,
  type = c("all", "series", "ac", "ch", "dist"),
  resid_type = c("standardized", "raw"),
  maxlag = 12
)
```

Arguments

stvar	object of class "stvar"
type	which type of diagnostic plot should be plotted? • "all" all below sequentially. • "series" the residual time series. • "ac" the residual autocorrelation and cross-correlation functions. • "ch" the squared residual autocorrelation and cross-correlation functions. • "dist" the residual histogram with theoretical density (dashed line) and QQ-plots.
resid_type	should standardized or raw residuals be used?
maxlag	the maximum lag considered in types "ac" and "ch".

Details

Auto- and cross-correlations (types "ac" and "ch") are calculated with the function acf from the package stats and the plot method for class 'acf' objects is employed. If cond_dist == "Student", the estimate of the degrees of freedom parameter is used in theoretical densities and quantiles.

Value

No return value, called for its side effect of plotting the diagnostic plot.

References

- Anderson H., Vahid F. 1998. Testing multiple equation systems for common nonlinear components. *Journal of Econometrics*, **84**:1, 1-36.
- Kheifets I.L., Saikkonen P.J. 2020. Stationarity and ergodicity of Vector STAR models. *Econometric Reviews*, **39**:4, 407-414.
- Lanne M., Virolainen S. 2024. A Gaussian smooth transition vector autoregressive model: An application to the macroeconomic effects of severe weather shocks. Unpublished working paper, available as arXiv:2403.14216.
- Lütkepohl H. 2005. New Introduction to Multiple Time Series Analysis, *Springer*.
- McElroy T. 2017. Computation of vector ARMA autocovariances. *Statistics and Probability Letters*, **124**, 92-96.
- Kilian L., Lütkepohl H. 20017. Structural Vector Autoregressive Analysis. 1st edition. *Cambridge University Press*, Cambridge.
- Tsay R. 1998. Testing and Modeling Multivariate Threshold Models. *Journal of the American Statistical Association*, **93**:443, 1188-1202.
- Virolainen S. 2024. Identification by non-Gaussianity in structural threshold and smooth transition vector autoregressive models. Unpublished working paper, available as arXiv:2404.19707.

See Also

Portmanteau_test, profile_logliks, fitSTVAR, STVAR, LR_test, Wald_test, Rao_test

Examples

```
## Gaussian STVAR p=1, M=2 model, with weighted relative stationary densities
# of the regimes as the transition weight function:
theta_122relg <- c(0.734054, 0.225598, 0.705744, 0.187897, 0.259626, -0.000863,
  -0.3124, 0.505251, 0.298483, 0.030096, -0.176925, 0.838898, 0.310863, 0.007512,
  0.018244, 0.949533, -0.016941, 0.121403, 0.573269)
mod122 <- STVAR(data=gdpdef, p=1, M=2, params=theta_122relg)

# Autocorelation function of raw residuals for checking remaining autocorrelation:
diagnostic_plot(mod122, type="ac", resid_type="raw")

# Autocorelation function of squared standardized residuals for checking remaining
# conditional heteroskedasticity:
diagnostic_plot(mod122, type="ch", resid_type="standardized")

# Below, ACF of squared raw residuals, which is not very informative for evaluating
# adequacy to capture conditional heteroskedasticity, since it doesn't take into account
# the time-varying conditional covariance matrix of the model:
diagnostic_plot(mod122, type="ch", resid_type="raw")

# Similarly, below the time series of raw residuals first, and then the
# time series of standardized residuals. The latter is more informative
# for evaluating adequacy:
diagnostic_plot(mod122, type="series", resid_type="raw")
diagnostic_plot(mod122, type="series", resid_type="standardized")

# Also similarly, histogram and Q-Q plots are more informative for standardized
# residuals when evaluating model adequacy:
diagnostic_plot(mod122, type="dist", resid_type="raw") # Bad fit for GDPDEF
diagnostic_plot(mod122, type="dist", resid_type="standardized") # Good fit for GDPDEF

## Linear Gaussian VAR p=1 model:
theta_112 <- c(0.649526, 0.066507, 0.288526, 0.021767, -0.144024, 0.897103,
  0.601786, -0.002945, 0.067224)
mod112 <- STVAR(data=gdpdef, p=1, M=1, params=theta_112)
diagnostic_plot(mod112, resid_type="standardized") # All plots for std. resids
diagnostic_plot(mod112, resid_type="raw") # All plots for raw residuals
```

---

diag_Omegas *Simultaneously diagonalize two covariance matrices*

Description

diag_Omegas Simultaneously diagonalizes two covariance matrices using eigenvalue decomposition.

Usage

```
diag_Omegas(Omega1, Omega2)
```

Arguments

Omega1	a positive definite (dxd) covariance matrix $(d > 1)$
Omega2	another positive definite (dxd) covariance matrix

Details

See the return value and Muirhead (1982), Theorem A9.9 for details.

Value

Returns a length $d^2 + d$ vector where the first d^2 elements are $vec(W)$ with the columns of W being (specific) eigenvectors of the matrix $\Omega_2\Omega_1^{-1}$ and the rest d elements are the corresponding eigenvalues "lambdas". The result satisfies $WW' = Omega1$ and $Wdiag(lambdas)W' = Omega2$.

If Omega2 is not supplied, returns a vectorized symmetric (and pos. def.) square root matrix of Omega1.

Warning

No argument checks! Does not work with dimension $d = 1$!

References

- Muirhead R.J. 1982. Aspects of Multivariate Statistical Theory, *Wiley*.

Examples

```
# Create two (2x2) coviance matrices using the parameters W and lambdas:
d <- 2 # The dimension
W0 <- matrix(1:(d^2), nrow=2) # W
lambdas0 <- 1:d # The eigenvalues
(Omg1 <- W0%*%t(W0)) # The first covariance matrix
(Omg2 <- W0%*%diag(lambdas0)%*%t(W0)) # The second covariance matrix

# Then simultaneously diagonalize the covariance matrices:
res <- diag_Omegas(Omg1, Omg2)

# Recover W:
W <- matrix(res[1:(d^2)], nrow=d, byrow=FALSE)
tcrossprod(W) # == Omg1, the first covariance matrix

# Recover lambdas:
lambdas <- res[(d^2 + 1):(d^2 + d)]
W%*%diag(lambdas)%*%t(W) # == Omg2, the second covariance matrix
```

`fitSSTVAR` *Maximum likelihood estimation of a structural STVAR model based on preliminary estimates from a reduced form model.*

Description

`fitSSTVAR` uses a robust method and a variable metric algorithm to estimate a structural STVAR model based on preliminary estimates from a reduced form model.

Usage

```
fitSSTVAR(
  stvar,
  identification = c("recursive", "heteroskedasticity", "non-Gaussianity"),
  B_constraints = NULL,
  maxit = 1000,
  maxit_robust = 1000,
  robust_method = c("Nelder-Mead", "SANN", "none"),
  print_res = TRUE,
  calc_std_errors = TRUE
)
```

Arguments

`stvar`	a an object of class `'stvar'`, created by, e.g., `fitSTVAR`, specifying a reduced form or a structural model
`identification`	Which identification should the structural model use? (see the vignette or the references for details) `"recursive"`**:** The usual lower-triangular recursive identification of the shocks via their impact responses. `"heteroskedasticity"`**:** Identification by conditional heteroskedasticity, which imposes constant relative impact responses for each shock.
`B_constraints`	Employ further constraints on the impact matrix? A $(d \times d)$ matrix with its entries imposing constraints on the impact matrix B_t: `NA` indicating that the element is unconstrained, a positive value indicating strict positive sign constraint, a negative value indicating strict negative sign constraint, and zero indicating that the element is constrained to zero. Currently only available for models with `identification="heteroskedasticity"` due to the (in)availability of appropriate parametrizations that allow such constraints to be imposed.
`maxit`	the maximum number of iterations in the variable metric algorithm.
`maxit_robust`	the maximum number of iterations on the first phase robust estimation, if employed.
`robust_method`	Should some robust estimation method be used in the estimation before switching to the gradient based variable metric algorithm? See details.
`print_res`	should summaries of estimation results be printed?

`calc_std_errors`
: should approximate standard errors be calculated?

Details

When the structural model does not impose overidentifying constraints, it is directly obtained from the reduced form model, and estimation is not required. When overidentifying constraints are imposed, the model is estimated via ..

Structural models can be provided in the argument `stvar` if overidentifying constraints should be imposed.

Using the robust estimation method before switching to the variable metric can be useful if the initial estimates are not very close to the ML estimate of the structural model, as the variable metric algorithm (usually) converges to a nearby local maximum or saddle point. However, if the initial estimates are far from the ML estimate, the resulting solution is likely local only due to the complexity of the model. Note that Nelder-Mead algorithm is much faster than SANN but can get stuck at a local solution. This is particularly the case when the imposed overidentifying restrictions are such that the unrestricted estimate is not close to satisfying them. Nevertheless, in most practical cases, the model is just identified and estimation is not required, and often reasonable overidentifying constraints are close to the unrestricted estimate.

Employs the estimation function `optim` from the package `stats` that implements the optimization algorithms. See `?optim` for the documentation on the

Value

Returns an S3 object of class `'stvar'` defining a smooth transition VAR model. The returned list contains the following components (some of which may be `NULL` depending on the use case):

`data`
: The input time series data.

`model`
: A list describing the model structure.

`params`
: The parameters of the model.

`std_errors`
: Approximate standard errors of the parameters, if calculated.

`transition_weights`
: The transition weights of the model.

`regime_cmeans`
: Conditional means of the regimes, if data is provided.

`total_cmeans`
: Total conditional means of the model, if data is provided.

`total_ccovs`
: Total conditional covariances of the model, if data is provided.

`uncond_moments`
: A list of unconditional moments including regime autocovariances, variances, and means.

`residuals_raw`
: Raw residuals, if data is provided.

`residuals_std`
: Standardized residuals, if data is provided.

`structural_shocks`
: Recovered structural shocks, if applicable.

`loglik`
: Log-likelihood of the model, if data is provided.

`IC`
: The values of the information criteria (AIC, HQIC, BIC) for the model, if data is provided.

`all_estimates`	The parameter estimates from all estimation rounds, if applicable.
`all_logliks`	The log-likelihood of the estimates from all estimation rounds, if applicable.
`which_converged`	Indicators of which estimation rounds converged, if applicable.
`which_round`	Indicators of which round of optimization each estimate belongs to, if applicable.

References

- Kilian L., Lütkepohl H. 20017. Structural Vector Autoregressive Analysis. 1st edition. *Cambridge University Press*, Cambridge.
- Lütkepohl H., Netšunajev A. 2017. Structural vector autoregressions with smooth transition in variances. *Journal of Economic Dynamics & Control*, **84**, 43-57.

See Also

`fitSTVAR`, `STVAR`, `optim`

Examples

```
## These are long running examples that take approximately 1 minute to run.

## Estimate first a reduced form Gaussian STVAR p=3, M=2 model with the weighted relative
# stationary densities of the regimes as the transition weight function, and the means and
# AR matrices constrained to be identical across the regimes:
fit32cm <- fitSTVAR(gdpdef, p=3, M=2, AR_constraints=rbind(diag(3*2^2), diag(3*2^2)),
  weight_function="relative_dens", mean_constraints=list(1:2), parametrization="mean",
  nrounds=1, seeds=1, ncores=1)

# Then, we estimate/create various structural models based on the reduced form model.
# Create a structural model with the shocks identified recursively:
fit32cms_rec <- fitSSTVAR(fit32cm, identification="recursive")

# Create a structural model with the shocks identified by conditional heteroskedasticity:
fit32cms_hetsked <- fitSSTVAR(fit32cm, identification="heteroskedasticity")
fit32cms_hetsked # Print the estimates

# Estimate a structural model with the shocks identified by conditional heteroskedasticity
# and overidentifying constraints imposed on the impact matrix: positive diagonal element
# and zero upper right element:
fit32cms_hs2 <- fitSSTVAR(fit32cm, identification="heteroskedasticity",
 B_constraints=matrix(c(1, NA, 0, 1), nrow=2))

# Estimate a structural model with the shocks identified by conditional heteroskedasticity
# and overidentifying constraints imposed on the impact matrix: positive diagonal element
# and zero off-diagonal elements:
fit32cms_hs3 <- fitSSTVAR(fit32cms_hs2, identification="heteroskedasticity",
 B_constraints=matrix(c(1, 0, 0, 1), nrow=2))

# Estimate first a reduced form three-regime Student's t Threshold VAR p=3 model with
# the first lag of the second variable as the switching variable, the means and AR
```

```
# matrices constrained to be identical across the regimes:
fit32cml <- fitSTVAR(gdpdef, p=3, M=3, cond_dist="Student",
 AR_constraints=rbind(diag(3*2^2), diag(3*2^2), diag(3*2^2)),
 weight_function="threshold", weightfun_pars=c(2, 1), mean_constraints=list(1:3),
 parametrization="mean", nrounds=1, seeds=1, ncores=1)

# Then, we estimate/create various structural models based on the reduced form model.
# Create a structural model with the shocks identified recusively:
fit32cmls_rec <- fitSSTVAR(fit32cml, identification="recursive")

# Estimate a structural model with the shocks identified by conditional heteroskedasticity,
# the model is overidentifying, as it has more than two regimes:
fit32cmls_hs <- fitSSTVAR(fit32cml, identification="heteroskedasticity")
fit32cmls_hs # Print the estimates

# Estimate a structural model with the shocks identified by conditional heteroskedasticity
# and overidentifying constraints imposed on the impact matrix: zero lower left element,
# estimation without employing a robust estimation method:
fit32cmls_hs2 <- fitSSTVAR(fit32cmls_hs, identification="heteroskedasticity",
 B_constraints=matrix(c(NA, 0, NA, NA), nrow=2),
 robust_method="none")

# Relax the zero constraint on the impact matrix and re-estimate the model:
fit32cmls_hs3 <- fitSSTVAR(fit32cmls_hs2, identification="heteroskedasticity",
 B_constraints=matrix(c(NA, NA, NA, NA), nrow=2),
 robust_method="none")
```

`fitSTVAR` *Two-phase maximum likelihood estimation of a reduced form smooth transition VAR model*

Description

`fitSTVAR` estimates a reduced form smooth transition VAR model in two phases: in the first phase, it uses a genetic algorithm to find starting values for a gradient based variable metric algorithm, which it then uses to finalize the estimation in the second phase. Parallel computing is utilized to perform multiple rounds of estimations in parallel.

Usage

```
fitSTVAR(
  data,
  p,
  M,
 weight_function = c("relative_dens", "logistic", "mlogit", "exponential", "threshold",
    "exogenous"),
  weightfun_pars = NULL,
  cond_dist = c("Gaussian", "Student", "ind_Student"),
```

```
  parametrization = c("intercept", "mean"),
  AR_constraints = NULL,
  mean_constraints = NULL,
  weight_constraints = NULL,
  nrounds = (M + 1)^5,
  ncores = 2,
  maxit = 1000,
  seeds = NULL,
  print_res = TRUE,
  use_parallel = TRUE,
  filter_estimates = TRUE,
  ...
)
```

Arguments

`data` a matrix or class `'ts'` object with `d>1` columns. Each column is taken to represent a univariate time series. Missing values are not supported.

`p` a positive integer specifying the autoregressive order

`M` a positive integer specifying the number of regimes

`weight_function`

What type of transition weights $\alpha_{m,t}$ should be used?

`"relative_dens"`: $\alpha_{m,t} = \frac{\alpha_m f_{m,dp}(y_{t-1},...,y_{t-p+1})}{\sum_{n=1}^{M} \alpha_n f_{n,dp}(y_{t-1},...,y_{t-p+1})}$, where $\alpha_m \in (0,1)$ are weight parameters that satisfy $\sum_{m=1}^{M} \alpha_m = 1$ and $f_{m,dp}(\cdot)$ is the dp-dimensional stationary density of the mth regime corresponding to p consecutive observations. Available for Gaussian conditional distribution only.

`"logistic"`: $M = 2$, $\alpha_{1,t} = 1 - \alpha_{2,t}$, and $\alpha_{2,t} = [1 + \exp\{-\gamma(y_{it-j} - c)\}]^{-1}$, where y_{it-j} is the lag j observation of the ith variable, c is a location parameter, and $\gamma > 0$ is a scale parameter.

`"mlogit"`: $\alpha_{m,t} = \frac{\exp\{\gamma_m' z_{t-1}\}}{\sum_{n=1}^{M} \exp\{\gamma_n' z_{t-1}\}}$, where γ_m are coefficient vectors, $\gamma_M = 0$, and z_{t-1} $(k \times 1)$ is the vector containing a constant and the (lagged) switching variables.

`"exponential"`: $M = 2$, $\alpha_{1,t} = 1 - \alpha_{2,t}$, and $\alpha_{2,t} = 1 - \exp\{-\gamma(y_{it-j} - c)\}$, where y_{it-j} is the lag j observation of the ith variable, c is a location parameter, and $\gamma > 0$ is a scale parameter.

`"threshold"`: $\alpha_{m,t} = 1$ if $r_{m-1} < y_{it-j} \leq r_m$ and 0 otherwise, where $-\infty \equiv r_0 < r_1 < \cdots < r_{M-1} < r_M \equiv \infty$ are thresholds y_{it-j} is the lag j observation of the ith variable.

`"exogenous"`: Exogenous nonrandom transition weights, specify the weight series in `weightfun_pars`.

See the vignette for more details about the weight functions.

`weightfun_pars` **If** `weight_function == "relative_dens"`: Not used.

If `weight_function %in% c("logistic", "exponential", "threshold")`: a numeric vector with the switching variable $i \in \{1, ..., d\}$ in the first and the lag $j \in \{1, ..., p\}$ in the second element.

If `weight_function == "mlogit"`**:** a list of two elements:

The first element `$vars`**:** a numeric vector containing the variables that should used as switching variables in the weight function in an increasing order, i.e., a vector with unique elements in $\{1, ..., d\}$.

The second element `$lags`**:** an integer in $\{1, ..., p\}$ specifying the number of lags to be used in the weight function.

If `weight_function == "exogenous"`**:** a size (`nrow(data) - p` x `M`) matrix containing the exogenous transition weights as `[t, m]` for time t and regime m. Each row needs to sum to one and only weakly positive values are allowed.

`cond_dist` specifies the conditional distribution of the model as `"Gaussian"`, `"Student"`, or `"ind_Student"`, where the latest is the Student's t distribution with independent components.

`parametrization`

`"intercept"` or `"mean"` determining whether the model is parametrized with intercept parameters $\phi_{m,0}$ or regime means μ_m, m=1,...,M.

`AR_constraints` a size (Mpd^2xq) constraint matrix C specifying linear constraints to the autoregressive parameters. The constraints are of the form $(\varphi_1, ..., \varphi_M) = C\psi$, where $\varphi_m = (vec(A_{m,1}), ..., vec(A_{m,p}))$ (pd^2x1), $m = 1, ..., M$, contains the coefficient matrices and ψ $(qx1)$ contains the related parameters. For example, to restrict the AR-parameters to be the identical across the regimes, set $C =$ `[I:...:I]'` (Mpd^2xpd^2) where `I = diag(p*d^2)`.

`mean_constraints`

Restrict the mean parameters of some regimes to be identical? Provide a list of numeric vectors such that each numeric vector contains the regimes that should share the common mean parameters. For instance, if `M=3`, the argument `list(1, 2:3)` restricts the mean parameters of the second and third regime to be identical but the first regime has freely estimated (unconditional) mean. Ignore or set to `NULL` if mean parameters should not be restricted to be the same among any regimes. This constraint is available only for mean parametrized models; that is, when `parametrization="mean"`.

`weight_constraints`

a list of two elements, R in the first element and r in the second element, specifying linear constraints on the transition weight parameters α. The constraints are of the form $\alpha = R\xi + r$, where R is a known $(a \times l)$ constraint matrix of full column rank (a is the dimension of α), r is a known $(a \times 1)$ constant, and ξ is an unknown $(l \times 1)$ parameter. **Alternatively**, set $R = 0$ in order to constrain the the weight parameter to the constant r (in this case, α is dropped from the constrained parameter vector).

`nrounds` the number of estimation rounds that should be performed.

`ncores` the number CPU cores to be used in parallel computing.

`maxit` the maximum number of iterations in the variable metric algorithm.

`seeds` a length `nrounds` vector containing the random number generator seed for each call to the genetic algorithm, or `NULL` for not initializing the seed.

`print_res` should summaries of estimation results be printed?

`use_parallel` employ parallel computing? If `use_parallel=FALSE && print_res=FALSE`, nothing is printed during the estimation process.

`filter_estimates`
: should the likely inappropriate estimates be filtered? See details.

`...`
: additional settings passed to the function `GAfit` employing the genetic algorithm.

Details

If you wish to estimate a structural model, estimate first the reduced form model and then use the use the function `fitSSTVAR` to create (and estimate if necessary) the structural model based on the estimated reduced form model.

Because of complexity and high multimodality of the log-likelihood function, it is **not certain** that the estimation algorithm will end up in the global maximum point. It is expected that many of the estimation rounds will end up in some local maximum or a saddle point instead. Therefore, a (sometimes very large) number of estimation rounds is required for reliable results. Due to identification problems and high complexity of the surface of the log-likelihood function, the estimation may fail especially in the cases where the number of regimes is chosen too large.

The estimation process is computationally heavy and it might take considerably long time for large models to estimate. Note that estimation of model with `cond_dist == "ind_Student"` is computationally substantially more demanding than standard Gaussian and Student's t models due to the different structure of the log-likelihood function (parametrized directly via impact matrices rather than covariance matrices of the regimes).

If the iteration limit `maxit` in the variable metric algorithm is reached, one can continue the estimation by iterating more with the function `iterate_more`. Alternatively, one may use the found estimates as starting values for the genetic algorithm and employ another round of estimation (see `??GAfit` how to set up an initial population with the dot parameters).

If the estimation algorithm performs poorly, it usually helps to scale the individual series so that they vary roughly in the same scale. This makes it is easier to draw reasonable AR coefficients and (with some weight functions) weight parameter values in the genetic algorithm. Even if the estimation algorithm somewhat works, it should be preferred to scale the data so that most of the AR coefficients will not be very large, as the estimation algorithm works better with relatively small AR coefficients. If needed, another package can be used to fit linear VARs to the series to see which scaling of the series results in relatively small AR coefficients. You should avoid very small (or very high) variance in the data as well so that the eigenvalues of the covariance matrices are in a reasonable range.

weight_constraints: If you are using weight constraints other than restricting some of the weight parameters to known constants, make sure the constraints are sensible. Otherwise, the estimation may fail due to the estimation algorithm not being able to generate reasonable random guesses for the values of the constrained weight parameters.

Filtering inappropriate estimates: If `filter_estimates == TRUE`, the code will automatically filter through estimates that it deems "inappropriate". That is, estimates that are not likely solutions of interest. Specifically, solutions that incorporate a near-singular error term covariance matrix (any eigenvalue less than 0.002), any modulus of the eigenvalues of the companion form AR matrices larger than 0.9985 (indicating the necessary condition for stationarity is close to break), or transition weights such that they are close to zero for almost all t for at least one regime. You can also set `filter_estimates=FALSE` and find the solutions of interest yourself by using the function `alt_stvar` (which can used with `filter_estimates=TRUE` as well since results from all estimation rounds are saved).

Value

Returns an S3 object of class `'stvar'` defining a smooth transition VAR model. The returned list contains the following components (some of which may be `NULL` depending on the use case):

`data`	The input time series data.
`model`	A list describing the model structure.
`params`	The parameters of the model.
`std_errors`	Approximate standard errors of the parameters, if calculated.
`transition_weights`	The transition weights of the model.
`regime_cmeans`	Conditional means of the regimes, if data is provided.
`total_cmeans`	Total conditional means of the model, if data is provided.
`total_ccovs`	Total conditional covariances of the model, if data is provided.
`uncond_moments`	A list of unconditional moments including regime autocovariances, variances, and means.
`residuals_raw`	Raw residuals, if data is provided.
`residuals_std`	Standardized residuals, if data is provided.
`structural_shocks`	Recovered structural shocks, if applicable.
`loglik`	Log-likelihood of the model, if data is provided.
`IC`	The values of the information criteria (AIC, HQIC, BIC) for the model, if data is provided.
`all_estimates`	The parameter estimates from all estimation rounds, if applicable.
`all_logliks`	The log-likelihood of the estimates from all estimation rounds, if applicable.
`which_converged`	Indicators of which estimation rounds converged, if applicable.
`which_round`	Indicators of which round of optimization each estimate belongs to, if applicable.

S3 methods

The following S3 methods are supported for class `'stvar'`: `logLik`, `residuals`, `print`, `summary`, `predict`, `simulate`, and `plot`.

References

- Anderson H., Vahid F. 1998. Testing multiple equation systems for common nonlinear components. *Journal of Econometrics*, **84**:1, 1-36.
- Hubrich K., Teräsvirta. T. 2013. Thresholds and Smooth Transitions in Vector Autoregressive Models. *CREATES Research Paper 2013-18, Aarhus University.*
- Lanne M., Virolainen S. 2024. A Gaussian smooth transition vector autoregressive model: An application to the macroeconomic effects of severe weather shocks. Unpublished working paper, available as arXiv:2403.14216.

- Kheifets I.L., Saikkonen P.J. 2020. Stationarity and ergodicity of Vector STAR models. *Econometric Reviews*, **39**:4, 407-414.
- Tsay R. 1998. Testing and Modeling Multivariate Threshold Models. *Journal of the American Statistical Association*, **93**:443, 1188-1202.
- Virolainen S. 2024. Identification by non-Gaussianity in structural threshold and smooth transition vector autoregressive models. Unpublished working paper, available as arXiv:2404.19707.

See Also

fitSSTVAR, STVAR, GAfit, iterate_more

Examples

```
## These are long running examples. Running all the below examples will take
## approximately three minutes.
# When estimating the models in empirical applications, typically a large number
# of estimation rounds (set by the argument 'nrounds') should be used. These examples
# use only a small number of rounds to make the running time of the examples reasonable.

# The below examples make use of the two-variate dataset 'gdpdef' containing
# the the quarterly U.S. GDP and GDP deflator from 1947Q1 to 2019Q4.

# Estimate Gaussian STVAR model of autoregressive order p=3 and two regimes (M=2),
# with the weighted relative stationary densities of the regimes as the transition
# weight function. The estimation is performed with 2 rounds and 2 CPU cores, with
# the random number generator seeds set for reproducibility.
fit32 <- fitSTVAR(gdpdef, p=3, M=2, weight_function="relative_dens",
 cond_dist="Gaussian", nrounds=2, ncores=2, seeds=1:2)

# Examine the results:
fit32 # Printout of the estimates
summary(fit32) # A more detailed summary printout
plot(fit32) # Plot the fitted transition weights
get_foc(fit32) # Gradient of the log-likelihood function about the estimate
get_soc(fit32) # Eigenvalues of the Hessian of the log-lik. fn. about the estimate
profile_logliks(fit32) # Profile log-likelihood functions about the estimate

# Estimate a two-regime Student's t STVAR p=3 model with logistic transition weights
# and the first lag of the second variable as the switching variable, only two
# estimation rounds using two CPU cores:
fitlogistict32 <- fitSTVAR(gdpdef, p=3, M=2, weight_function="logistic", weightfun_pars=c(2, 1),
 cond_dist="Student", nrounds=2, ncores=2, seeds=1:2)
summary(fitlogistict32) # Summary printout of the estimates

# Estimate a two-regime threshold VAR p=3 model with independent Student's t shocks.
# The first lag of the the second variable is specified as the switching variable,
# and the threshold parameter constrained to the fixed value 1.
fitthres32wit <- fitSTVAR(gdpdef, p=3, M=2, weight_function="threshold", weightfun_pars=c(2, 1),
  cond_dist="ind_Student", weight_constraints=list(R=0, r=1), nrounds=2, ncores=2, seeds=1:2)
plot(fitthres32wit) # Plot the fitted transition weights

# Estimate a two-regime STVAR p=1 model with exogenous transition weights defined as the indicator
```

```
# of NBER based U.S. recessions (source: St. Louis Fed database). Moreover, restrict the AR matrices
# to be identical across the regimes (i.e., allowing for time-variation in the intercepts and the
# covariance matrix only):

# Step 1: Define transition weights of Regime 1 as the indicator of NBER based U.S. recessions
# (the start date of weights is start of data + p, since the first p values are used as the initial
# values):
tw1 <- c(0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1,
 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)

# Step 2: Define the transition weights of Regime 2 as one minus the weights of Regime 1, and
# combine the weights to matrix of transition weights:
twmat <- cbind(tw1, 1 - tw1)

# Step 3: Create the appropriate constraint matrix:
C_122 <- rbind(diag(1*2^2), diag(1*2^2))

# Step 4: Estimate the model by specifying the weights in the argument 'weightfun_pars'
# and the constraint matrix in the argument 'AR_constraints':
fitexo12cit <- fitSTVAR(gdpdef, p=1, M=2, weight_function="exogenous", weightfun_pars=twmat,
  cond_dist="ind_Student", AR_constraints=C_122, nrounds=2, ncores=2, seeds=1:2)
plot(fitexo12cit) # Plot the transition weights
summary(fitexo12cit) # Summary printout of the estimates

# Estimate a two-regime Gaussian STVAR p=1 model with the weighted relative stationary densities
# of the regimes as the transition weight function, and the means of the regimes
# and AR matrices constrained to be identical across the regimes (i.e., allowing for time-varying
# conditional covariance matrix only):
fit12cm <- fitSTVAR(gdpdef, p=1, M=2, weight_function="relative_dens", cond_dist="Gaussian",
 AR_constraints=C_122, mean_constraints=list(1:2), parametrization="mean", nrounds=2, seeds=1:2)
fit12cm # Print the estimates

# Estimate a two-regime Gaussian STVAR p=1 model with the weighted relative stationary densities
# of the regimes as the transition weight function; constrain AR matrices to be identical
# across the regimes and also constrain the off-diagonal elements of the AR matrices to be zero.
mat0 <- matrix(c(1, rep(0, 10), 1, rep(0, 8), 1, rep(0, 10), 1), nrow=2*2^2, byrow=FALSE)
C_222 <- rbind(mat0, mat0) # The constraint matrix
fit22c <- fitSTVAR(gdpdef, p=2, M=2, weight_function="relative_dens", cond_dist="Gaussian",
 AR_constraints=C_222, nrounds=2, seeds=1:2)
fit22c # Print the estimates

# Estimate a two-regime Student's t STVAR p=3 model with logistic transition weights
# and the first lag of the second variable as the switching variable. Constraint the location
# parameter to the fixed value 1 and leave the scale parameter unconstrained.
fitlogistic32w <- fitSTVAR(gdpdef, p=3, M=2, weight_function="logistic", weightfun_pars=c(2, 1),
 weight_constraints=list(R=matrix(c(0, 1), nrow=2), r=c(1, 0)), nrounds=2, seeds=1:2)
plot(fitlogistic32w) # Plot the fitted transition weights
```

```
# Estimate a two-regime Gaussian STVAR p=3 model with multinomial logit transition weights
# using the second variable is the switching variable with two lags. Constrain the AR matrices
# identical across the regimes (allowing for time-variation in the intercepts and covariance
# matrix).
C_322 <- rbind(diag(3*2^2), diag(3*2^2)) # The constraint matrix
fitmlogit32c <- fitSTVAR(gdpdef, p=3, M=2, weight_function="mlogit", cond_dist="Gaussian",
 weightfun_pars=list(vars=2, lags=2), AR_constraints=C_322, nrounds=1, seeds=3, ncores=1)
plot(fitmlogit32c) # Plot the fitted transition weights

# Estimate a two-regime Gaussian STVAR p=3 model with exponential transition weights and the first
# lag of the second variable as switching variable, and AR parameter constrained identical across
# the regimes, means constrained identical across the regimes, and the location parameter
# constrained to 0.5 (but scale parameter unconstrained).
fitexp32cmw <- fitSTVAR(gdpdef, p=3, M=2, weight_function="exponential", weightfun_pars=c(2, 1),
 cond_dist="Student", AR_constraints=C_322, mean_constraints=list(1:2),
 weight_constraints=list(R=matrix(c(0, 1), nrow=2), r=c(0.5, 0)), nrounds=1, seeds=1, ncores=1)
summary(fitexp32cmw) # Summary printout of the estimates
```

GAfit *Genetic algorithm for preliminary estimation of a STVAR models*

Description

`GAfit` estimates the specified STVAR model using a genetic algorithm. It is designed to find starting values for gradient based methods and NOT to obtain final estimates constituting a local maximum.

Usage

```
GAfit(
  data,
  p,
  M,
 weight_function = c("relative_dens", "logistic", "mlogit", "exponential", "threshold",
    "exogenous"),
  weightfun_pars = NULL,
  cond_dist = c("Gaussian", "Student", "ind_Student"),
  parametrization = c("intercept", "mean"),
  AR_constraints = NULL,
  mean_constraints = NULL,
  weight_constraints = NULL,
  ngen = 200,
  popsize,
  smart_mu = min(100, ceiling(0.5 * ngen)),
  initpop = NULL,
  mu_scale,
  mu_scale2,
```

```
  omega_scale,
  B_scale,
  weight_scale,
  ar_scale = 0.2,
  upper_ar_scale = 1,
  ar_scale2 = 1,
  regime_force_scale = 1,
  red_criteria = c(0.05, 0.01),
  pre_smart_mu_prob = 0,
  to_return = c("alt_ind", "best_ind"),
  minval,
  seed = NULL
)
```

Arguments

`data` a matrix or class `'ts'` object with `d>1` columns. Each column is taken to represent a univariate time series. Missing values are not supported.

`p` a positive integer specifying the autoregressive order

`M` a positive integer specifying the number of regimes

`weight_function`

What type of transition weights $\alpha_{m,t}$ should be used?

`"relative_dens"`: $\alpha_{m,t} = \frac{\alpha_m f_{m,dp}(y_{t-1},...,y_{t-p+1})}{\sum_{n=1}^{M} \alpha_n f_{n,dp}(y_{t-1},...,y_{t-p+1})}$, where $\alpha_m \in (0,1)$ are weight parameters that satisfy $\sum_{m=1}^{M} \alpha_m = 1$ and $f_{m,dp}(\cdot)$ is the dp-dimensional stationary density of the mth regime corresponding to p consecutive observations. Available for Gaussian conditional distribution only.

`"logistic"`: $M = 2$, $\alpha_{1,t} = 1 - \alpha_{2,t}$, and $\alpha_{2,t} = [1 + \exp\{-\gamma(y_{it-j} - c)\}]^{-1}$, where y_{it-j} is the lag j observation of the ith variable, c is a location parameter, and $\gamma > 0$ is a scale parameter.

`"mlogit"`: $\alpha_{m,t} = \frac{\exp\{\gamma_m' z_{t-1}\}}{\sum_{n=1}^{M} \exp\{\gamma_n' z_{t-1}\}}$, where γ_m are coefficient vectors, $\gamma_M = 0$, and z_{t-1} $(k \times 1)$ is the vector containing a constant and the (lagged) switching variables.

`"exponential"`: $M = 2$, $\alpha_{1,t} = 1 - \alpha_{2,t}$, and $\alpha_{2,t} = 1 - \exp\{-\gamma(y_{it-j} - c)\}$, where y_{it-j} is the lag j observation of the ith variable, c is a location parameter, and $\gamma > 0$ is a scale parameter.

`"threshold"`: $\alpha_{m,t} = 1$ if $r_{m-1} < y_{it-j} \leq r_m$ and 0 otherwise, where $-\infty \equiv r_0 < r_1 < \cdots < r_{M-1} < r_M \equiv \infty$ are thresholds y_{it-j} is the lag j observation of the ith variable.

`"exogenous"`: Exogenous nonrandom transition weights, specify the weight series in `weightfun_pars`.

See the vignette for more details about the weight functions.

`weightfun_pars` **If** `weight_function == "relative_dens"`: Not used.

If `weight_function %in% c("logistic", "exponential", "threshold")`: a numeric vector with the switching variable $i \in \{1, ..., d\}$ in the first and the lag $j \in \{1, ..., p\}$ in the second element.

If `weight_function == "mlogit"`**:** a list of two elements:

The first element `$vars`**:** a numeric vector containing the variables that should used as switching variables in the weight function in an increasing order, i.e., a vector with unique elements in $\{1, ..., d\}$.

The second element `$lags`**:** an integer in $\{1, ..., p\}$ specifying the number of lags to be used in the weight function.

If `weight_function == "exogenous"`**:** a size (`nrow(data) - p` x `M`) matrix containing the exogenous transition weights as `[t, m]` for time t and regime m. Each row needs to sum to one and only weakly positive values are allowed.

`cond_dist` specifies the conditional distribution of the model as `"Gaussian"`, `"Student"`, or `"ind_Student"`, where the latest is the Student's t distribution with independent components.

`parametrization`

`"intercept"` or `"mean"` determining whether the model is parametrized with intercept parameters $\phi_{m,0}$ or regime means μ_m, m=1,...,M.

`AR_constraints` a size (Mpd^2xq) constraint matrix C specifying linear constraints to the autoregressive parameters. The constraints are of the form $(\varphi_1, ..., \varphi_M) = C\psi$, where $\varphi_m = (vec(A_{m,1}), ..., vec(A_{m,p}))$ (pd^2x1), $m = 1, ..., M$, contains the coefficient matrices and ψ $(qx1)$ contains the related parameters. For example, to restrict the AR-parameters to be the identical across the regimes, set $C =$ `[I:...:I]'` (Mpd^2xpd^2) where `I = diag(p*d^2)`.

`mean_constraints`

Restrict the mean parameters of some regimes to be identical? Provide a list of numeric vectors such that each numeric vector contains the regimes that should share the common mean parameters. For instance, if `M=3`, the argument `list(1, 2:3)` restricts the mean parameters of the second and third regime to be identical but the first regime has freely estimated (unconditional) mean. Ignore or set to `NULL` if mean parameters should not be restricted to be the same among any regimes. This constraint is available only for mean parametrized models; that is, when `parametrization="mean"`.

`weight_constraints`

a list of two elements, R in the first element and r in the second element, specifying linear constraints on the transition weight parameters α. The constraints are of the form $\alpha = R\xi + r$, where R is a known $(a \times l)$ constraint matrix of full column rank (a is the dimension of α), r is a known $(a \times 1)$ constant, and ξ is an unknown $(l \times 1)$ parameter. **Alternatively**, set $R = 0$ in order to constrain the the weight parameter to the constant r (in this case, α is dropped from the constrained parameter vector).

`ngen` a positive integer specifying the number of generations to be ran through in the genetic algorithm.

`popsize` a positive even integer specifying the population size in the genetic algorithm. Default is `10*n_params`.

`smart_mu` a positive integer specifying the generation after which the random mutations in the genetic algorithm are "smart". This means that mutating individuals will mostly mutate fairly close (or partially close) to the best fitting individual (which has the least regimes with time varying mixing weights practically at zero) so far.

`initpop` a list of parameter vectors from which the initial population of the genetic algorithm will be generated from. The parameter vectors should have the form $\theta = (\phi_{1,0}, ..., \phi_{M,0}, \varphi_1, ..., \varphi_M, \sigma, \alpha, \nu)$, where

- $\phi_{m,0}$ = the $(d \times 1)$ intercept (or mean) vector of the mth regime.
- $\varphi_m = (vec(A_{m,1}), ..., vec(A_{m,p}))$ $(pd^2 \times 1)$.
- **if** `cond_dist="Gaussian"` **or** `"Student"`: $\sigma = (vech(\Omega_1), ..., vech(\Omega_M))$ $(Md(d+1)/2 \times 1)$.

  if `cond_dist="ind_Student"`: $\sigma = (vec(B_1), ..., vec(B_M)$ $(Md^2 \times 1)$.
- α contains the transition weights parameters (see below)
- **if** `cond_dist = "Gaussian"`): Omit ν from the parameter vector.

  if `cond_dist="Student"`: $\nu > 2$ is the single degrees of freedom parameter.

  if `cond_dist="ind_Student"`: $\nu = (\nu_1, ..., \nu_M)$ $(M \times 1)$, $nu_m > 2$.

`weight_function="relative_dens"`: $\alpha = (\alpha_1, ..., \alpha_{M-1})$ $(M-1 \times 1)$, where α_m $(1 \times 1), m = 1, ..., M-1$ are the transition weight parameters.

`weight_function="logistic"`: $\alpha = (c, \gamma)$ (2×1), where $c \in \mathbb{R}$ is the location parameter and $\gamma > 0$ is the scale parameter.

`weight_function="mlogit"`: $\alpha = (\gamma_1, ..., \gamma_M)$ $((M-1)k \times 1)$, where γ_m $(k \times 1)$, $m = 1, ..., M-1$ contains the multinomial logit-regression coefficients of the mth regime. Specifically, for switching variables with indices in $I \subset \{1, ..., d\}$, and with $\tilde{p} \in \{1, ..., p\}$ lags included, γ_m contains the coefficients for the vector $z_{t-1} = (1, \tilde{z}_{\min\{I\}}, ..., \tilde{z}_{\max\{I\}})$, where $\tilde{z}_i = (y_{it-1}, ..., y_{it-\tilde{p}})$, $i \in I$. So $k = 1 + |I|\tilde{p}$ where $|I|$ denotes the number of elements in I.

`weight_function="exponential"`: $\alpha = (c, \gamma)$ (2×1), where $c \in \mathbb{R}$ is the location parameter and $\gamma > 0$ is the scale parameter.

`weight_function="threshold"`: $\alpha = (r_1, ..., r_{M-1})$ $(M - 1 \times 1)$, where $r_1, ..., r_{M-1}$ are the threshold values.

`weight_function="exogenous"`: Omit α from the parameter vector.

AR_constraints: Replace $\varphi_1, ..., \varphi_M$ with ψ as described in the argument `AR_constraints`.

mean_constraints: Replace $\phi_{1,0}, ..., \phi_{M,0}$ with $(\mu_1, ..., \mu_g)$ where μ_i, $(d \times 1)$ is the mean parameter for group i and g is the number of groups.

weight_constraints: If linear constraints are imposed, replace α with ξ as described in the argument `weigh_constraints`. If weight functions parameters are imposed to be fixed values, simply drop α from the parameter vector.

Above, $\phi_{m,0}$ is the intercept parameter, $A_{m,i}$ denotes the ith coefficient matrix of the mth regime, Ω_m denotes the positive definite error term covariance matrix of the mth regime, and B_m is the invertible $(d \times d)$ impact matrix of the mth regime. ν_m is the degrees of freedom parameter of the mth regime. If `parametrization=="mean"`, just replace each $\phi_{m,0}$ with regimewise mean μ_m. $vec()$ is vectorization operator that stacks columns of a given matrix into a vector. $vech()$ stacks columns of a given matrix from the principal diagonal downwards (including elements on the diagonal) into a vector. $Bvec()$ is a vectorization operator that stacks the columns of a given impact matrix B_m

into a vector so that the elements that are constrained to zero by the argument `B_constraints` are excluded.

`mu_scale` a size $(dx1)$ vector defining **means** of the normal distributions from which each mean parameter μ_m is drawn from in random mutations. Default is `colMeans(data)`. Note that mean-parametrization is always used for optimization in `GAfit` - even when `parametrization=="intercept"`. However, input (in `initpop`) and output (return value) parameter vectors can be intercept-parametrized.

`mu_scale2` a size $(dx1)$ strictly positive vector defining **standard deviations** of the normal distributions from which each mean parameter μ_m is drawn from in random mutations. Default is `vapply(1:d, function(i1) sd(data[,i1]), numeric(1))`.

`omega_scale` a size $(dx1)$ strictly positive vector specifying the scale and variability of the random covariance matrices in random mutations. The covariance matrices are drawn from (scaled) Wishart distribution. Expected values of the random covariance matrices are `diag(omega_scale)`. Standard deviations of the diagonal elements are `sqrt(2/d)*omega_scale[i]` and for non-diagonal elements they are `sqrt(1/d*omega_scale[i]*omega_scale[j])`. Note that for `d>4` this scale may need to be chosen carefully. Default in `GAfit` is `var(stats::ar(data[,i], order.max=10)$resid, na.rm=TRUE), i=1,...,d`. This argument is ignored if `cond_dist == "ind_Student"`.

`B_scale` a size $(d \times 1)$ strictly positive vector specifying the mean and variability of the random impact matrices in random mutations. In Regime 1, the mean of the error term covariance matrix implied by the random impact matrix will be `0.95*diag(B_scale)` and in the rest of the regimes `diag(B_scale)`, whereas the variability increases with `B_scale`. Default in `GAfit` is `var(stats::ar(data[,i], order.max=10)$resid, na.rm=TRUE), i=1,...,d`. This argument is ignored if `cond_dist != "ind_Student"`.

`weight_scale` For...

`weight_function %in% c("relative_dens", "exogenous"):` not used.

`weight_function %in% c("logistic", "exponential"):` length three vector with the mean (in the first element) and standard deviation (in the second element) of the normal distribution the location parameter is drawn from in random mutations. The third element is the standard deviation of the normal distribution from whose absolute value the location parameter is drawn from.

`weight_function == "mlogit":` length two vector with the mean (in the first element) and standard deviation (in the second element) of the normal distribution the coefficients of the logit sub model's constant terms are drawn from in random mutations. The third element is the standard deviation of the normal distribution from which the non-constant regressors' coefficients are drawn from.

`weight_function == "threshold":` a lenght two vector with the lower bound, in the first element and the upper bound, in the second element, of the uniform distribution threshold parameters are drawn from in random mutations.

`ar_scale` a positive real number between zero and one adjusting how large AR parameter values are typically proposed in construction of the initial population: larger

value implies larger coefficients (in absolute value). After construction of the initial population, a new scale is drawn from `(0, upper_ar_scale)` uniform distribution in each iteration.

`upper_ar_scale` the upper bound for `ar_scale` parameter (see above) in the random mutations. Setting this too high might lead to failure in proposing new parameters that are well enough inside the parameter space, and especially with large `p` one might want to try smaller upper bound (e.g., 0.5). With large `p` or `d`, `upper_ar_scale` is restricted from above, see the details section.

`ar_scale2` a positive real number adjusting how large AR parameter values are typically proposed in some random mutations (if AR constraints are employed, in all random mutations): larger value implies **smaller** coefficients (in absolute value). **Values larger than 1 can be used if the AR coefficients are expected to be very small. If set smaller than 1, be careful as it might lead to failure in the creation of parameter candidates that satisfy the stability condition.**

`regime_force_scale`
a non-negative real number specifying how much should natural selection favor individuals with less regimes that have almost all mixing weights (practically) at zero. Set to zero for no favoring or large number for heavy favoring. Without any favoring the genetic algorithm gets more often stuck in an area of the parameter space where some regimes are wasted, but with too much favouring the best genes might never mix into the population and the algorithm might converge poorly. Default is `1` and it gives $2x$ larger surviving probability weights for individuals with no wasted regimes compared to individuals with one wasted regime. Number `2` would give $3x$ larger probability weights etc.

`red_criteria` a length 2 numeric vector specifying the criteria that is used to determine whether a regime is redundant (or "wasted") or not. Any regime `m` which satisfies `sum(transitionWeights[,m] > red_criteria[1]) < red_criteria[2]*n_obs` will be considered "redundant". One should be careful when adjusting this argument (set `c(0, 0)` to fully disable the 'redundant regime' features from the algorithm).

`pre_smart_mu_prob`
A number in $[0, 1]$ giving a probability of a "smart mutation" occuring randomly in each iteration before the iteration given by the argument `smart_mu`.

`to_return` should the genetic algorithm return the best fitting individual which has "positive enough" mixing weights for as many regimes as possible (`"alt_ind"`) or the individual which has the highest log-likelihood in general (`"best_ind"`) but might have more wasted regimes?

`minval` a real number defining the minimum value of the log-likelihood function that will be considered. Values smaller than this will be treated as they were `minval` and the corresponding individuals will never survive. The default is `-(10^(ceiling(log10(n_obs)) + d) - 1)`.

`seed` a single value, interpreted as an integer, or NULL, that sets seed for the random number generator in the beginning of the function call. If calling `GAfit` from `fitSTVAR`, use the argument `seeds` instead of passing the argument `seed`.

Details

Only reduced form models are supported!

The core of the genetic algorithm is mostly based on the description by *Dorsey and Mayer (1995)*. It utilizes a slightly modified version of the individually adaptive crossover and mutation rates described by *Patnaik and Srinivas (1994)* and employs (50%) fitness inheritance discussed by *Smith, Dike and Stegmann (1995)*.

By "redundant" or "wasted" regimes we mean regimes that have the time varying mixing weights practically at zero for almost all t. A model including redundant regimes would have about the same log-likelihood value without the redundant regimes and there is no purpose to have redundant regimes in a model.

Some of the AR coefficients are drawn with the algorithm by Ansley and Kohn (1986). However, when using large `ar_scale` with large `p` or `d`, numerical inaccuracies caused by the imprecision of the float-point presentation may result in errors or nonstationary AR-matrices. Using smaller `ar_scale` facilitates the usage of larger `p` or `d`. Therefore, we bound `upper_ar_scale` from above by $1 - pd/150$ when `p*d>40` and by 1 otherwise.

Structural models are not supported here, as they are best estimated based on reduced form parameter estimates using the function `fitSSTVAR`.

Value

Returns the estimated parameter vector which has the form described in `initpop`, **with the exception** that for models with `cond_dist == "ind_Student"` or `identification="non-Gaussianity"`, the parameter vector is parametrized with $B_1, B_2^*, ..., B_M^*$ instead of $B_1, B_2, ..., B_M$, where $B_m^* = B_m - B_1$. Use the function `change_parametrization` to change back to the original parametrization if desired.

References

- Ansley C.F., Kohn R. 1986. A note on reparameterizing a vector autoregressive moving average model to enforce stationarity. *Journal of statistical computation and simulation*, **24**:2, 99-106.
- Dorsey R. E. and Mayer W. J. 1995. Genetic algorithms for estimation problems with multiple optima, nondifferentiability, and other irregular features. *Journal of Business & Economic Statistics*, **13**, 53-66.
- Patnaik L.M. and Srinivas M. 1994. Adaptive Probabilities of Crossover and Mutation in Genetic Algorithms. *Transactions on Systems, Man and Cybernetics* **24**, 656-667.
- Smith R.E., Dike B.A., Stegmann S.A. 1995. Fitness inheritance in genetic algorithms. *Proceedings of the 1995 ACM Symposium on Applied Computing*, 345-350.

`gdpdef` *U.S. real GDP percent change and GDP implicit price deflator percent change.*

Description

A dataset containing a quarterly U.S. time series with two components: the percentage change of real GDP and the percentage change of GDP implicit price deflator, covering the period from 1959Q1 - 2019Q4.

Usage

```
gdpdef
```

Format

A numeric matrix of class 'ts' with 244 rows and 2 columns with one time series in each column:

First column (GDP): The quarterly percent change of real U.S. GDP, from 1959Q1 to 2019Q4, https://fred.stlouisfed.org/series/GDPC1.

Second column (GDPDEF): The quarterly percent change of U.S. GDP implicit price deflator, from 1959Q1 to 2019Q4, https://fred.stlouisfed.org/series/GDPDEF.

Source

The Federal Reserve Bank of St. Louis database

---

get_hetsked_sstvar *Switch from two-regime reduced form STVAR model to a structural model identified by heteroskedasticity*

Description

get_hetsked_sstvar constructs structural STVAR model identified by heteroskedasticity based on a reduced form STVAR model.

Usage

```
get_hetsked_sstvar(stvar, calc_std_errors = FALSE)
```

Arguments

stvar	a an object of class 'stvar', created by, e.g., fitSTVAR, specifying a reduced form or a structural model
calc_std_errors	should approximate standard errors be calculated?

Details

The switch is made by simultaneously diagonalizing the two error term covariance matrices with a well known matrix decomposition (Muirhead, 1982, Theorem A9.9) and then normalizing the diagonal of the matrix W positive (which implies positive diagonal of the impact matrix). Models with more that two regimes are not supported because the matrix decomposition does not generally exists for more than two covariance matrices.

Value

Returns an object of class 'stvar' defining a structural STVAR model identified by heteroskedasticity, with the main diagonal of the impact matrix normalized to be positive.

See Also

fitSSTVAR, STVAR, fitSTVAR

- Muirhead R.J. 1982. Aspects of Multivariate Statistical Theory, *Wiley*.

GFEVD *Estimate generalized forecast error variance decomposition for structural STVAR models.*

Description

GFEVD estimates generalized forecast error variance decomposition for structural STVAR models.

Usage

```
GFEVD(
  stvar,
  shock_size = 1,
  N = 30,
  initval_type = c("data", "random", "fixed"),
  use_data_shocks = FALSE,
  R1 = 250,
  R2 = 250,
  init_regime = 1,
  init_values = NULL,
  which_cumulative = numeric(0),
  ncores = 2,
  burn_in = 1000,
  exo_weights = NULL,
  seeds = NULL,
  use_parallel = TRUE
)

## S3 method for class 'gfevd'
plot(x, ..., data_shock_pars = NULL)

## S3 method for class 'gfevd'
print(x, ..., digits = 2, N_to_print)
```

Arguments

stvar	an object of class 'stvar', created by, e.g., fitSTVAR or fitSSTVAR.
shock_size	What sign and size should be used for all shocks? By the normalization, the conditional covariance matrix of the structural error is an identity matrix.
N	a positive integer specifying the horizon how far ahead should the GFEVD be calculated.

`initval_type` What type initial values are used for estimating the GIRFs that the GFEVD is based on?

`"data"`: Estimate the GIRF for all the possible length p histories in the data.

`"random"`: Estimate the GIRF for several random initial values generated from the a specific regime specified by the argument `init_regimes`. The number of initial values is set with the argument `R2`.

`"fixed"`: Estimate the GIRF for the initial values specified with the argument `init_values`.

`use_data_shocks`

`TRUE` for a special feature in which for every possible length p history in the data, the GFEVD is estimated for a shock that has the sign and size of the corresponding structural shock recovered from the data. See the details section.

`R1` the number of repetitions used to estimate GIRF for each initial value.

`R2` the number of initial values to be drawn/used if `initval_type="random"` or `"fixed"`.

`init_regime` an integer in $1, ..., M$ specifying the regime from which the initial values should be generated from. The initial values will be generated from the stationary distribution of the specific regime. Due to the lack of knowledge of the stationary distribution, models with other than Gaussian conditional distribution uses a simulation procedure with a burn-in period. See the details section.

`init_values` a size `[p, d, R2]` array specifying the initial values in each slice for each Monte Carlo repetition, where `d` is the number of time series in the system and `R2` is an argument of this function. In each slice, the **last** row will be used as initial values for the first lag, the second last row for second lag etc. If not specified, initial values will be drawn from the regime specified in `init_regimes`.

`which_cumulative`

a numeric vector with values in $1, ..., d$ (`d=ncol(data)`) specifying which the variables for which the impulse responses should be cumulative. Default is none.

`ncores` the number CPU cores to be used in parallel computing. Only single core computing is supported if an initial value is specified (and the GIRF won't thus be estimated multiple times).

`burn_in` Burn-in period for simulating initial values from a regime when `cond_dist!="Gaussian"`. See the details section.

`exo_weights` if `weight_function="exogenous"`, provide a size $(N + 1xM)$ matrix of exogenous transition weights for the regimes: `[h, m]` for the (after-the-impact) period $h - 1$ and regime m weight (`[1, m]` is for the impact period). Ignored if `weight_function!="exogenous"`.

`seeds` a numeric vector containing the random number generator seed for estimation of each GIRF. Should have the length...

- ...`nrow(data) - p + 1` if `initval_type="data"`.
- ...`R2` if `initval_type="random"`.
- ...1 if `initval_type="fixed."`.

Set to `NULL` for not initializing the seed. Exists for creating reproducible results.

`use_parallel` employ parallel computing? If `FALSE`, does not print anything.

x	object of class 'gfevd' generated by the function GFEVD.
...	graphical parameters passed to the 'ts' plot method when using data_shock_pars.
data_shock_pars	if use_data_shocks, alternative plot method can be used that plots the relative contribution of a given shock to the forecast error variance of each variable at a given horizon. Should be a length two numeric vector with the number of the shock (1,..,d) in the first element and the horizon (0,1,2,...,N) in the second element.
digits	the number of decimals to print
N_to_print	an integer specifying the horizon how far to print the estimates. The default is that all the values are printed.

Details

The GFEVD is a forecast error variance decomposition calculated with the generalized impulse response function (GIRF). See Lanne and Nyberg (2016) for details.

If use_data_shocks == TRUE, the GIRF is estimated for a shock that has the sign and size of the corresponding structural shock recovered from the fitted model. This is done for every possible length p history in the data. The GFEVD is then calculated as the average of the GFEVDs obtained from the GIRFs estimated for the data shocks. The plot and print methods can be used as usual for this GFEVD. However, this feature also obtain the contribution of each shock to the variance of the forecast errors at various horizons in specific historical points of time. This can be done by using the plot method with the argument data_shock_pars. Note that the arguments shock_size, initval_type, and init_regime are ignored if use_data_shocks == TRUE.

Value

Returns and object of class 'gfevd' containing the GFEVD for all the variables and to the transition weights. Note that the decomposition does not exist at horizon zero for transition weights because the related GIRFs are always zero at impact. If use_data_shocks=TRUE, also contains the GFEVDs for each length p history in the data as 4D array with dimensions [horizon, variable, shock, time].

Functions

- plot(gfevd): plot method
- print(gfevd): print method

References

- Lanne M. and Nyberg H. 2016. Generalized Forecast Error Variance Decomposition for Linear and Nonlineae Multivariate Models. *Oxford Bulletin of Economics and Statistics*, **78**, 4, 595-603.

See Also

GIRF, linear_IRF, fitSSTVAR

Examples

```
# These are long-running examples that use parallel computing.
# It takes approximately 30 seconds to run all the below examples.
# Note that larger R1 and R2 should be used for more reliable results;
# small R1 and R2 are used here to shorten the estimation time.

# Recursively identifed logistic Student's t STVAR(p=3, M=2) model with the first
# lag of the second variable as the switching variable:
params32logt <- c(0.5959, 0.0447, 2.6279, 0.2897, 0.2837, 0.0504, -0.2188, 0.4008,
 0.3128, 0.0271, -0.1194, 0.1559, -0.0972, 0.0082, -0.1118, 0.2391, 0.164, -0.0363,
 -1.073, 0.6759, 3e-04, 0.0069, 0.4271, 0.0533, -0.0498, 0.0355, -0.4686, 0.0812,
  0.3368, 0.0035, 0.0325, 1.2289, -0.047, 0.1666, 1.2067, 7.2392, 11.6091)
mod32logt <- STVAR(gdpdef, p=3, M=2, params=params32logt, weight_function="logistic",
 weightfun_pars=c(2, 1), cond_dist="Student", identification="recursive")

# GFEVD for one-standard-error positive structural shocks, N=30 steps ahead,
# with fix initial values assuming all possible histories in the data.
gfevd1 <- GFEVD(mod32logt, shock_size=1, N=30, initval_type="data", R1=10,
  seeds=1:(nrow(mod32logt$data)-2))
print(gfevd1) # Print the results
plot(gfevd1) # Plot the GFEVD

# GFEVD for one-standard-error positive structural shocks, N=30 steps ahead,
# with fix initial values that are the last p observations of the data.
gfevd2 <- GFEVD(mod32logt, shock_size=1, N=30, initval_type="fixed", R1=100, R2=1,
 init_values=array(mod32logt$data[(nrow(mod32logt$data) - 2):nrow(mod32logt$data),],
 dim=c(3, 2, 1)), seeds=1)
plot(gfevd2) # Plot the GFEVD

# GFEVD for two-standard-error negative structural shocks, N=50 steps ahead
# with the inital values drawn from the first regime. The responses of both
# variables are accumulated.
gfevd3 <- GFEVD(mod32logt, shock_size=-2, N=50, initval_type="random",
 R1=50, R2=50, init_regime=1)
plot(gfevd3) # Plot the GFEVD

# GFEVD calculated for each lenght p history in the data in such a way that
# for each history, the structural shock recoved from the fitted model is
# used.
gfevd4 <- GFEVD(mod32logt, N=20, use_data_shocks=TRUE, R1=10)
plot(gfevd4) # Usual plot method

# Plot the contribution of the first to the variance of the forecast errors at
# the historial points of time using the structural shocks recovered from the data:
plot(gfevd4, data_shock_pars=c(1, 0)) # Contribution at impact
plot(gfevd4, data_shock_pars=c(1, 2)) # Contribution after two periods
plot(gfevd4, data_shock_pars=c(1, 4)) # Contribution after four periods
```

GIRF *Estimate generalized impulse response function for structural STVAR models.*

Description

GIRF estimates generalized impulse response function for structural STVAR models.

Usage

```
GIRF(
  stvar,
  which_shocks,
  shock_size = 1,
  N = 30,
  R1 = 250,
  R2 = 250,
  init_regime = 1,
  init_values = NULL,
  which_cumulative = numeric(0),
  scale = NULL,
  scale_type = c("instant", "peak"),
  scale_horizon = N,
  ci = c(0.95, 0.8),
  ncores = 2,
  burn_in = 1000,
  exo_weights = NULL,
  seeds = NULL,
  use_parallel = TRUE
)

## S3 method for class 'girf'
plot(x, margs, ...)

## S3 method for class 'girf'
print(x, ..., digits = 2, N_to_print)
```

Arguments

stvar	an object of class 'stvar', created by, e.g., fitSTVAR or fitSSTVAR.
which_shocks	a numeric vector of length at most d (=ncol(data)) and elements in $1, ..., d$ specifying the structural shocks for which the GIRF should be estimated.
shock_size	a non-zero scalar value specifying the common size for all scalar components of the structural shock. Note that the conditional covariance matrix of the structural shock is normalized to an identity matrix and that the (generalized) impulse responses may not be symmetric with respect to the sign and size of the shock.
N	a positive integer specifying the horizon how far ahead should the generalized impulse responses be calculated.

R1 the number of repetitions used to estimate GIRF for each initial value.

R2 the number of initial values to use, i.e., to draw from `init_regime` if `init_values` are not specified. The confidence bounds will be sample quantiles of the GIRFs based on different initial values. Ignored if the argument `init_value` is specified.

`init_regime` an integer in $1, ..., M$ specifying the regime from which the initial values should be generated from. The initial values will be generated from the stationary distribution of the specific regime. Due to the lack of knowledge of the stationary distribution, models with other than Gaussian conditional distribution uses a simulation procedure with a burn-in period. See the details section.

`init_values` a size `[p, d, R2]` array specifying the initial values in each slice for each Monte Carlo repetition, where d is the number of time series in the system and `R2` is an argument of this function. In each slice, the **last** row will be used as initial values for the first lag, the second last row for second lag etc. If not specified, initial values will be drawn from the regime specified in `init_regimes`.

`which_cumulative`

a numeric vector with values in $1, ..., d$ (`d=ncol(data)`) specifying which the variables for which the impulse responses should be cumulative. Default is none.

`scale` should the GIRFs to some of the shocks be scaled so that they correspond to a specific magnitude of instantaneous or peak response of some specific variable (see the argument `scale_type`)? Provide a length three vector where the shock of interest is given in the first element (an integer in $1, ..., d$), the variable of interest is given in the second element (an integer in $1, ..., d$), and the magnitude of its instantaneous or peak response in the third element (a non-zero real number). If the GIRFs of multiple shocks should be scaled, provide a matrix which has one column for each of the shocks with the columns being the length three vectors described above.

`scale_type` If argument `scale` is specified, should the GIRFs be scaled to match an instantaneous response (`"instant"`) or peak response (`"peak"`). If `"peak"`, the scale is based on the largest magnitude of peak response in absolute value. Ignored if `scale` is not specified.

`scale_horizon` If `scale_type == "peak"` what the maximum horizon up to which peak response is expected? Scaling won't based on values after this.

`ci` a numeric vector with elements in $(0, 1)$ specifying the confidence levels of the confidence intervals.

`ncores` the number CPU cores to be used in parallel computing. Only single core computing is supported if an initial value is specified (and the GIRF won't thus be estimated multiple times).

`burn_in` Burn-in period for simulating initial values from a regime when `cond_dist!="Gaussian"`. See the details section.

`exo_weights` if `weight_function="exogenous"`, provide a size $(N + 1xM)$ matrix of exogenous transition weights for the regimes: `[h, m]` for the (after-the-impact) period $h - 1$ and regime m weight (`[1, m]` is for the impact period). Ignored if `weight_function!="exogenous"`.

seeds	a length R2 vector containing the random number generator seed for estimation of each GIRF. A single number of an initial value is specified. or NULL for not initializing the seed. Exists for creating reproducible results.
use_parallel	employ parallel computing? If FALSE, does not print anything.
x	object of class 'girf' generated by the function GIRF.
margs	numeric vector of length four that adjusts the [bottom_marginal, left_marginal, top_marginal, right_marginal] as the relative sizes of the marginals to the figures of the responses.
...	graphical parameters passed to plot method plotting the GIRFs
digits	the number of decimals to print
N_to_print	an integer specifying the horizon how far to print the estimates and confidence intervals. The default is that all the values are printed.

Details

The confidence bounds reflect uncertainty about the initial state (but not about the parameter estimates) if initial values are not specified. If initial values are specified, confidence intervals won't be estimated.

Note that if the argument scale is used, the scaled responses of the transition weights might be more than one in absolute value.

If weight_function="exogenous", exogenous transition weights used in the Monte Carlo simulations for the future sample paths of the process must the given in the argument exo_weights. The same weights are used as the transition weights across the Monte Carlo repetitions.

Value

Returns a class 'girf' list with the GIRFs in the first element ($girf_res) and the used arguments the rest. The first element containing the GIRFs is a list with the mth element containing the point estimates for the GIRF in $point_est (the first element) and confidence intervals in $conf_ints (the second element). The first row is for the GIRF at impact $(n = 0)$, the second for $n = 1$, the third for $n = 2$, and so on.

The element $all_girfs is a list containing results from all the individual GIRFs obtained from the MC repetitions. Each element is for one shock and results are in array of the form [horizon, variables, MC-repetitions].

Functions

- plot(girf): plot method
- print(girf): print method

See Also

GFEVD, linear_IRF, fitSSTVAR

- Kilian L., Lütkepohl H. 20017. Structural Vector Autoregressive Analysis. 1st edition. *Cambridge University Press*, Cambridge.

Examples

```
# These are long-running examples that use parallel computing.
# It takes approximately 30 seconds to run all the below examples.
# Note that larger R1 and R2 should be used for more reliable results;
# small R1 and R2 are used here to shorten the estimation time.

# Recursively identified logistic Student's t STVAR(p=3, M=2) model with the first
# lag of the second variable as the switching variable:
params32logt <- c(0.5959, 0.0447, 2.6279, 0.2897, 0.2837, 0.0504, -0.2188, 0.4008,
 0.3128, 0.0271, -0.1194, 0.1559, -0.0972, 0.0082, -0.1118, 0.2391, 0.164, -0.0363,
 -1.073, 0.6759, 3e-04, 0.0069, 0.4271, 0.0533, -0.0498, 0.0355, -0.4686, 0.0812,
  0.3368, 0.0035, 0.0325, 1.2289, -0.047, 0.1666, 1.2067, 7.2392, 11.6091)
mod32logt <- STVAR(gdpdef, p=3, M=2, params=params32logt, weight_function="logistic",
 weightfun_pars=c(2, 1), cond_dist="Student", identification="recursive")

# GIRF for one-standard-error positive structural shocks, N=30 steps ahead,
# with the inital values drawn from the first regime.
girf1 <- GIRF(mod32logt, which_shocks=1:2, shock_size=1, N=30, R1=50, R2=50,
 init_regime=2)
print(girf1) # Print the results
plot(girf1) # Plot the GIRFs

# GIRF for one-standard-error positive structural shocks, N=30 steps ahead,
# with the inital values drawn from the second regime. The responses of the
# GDP and GDP deflator growth rates are accumulated.
girf2 <- GIRF(mod32logt, which_shocks=1:2, which_cumulative=1:2, shock_size=1,
 N=30, R1=50, R2=50, init_regime=2)
plot(girf2) # Plot the GIRFs

# GIRF for two-standard-error negative structural shock - the first shock only.
# N=50 steps ahead with the inital values drawn from the first regime. The responses
# are scaled to correspond an instantanous increase of 0.5 of the first variable.
girf3 <- GIRF(mod32logt, which_shocks=1, shock_size=-2, N=50, R1=50, R2=50,
 init_regime=1, scale_type="instant", scale=c(1, 1, 0.5))
plot(girf3) # Plot the GIRFs
```

---

in_paramspace *Determine whether the parameter vector is in the parameter space*

Description

in_paramspace checks whether the parameter vector is in the parameter space.

Usage

```
in_paramspace(
  p,
  M,
```

```
  d,
  params,
  weight_function = c("relative_dens", "logistic", "mlogit", "exponential", "threshold",
    "exogenous"),
  weightfun_pars = NULL,
  cond_dist = c("Gaussian", "Student", "ind_Student"),
  identification = c("reduced_form", "recursive", "heteroskedasticity",
    "non-Gaussianity"),
  B_constraints = NULL,
  other_constraints = NULL,
  all_boldA,
  all_Omegas,
  weightpars,
  distpars,
  transition_weights,
  stab_tol = 0.001,
  posdef_tol = 1e-08,
  distpar_tol = 1e-08,
  weightpar_tol = 1e-08
)
```

Arguments

p	a positive integer specifying the autoregressive order
M	a positive integer specifying the number of regimes
d	the number of time series in the system, i.e., the dimension
params	a real valued vector specifying the parameter values. Should have the form $\theta = (\phi_{1,0}, ..., \phi_{M,0}, \varphi_1, ..., \varphi_M, \sigma, \alpha, \nu)$, where (see exceptions below): • $\phi_{m,0}$ = the $(d \times 1)$ intercept (or mean) vector of the mth regime. • $\varphi_m = (vec(A_{m,1}), ..., vec(A_{m,p}))$ $(pd^2 \times 1)$. • **if** `cond_dist="Gaussian"` **or** `"Student"`**:** $\sigma = (vech(\Omega_1), ..., vech(\Omega_M))$ $(Md(d+1)/2 \times 1)$. **if** `cond_dist="ind_Student"`**:** $\sigma = (vec(B_1), ..., vec(B_M)$ $(Md^2 \times 1)$. • α = the $(a \times 1)$ vector containing the transition weight parameters (see below). • **if** `cond_dist = "Gaussian"`**):** Omit ν from the parameter vector. **if** `cond_dist="Student"`**:** $\nu > 2$ is the single degrees of freedom parameter. **if** `cond_dist="ind_Student"`**:** $\nu = (\nu_1, ..., \nu_M)$ $(M \times 1)$, $nu_m > 2$. For models with... `weight_function="relative_dens"`**:** $\alpha = (\alpha_1, ..., \alpha_{M-1})$ $(M-1 \times 1)$, where α_m $(1 \times 1), m = 1, ..., M-1$ are the transition weight parameters. `weight_function="logistic"`**:** $\alpha = (c, \gamma)$ (2×1), where $c \in \mathbb{R}$ is the location parameter and $\gamma > 0$ is the scale parameter.

`weight_function="mlogit"`: $\alpha = (\gamma_1, ..., \gamma_M)$ $((M-1)k \times 1)$, where γ_m $(k \times 1)$, $m = 1, ..., M-1$ contains the multinomial logit-regression coefficients of the mth regime. Specifically, for switching variables with indices in $I \subset \{1, ..., d\}$, and with $\tilde{p} \in \{1, ..., p\}$ lags included, γ_m contains the coefficients for the vector $z_{t-1} = (1, \tilde{z}_{\min\{I\}}, ..., \tilde{z}_{\max\{I\}})$, where $\tilde{z}_i = (y_{it-1}, ..., y_{it-\tilde{p}})$, $i \in I$. So $k = 1 + |I|\tilde{p}$ where $|I|$ denotes the number of elements in I.

`weight_function="exponential"`: $\alpha = (c, \gamma)$ (2×1), where $c \in \mathbb{R}$ is the location parameter and $\gamma > 0$ is the scale parameter.

`weight_function="threshold"`: $\alpha = (r_1, ..., r_{M-1})$ $(M - 1 \times 1)$, where $r_1, ..., r_{M-1}$ are the threshold values.

`weight_function="exogenous"`: Omit α from the parameter vector.

AR_constraints: Replace $\varphi_1, ..., \varphi_M$ with ψ as described in the argument `AR_constraints`.

mean_constraints: Replace $\phi_{1,0}, ..., \phi_{M,0}$ with $(\mu_1, ..., \mu_g)$ where μ_i, $(d \times 1)$ is the mean parameter for group i and g is the number of groups.

weight_constraints: If linear constraints are imposed, replace α with ξ as described in the argument `weigh_constraints`. If weight functions parameters are imposed to be fixed values, simply drop α from the parameter vector.

`identification="heteroskedasticity"`: $\sigma = (vec(W), \lambda_2, ..., \lambda_M)$, where W $(d \times d)$ and λ_m $(d \times 1)$, $m = 2, ..., M$, satisfy $\Omega_1 = WW'$ and $\Omega_m = W\Lambda_m W'$, $\Lambda_m = diag(\lambda_{m1}, ..., \lambda_{md})$, $\lambda_{mi} > 0$, $m = 2, ..., M$, $i = 1, ..., d$.

B_constraints (only for structural models identified by heteroskedasticity): Replace $vec(W)$ with $v\tilde{e}c(W)$ that stacks the columns of the matrix W in to vector so that the elements that are constrained to zero are not included.

Above, $\phi_{m,0}$ is the intercept parameter, $A_{m,i}$ denotes the ith coefficient matrix of the mth regime, Ω_m denotes the positive definite error term covariance matrix of the mth regime, and B_m is the invertible $(d \times d)$ impact matrix of the mth regime. ν_m is the degrees of freedom parameter of the mth regime. If `parametrization=="mean"`, just replace each $\phi_{m,0}$ with regimewise mean μ_m. $vec()$ is vectorization operator that stacks columns of a given matrix into a vector. $vech()$ stacks columns of a given matrix from the principal diagonal downwards (including elements on the diagonal) into a vector. $Bvec()$ is a vectorization operator that stacks the columns of a given impact matrix B_m into a vector so that the elements that are constrained to zero by the argument `B_constraints` are excluded.

`weight_function`

What type of transition weights $\alpha_{m,t}$ should be used?

`"relative_dens"`: $\alpha_{m,t} = \frac{\alpha_m f_{m,dp}(y_{t-1},...,y_{t-p+1})}{\sum_{n=1}^{M} \alpha_n f_{n,dp}(y_{t-1},...,y_{t-p+1})}$, where $\alpha_m \in (0, 1)$ are weight parameters that satisfy $\sum_{m=1}^{M} \alpha_m = 1$ and $f_{m,dp}(\cdot)$ is the dp-dimensional stationary density of the mth regime corresponding to p consecutive observations. Available for Gaussian conditional distribution only.

`"logistic"`: $M = 2$, $\alpha_{1,t} = 1 - \alpha_{2,t}$, and $\alpha_{2,t} = [1 + \exp\{-\gamma(y_{it-j} - c)\}]^{-1}$, where y_{it-j} is the lag j observation of the ith variable, c is a location parameter, and $\gamma > 0$ is a scale parameter.

"mlogit": $\alpha_{m,t} = \frac{\exp\{\gamma_m' z_{t-1}\}}{\sum_{n=1}^{M}\exp\{\gamma_n' z_{t-1}\}}$, where γ_m are coefficient vectors, $\gamma_M = 0$, and z_{t-1} $(k \times 1)$ is the vector containing a constant and the (lagged) switching variables.

"exponential": $M = 2$, $\alpha_{1,t} = 1 - \alpha_{2,t}$, and $\alpha_{2,t} = 1 - \exp\{-\gamma(y_{it-j} - c)\}$, where y_{it-j} is the lag j observation of the ith variable, c is a location parameter, and $\gamma > 0$ is a scale parameter.

"threshold": $\alpha_{m,t} = 1$ if $r_{m-1} < y_{it-j} \leq r_m$ and 0 otherwise, where $-\infty \equiv r_0 < r_1 < \cdots < r_{M-1} < r_M \equiv \infty$ are thresholds y_{it-j} is the lag j observation of the ith variable.

"exogenous": Exogenous nonrandom transition weights, specify the weight series in `weightfun_pars`.

See the vignette for more details about the weight functions.

`weightfun_pars` **If** `weight_function == "relative_dens"`: Not used.

If `weight_function %in% c("logistic", "exponential", "threshold")`: a numeric vector with the switching variable $i \in \{1, ..., d\}$ in the first and the lag $j \in \{1, ..., p\}$ in the second element.

If `weight_function == "mlogit"`: a list of two elements:

The first element `$vars`: a numeric vector containing the variables that should used as switching variables in the weight function in an increasing order, i.e., a vector with unique elements in $\{1, ..., d\}$.

The second element `$lags`: an integer in $\{1, ..., p\}$ specifying the number of lags to be used in the weight function.

If `weight_function == "exogenous"`: a size (`nrow(data) - p` x `M`) matrix containing the exogenous transition weights as `[t, m]` for time t and regime m. Each row needs to sum to one and only weakly positive values are allowed.

`cond_dist` specifies the conditional distribution of the model as `"Gaussian"`, `"Student"`, or `"ind_Student"`, where the latest is the Student's t distribution with independent components.

`identification` is it reduced form model or an identified structural model; if the latter, how is it identified (see the vignette or the references for details)?

"reduced_form": Reduced form model.

"recursive": The usual lower-triangular recursive identification of the shocks via their impact responses.

"heteroskedasticity": Identification by conditional heteroskedasticity, which imposes constant relative impact responses for each shock.

"non-Gaussianity": Identification by non-Gaussianity; requires mutually independent non-Gaussian shocks, thus, currently available only with the conditional distribution `"ind_Student"`.

`B_constraints` a $(d \times d)$ matrix with its entries imposing constraints on the impact matrix B_t: `NA` indicating that the element is unconstrained, a positive value indicating strict positive sign constraint, a negative value indicating strict negative sign constraint, and zero indicating that the element is constrained to zero. Currently only available for models with `identification="heteroskedasticity"` or `"non-Gaussianity"` due to the (in)availability of appropriate parametrizations that allow such constraints to be imposed.

other_constraints
: A list containing internally used additional type of constraints (see the options below).

  $fixed_lambdas (only if `identification="heteroskedasticity"`**):** a length $d(M-1)$ numeric vector $(\lambda_2, ..., \lambda_M)$ with elements strictly larger than zero specifying the fixed parameter values for the parameters λ_{mi} should be constrained to.

  $B1_constraints (only if `identification="non-Gaussianity"`**):** set to the string "fixed_sign_and_order" to impose the constraints that the elements of the first impact matrix B_1 are strictly positive and that they are in a decreasing order.

all_boldA
: 3D array containing the $((dp)x(dp))$ "bold A" (companion form) matrices of each regime, obtained from `form_boldA`. Will be computed if not given.

all_Omegas
: A 3D array containing the covariance matrix parameters obtain from `pick_Omegas`...

  If `cond_dist %in% c("Gaussian", "Student")`**:** all covariance matrices Ω_m in `[, , m]`.

  If `cond_dist=="ind_Student"`**:** all impact matrices B_m of the regimes in `[, , m]`.

weightpars
: numerical vector containing the transition weight parameters, obtained from `pick_weightpars`.

distpars
: A numeric vector containing the distribution parameters...

  If `cond_dist=="Gaussian"`**:** Not used, i.e., a numeric vector of length zero.

  If `cond_dist=="Student"`**:** The degrees of freedom parameter, i.e., a numeric vector of length one.

transition_weights
: (optional; only for models with `cond_dist="ind_Student"` or `identification="non-Gaussianity"`) A $T \times M$ matrix containing the transition weights. If `cond_dist="ind_Student"` checks that the impact matrix $\sum_{m=1}^{M} \alpha_{m,t}^{1/2} B_m$ is invertible for all $t = 1, ..., T$.

stab_tol
: numerical tolerance for stability of condition of the regimes: if the "bold A" matrix of any regime has eigenvalues larger that `1 - stat_tol` the parameter is considered to be outside the parameter space. Note that if tolerance is too small, numerical evaluation of the log-likelihood might fail and cause error.

posdef_tol
: numerical tolerance for positive definiteness of the error term covariance matrices: if the error term covariance matrix of any regime has eigenvalues smaller than this, the parameter is considered to be outside the parameter space. Note that if the tolerance is too small, numerical evaluation of the log-likelihood might fail and cause error.

distpar_tol
: the parameter vector is considered to be outside the parameter space if the degrees of freedom parameters is not larger than `2 + distpar_tol` (applies only if `cond_dist="Student"`).

weightpar_tol
: numerical tolerance for weight parameters being in the parameter space. Values closer to to the border of the parameter space than this are considered to be "outside" the parameter space.

Details

The parameter vector in the argument `params` should be unconstrained and reduced form.

Value

Returns `TRUE` if the given parameter values are in the parameter space and `FALSE` otherwise. This function does NOT consider identification conditions!

References

- Kheifets I.L., Saikkonen P.J. 2020. Stationarity and ergodicity of Vector STAR models. *Econometric Reviews*, **39**:4, 407-414.
- Lütkepohl H. 2005. New Introduction to Multiple Time Series Analysis, *Springer*.
- Lanne M., Virolainen S. 2024. A Gaussian smooth transition vector autoregressive model: An application to the macroeconomic effects of severe weather shocks. Unpublished working paper, available as arXiv:2403.14216.
- Virolainen S. 2024. Identification by non-Gaussianity in structural threshold and smooth transition vector autoregressive models. Unpublished working paper, available as arXiv:2404.19707.

@keywords internal

`iterate_more` *Maximum likelihood estimation of a reduced form or structural STVAR model based on preliminary estimates*

Description

`iterate_more` uses a variable metric algorithm to estimate a reduced form or structural STVAR model (object of class `'stvar'`) based on preliminary estimates.

Usage

```
iterate_more(stvar, maxit = 100, calc_std_errors = TRUE)
```

Arguments

`stvar`	an object of class `'stvar'`, created by, e.g., `fitSTVAR` or `fitSSTVAR`.
`maxit`	the maximum number of iterations in the variable metric algorithm.
`calc_std_errors`	should approximate standard errors be calculated?

Details

The purpose of `iterate_more` is to provide a simple and convenient tool to finalize the estimation when the maximum number of iterations is reached when estimating a STVAR model with the main estimation function `fitSTVAR` or `fitSSTVAR`.

Value

Returns an S3 object of class `'stvar'` defining a smooth transition VAR model. The returned list contains the following components (some of which may be `NULL` depending on the use case):

`data`	The input time series data.
`model`	A list describing the model structure.
`params`	The parameters of the model.
`std_errors`	Approximate standard errors of the parameters, if calculated.
`transition_weights`	The transition weights of the model.
`regime_cmeans`	Conditional means of the regimes, if data is provided.
`total_cmeans`	Total conditional means of the model, if data is provided.
`total_ccovs`	Total conditional covariances of the model, if data is provided.
`uncond_moments`	A list of unconditional moments including regime autocovariances, variances, and means.
`residuals_raw`	Raw residuals, if data is provided.
`residuals_std`	Standardized residuals, if data is provided.
`structural_shocks`	Recovered structural shocks, if applicable.
`loglik`	Log-likelihood of the model, if data is provided.
`IC`	The values of the information criteria (AIC, HQIC, BIC) for the model, if data is provided.
`all_estimates`	The parameter estimates from all estimation rounds, if applicable.
`all_logliks`	The log-likelihood of the estimates from all estimation rounds, if applicable.
`which_converged`	Indicators of which estimation rounds converged, if applicable.
`which_round`	Indicators of which round of optimization each estimate belongs to, if applicable.

References

- Anderson H., Vahid F. 1998. Testing multiple equation systems for common nonlinear components. *Journal of Econometrics*, **84**:1, 1-36.
- Hubrich K., Teräsvirta. T. 2013. Thresholds and Smooth Transitions in Vector Autoregressive Models. *CREATES Research Paper 2013-18, Aarhus University.*
- Lanne M., Virolainen S. 2024. A Gaussian smooth transition vector autoregressive model: An application to the macroeconomic effects of severe weather shocks. Unpublished working paper, available as arXiv:2403.14216.
- Kheifets I.L., Saikkonen P.J. 2020. Stationarity and ergodicity of Vector STAR models. *Econometric Reviews*, **39**:4, 407-414.
- Tsay R. 1998. Testing and Modeling Multivariate Threshold Models. *Journal of the American Statistical Association*, **93**:443, 1188-1202.
- Virolainen S. 2024. Identification by non-Gaussianity in structural threshold and smooth transition vector autoregressive models. Unpublished working paper, available as arXiv:2404.19707.

See Also

fitSTVAR, STVAR, optim, swap_B_signs, reorder_B_columns

Examples

```
## These are long running examples that take approximately 20 seconds to run.

# Estimate two-regime Gaussian STVAR p=1 model with the weighted relative stationary densities
# of the regimes as the transition weight function, but only 5 iterations of the variable matrix
# algorithm:
fit12 <- fitSTVAR(gdpdef, p=1, M=2, nrounds=1, seeds=1, ncores=1, maxit=5)

# The iteration limit was reached, so the estimate is not local maximum.
# The gradient of the log-likelihood function:
get_foc(fit12) # Not close to zero!

# So, we run more iterations of the variable metric algorithm:
fit12 <- iterate_more(fit12)

# The gradient of the log-likelihood function after iterating more:
get_foc(fit12) # Close to zero!
```

---

linear_IRF *Estimate linear impulse response function based on a single regime of a structural STVAR model.*

Description

linear_IRF estimates linear impulse response function based on a single regime of a structural STVAR model.

Usage

```
linear_IRF(
  stvar,
  N = 30,
  regime = 1,
  which_cumulative = numeric(0),
  scale = NULL,
  ci = NULL,
  bootstrap_reps = 100,
  ncores = 2,
  robust_method = c("Nelder-Mead", "SANN", "none"),
  maxit_robust = 1000,
  seed = NULL,
  ...
)
```

```
## S3 method for class 'irf'
plot(x, shocks_to_plot, ...)

## S3 method for class 'irf'
print(x, ..., digits = 2, N_to_print, shocks_to_print)
```

Arguments

stvar	an object of class 'stvar' defining a structural or reduced form STVAR model. For a reduced form model, the shocks are automatically identified by the lower triangular Cholesky decomposition.
N	a positive integer specifying the horizon how far ahead should the linear impulse responses be calculated.
regime	Based on which regime the linear IRF should be calculated? An integer in $1, ..., M$.
which_cumulative	a numeric vector with values in $1, ..., d$ (d=ncol(data)) specifying which the variables for which the linear impulse responses should be cumulative. Default is none.
scale	should the linear IRFs to some of the shocks be scaled so that they correspond to a specific instantaneous response of some specific variable? Provide a length three vector where the shock of interest is given in the first element (an integer in $1, ..., d$), the variable of interest is given in the second element (an integer in $1, ..., d$), and its instantaneous response in the third element (a non-zero real number). If the linear IRFs of multiple shocks should be scaled, provide a matrix which has one column for each of the shocks with the columns being the length three vectors described above.
ci	a real number in $(0, 1)$ specifying the confidence level of the confidence intervals calculated via a fixed-design wild residual bootstrap method. Available only for models that impose linear autoregressive dynamics (excluding changes in the volatility regime).
bootstrap_reps	the number of bootstrap repetitions for estimating confidence bounds.
ncores	the number of CPU cores to be used in parallel computing when bootstrapping confidence bounds.
robust_method	Should some robust estimation method be used in the estimation before switching to the gradient based variable metric algorithm? See details.
maxit_robust	the maximum number of iterations on the first phase robust estimation, if employed.
seed	a real number initializing the seed for the random generator.
...	currently not used.
x	object of class 'irf' generated by the function linear_IRF.
shocks_to_plot	IRFs of which shocks should be plotted? A numeric vector with elements in 1,...,d.
digits	the number of decimals to print

`N_to_print` an integer specifying the horizon how far to print the estimates and confidence intervals. The default is that all the values are printed.

`shocks_to_print`
the responses to which should should be printed? A numeric vector with elements in `1,...,d`. The default is that responses to all the shocks are printed.

Details

If the autoregressive dynamics of the model are linear (i.e., either M == 1 or mean and AR parameters are constrained identical across the regimes), confidence bounds can be calculated based on a fixed-design wild residual bootstrap method. We employ the method described in Herwartz and Lütkepohl (2014); see also the relevant chapters in Kilian and Lütkepohl (2017).

Employs the estimation function `optim` from the package `stats` that implements the optimization algorithms. The robust optimization method Nelder-Mead is much faster than SANN but can get stuck at a local solution. See `?optim` and the references therein for further details.

For model identified by non-Gaussianity, the signs and ordering of the shocks are normalized by assuming that the first non-zero element of each column of the impact matrix of Regime 1 is strictly positive and they are in a decreasing order. Use the argument `scale` to obtain IRFs scaled for specific impact responses.

Value

Returns a class `'irf'` list with with the following elements:

`$point_est`**:** a 3D array `[variables, shock, horizon]` containing the point estimates of the IRFs. Note that the first slice is for the impact responses and the slice i+1 for the period i. The response of the variable 'i1' to the shock 'i2' is subsetted as `$point_est[i1, i2, ]`.

`$conf_ints`**:** bootstrapped confidence intervals for the IRFs in a `[variables, shock, horizon, bound]` 4D array. The lower bound is obtained as `$conf_ints[, , , 1]`, and similarly the upper bound as `$conf_ints[, , , 2]`. The subsetted 3D array is then the bound in a form similar to `$point_est`.

`$all_bootstrap_reps`**:** IRFs from all of the bootstrap replications in a `[variables, shock, horizon, rep]`. 4D array. The IRF from replication i1 is obtained as `$all_bootstrap_reps[, , , i1]`, and the subsetted 3D array is then the in a form similar to `$point_est`.

Other elements: contains some of the arguments the `linear_IRF` was called with.

Functions

- `plot(irf)`: plot method
- `print(irf)`: print method

References

- Herwartz H. and Lütkepohl H. 2014. Structural vector autoregressions with Markov switching: Combining conventional with statistical identification of shocks. *Journal of Econometrics*, 183, pp. 104-116.
- Kilian L. and Lütkepohl H. 2017. Structural Vectors Autoregressive Analysis. *Cambridge University Press*, Cambridge.

See Also

GIRF, GFEVD, fitSTVAR, STVAR, reorder_B_columns, swap_B_signs

Examples

```
## These are long running examples that take approximately 10 seconds to run.
## A small number of bootstrap replications is used below to shorten the
## running time (in practice, a larger number of replications should be used).

# p=1, M=1, d=2, linear VAR model with independent Student's t shocks identified
# by non-Gaussianity (arbitrary weight function applied here):
theta_112it <- c(0.644, 0.065, 0.291, 0.021, -0.124, 0.884, 0.717, 0.105, 0.322,
  -0.25, 4.413, 3.912)
mod112 <- STVAR(data=gdpdef, p=1, M=1, params=theta_112it, cond_dist="ind_Student",
 identification="non-Gaussianity", weight_function="threshold", weightfun_pars=c(1, 1))
mod112 <- swap_B_signs(mod112, which_to_swap=1:2)

# Estimate IRFs 20 periods ahead, bootstrapped 90% confidence bounds based on
# 10 bootstrap replications. Linear model so robust estimation methods are
# not required.
irf1 <- linear_IRF(stvar=mod112, N=20, regime=1, ci=0.90, bootstrap_reps=1,
 robust_method="none", seed=1, ncores=1)
plot(irf1)
print(irf1, digits=3)

# p=1, M=2, d=2, Gaussian STVAR with relative dens weight function,
# shocks identified recursively.
theta_122relg <- c(0.734054, 0.225598, 0.705744, 0.187897, 0.259626, -0.000863,
  -0.3124, 0.505251, 0.298483, 0.030096, -0.176925, 0.838898, 0.310863, 0.007512,
  0.018244, 0.949533, -0.016941, 0.121403, 0.573269)
mod122 <- STVAR(data=gdpdef, p=1, M=2, params=theta_122relg, identification="recursive")

# Estimate IRF based on the first regime 30 period ahead. Scale IRFs so that
# the instantaneous response of the first variable to the first shock is 0.3,
# and the response of the second variable to the second shock is 0.5.
# response of the Confidence bounds
# are not available since the autoregressive dynamics are nonlinear.
irf2 <- linear_IRF(stvar=mod122, N=30, regime=1, scale=cbind(c(1, 1, 0.3), c(2, 2, 0.5)))
plot(irf2)

 # Estimate IRF based on the second regime without scaling the IRFs:
irf3 <- linear_IRF(stvar=mod122, N=30, regime=2)
plot(irf3)

# p=3, M=2, d=3, Students't logistic STVAR model with the first lag of the second
# variable as the switching variable. Autoregressive dynamics restricted linear,
# but the volatility regime varies in time, allowing the shocks to be identified
# by conditional heteroskedasticity.
theta_322 <- c(0.7575, 0.6675, 0.2634, 0.031, -0.007, 0.5468, 0.2508, 0.0217, -0.0356,
 0.171, -0.083, 0.0111, -0.1089, 0.1987, 0.2181, -0.1685, 0.5486, 0.0774, 5.9398, 3.6945,
 1.2216, 8.0716, 8.9718)
mod322 <- STVAR(data=gdpdef, p=3, M=2, params=theta_322, weight_function="logistic",
```

```
  weightfun_pars=c(2, 1), cond_dist="Student", mean_constraints=list(1:2),
  AR_constraints=rbind(diag(3*2^2), diag(3*2^2)), identification="heteroskedasticity",
  parametrization="mean")

## Estimate IRFs 30 periods ahead, bootstrapped 90% confidence bounds based on
# 10 bootstrap replications. Responses of the second variable are accumulated.
irf4 <- linear_IRF(stvar=mod322, N=30, regime=1, ci=0.90, bootstrap_reps=10,
 which_cumulative=2, seed=1)
plot(irf4)
```

`LR_test` *Perform likelihood ratio test for a STVAR model*

Description

`LR_test` performs a likelihood ratio test for a STVAR model

Usage

```
LR_test(stvar1, stvar2)
```

Arguments

`stvar1`	an object of class `'stvar'` generated by `fitSTVAR` or `STVAR`, containing the **freely estimated** model.
`stvar2`	an object of class `'stvar'` generated by `fitSTVAR` or `STVAR`, containing the **constrained** model.

Details

Performs a likelihood ratio test, testing the null hypothesis that the true parameter value lies in the constrained parameter space. Under the null, the test statistic is asymptotically χ^2-distributed with k degrees of freedom, k being the difference in the dimensions of the unconstrained and constrained parameter spaces.

The test is based on the assumption of the standard result of asymptotic normality! Also, note that this function does **not** verify that the two models are actually nested.

Value

A list with class "hypotest" containing the test results and arguments used to calculate the test.

References

- Buse A. (1982). The Likelihood Ratio, Wald, and Lagrange Multiplier Tests: An Expository Note. *The American Statistician*, 36(3a), 153-157.

See Also

Wald_test, Rao_test, fitSTVAR, STVAR, diagnostic_plot, profile_logliks, Portmanteau_test

Examples

```
# Logistic Student's t STVAR with p=1, M=2, and the first lag of the second variable
# as the switching variable (parameter values were obtained by maximum likelihood estimation;
 # fitSTVAR is not used here because the estimation is computationally demanding).
params12 <- c(0.62906848, 0.14245295, 2.41245785, 0.66719269, 0.3534745, 0.06041779, -0.34909745,
  0.61783824, 0.125769, -0.04094521, -0.99122586, 0.63805416, 0.371575, 0.00314754, 0.03440824,
   1.29072533, -0.06067807, 0.18737385, 1.21813844, 5.00884263, 7.70111672)
fit12 <- STVAR(data=gdpdef, p=1, M=2, params=params12, weight_function="logistic",
  weightfun_pars=c(2, 1), cond_dist="Student")
fit12

## Test whether the location parameter equals 1:

# Same as the original model but with the location parameter constrained to 1
# (parameter values were obtained by maximum likelihood estimation; fitSTVAR
# is not used here because the estimation is computationally demanding).
params12w <- c(0.6592583, 0.16162866, 1.7811393, 0.38876396, 0.35499367, 0.0576433,
 -0.43570508, 0.57337706, 0.16449607, -0.01910167, -0.70747014, 0.75386158, 0.3612087,
  0.00241419, 0.03202824, 1.07459924, -0.03432236, 0.14982445, 6.22717097, 8.18575651)
fit12w <- STVAR(data=gdpdef, p=1, M=2, params=params12w, weight_function="logistic",
 weightfun_pars=c(2, 1), cond_dist="Student",
 weight_constraints=list(R=matrix(c(0, 1), nrow=2), r=c(1, 0)))

# Test the null hypothesis of the location parameter equal 1:
LR_test(fit12, fit12w)

## Test whether the means and AR matrices are identical across the regimes:

# Same as the original model but with the mean and AR matrices constrained identical
# across the regimes (parameter values were obtained by maximum likelihood estimation;
# fitSTVAR is not used here because the estimation is computationally demanding).
params12cm <- c(0.76892423, 0.67128089, 0.30824474, 0.03530802, -0.11498402, 0.85942541,
 0.39106754, 0.0049437, 0.03897287, 1.44457723, -0.05939876, 0.20885008, 1.23568782,
 6.42128475, 7.28733557)
fit12cm <- STVAR(data=gdpdef, p=1, M=2, params=params12cm, weight_function="logistic",
 weightfun_pars=c(2, 1), parametrization="mean", cond_dist="Student",
 mean_constraints=list(1:2), AR_constraints=rbind(diag(4), diag(4)))

# Test the null hypothesis of the means and AR matrices being identical across the regimes:
LR_test(fit12, fit12cm)
```

`plot.stvarpred` *Predict method for class 'stvar' objects*

Description

`predict.stvar` is a predict method for class `'stvar'` objects.

Usage

```
## S3 method for class 'stvarpred'
plot(x, ..., nt, trans_weights = TRUE)

## S3 method for class 'stvar'
predict(
  object,
  ...,
  nsteps,
  nsim = 1000,
  pi = c(0.95, 0.8),
  pred_type = c("mean", "median"),
  exo_weights = NULL
)

## S3 method for class 'stvarpred'
print(x, ..., digits = 2)
```

Arguments

x	object of class 'stvarpred'
...	currently not used.
nt	a positive integer specifying the number of observations to be plotted along with the forecast.
trans_weights	should forecasts for transition weights be plotted?
object	an object of class 'stvar'.
nsteps	how many steps ahead should be predicted?
nsim	to how many independent simulations should the forecast be based on?
pi	a numeric vector specifying the confidence levels of the prediction intervals.
pred_type	should the pointforecast be based on sample "median" or "mean"?
exo_weights	if weight_function="exogenous", provide a size $(nstepsxM)$ matrix of exogenous transition weights for the regimes: [step, m] for $step$ steps ahead and regime m weight. Ignored if weight_function!="exogenous".
digits	the number of decimals to print

Details

The forecasts are computed by simulating multiple sample paths of the future observations and using the sample medians or means as point forecasts and empirical quantiles as prediction intervals.

Value

Returns a class 'stvarpred' object containing, among the specifications,...

$pred Point forecasts

$pred_ints Prediction intervals, as [, , d].

$trans_pred Point forecasts for the transition weights

$trans_pred_ints Individual prediction intervals for transition weights, as `[, , m]`, m=1,..,M.

Functions

- `plot(stvarpred)`: predict method
- `print(stvarpred)`: print method

References

- Anderson H., Vahid F. 1998. Testing multiple equation systems for common nonlinear components. *Journal of Econometrics*, **84**:1, 1-36.
- Kheifets I.L., Saikkonen P.J. 2020. Stationarity and ergodicity of Vector STAR models. *Econometric Reviews*, **39**:4, 407-414.
- Lanne M., Virolainen S. 2024. A Gaussian smooth transition vector autoregressive model: An application to the macroeconomic effects of severe weather shocks. Unpublished working paper, available as arXiv:2403.14216.
- Lütkepohl H. 2005. New Introduction to Multiple Time Series Analysis, *Springer*.
- McElroy T. 2017. Computation of vector ARMA autocovariances. *Statistics and Probability Letters*, **124**, 92-96.
- Kilian L., Lütkepohl H. 20017. Structural Vector Autoregressive Analysis. 1st edition. *Cambridge University Press*, Cambridge.
- Tsay R. 1998. Testing and Modeling Multivariate Threshold Models. *Journal of the American Statistical Association*, **93**:443, 1188-1202.
- Virolainen S. 2024. Identification by non-Gaussianity in structural threshold and smooth transition vector autoregressive models. Unpublished working paper, available as arXiv:2404.19707.

See Also

`simulate.stvar`

Examples

```
# p=2, M=2, d=2, Gaussian relative dens weights
theta_222relg <- c(0.356914, 0.107436, 0.356386, 0.08633, 0.13996, 0.035172,
  -0.164575, 0.386816, 0.451675, 0.013086, 0.227882, 0.336084, 0.239257, 0.024173,
  -0.021209, 0.707502, 0.063322, 0.027287, 0.009182, 0.197066, 0.205831, 0.005157,
  0.025877, 1.092094, -0.009327, 0.116449, 0.592446)
mod222relg <- STVAR(data=gdpdef, p=2, M=2, d=2, params=theta_222relg,
  weight_function="relative_dens")

# Predict 10 steps ahead, point forecast based on the conditional
# mean and 90% prediction intervals; prediction based on 100 sample paths:
pred1 <- predict(mod222relg, nsteps=10, nsim=100, pi=0.9, pred_type="mean")
pred1
plot(pred1)

# Predict 7 steps ahead, point forecast based on median and  90%, 80%,
```

```
# and 70% prediction intervals; prediction based on 80 sample paths:
pred2 <- predict(mod222relg, nsteps=7, nsim=80, pi=c(0.9, 0.8, 0.7),
 pred_type="median")
pred2
plot(pred2)
```

`Portmanteau_test` *Perform adjusted Portmanteau test for a STVAR model*

Description

`Portmanteau_test` performs adjusted Portmanteau test for remaining autocorrelation (or heteroskedasticity) in the residuals of a STVAR model.

Usage

```
Portmanteau_test(stvar, nlags = 20, which_test = c("autocorr", "het.sked"))
```

Arguments

`stvar`	an object of class `'stvar'` generated by `fitSTVAR` or `STVAR`.
`nlags`	a strictly positive integer specifying the number of lags to be tested.
`which_test`	should test for remaining autocorrelation or heteroskedasticity be calculated?

Details

The implemented adjusted Portmanteau test is based on Lütkepohl (2005), Section 4.4.3. When testing for remaining heteroskedasticity, the Portmanteau test is applied to squared standardized residuals that are centered to have zero mean. Note that the validity of the heteroskedasticity test requires that the residuals are not autocorrelated.

Value

A list with class "hypotest" containing the test results and arguments used to calculate the test.

References

- Lütkepohl H. 2005. New Introduction to Multiple Time Series Analysis, *Springer*.

See Also

`LR_test`, `Rao_test`, `fitSTVAR`, `STVAR`, `diagnostic_plot`, `profile_logliks`,

Examples

```
# Gaussian STVAR p=2, M=2, model with weighted relative stationary densities
# of the regimes as the transition weight function:
theta_222relg <- c(0.357, 0.107, 0.356, 0.086, 0.14, 0.035, -0.165, 0.387, 0.452,
 0.013, 0.228, 0.336, 0.239, 0.024, -0.021, 0.708, 0.063, 0.027, 0.009, 0.197,
 0.206, 0.005, 0.026, 1.092, -0.009, 0.116, 0.592)
mod222relg <- STVAR(data=gdpdef, p=2, M=2, d=2, params=theta_222relg,
 weight_function="relative_dens")

# Test for remaining autocorrelation taking into account the first 20 lags:
Portmanteau_test(mod222relg, nlags=20)

# Test for remaining heteroskedasticity taking into account the first 20 lags:
Portmanteau_test(mod222relg, nlags=20, which_test="het.sked")

# Two-regime Student's t Threhold VAR p=3 model with the first lag of the second
# variable as the switching variable:
theta_322thres <- c(0.527, 0.039, 1.922, 0.154, 0.284, 0.053, 0.033, 0.453, 0.291,
 0.024, -0.108, 0.153, -0.108, 0.003, -0.128, 0.219, 0.195, -0.03, -0.893, 0.686,
 0.047, 0.016, 0.524, 0.068, -0.025, 0.044, -0.435, 0.119, 0.359, 0.002, 0.038,
 1.252, -0.041, 0.151, 1.196, 12.312)
mod322thres <- STVAR(data=gdpdef, p=3, M=2, d=2, params=theta_322thres,
 weight_function="threshold", weightfun_pars=c(2, 1), cond_dist="Student")

# Test for remaining autocorrelation taking into account the first 25 lags:
Portmanteau_test(mod322thres, nlags=25)

# Test for remaining heteroskedasticity taking into account the first 25 lags:
Portmanteau_test(mod322thres, nlags=25, which_test="het.sked")
```

---

print.hypotest *Print method for the class hypotest*

Description

`print.hypotest` is the print method for the class hypotest objects.

Usage

```
## S3 method for class 'hypotest'
print(x, ..., digits = 4)
```

Arguments

`x`	object of class `'hypotest'` generated by the function `Wald_test`, `LR_test`, `Rao_test`, or `Portmenteau_test`.
`...`	currently not in use.
`digits`	how many significant digits to print?

Value

Returns the input object x invisibly.

`print.stvarsum` *Summary print method from objects of class 'stvarsum'*

Description

`print.stvarsum` is a print method for object `'stvarsum'` generated by `summary.stvar`.

Usage

```
## S3 method for class 'stvarsum'
print(x, ..., digits)
```

Arguments

x	object of class 'stvarsum' generated by `summary.stvar`.
...	currently not used.
digits	the number of digits to be printed.

Value

Returns the input object x invisibly.

`profile_logliks` *Plot profile log-likelihood functions about the estimates*

Description

`profile_logliks` plots profile log-likelihood functions about the estimates.

Usage

```
profile_logliks(
  stvar,
  which_pars,
  scale = 0.02,
  nrows,
  ncols,
  precision = 50,
  stab_tol = 0.001,
  posdef_tol = 1e-08,
  distpar_tol = 1e-08,
  weightpar_tol = 1e-08
)
```

Arguments

stvar	an object of class 'stvar', created by, e.g., fitSTVAR or fitSSTVAR.
which_pars	the profile log-likelihood function of which parameters should be plotted? An integer vector specifying the positions of the parameters in the parameter vector. The parameter vector has the form...
scale	a numeric scalar specifying the interval plotted for each estimate: the estimate plus-minus abs(scale*estimate).
nrows	how many rows should be in the plot-matrix? The default is max(ceiling(log2(length(which_pars)) - 1), 1).
ncols	how many columns should be in the plot-matrix? The default is ceiling(length(which_pars)/nrows). Note that nrows*ncols should not be smaller than the length of which_pars.
precision	at how many points should each profile log-likelihood function be evaluated at?
stab_tol	numerical tolerance for stability of condition of the regimes: if the "bold A" matrix of any regime has eigenvalues larger that 1 - stat_tol the parameter is considered to be outside the parameter space. Note that if tolerance is too small, numerical evaluation of the log-likelihood might fail and cause error.
posdef_tol	numerical tolerance for positive definiteness of the error term covariance matrices: if the error term covariance matrix of any regime has eigenvalues smaller than this, the parameter is considered to be outside the parameter space. Note that if the tolerance is too small, numerical evaluation of the log-likelihood might fail and cause error.
distpar_tol	the parameter vector is considered to be outside the parameter space if the degrees of freedom parameters is not larger than 2 + distpar_tol (applies only if cond_dist="Student").
weightpar_tol	numerical tolerance for weight parameters being in the parameter space. Values closer to to the border of the parameter space than this are considered to be "outside" the parameter space.

Details

When the number of parameters is large, it might be better to plot a smaller number of profile log-likelihood functions at a time using the argument which_pars.

The red vertical line points the estimate.

Value

Only plots to a graphical device and doesn't return anything.

References

- Anderson H., Vahid F. 1998. Testing multiple equation systems for common nonlinear components. *Journal of Econometrics*, **84**:1, 1-36.
- Kheifets I.L., Saikkonen P.J. 2020. Stationarity and ergodicity of Vector STAR models. *Econometric Reviews*, **39**:4, 407-414.

- Lanne M., Virolainen S. 2024. A Gaussian smooth transition vector autoregressive model: An application to the macroeconomic effects of severe weather shocks. Unpublished working paper, available as arXiv:2403.14216.
- Lütkepohl H. 2005. New Introduction to Multiple Time Series Analysis, *Springer*.
- McElroy T. 2017. Computation of vector ARMA autocovariances. *Statistics and Probability Letters*, **124**, 92-96.
- Kilian L., Lütkepohl H. 20017. Structural Vector Autoregressive Analysis. 1st edition. *Cambridge University Press*, Cambridge.
- Tsay R. 1998. Testing and Modeling Multivariate Threshold Models. *Journal of the American Statistical Association*, **93**:443, 1188-1202.
- Virolainen S. 2024. Identification by non-Gaussianity in structural threshold and smooth transition vector autoregressive models. Unpublished working paper, available as arXiv:2404.19707.

See Also

get_foc, get_soc, diagnostic_plot

Examples

```
# Threshold STVAR with p=1, M=2, the first lag of the second variable as switching variable:
pars <- c(0.5231, 0.1015, 1.9471, 0.3253, 0.3476, 0.0649, -0.035, 0.7513, 0.1651,
 -0.029, -0.7947, 0.7925, 0.4233, 5e-04, 0.0439, 1.2332, -0.0402, 0.1481, 1.2036)
mod12thres <- STVAR(data=gdpdef, p=1, M=2, params=pars, weight_function="threshold",
  weightfun_pars=c(2, 1))

# Plot the profile log-likelihood functions of all parameters:
profile_logliks(mod12thres, precision=50) # Plots fast with precision=50

# Plot only the profile log-likelihood function of the threshold parameter
# (which is the last parameter in the parameter vector):
profile_logliks(mod12thres, which_pars=length(pars), precision=100)

# Plot only the profile log-likelihood functions of the intercept parameters
# (which are the first four parameters in the parameter vector, as d=2 and M=2):
profile_logliks(mod12thres, which_pars=1:4, precision=100)
```

Rao_test *Perform Rao's score test for a STVAR model*

Description

Rao_test performs Rao's score test for a STVAR model

Usage

```
Rao_test(stvar)
```

Arguments

stvar	an object of class 'stvar' generated by fitSTVAR or STVAR, containing the model specified by the null hypothesis (i.e., **the constrained model**).

Details

Tests the constraints imposed in the model given in the argument stvar. This implementation uses the outer product of gradients approximation in the test statistic.

The test is based on the assumption of the standard result of asymptotic normality!

Value

A list with class "hypotest" containing the test results and arguments used to calculate the test.

References

- Buse A. (1982). The Likelihood Ratio, Wald, and Lagrange Multiplier Tests: An Expository Note. *The American Statistician*, 36(3a), 153-157.

See Also

LR_test, Wald_test, fitSTVAR, STVAR, diagnostic_plot, profile_logliks, Portmanteau_test

Examples

```
## These are long running examples that take approximately 10 seconds to run.

# Logistic Student's t STVAR with p=1, M=2, and the first lag of the second variable
# as the switching variable.

## Test whether the location parameter equal 1:

# The model imposing the constraint on the location parameter (parameter values
# were obtained by maximum likelihood estimation; fitSTVAR is not used here
# because the estimation is computationally demanding):
params12w <- c(0.6592583, 0.16162866, 1.7811393, 0.38876396, 0.35499367, 0.0576433,
  -0.43570508, 0.57337706, 0.16449607, -0.01910167, -0.70747014, 0.75386158, 0.3612087,
  0.00241419, 0.03202824, 1.07459924, -0.03432236, 0.14982445, 6.22717097, 8.18575651)
fit12w <- STVAR(data=gdpdef, p=1, M=2, params=params12w, weight_function="logistic",
 weightfun_pars=c(2, 1), cond_dist="Student",
 weight_constraints=list(R=matrix(c(0, 1), nrow=2), r=c(1, 0)))
fit12w

# Test the null hypothesis of the location parameter equal 1:
Rao_test(fit12w)

## Test whether the means and AR matrices are identical across the regimes:

# The model imposing the constraint on the location parameter (parameter values
# were obtained by maximum likelihood estimation; fitSTVAR is not used here
# because the estimation is computationally demanding):
```

```
params12cm <- c(0.76892423, 0.67128089, 0.30824474, 0.03530802, -0.11498402, 0.85942541,
 0.39106754, 0.0049437, 0.03897287, 1.44457723, -0.05939876, 0.20885008, 1.23568782,
 6.42128475, 7.28733557)
fit12cm <- STVAR(data=gdpdef, p=1, M=2, params=params12cm, weight_function="logistic",
 weightfun_pars=c(2, 1), parametrization="mean", cond_dist="Student",
 mean_constraints=list(1:2), AR_constraints=rbind(diag(4), diag(4)))

# Test the null hypothesis of the means and AR matrices being identical across the regimes:
Rao_test(fit12cm)
```

`redecompose_Omegas` *In the decomposition of the covariance matrices (Muirhead, 1982, Theorem A9.9), change the ordering of the covariance matrices.*

Description

`redecompose_Omegas` exchanges the order of the covariance matrices in the decomposition of Muirhead (1982, Theorem A9.9) and returns the new decomposition.

Usage

```
redecompose_Omegas(M, d, W, lambdas, perm = 1:M)
```

Arguments

`M`	the number of regimes in the model
`d`	the number of time series in the system
`W`	a length `d^2` vector containing the vectorized W matrix.
`lambdas`	a length `d*(M-1)` vector of the form $\lambda_2, ..., \lambda_M$ where $\lambda_m = (\lambda_{m1}, ..., \lambda_{md})$
`perm`	a vector of length `M` giving the new order of the covariance matrices (relative to the current order)

Details

We consider the following decomposition of positive definite covariannce matrices: $\Omega_1 = WW'$, $\Omega_m = W\Lambda_m W'$, $m = 2, .., M$ where $\Lambda_m = diag(\lambda_{m1}, ..., \lambda_{md})$ contains the strictly postive eigenvalues of $\Omega_m\Omega_1^{-1}$ and the column of the invertible W are the corresponding eigenvectors. Note that this decomposition does not necessarily exists for $M > 2$.

See Muirhead (1982), Theorem A9.9 for more details on the decomposition and the source code for more details on the reparametrization.

Value

Returns a $d^2 + (M-1)d \times 1$ vector of the form `c(vec(new_W), new_lambdas)` where the lambdas parameters are in the regimewise order (first regime 2, then 3, etc) and the "new W" and "new lambdas" are constitute the new decomposition with the order of the covariance matrices given by the argument `perm`. Notice that if the first element of `perm` is one, the W matrix will be the same and the lambdas are just re-ordered.

Note that unparametrized zero elements ARE present in the returned W!

Warning

No argument checks! Does not work with dimension $d = 1$ or with only one mixture component $M = 1$.

References

- Muirhead R.J. 1982. Aspects of Multivariate Statistical Theory, *Wiley*.

Examples

```
# Create two (2x2) coviance matrices:
d <- 2 # The dimension
M <- 2 # The number of covariance matrices
Omega1 <- matrix(c(2, 0.5, 0.5, 2), nrow=d)
Omega2 <- matrix(c(1, -0.2, -0.2, 1), nrow=d)

# The decomposition with Omega1 as the first covariance matrix:
decomp1 <- diag_Omegas(Omega1, Omega2)
W <- matrix(decomp1[1:d^2], nrow=d, ncol=d) # Recover W
lambdas <- decomp1[(d^2 + 1):length(decomp1)] # Recover lambdas
tcrossprod(W) # = Omega1
W%*%tcrossprod(diag(lambdas), W) # = Omega2

# Reorder the covariance matrices in the decomposition so that now
# the first covariance matrix is Omega2:
decomp2 <- redecompose_Omegas(M=M, d=d, W=as.vector(W), lambdas=lambdas,
                              perm=2:1)
new_W <- matrix(decomp2[1:d^2], nrow=d, ncol=d) # Recover W
new_lambdas <- decomp2[(d^2 + 1):length(decomp2)] # Recover lambdas
tcrossprod(new_W) # = Omega2
new_W%*%tcrossprod(diag(new_lambdas), new_W) # = Omega1
```

`reorder_B_columns` *Reorder columns of impact matrix B (and lambda parameters if any) of a structural STVAR model that is identified by heteroskedasticity or non-Gaussianity.*

Description

`reorder_B_columns` reorder columns of impact matrix B (and lambda parameters if any) of a structural STVAR model that is identified by heteroskedasticity or non-Gaussianity.

Usage

```
reorder_B_columns(stvar, perm, calc_std_errors = FALSE)
```

Arguments

`stvar`	a class 'stvar' object defining a structural STVAR model that is identified by heteroskedasticity or non-Gaussianity, typically created with `fitSSTVAR`.
`perm`	an integer vector of length d specifying the new order of the columns of the impact matrix. For model identified by...
	heteroskedasticity also lambda parameters of each regime will be reordered accordingly.
	non-Gaussianity the columns of the impact matrices of all the regimes and the component specific distribution parameters (degrees of freedom parameters) are reordered accordingly.
`calc_std_errors`	should approximate standard errors be calculated?

Details

The order of the columns of the impact matrix can be changed without changing the implied reduced form model (as long as, for models identified by heteroskedasticity, the order of lambda parameters is also changed accordingly; and for model identified by non-Gaussianity, ordering of the columns of all the impact matrices and the component specific distribution parameters is also changed accordingly). Note that constraints imposed on the impact matrix via `B_constraints` will also be modified accordingly.

Also all signs in any column of impact matrix can be swapped (without changing the implied reduced form model) with the function `swap_B_signs`. This obviously also swaps the sign constraints (if any) in the corresponding columns of the impact matrix.

Value

Returns an S3 object of class `'stvar'` defining a smooth transition VAR model. The returned list contains the following components (some of which may be `NULL` depending on the use case):

`data`	The input time series data.
`model`	A list describing the model structure.
`params`	The parameters of the model.
`std_errors`	Approximate standard errors of the parameters, if calculated.
`transition_weights`	The transition weights of the model.
`regime_cmeans`	Conditional means of the regimes, if data is provided.

total_cmeans	Total conditional means of the model, if data is provided.
total_ccovs	Total conditional covariances of the model, if data is provided.
uncond_moments	A list of unconditional moments including regime autocovariances, variances, and means.
residuals_raw	Raw residuals, if data is provided.
residuals_std	Standardized residuals, if data is provided.
structural_shocks	Recovered structural shocks, if applicable.
loglik	Log-likelihood of the model, if data is provided.
IC	The values of the information criteria (AIC, HQIC, BIC) for the model, if data is provided.
all_estimates	The parameter estimates from all estimation rounds, if applicable.
all_logliks	The log-likelihood of the estimates from all estimation rounds, if applicable.
which_converged	Indicators of which estimation rounds converged, if applicable.
which_round	Indicators of which round of optimization each estimate belongs to, if applicable.

References

- Lütkepohl H., Netšunajev A. 2018. Structural vector autoregressions with smooth transition in variances. *Journal of Economic Dynamics & Control*, **84**, 43-57.

See Also

GIRF, fitSSTVAR, swap_B_signs

Examples

```
# Create a structural two-variate Student's t STVAR p=2, M=2 model with logistic transition
# weights and the first lag of the second variable  as the switching variable, and shocks
# identified by heteroskedasticity:
theta_222logt <- c(0.356914, 0.107436, 0.356386, 0.086330, 0.139960, 0.035172, -0.164575,
 0.386816, 0.451675, 0.013086, 0.227882, 0.336084, 0.239257, 0.024173, -0.021209, 0.707502,
 0.063322, 0.027287, 0.009182, 0.197066, -0.03, 0.24, -0.76, -0.02, 3.36, 0.86, 0.1, 0.2, 7)
mod222logt <- STVAR(p=2, M=2, d=2, params=theta_222logt, weight_function="logistic",
  weightfun_pars=c(2, 1), cond_dist="Student", identification="heteroskedasticity")

# Print the parameter values, W and lambdas are printed in the bottom:
mod222logt

# Reverse the ordering of the columns of W (or equally the impact matrix):
mod222logt_rev <- reorder_B_columns(mod222logt, perm=c(2, 1))
mod222logt_rev # The columns of the impact matrix are in a reversed order

# Swap the ordering of the columns of the impact matrix back to the original:
mod222logt_rev2 <- reorder_B_columns(mod222logt_rev, perm=c(2, 1))
```

```
mod222logt_rev2 # The columns of the impact matrix are back in the original ordering

# Below code does not do anything, as perm=1:2, so the ordering does not change:
mod222logt3 <- reorder_B_columns(mod222logt, perm=c(1, 2))
mod222logt3 # The ordering of the columns did not change from the original
```

`simulate.stvar` *Simulate method for class 'stvar' objects*

Description

`simulate.stvar` is a simulate method for class 'stvar' objects.

Usage

```
## S3 method for class 'stvar'
simulate(
  object,
  nsim = 1,
  seed = NULL,
  ...,
  init_values = NULL,
  init_regime,
  ntimes = 1,
  burn_in = 1000,
  exo_weights = NULL,
  drop = TRUE,
  girf_pars = NULL
)
```

Arguments

`object`	an object of class `'stvar'`.
`nsim`	number of observations to be simulated.
`seed`	set seed for the random number generator?
`...`	currently not in use.
`init_values`	a size $(p \times d)$ matrix specifying the initial values, where d is the number of time series in the system. The **last** row will be used as initial values for the first lag, the second last row for second lag etc. If not specified, initial values will be drawn from the regime specified in `init_regimes` (for Gaussian models only).
`init_regime`	an integer in $1, ..., M$ specifying the regime from which the initial values should be generated from. The initial values will be generated from the stationary distribution of the specific regime. Due to the lack of knowledge of the stationary distribution, models with other than Gaussian conditional distribution uses a simulation procedure with a burn-in period. See the details section.
`ntimes`	how many sets of simulations should be performed?

`burn_in` Burn-in period for simulating initial values from a regime when `cond_dist!="Gaussian"`. See the details section.

`exo_weights` if `weight_function="exogenous"`, provide a size $(nsim \times M)$ matrix of exogenous transition weights for the regimes: `[t, m]` for a time t and regime m weight. Ignored if `weight_function!="exogenous"`.

`drop` if `TRUE` (default) then the components of the returned list are coerced to lower dimension if `ntimes==1`, i.e., `$sample` and `$transition_weights` will be matrices, and `$component` will be vector.

`girf_pars` This argument is used internally in the estimation of generalized impulse response functions (see `?GIRF`). You should ignore it (specifying something else than null to it will change how the function behaves).

Details

The stationary distribution of each regime is not known when `cond_dist!="Gaussian"`. Therefore, when using `init_regime` to simulate the initial values from a given regime, we employ the following simulation procedure to obtain the initial values. First, we set the initial values to the unconditional mean of the specified regime. Then, we simulate a large number observations from that regime as specified in the argument `burn_in`. Then, we simulate $p + 100$ observations more after the burn in period, and for the 100 observations calculate the transition weights for them and take the consecutive p observations that yield the highest transition weight for the given regime. For models with exogenous transition weights, takes just the last p observations after the burn-in period.

The argument `ntimes` is intended for forecasting, which is used by the predict method (see `?predict.stvar`).

Value

Returns a list containing the simulation results. If `drop==TRUE` and `ntimes==1` (default), contains the following entries:

`sample` a size (`nsim`$\times d$) matrix containing the simulated time series.

`transition weights:`
a size (`nsim`$\times M$) matrix containing the transition weights corresponding to the simulated sample.

Otherwise, returns a list with the following entries:

`$sample` a size (`nsim`$\times d \times$`ntimes`) array containing the samples: the dimension `[t, , ]` is the time index, the dimension `[, d, ]` indicates the marginal time series, and the dimension `[, , i]` indicates the i:th set of simulations.

`$transition_weights`
a size (`nsim`$\times M \times$`ntimes`) array containing the transition weights corresponding to the sample: the dimension `[t, , ]` is the time index, the dimension `[, m, ]` indicates the regime, and the dimension `[, , i]` indicates the i:th set of simulations.

References

- Anderson H., Vahid F. 1998. Testing multiple equation systems for common nonlinear components. *Journal of Econometrics*, **84**:1, 1-36.

- Kheifets I.L., Saikkonen P.J. 2020. Stationarity and ergodicity of Vector STAR models. *Econometric Reviews*, **39**:4, 407-414.
- Lanne M., Virolainen S. 2024. A Gaussian smooth transition vector autoregressive model: An application to the macroeconomic effects of severe weather shocks. Unpublished working paper, available as arXiv:2403.14216.
- Lütkepohl H. 2005. New Introduction to Multiple Time Series Analysis, *Springer*.
- McElroy T. 2017. Computation of vector ARMA autocovariances. *Statistics and Probability Letters*, **124**, 92-96.
- Kilian L., Lütkepohl H. 20017. Structural Vector Autoregressive Analysis. 1st edition. *Cambridge University Press*, Cambridge.
- Tsay R. 1998. Testing and Modeling Multivariate Threshold Models. *Journal of the American Statistical Association*, **93**:443, 1188-1202.
- Virolainen S. 2024. Identification by non-Gaussianity in structural threshold and smooth transition vector autoregressive models. Unpublished working paper, available as arXiv:2404.19707.

See Also

predict.stvar,GIRF, GFEVD, fitSTVAR, fitSSTVAR STVAR

Examples

```
 # Gaussian STVAR(p=2, M=2) model with weighted relative stationary densities
 # of the regimes as the transition weight function:
 theta_222relg <- c(0.356914, 0.107436, 0.356386, 0.08633, 0.13996, 0.035172,
   -0.164575, 0.386816, 0.451675, 0.013086, 0.227882, 0.336084, 0.239257, 0.024173,
   -0.021209, 0.707502, 0.063322, 0.027287, 0.009182, 0.197066, 0.205831, 0.005157,
   0.025877, 1.092094, -0.009327, 0.116449, 0.592446)
 mod222relg <- STVAR(data=gdpdef, p=2, M=2, d=2, params=theta_222relg,
   weight_function="relative_dens")

 # Simulate T=200 observations using given initial values:
 init_vals <- matrix(c(0.5, 1.0, 0.5, 1), nrow=2)
 sim1 <- simulate(mod222relg, nsim=200, seed=1, init_values=init_vals)
 plot.ts(sim1$sample) # Sample
 plot.ts(sim1$transition_weights) # Transition weights

 # Simulate T=100 observations, with initial values drawn from the stationary
 # distribution of the 1st regime:
 sim2 <- simulate(mod222relg, nsim=200, seed=1, init_regime=1)
 plot.ts(sim2$sample) # Sample
 plot.ts(sim2$transition_weights) # Transition weights

# Logistic Student's t STVAR with p=1, M=2, and the first lag of the second variable
# as the switching variable.
params12 <- c(0.62906848, 0.14245295, 2.41245785, 0.66719269, 0.3534745, 0.06041779, -0.34909745,
  0.61783824, 0.125769, -0.04094521, -0.99122586, 0.63805416, 0.371575, 0.00314754, 0.03440824,
  1.29072533, -0.06067807, 0.18737385, 1.21813844, 5.00884263, 7.70111672)
fit12 <- STVAR(data=gdpdef, p=1, M=2, params=params12, weight_function="logistic",
  weightfun_pars=c(2, 1), cond_dist="Student")
```

```
# Simulate T=100 observations with initial values drawn from the second regime.
# Since the stationary distribution of the Student's regime is not known, we
# use a simulation procedure that starts from the unconditional mean of the regime,
# then simulates a number of observations from the regime for a "burn-in" period,
# and finally takes the last p observations generated from the regime as the initial
# values for the simulation from the STVAR model:
sim3 <- simulate(fit12, nsim=100, init_regime=1, burn_in=1000)
plot.ts(sim3$sample) # Sample
plot.ts(sim3$transition_weights) # Transition weights
```

STVAR *Create a class 'stvar' object defining a reduced form or structural smooth transition VAR model*

Description

STVAR creates a class 'stvar' object that defines a reduced form or structural smooth transition VAR model

Usage

```
STVAR(
  data,
  p,
  M,
  d,
  params,
 weight_function = c("relative_dens", "logistic", "mlogit", "exponential", "threshold",
    "exogenous"),
  weightfun_pars = NULL,
  cond_dist = c("Gaussian", "Student", "ind_Student"),
  parametrization = c("intercept", "mean"),
  identification = c("reduced_form", "recursive", "heteroskedasticity",
    "non-Gaussianity"),
  AR_constraints = NULL,
  mean_constraints = NULL,
  weight_constraints = NULL,
  B_constraints = NULL,
  calc_std_errors = FALSE
)

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

## S3 method for class 'stvar'
residuals(object, ...)

## S3 method for class 'stvar'
```

```
summary(object, ..., digits = 2)

## S3 method for class 'stvar'
plot(x, ..., plot_type = c("trans_weights", "cond_mean"))

## S3 method for class 'stvar'
print(x, ..., digits = 2, summary_print = FALSE, standard_error_print = FALSE)
```

Arguments

`data`	a matrix or class `'ts'` object with `d>1` columns. Each column is taken to represent a single times series. `NA` values are not supported. Ignore if defining a model without data is desired.
`p`	a positive integer specifying the autoregressive order
`M`	a positive integer specifying the number of regimes
`d`	number of times series in the system, i.e. `ncol(data)`. This can be used to define STVAR models without data and can be ignored if `data` is provided.
`params`	a real valued vector specifying the parameter values. Should have the form $\theta = (\phi_{1,0}, ..., \phi_{M,0}, \varphi_1, ..., \varphi_M, \sigma, \alpha, \nu)$, where (see exceptions below): • $\phi_{m,0}$ = the $(d \times 1)$ intercept (or mean) vector of the mth regime. • $\varphi_m = (vec(A_{m,1}), ..., vec(A_{m,p}))$ $(pd^2 \times 1)$. • **if** `cond_dist="Gaussian"` **or** `"Student"`**:** $\sigma = (vech(\Omega_1), ..., vech(\Omega_M))$ $(Md(d+1)/2 \times 1)$. **if** `cond_dist="ind_Student"`**:** $\sigma = (vec(B_1), ..., vec(B_M)$ $(Md^2 \times 1)$. • α = the $(a \times 1)$ vector containing the transition weight parameters (see below). • **if** `cond_dist = "Gaussian"`**):** Omit ν from the parameter vector. **if** `cond_dist="Student"`**:** $\nu > 2$ is the single degrees of freedom parameter. **if** `cond_dist="ind_Student"`**:** $\nu = (\nu_1, ..., \nu_M)$ $(M \times 1)$, $nu_m > 2$. For models with... `weight_function="relative_dens"`**:** $\alpha = (\alpha_1, ..., \alpha_{M-1})$ $(M-1 \times 1)$, where α_m $(1 \times 1), m = 1, ..., M-1$ are the transition weight parameters. `weight_function="logistic"`**:** $\alpha = (c, \gamma)$ (2×1), where $c \in \mathbb{R}$ is the location parameter and $\gamma > 0$ is the scale parameter. `weight_function="mlogit"`**:** $\alpha = (\gamma_1, ..., \gamma_M)$ $((M-1)k \times 1)$, where γ_m $(k \times 1)$, $m = 1, ..., M-1$ contains the multinomial logit-regression coefficients of the mth regime. Specifically, for switching variables with indices in $I \subset \{1, ..., d\}$, and with $\tilde{p} \in \{1, ..., p\}$ lags included, γ_m contains the coefficients for the vector $z_{t-1} = (1, \tilde{z}_{\min\{I\}}, ..., \tilde{z}_{\max\{I\}})$, where $\tilde{z}_i = (y_{it-1}, ..., y_{it-\tilde{p}})$, $i \in I$. So $k = 1 + |I|\tilde{p}$ where $|I|$ denotes the number of elements in I. `weight_function="exponential"`**:** $\alpha = (c, \gamma)$ (2×1), where $c \in \mathbb{R}$ is the location parameter and $\gamma > 0$ is the scale parameter.

`weight_function="threshold"`: $\alpha = (r_1, ..., r_{M-1})$ $(M - 1 \times 1)$, where $r_1, ..., r_{M-1}$ are the threshold values.

`weight_function="exogenous"`: Omit α from the parameter vector.

AR_constraints: Replace $\varphi_1, ..., \varphi_M$ with ψ as described in the argument `AR_constraints`.

mean_constraints: Replace $\phi_{1,0}, ..., \phi_{M,0}$ with $(\mu_1, ..., \mu_g)$ where μ_i, $(d \times 1)$ is the mean parameter for group i and g is the number of groups.

weight_constraints: If linear constraints are imposed, replace α with ξ as described in the argument `weigh_constraints`. If weight functions parameters are imposed to be fixed values, simply drop α from the parameter vector.

`identification="heteroskedasticity"`: $\sigma = (vec(W), \lambda_2, ..., \lambda_M)$, where W $(d \times d)$ and λ_m $(d \times 1)$, $m = 2, ..., M$, satisfy $\Omega_1 = WW'$ and $\Omega_m = W\Lambda_m W'$, $\Lambda_m = diag(\lambda_{m1}, ..., \lambda_{md})$, $\lambda_{mi} > 0$, $m = 2, ..., M$, $i = 1, ..., d$.

B_constraints (only for structural models identified by heteroskedasticity): Replace $vec(W)$ with $v\tilde{e}c(W)$ that stacks the columns of the matrix W in to vector so that the elements that are constrained to zero are not included.

Above, $\phi_{m,0}$ is the intercept parameter, $A_{m,i}$ denotes the ith coefficient matrix of the mth regime, Ω_m denotes the positive definite error term covariance matrix of the mth regime, and B_m is the invertible $(d \times d)$ impact matrix of the mth regime. ν_m is the degrees of freedom parameter of the mth regime. If `parametrization=="mean"`, just replace each $\phi_{m,0}$ with regimewise mean μ_m. $vec()$ is vectorization operator that stacks columns of a given matrix into a vector. $vech()$ stacks columns of a given matrix from the principal diagonal downwards (including elements on the diagonal) into a vector. $Bvec()$ is a vectorization operator that stacks the columns of a given impact matrix B_m into a vector so that the elements that are constrained to zero by the argument `B_constraints` are excluded.

`weight_function`

What type of transition weights $\alpha_{m,t}$ should be used?

`"relative_dens"`: $\alpha_{m,t} = \frac{\alpha_m f_{m,dp}(y_{t-1},...,y_{t-p+1})}{\sum_{n=1}^{M} \alpha_n f_{n,dp}(y_{t-1},...,y_{t-p+1})}$, where $\alpha_m \in (0, 1)$ are weight parameters that satisfy $\sum_{m=1}^{M} \alpha_m = 1$ and $f_{m,dp}(\cdot)$ is the dp-dimensional stationary density of the mth regime corresponding to p consecutive observations. Available for Gaussian conditional distribution only.

`"logistic"`: $M = 2$, $\alpha_{1,t} = 1 - \alpha_{2,t}$, and $\alpha_{2,t} = [1 + \exp\{-\gamma(y_{it-j} - c)\}]^{-1}$, where y_{it-j} is the lag j observation of the ith variable, c is a location parameter, and $\gamma > 0$ is a scale parameter.

`"mlogit"`: $\alpha_{m,t} = \frac{\exp\{\gamma_m' z_{t-1}\}}{\sum_{n=1}^{M} \exp\{\gamma_n' z_{t-1}\}}$, where γ_m are coefficient vectors, $\gamma_M = 0$, and z_{t-1} $(k \times 1)$ is the vector containing a constant and the (lagged) switching variables.

`"exponential"`: $M = 2$, $\alpha_{1,t} = 1 - \alpha_{2,t}$, and $\alpha_{2,t} = 1 - \exp\{-\gamma(y_{it-j} - c)\}$, where y_{it-j} is the lag j observation of the ith variable, c is a location parameter, and $\gamma > 0$ is a scale parameter.

`"threshold"`: $\alpha_{m,t} = 1$ if $r_{m-1} < y_{it-j} \leq r_m$ and 0 otherwise, where $-\infty \equiv r_0 < r_1 < \cdots < r_{M-1} < r_M \equiv \infty$ are thresholds y_{it-j} is the lag j observation of the ith variable.

"exogenous": Exogenous nonrandom transition weights, specify the weight series in weightfun_pars.

See the vignette for more details about the weight functions.

weightfun_pars **If** weight_function == "relative_dens": Not used.

If weight_function %in% c("logistic", "exponential", "threshold"): a numeric vector with the switching variable $i \in \{1, ..., d\}$ in the first and the lag $j \in \{1, ..., p\}$ in the second element.

If weight_function == "mlogit": a list of two elements:

The first element $vars: a numeric vector containing the variables that should used as switching variables in the weight function in an increasing order, i.e., a vector with unique elements in $\{1, ..., d\}$.

The second element $lags: an integer in $\{1, ..., p\}$ specifying the number of lags to be used in the weight function.

If weight_function == "exogenous": a size (nrow(data) - p x M) matrix containing the exogenous transition weights as [t, m] for time t and regime m. Each row needs to sum to one and only weakly positive values are allowed.

cond_dist specifies the conditional distribution of the model as "Gaussian", "Student", or "ind_Student", where the latest is the Student's t distribution with independent components.

parametrization

"intercept" or "mean" determining whether the model is parametrized with intercept parameters $\phi_{m,0}$ or regime means μ_m, m=1,...,M.

identification is it reduced form model or an identified structural model; if the latter, how is it identified (see the vignette or the references for details)?

"reduced_form": Reduced form model.

"recursive": The usual lower-triangular recursive identification of the shocks via their impact responses.

"heteroskedasticity": Identification by conditional heteroskedasticity, which imposes constant relative impact responses for each shock.

"non-Gaussianity": Identification by non-Gaussianity; requires mutually independent non-Gaussian shocks, thus, currently available only with the conditional distribution "ind_Student".

AR_constraints a size (Mpd^2xq) constraint matrix C specifying linear constraints to the autoregressive parameters. The constraints are of the form $(\varphi_1, ..., \varphi_M) = C\psi$, where $\varphi_m = (vec(A_{m,1}), ..., vec(A_{m,p}))$ (pd^2x1), $m = 1, ..., M$, contains the coefficient matrices and ψ $(qx1)$ contains the related parameters. For example, to restrict the AR-parameters to be the identical across the regimes, set C = [I:...:I]' (Mpd^2xpd^2) where I = diag(p*d^2).

mean_constraints

Restrict the mean parameters of some regimes to be identical? Provide a list of numeric vectors such that each numeric vector contains the regimes that should share the common mean parameters. For instance, if M=3, the argument list(1, 2:3) restricts the mean parameters of the second and third regime to be identical but the first regime has freely estimated (unconditional) mean. Ignore or set to NULL if mean parameters should not be restricted to be the same among any

regimes. This constraint is available only for mean parametrized models; that is, when `parametrization="mean"`.

`weight_constraints`
: a list of two elements, R in the first element and r in the second element, specifying linear constraints on the transition weight parameters α. The constraints are of the form $\alpha = R\xi + r$, where R is a known $(a \times l)$ constraint matrix of full column rank (a is the dimension of α), r is a known $(a \times 1)$ constant, and ξ is an unknown $(l \times 1)$ parameter. **Alternatively**, set $R = 0$ in order to constrain the the weight parameter to the constant r (in this case, α is dropped from the constrained parameter vector).

`B_constraints`
: a $(d \times d)$ matrix with its entries imposing constraints on the impact matrix B_t: `NA` indicating that the element is unconstrained, a positive value indicating strict positive sign constraint, a negative value indicating strict negative sign constraint, and zero indicating that the element is constrained to zero. Currently only available for models with `identification="heteroskedasticity"` or `"non-Gaussianity"` due to the (in)availability of appropriate parametrizations that allow such constraints to be imposed.

`calc_std_errors`
: should approximate standard errors be calculated?

`object`
: object of class `'stvar'`.

`...`
: currently not used.

`digits`
: number of digits to be printed.

`x`
: an object of class `'stvar'`.

`plot_type`
: should the series be plotted with the estimated transition weights or conditional means?

`summary_print`
: if set to `TRUE` then the print will include log-likelihood and information criteria values.

`standard_error_print`
: if set to `TRUE`, instead of printing the estimates, prints the approximate standard errors using square roots of the diagonal of inverse of the observed information matrix.

Details

If data is provided, then also residuals are computed and included in the returned object.

The plot displays the time series together with estimated transition weights.

Value

Returns an S3 object of class `'stvar'` defining a smooth transition VAR model. The returned list contains the following components (some of which may be `NULL` depending on the use case):

`data`
: The input time series data.

`model`
: A list describing the model structure.

`params`
: The parameters of the model.

`std_errors`
: Approximate standard errors of the parameters, if calculated.

`transition_weights`
: The transition weights of the model.

`regime_cmeans`
: Conditional means of the regimes, if data is provided.

`total_cmeans`
: Total conditional means of the model, if data is provided.

`total_ccovs`
: Total conditional covariances of the model, if data is provided.

`uncond_moments`
: A list of unconditional moments including regime autocovariances, variances, and means.

`residuals_raw`
: Raw residuals, if data is provided.

`residuals_std`
: Standardized residuals, if data is provided.

`structural_shocks`
: Recovered structural shocks, if applicable.

`loglik`
: Log-likelihood of the model, if data is provided.

`IC`
: The values of the information criteria (AIC, HQIC, BIC) for the model, if data is provided.

`all_estimates`
: The parameter estimates from all estimation rounds, if applicable.

`all_logliks`
: The log-likelihood of the estimates from all estimation rounds, if applicable.

`which_converged`
: Indicators of which estimation rounds converged, if applicable.

`which_round`
: Indicators of which round of optimization each estimate belongs to, if applicable.

Functions

- `logLik(stvar)`: Log-likelihood method
- `residuals(stvar)`: residuals method to extract Pearson residuals
- `summary(stvar)`: summary method
- `plot(stvar)`: plot method for class 'stvar'
- `print(stvar)`: print method

About S3 methods

If data is not provided, only the `print` and `simulate` methods are available. If data is provided, then in addition to the ones listed above, `predict` method is also available. See `?simulate.stvar` and `?predict.stvar` for details about the usage.

References

- Anderson H., Vahid F. 1998. Testing multiple equation systems for common nonlinear components. *Journal of Econometrics*, **84**:1, 1-36.
- Hubrich K., Teräsvirta. T. 2013. Thresholds and Smooth Transitions in Vector Autoregressive Models. *CREATES Research Paper 2013-18, Aarhus University.*
- Lanne M., Virolainen S. 2024. A Gaussian smooth transition vector autoregressive model: An application to the macroeconomic effects of severe weather shocks. Unpublished working paper, available as arXiv:2403.14216.

- Kheifets I.L., Saikkonen P.J. 2020. Stationarity and ergodicity of Vector STAR models. *Econometric Reviews*, **39**:4, 407-414.
- Lütkepohl H., Netšunajev A. 2017. Structural vector autoregressions with smooth transition in variances. *Journal of Economic Dynamics & Control*, **84**, 43-57.
- Tsay R. 1998. Testing and Modeling Multivariate Threshold Models. *Journal of the American Statistical Association*, **93**:443, 1188-1202.
- Virolainen S. 2024. Identification by non-Gaussianity in structural threshold and smooth transition vector autoregressive models. Unpublished working paper, available as arXiv:2404.19707.

See Also

fitSTVAR, swap_parametrization, alt_stvar

Examples

```
# Below examples use the example data "gdpdef", which is a two-variate quarterly data
# of U.S. GDP and GDP implicit price deflator covering the period from 1959Q1 to 2019Q4.

# Gaussian STVAR p=1, M=2, model with the weighted relative stationary densities
# of the regimes as the transition weight function:
theta_122relg <- c(0.734054, 0.225598, 0.705744, 0.187897, 0.259626, -0.000863,
  -0.3124, 0.505251, 0.298483, 0.030096, -0.176925, 0.838898, 0.310863, 0.007512,
  0.018244, 0.949533, -0.016941, 0.121403, 0.573269)
mod122 <- STVAR(data=gdpdef, p=1, M=2, params=theta_122relg)
print(mod122) # Printout of the model
summary(mod122) # Summary printout
plot(mod122) # Plot the transition weights
plot(mod122, plot_type="cond_mean") # Plot one-step conditional means

# Logistic Student's t STVAR with p=1, M=2, and the first lag of the second variable
# as the switching variable:
params12 <- c(0.62906848, 0.14245295, 2.41245785, 0.66719269, 0.3534745, 0.06041779, -0.34909745,
 0.61783824, 0.125769, -0.04094521, -0.99122586, 0.63805416, 0.371575, 0.00314754, 0.03440824,
  1.29072533, -0.06067807, 0.18737385, 1.21813844, 5.00884263, 7.70111672)
fit12 <- STVAR(data=gdpdef, p=1, M=2, params=params12, weight_function="logistic",
  weightfun_pars=c(2, 1), cond_dist="Student")
summary(fit12) # Summary printout
plot(fit12) # Plot the transition weights

# Threshold STVAR with p=1, M=2, the first lag of the second variable as switching variable:
params12thres <- c(0.5231, 0.1015, 1.9471, 0.3253, 0.3476, 0.0649, -0.035, 0.7513, 0.1651,
 -0.029, -0.7947, 0.7925, 0.4233, 5e-04, 0.0439, 1.2332, -0.0402, 0.1481, 1.2036)
mod12thres <- STVAR(data=gdpdef, p=1, M=2, params=params12thres, weight_function="threshold",
  weightfun_pars=c(2, 1))
mod12thres # Printout of the model

# Student's t logistic STVAR with p=2, M=2 with the second lag of the second variable
# as the switching variable and structural shocks identified by heteroskedasticity;
# the model created without data:
params22log <- c(0.357, 0.107, 0.356, 0.086, 0.14, 0.035, -0.165, 0.387, 0.452,
 0.013, 0.228, 0.336, 0.239, 0.024, -0.021, 0.708, 0.063, 0.027, 0.009, 0.197,
```

```
  -0.03, 0.24, -0.76, -0.02, 3.36, 0.86, 0.1, 0.2, 7)
mod222logtsh <- STVAR(p=2, M=2, d=2, params=params22log, weight_function="logistic",
 weightfun_pars=c(2, 2), cond_dist="Student", identification="heteroskedasticity")
print(mod222logtsh) # Printout of the model

# STVAR p=2, M=2, model with exogenous transition weights and mutually independent
# Student's t shocks:
set.seed(1); tw1 <- runif(nrow(gdpdef)-2) # Transition weights of Regime 1
params22exoit <- c(0.357, 0.107, 0.356, 0.086, 0.14, 0.035, -0.165, 0.387, 0.452,
 0.013, 0.228, 0.336, 0.239, 0.024, -0.021, 0.708, 0.063, 0.027, 0.009, 0.197,
 -0.1, 0.2, -0.15, 0.13, 0.21, 0.15, 0.11, -0.09, 3, 4)
mod222exoit <- STVAR(p=2, M=2, d=2, params=params22exoit, weight_function="exogenous",
 weightfun_pars=cbind(tw1, 1-tw1), cond_dist="ind_Student")
print(mod222exoit) # Printout of the model

# Linear Gaussian VAR(p=1) model:
theta_112 <- c(0.649526, 0.066507, 0.288526, 0.021767, -0.144024, 0.897103,
  0.601786, -0.002945, 0.067224)
mod112 <- STVAR(data=gdpdef, p=1, M=1, params=theta_112)
summary(mod112) # Summary printout
```

`swap_B_signs` *Swap all signs in pointed columns of the impact matrix of a structural STVAR model that is identified by heteroskedasticity or non-Gaussianity*

Description

`swap_B_signs` swaps all signs in pointed columns of the impact matrix of a structural STVAR model that is identified by heteroskedasticity or non-Gaussianity.

Usage

```
swap_B_signs(stvar, which_to_swap, calc_std_errors = FALSE)
```

Arguments

`stvar`	a class 'stvar' object defining a structural STVAR model that is identified by heteroskedasticity or non-Gaussianity, typically created with `fitSSTVAR`.
`which_to_swap`	a numeric vector of length at most d and elemnts in $1, .., d$ specifying the columns of the impact matrix whose sign should be swapped.
`calc_std_errors`	should approximate standard errors be calculated?

Details

All signs in any column of the impact matrix can be swapped without changing the implied reduced form model. For model identified by non-Gaussianity, the signs of the columns of the impact matrices of all the regimes are swapped accordingly. Note that the sign constraints imposed on the impact matrix via `B_constraints` are also swapped in the corresponding columns accordingly.

Also the order of the columns of the impact matrix can be changed (without changing the implied reduced form model) as long as the ordering of other related parameters is also changed accordingly. This can be done with the function `reorder_B_columns`.

Value

Returns an S3 object of class `'stvar'` defining a smooth transition VAR model. The returned list contains the following components (some of which may be `NULL` depending on the use case):

`data`	The input time series data.
`model`	A list describing the model structure.
`params`	The parameters of the model.
`std_errors`	Approximate standard errors of the parameters, if calculated.
`transition_weights`	The transition weights of the model.
`regime_cmeans`	Conditional means of the regimes, if data is provided.
`total_cmeans`	Total conditional means of the model, if data is provided.
`total_ccovs`	Total conditional covariances of the model, if data is provided.
`uncond_moments`	A list of unconditional moments including regime autocovariances, variances, and means.
`residuals_raw`	Raw residuals, if data is provided.
`residuals_std`	Standardized residuals, if data is provided.
`structural_shocks`	Recovered structural shocks, if applicable.
`loglik`	Log-likelihood of the model, if data is provided.
`IC`	The values of the information criteria (AIC, HQIC, BIC) for the model, if data is provided.
`all_estimates`	The parameter estimates from all estimation rounds, if applicable.
`all_logliks`	The log-likelihood of the estimates from all estimation rounds, if applicable.
`which_converged`	Indicators of which estimation rounds converged, if applicable.
`which_round`	Indicators of which round of optimization each estimate belongs to, if applicable.

References

- Lütkepohl H., Netšunajev A. 2018. Structural vector autoregressions with smooth transition in variances. *Journal of Economic Dynamics & Control*, **84**, 43-57.

See Also

GIRF, fitSSTVAR, reorder_B_columns

Examples

```
# Create a structural two-variate Student's t STVAR p=2, M=2, model with logistic transition
# weights and the first lag of the second variable as the switching variable, and shocks
# identified by heteroskedasticity:
theta_222logt <- c(0.356914, 0.107436, 0.356386, 0.086330, 0.139960, 0.035172, -0.164575,
 0.386816, 0.451675, 0.013086, 0.227882, 0.336084, 0.239257, 0.024173, -0.021209, 0.707502,
 0.063322, 0.027287, 0.009182, 0.197066, -0.03, 0.24, -0.76, -0.02, 3.36, 0.86, 0.1, 0.2, 7)
mod222logt <- STVAR(p=2, M=2, d=2, params=theta_222logt, weight_function="logistic",
  weightfun_pars=c(2, 1), cond_dist="Student", identification="heteroskedasticity")

# Print the parameter values, W and lambdas are printed in the bottom:
mod222logt

# Swap the signs of the first column of W (or equally the impact matrix):
mod222logt2 <- swap_B_signs(mod222logt, which_to_swap=1)
mod222logt2 # The signs of the first column of the impact matrix are swapped

# Swap the signs of the second column of the impact matrix:
mod222logt3 <- swap_B_signs(mod222logt, which_to_swap=2)
mod222logt3 # The signs of the second column of the impact matrix are swapped

# Swap the signs of both columns of the impact matrix:
mod222logt4 <- swap_B_signs(mod222logt, which_to_swap=1:2)
mod222logt4 # The signs of both columns of the impact matrix are swapped
```

---

swap_parametrization *Swap the parametrization of a STVAR model*

Description

swap_parametrization swaps the parametrization of a STVAR model to "mean" if the current parametrization is "intercept", and vice versa.

Usage

```
swap_parametrization(stvar, calc_std_errors = FALSE)
```

Arguments

stvar	object of class "stvar"
calc_std_errors	should approximate standard errors be calculated?

Details

`swap_parametrization` is a convenient tool if you have estimated the model in "intercept" parametrization but wish to work with "mean" parametrization in the future, or vice versa.

Value

Returns an S3 object of class `'stvar'` defining a smooth transition VAR model. The returned list contains the following components (some of which may be `NULL` depending on the use case):

`data`	The input time series data.
`model`	A list describing the model structure.
`params`	The parameters of the model.
`std_errors`	Approximate standard errors of the parameters, if calculated.
`transition_weights`	The transition weights of the model.
`regime_cmeans`	Conditional means of the regimes, if data is provided.
`total_cmeans`	Total conditional means of the model, if data is provided.
`total_ccovs`	Total conditional covariances of the model, if data is provided.
`uncond_moments`	A list of unconditional moments including regime autocovariances, variances, and means.
`residuals_raw`	Raw residuals, if data is provided.
`residuals_std`	Standardized residuals, if data is provided.
`structural_shocks`	Recovered structural shocks, if applicable.
`loglik`	Log-likelihood of the model, if data is provided.
`IC`	The values of the information criteria (AIC, HQIC, BIC) for the model, if data is provided.
`all_estimates`	The parameter estimates from all estimation rounds, if applicable.
`all_logliks`	The log-likelihood of the estimates from all estimation rounds, if applicable.
`which_converged`	Indicators of which estimation rounds converged, if applicable.
`which_round`	Indicators of which round of optimization each estimate belongs to, if applicable.

References

- Anderson H., Vahid F. 1998. Testing multiple equation systems for common nonlinear components. *Journal of Econometrics*, **84**:1, 1-36.
- Hubrich K., Teräsvirta. T. 2013. Thresholds and Smooth Transitions in Vector Autoregressive Models. *CREATES Research Paper 2013-18, Aarhus University.*
- Lanne M., Virolainen S. 2024. A Gaussian smooth transition vector autoregressive model: An application to the macroeconomic effects of severe weather shocks. Unpublished working paper, available as arXiv:2403.14216.

- Kheifets I.L., Saikkonen P.J. 2020. Stationarity and ergodicity of Vector STAR models. *Econometric Reviews*, **39**:4, 407-414.
- Lütkepohl H., Netšunajev A. 2017. Structural vector autoregressions with smooth transition in variances. *Journal of Economic Dynamics & Control*, **84**, 43-57.
- Tsay R. 1998. Testing and Modeling Multivariate Threshold Models. *Journal of the American Statistical Association*, **93**:443, 1188-1202.
- Virolainen S. 2024. Identification by non-Gaussianity in structural threshold and smooth transition vector autoregressive models. Unpublished working paper, available as arXiv:2404.19707.

Examples

```
## Create a Gaussian STVAR p=1, M=2 model with the weighted relative stationary densities
# of the regimes as the transition weight function; use the intercept parametrization:
theta_122relg <- c(0.734054, 0.225598, 0.705744, 0.187897, 0.259626, -0.000863,
  -0.3124, 0.505251, 0.298483, 0.030096, -0.176925, 0.838898, 0.310863, 0.007512,
  0.018244, 0.949533, -0.016941, 0.121403, 0.573269)
mod122 <- STVAR(p=1, M=2, d=2, params=theta_122relg, parametrization="intercept")
mod122$params[1:4] # The intercept parameters

# Swap from the intercept parametrization to mean parametrization:
mod122mu <- swap_parametrization(mod122)
mod122mu$params[1:4] # The mean parameters

# Swap back to the intercept parametrization:
mod122int <- swap_parametrization(mod122mu)
mod122int$params[1:4] # The intercept parameters

## Create a linear VAR(p=1) model with the intercept parametrization, include
# the two-variate data gdpdef to the model and calculate approximate standard errors:
theta_112 <- c(0.649526, 0.066507, 0.288526, 0.021767, -0.144024, 0.897103,
  0.601786, -0.002945, 0.067224)
mod112 <- STVAR(data=gdpdef, p=1, M=1, params=theta_112, parametrization="intercept",
  calc_std_errors=TRUE)
print(mod112, standard_error_print=TRUE) # Standard errors are printed for the intercepts

# To obtain standard errors for the unconditional means instead of the intercepts,
# swap to mean parametrization:
mod112mu <- swap_parametrization(mod112, calc_std_errors=TRUE)
print(mod112mu, standard_error_print=TRUE) # Standard errors are printed for the means
```

`uncond_moments` *Calculate the unconditional means, variances, the first p autocovariances, and the first p autocorrelations of the regimes of the model.*

Description

`uncond_moments` calculates the unconditional means, variances, the first p autocovariances, and the first p autocorrelations of the regimes of the model.

Usage

```
uncond_moments(stvar)
```

Arguments

stvar	object of class "stvar"

Value

Returns a list with three components:

$regime_means a $M \times d$ matrix vector containing the unconditional mean of the regime m in the mth column.

$regime_vars a $M \times d$ matrix vector containing the unconditional marginal variances of the regime m in the mth column.

$regime_autocovs an $(dxdxp + 1, M)$ array containing the lag 0,1,...,p autocovariances of the process. The subset [, , j, m] contains the lag j-1 autocovariance matrix (lag zero for the variance) for the regime m.

$regime_autocors the autocovariance matrices scaled to autocorrelation matrices.

References

- Lütkepohl H. 2005. New Introduction to Multiple Time Series Analysis, *Springer*.

Examples

```
# Two-variate Gaussian STVAR p=1, M=2 model with the weighted relative stationary
# densities of the regimes as the transition weight function:
theta_122relg <- c(0.734054, 0.225598, 0.705744, 0.187897, 0.259626, -0.000863,
-0.3124, 0.505251, 0.298483, 0.030096, -0.176925, 0.838898, 0.310863, 0.007512,
0.018244, 0.949533, -0.016941, 0.121403, 0.573269)
mod122 <- STVAR(data=gdpdef, p=1, M=2, params=theta_122relg, weight_function="relative_dens")

# Calculate the unconditional moments of model:
tmp122 <- uncond_moments(mod122)

# Print the various unconditional moments calculated:
tmp122$regime_means[,1] # Unconditional means of the first regime
tmp122$regime_means[,2] # Unconditional means of the second regime
tmp122$regime_vars[,1] # Unconditional variances of the first regime
tmp122$regime_vars[,2] # Unconditional variances of the second regime
tmp122$regime_autocovs[, , , 1] # a.cov. matrices of the first regime
tmp122$regime_autocovs[, , , 2] # a.cov. matrices of the second regime
tmp122$regime_autocors[, , , 1] # a.cor. matrices of the first regime
tmp122$regime_autocors[, , , 2] # a.cor. matrices of the second regime

# A two-variate linear Gaussian VAR p=1 model:
theta_112 <- c(0.649526, 0.066507, 0.288526, 0.021767, -0.144024, 0.897103,
 0.601786, -0.002945, 0.067224)
mod112 <- STVAR(data=gdpdef, p=1, M=1, params=theta_112)
```

```
# Calculate the unconditional moments of model:
tmp112 <- uncond_moments(mod112)

# Print the various unconditional moments calculated:
tmp112$regime_means # Unconditional means
tmp112$regime_vars # Unconditional variances
tmp112$regime_autocovs # Unconditional autocovariance matrices
tmp112$regime_autocovs[, , 1, 1] # a.cov. matrix of lag zero (of the first regime)
tmp112$regime_autocovs[, , 2, 1] # a.cov. matrix of lag one (of the first regime)
tmp112$regime_autocors # Unconditional autocorrelation matrices
```

---

usacpu

A monthly U.S. data covering the period from 1987:4 to 2024:2 (443 observations) and consisting six variables. First, the climate policy uncertainty index (CPUI) (Gavridiilis, 2021), which is a news based measure of climate policy uncertainty. Second, the economic policy uncertainty index (EPUI), which is a news based measure of economic policy uncertainty. Third, the log-difference of real indsitrial production index (IPI). Fourth, the log-difference of the consumer price index (CPI). Fifth, the log-difference of the producer price index (PPI). Sixth, an interest rate variable, which is the effective federal funds rate that is replaced by the the Wu and Xia (2016) shadow rate during zero-lower-bound periods. The Wu and Xia (2016) shadow rate is not bounded by the zero lower bound and also quantifies unconventional monetary policy measures, while it closely follows the federal funds rate when the zero lower bound does not bind.

Description

A monthly U.S. data covering the period from 1987:4 to 2024:2 (443 observations) and consisting six variables. First, the climate policy uncertainty index (CPUI) (Gavridiilis, 2021), which is a news based measure of climate policy uncertainty. Second, the economic policy uncertainty index (EPUI), which is a news based measure of economic policy uncertainty. Third, the log-difference of real indsitrial production index (IPI). Fourth, the log-difference of the consumer price index (CPI). Fifth, the log-difference of the producer price index (PPI). Sixth, an interest rate variable, which is the effective federal funds rate that is replaced by the the Wu and Xia (2016) shadow rate during zero-lower-bound periods. The Wu and Xia (2016) shadow rate is not bounded by the zero lower bound and also quantifies unconventional monetary policy measures, while it closely follows the federal funds rate when the zero lower bound does not bind.

Usage

```
usacpu
```

Format

A numeric matrix of class `'ts'` with 443 rows and 4 columns with one time series in each column:

First column (CPUI): The climate policy uncertainty index, https://www.policyuncertainty.com/climate_uncertainty.html.

Second column (EPUI): The economic policy uncertainty index, https://www.policyuncertainty.com/us_monthly.html.

Third column (IPI): The log-difference of real indsitrial production index, https://fred.stlouisfed.org/series/INDPRO.

Fourth column (CPI): The log-difference of the consumer price index, https://fred.stlouisfed.org/series/CPIAUCSL.

Fifth column (PPI): The log-difference of the producer price index, https://fred.stlouisfed.org/series/PPIACO.

Sixth column (RATE): The Federal funds rate from 1954Q3 to 2008Q2 and after that the Wu and Xia (2016) shadow rate, https://fred.stlouisfed.org/series/FEDFUNDS, https://www.atlantafed.org/cqer/research/wu-xia-shadow-federal-funds-rate.

Source

The Federal Reserve Bank of St. Louis database and the Federal Reserve Bank of Atlanta's website

References

- K. Gavriilidis, 2021. Measuring climate policy uncertainty. https://www.ssrn.com/abstract=3847388.
- Federal Reserve Bank of Chicago. 2023. Monthly GDP Growth Rate Data. https://www.chicagofed.org/publications/bbki/index.
- Wu J. and Xia F. 2016. Measuring the macroeconomic impact of monetary policy at the zero lower bound. *Journal of Money, Credit and Banking*, 48(2-3): 253-291.

`usamone`

A quarterly U.S. data covering the period from 1954Q3 to 2021Q4 (270 observations) and consisting three variables: cyclical component of the log of real GDP, the log-difference of GDP implicit price deflator, and an interest rate variable. The interest rate variable is the effective federal funds rate from 1954Q3 to 2008Q2 and after that the Wu and Xia (2016) shadow rate, which is not constrained by the zero lower bound and also quantifies unconventional monetary policy measures. The log-differences of the GDP deflator and producer price index are multiplied by hundred.

Description

The cyclical component of the log of real GDP was obtained by applying a one-sided Hodrick-Prescott (HP) filter with the standard smoothing parameter lambda=1600. The one-sided filter was obtained from the two-sided HP filter by applying the filter up to horizon t, taking the last observation, and repeating this procedure for the full sample t=1,...,T. In order to allow the series to start from any phase of the cycle, we applied the one-sided filter to the full available sample from 1947Q1 to 2021Q1 before extracting our sample period from it. We computed the two-sided HP filters with the R package lpirfs (Adämmer, 2021)

Usage

```
usamone
```

Format

A numeric matrix of class 'ts' with 270 rows and 4 columns with one time series in each column:

First column (GDP): The cyclical component of the log of real GDP, https://fred.stlouisfed.org/series/GDPC1.

Second column (GDPDEF): The log-difference of GDP implicit price deflator, https://fred.stlouisfed.org/series/GDPDEF.

Third column (RATE): The Federal funds rate from 1954Q3 to 2008Q2 and after that the Wu and Xia (2016) shadow rate, https://fred.stlouisfed.org/series/FEDFUNDS, https://www.atlantafed.org/cqer/research/wu-xia-shadow-federal-funds-rate.

Source

The Federal Reserve Bank of St. Louis database and the Federal Reserve Bank of Atlanta's website

References

- Adämmer P. 2021. lprfs: Local Projections Impulse Response Functions. R package version: 0.2.0, https://CRAN.R-project.org/package=lpirfs.
- Wu J. and Xia F. 2016. Measuring the macroeconomic impact of monetary policy at the zero lower bound. *Journal of Money, Credit and Banking*, 48(2-3): 253-291.

Wald_test *Perform Wald test for a STVAR model*

Description

Wald_test performs a Wald test for a STVAR model

Usage

```
Wald_test(stvar, A, c)
```

Arguments

stvar	an object of class 'stvar' generated by fitSTVAR or STVAR, containing the model specified by the alternative hypothesis (i.e., **the unconstrained model**).
A	a size (kxn_params) matrix with full row rank specifying a part of the null hypothesis where n_params is the number of parameters in the (unconstrained) model. See details for more information.
c	a length k vector specifying a part of the null hypothesis. See details for more information.

Details

Denoting the true parameter value by θ_0, we test the null hypothesis $A\theta_0 = c$. Under the null, the test statistic is asymptotically χ^2-distributed with k (=nrow(A)) degrees of freedom. The parameter θ_0 is assumed to have the same form as in the model supplied in the argument stvar and it is presented in the documentation of the argument params in the function STVAR (see ?STVAR).

The test is based on the assumption of the standard result of asymptotic normality! Also note that this function does **not** check whether the model assumptions hold under the null.

Value

A list with class "hypotest" containing the test results and arguments used to calculate the test.

References

- Buse A. (1982). The Likelihood Ratio, Wald, and Lagrange Multiplier Tests: An Expository Note. *The American Statistician*, 36(3a), 153-157.

See Also

LR_test, Rao_test, fitSTVAR, STVAR, diagnostic_plot, profile_logliks, Portmanteau_test

Examples

```
# Logistic Student's t STVAR with p=1, M=2, and the first lag of the second variable
# as the switching variable (parameter values were obtained by maximum likelihood estimation;
# fitSTVAR is not used here because the estimation is computationally demanding).
params12 <- c(0.62906848, 0.14245295, 2.41245785, 0.66719269, 0.3534745, 0.06041779, -0.34909745,
 0.61783824, 0.125769, -0.04094521, -0.99122586, 0.63805416, 0.371575, 0.00314754, 0.03440824,
  1.29072533, -0.06067807, 0.18737385, 1.21813844, 5.00884263, 7.70111672)
fit12 <- STVAR(data=gdpdef, p=1, M=2, params=params12, weight_function="logistic",
 weightfun_pars=c(2, 1), cond_dist="Student")
fit12

# Test whether the location parameter equals 1.
# For this model, the parameter vector has the length 21 and
# location parameter is in the 19th element:
A <- matrix(c(rep(0, times=18), 1, 0, 0), nrow=1, ncol=21)
c <- 1
Wald_test(fit12, A=A, c=c)

# Test whether the intercepts and autoregressive matrices are identical across the regimes:
# fit12 has parameter vector of length 21. In the first regime, the intercepts are in the
# elements 1,2 and the AR parameters in the elements 5,...,8. In the second regime,
# the intercepts are in the elements 3,4, and the AR parameters the elements 9,...,12.
A <- rbind(cbind(diag(2), -diag(2), matrix(0, nrow=2, ncol=17)), # intercepts
       cbind(matrix(0, nrow=4, ncol=4), diag(4), -diag(4), matrix(0, nrow=4, ncol=9))) # AR
c <- rep(0, times=6)
Wald_test(fit12, A=A, c=c)
```

Index