Package ‘bvhar’

December 6, 2023

Type Package

Title Bayesian Vector Heterogeneous Autoregressive Modeling

Version 1.0.2

Description Tools to research Bayesian Vector heterogeneous autoregressive (VHAR) model, referring to Kim & Baek (2023+) (<doi:10.1080/00949655.2023.2281644>). 'bvhar' can model Vector Autoregressive (VAR), VHAR, Bayesian VAR (BVAR), and Bayesian VHAR (BVHAR) models.

License GPL (>= 3)


BugReports https://github.com/ygeunkim/bvhar/issues

Suggests covr, knitr, parallel, rmarkdown, testthat (>= 3.0.0)

Encoding UTF-8

LazyData true

RoxygenNote 7.2.3

Imports lifecycle, magrittr, Rcpp, ggplot2, tidyr, tibble, dplyr, foreach, purrr, stats, optimParallel, posterior, bayesplot, doRNG, forcats

LinkingTo Rcpp, RcppEigen, RcppProgress

VignetteBuilder knitr

Depends R (>= 3.6.0)

Config/testthat/edition 3

NeedsCompilation yes

Author Young Geun Kim [aut, cre, cph]
  (<https://orcid.org/0000-0001-8651-1167>),
  Changryong Baek [ctb]

Maintainer Young Geun Kim <dudrms33@g.skku.edu>

Repository CRAN

Date/Publication 2023-12-06 11:20:02 UTC
R topics documented:

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIC.varlse</td>
<td>4</td>
</tr>
<tr>
<td>analyze_ir.varlse</td>
<td>5</td>
</tr>
<tr>
<td>autoplot.bvharirf</td>
<td>7</td>
</tr>
<tr>
<td>autoplot.bvharsp</td>
<td>7</td>
</tr>
<tr>
<td>autoplot.normaliw</td>
<td>8</td>
</tr>
<tr>
<td>autoplot.preditvar</td>
<td>9</td>
</tr>
<tr>
<td>autoplot.summary.bvharsp</td>
<td>10</td>
</tr>
<tr>
<td>autoplot.summary.normaliw</td>
<td>10</td>
</tr>
<tr>
<td>BIC.varlse</td>
<td>11</td>
</tr>
<tr>
<td>bound_bvhar</td>
<td>12</td>
</tr>
<tr>
<td>bvar_flat</td>
<td>13</td>
</tr>
<tr>
<td>bvar_minnesota</td>
<td>15</td>
</tr>
<tr>
<td>bvar_niwhm</td>
<td>17</td>
</tr>
<tr>
<td>bvar_ssvs</td>
<td>19</td>
</tr>
<tr>
<td>bvar_sv</td>
<td>22</td>
</tr>
<tr>
<td>bvhar_minnesota</td>
<td>23</td>
</tr>
<tr>
<td>bvhar_ssvs</td>
<td>25</td>
</tr>
<tr>
<td>bvhar_sv</td>
<td>28</td>
</tr>
<tr>
<td>choose_bayes</td>
<td>29</td>
</tr>
<tr>
<td>choose_bvar</td>
<td>30</td>
</tr>
<tr>
<td>choose_ssvs</td>
<td>32</td>
</tr>
<tr>
<td>choose_var</td>
<td>34</td>
</tr>
<tr>
<td>coef.varlse</td>
<td>34</td>
</tr>
<tr>
<td>compute_dic</td>
<td>35</td>
</tr>
<tr>
<td>compute_logml</td>
<td>36</td>
</tr>
<tr>
<td>confusion</td>
<td>37</td>
</tr>
<tr>
<td>conf_fdr</td>
<td>38</td>
</tr>
<tr>
<td>conf_fnr</td>
<td>39</td>
</tr>
<tr>
<td>conf_fscore</td>
<td>40</td>
</tr>
<tr>
<td>conf_prec</td>
<td>41</td>
</tr>
<tr>
<td>conf_recall</td>
<td>42</td>
</tr>
<tr>
<td>divide_ts</td>
<td>43</td>
</tr>
<tr>
<td>ETF_VIX</td>
<td>43</td>
</tr>
<tr>
<td>fitted.varlse</td>
<td>45</td>
</tr>
<tr>
<td>forecast_expand</td>
<td>45</td>
</tr>
<tr>
<td>forecast_roll</td>
<td>46</td>
</tr>
<tr>
<td>FPE</td>
<td>47</td>
</tr>
<tr>
<td>FPE.varlse</td>
<td>48</td>
</tr>
<tr>
<td>fromse</td>
<td>49</td>
</tr>
<tr>
<td>geom_eval</td>
<td>50</td>
</tr>
<tr>
<td>gg_loss</td>
<td>50</td>
</tr>
<tr>
<td>HQ</td>
<td>51</td>
</tr>
<tr>
<td>HQ.varlse</td>
<td>52</td>
</tr>
<tr>
<td>init_ssvs</td>
<td>53</td>
</tr>
<tr>
<td>is.stable</td>
<td>55</td>
</tr>
<tr>
<td>is.stable.varlse</td>
<td>55</td>
</tr>
</tbody>
</table>
Akaike’s Information Criterion of Multivariate Time Series Model

Description

Compute AIC of VAR(p), VHAR, BVAR(p), and BVHAR

Usage

```r
## S3 method for class 'varlse'
AIC(object, ...)

## S3 method for class 'vharlse'
AIC(object, ...)

## S3 method for class 'bvarmn'
AIC(object, ...)

## S3 method for class 'bvarflat'
AIC(object, ...)

## S3 method for class 'bvharmn'
AIC(object, ...)
```

Arguments

- `object` Model fit
- `...` not used

Details

Let $\hat{\Sigma}_{e}$ be the MLE and let $\hat{\Sigma}_{c}$ be the unbiased estimator (covmat) for $\Sigma_{e}$. Note that

$$
\hat{\Sigma}_{c} = \frac{s - k}{s} \hat{\Sigma}_{e}
$$

Then

$$
AIC(p) = \log \det \Sigma_{e} + \frac{2}{s} (\text{number of freely estimated parameters})
$$

where the number of freely estimated parameters is $mk$, i.e. $pm^{2}$ or $pm^{2} + m$.

Value

AIC value.
References


---

**analyze_ir.varlse**

**Impulse Response Analysis**

**Description**

Computes responses to impulses or orthogonal impulses

**Usage**

```r
## S3 method for class 'varlse'
analyze_ir(
  object,
  lag_max = 10,
  orthogonal = TRUE,
  impulse_var,
  response_var,
  ...
)

## S3 method for class 'vharlse'
analyze_ir(
  object,
  lag_max = 10,
  orthogonal = TRUE,
  impulse_var,
  response_var,
  ...
)

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

analyze_ir(object, lag_max, orthogonal, impulse_var, response_var, ...)

knit_print.bvharirf(x, ...)
```
Arguments

object Model object
lag_max Maximum lag to investigate the impulse responses (By default, 10)
orthogonal Orthogonal impulses (TRUE) or just impulses (FALSE)
impulse_var Impulse variables character vector. If not specified, use every variable.
response_var Response variables character vector. If not specified, use every variable.
... not used
x bvhairf object
digits digit option to print

Value

bvhairf class

Responses to forecast errors

If orthogonal = FALSE, the function gives $W_j$ VMA representation of the process such that

$$Y_t = \sum_{j=0}^{\infty} W_j \epsilon_{t-j}$$

Responses to orthogonal impulses

If orthogonal = TRUE, it gives orthogonalized VMA representation

$$\Theta$$

. Based on variance decomposition (Cholesky decomposition)

$$\Sigma = PP^T$$

where $P$ is lower triangular matrix, impulse response analysis if performed under MA representation

$$y_t = \sum_{i=0}^{\infty} \Theta_i v_{t-i}$$

Here,

$$\Theta_i = W_i P$$

and $v_t = P^{-1} \epsilon_t$ are orthogonal.

References


See Also

VARtoVMA()
VHARtoVMA()
autoplot.bvharirf  

**Plot Impulse Responses**

**Description**

Draw impulse responses of response ~ impulse in the facet.

**Usage**

```r
## S3 method for class 'bvharirf'
autoplot(object, ...)
```

**Arguments**

- `object` : bvharirf object
- `...` : Other arguments passed on the `ggplot2::geom_path()`.

**Value**

A ggplot object

**See Also**

- `analyze_ir()`

autoplot.bvharsp  

**Plot the Result of BVAR and BVHAR MCMC**

**Description**

Draw BVAR and BVHAR MCMC plots.

**Usage**

```r
## S3 method for class 'bvharsp'
autoplot(
  object,
  type = c("trace", "dens", "area"),
  pars = character(),
  regex_pars = character(),
  ...)
```
Arguments

- **object**: bvhar object
- **type**: The type of the plot. Trace plot ("trace"), kernel density plot ("dens"), and interval estimates plot ("area").
- **pars**: Parameter names to draw.
- **regex_pars**: Regular expression parameter names to draw.
- **...**: Other options for each bayesplot::mcmc_trace(), bayesplot::mcmc_dens(), and bayesplot::mcmc_areas().

Value

A ggplot object

---

**autoplot.normaliw**  
**Residual Plot for Minnesota Prior VAR Model**

Description

This function draws residual plot for covariance matrix of Minnesota prior VAR model.

Usage

```r
## S3 method for class 'normaliw'
autoplot(object, hcol = "grey", hsize = 1.5, ...)
```

Arguments

- **object**: normaliw object
- **hcol**: color of horizontal line = 0 (By default, grey)
- **hsize**: size of horizontal line = 0 (By default, 1.5)
- **...**: additional options for geom_point

Value

A ggplot object
**Description**

Plots the forecasting result with forecast regions.

**Usage**

```r
## S3 method for class 'predbvhar'
autoplot(
  object,
  type = c("grid", "wrap"),
  ci_alpha = 0.7,
  alpha_scale = 0.3,
  x_cut = 1,
  viridis = FALSE,
  viridis_option = "D",
  NROW = NULL,
  NCOL = NULL,
  ...
)
```

```r
## S3 method for class 'predbvhar'
autolayer(object, ci_fill = "grey70", ci_alpha = 0.5, alpha_scale = 0.3, ...)
```

**Arguments**

- **object**: predbvhar object
- **type**: Divide variables using `ggplot2::facet_grid()` ("grid": default) or `ggplot2::facet_wrap()` ("wrap")
- **ci_alpha**: Transparency of CI
- **alpha_scale**: Scale of transparency parameter (alpha) between the two layers. alpha of CI ribbon = alpha_scale * alpha of path (By default, .5)
- **x_cut**: plot x axes from x_cut for visibility
- **viridis**: If TRUE, scale CI and forecast line using `ggplot2::scale_fill_viridis_d()` and `ggplot2::scale_colour_viridis_d`, respectively.
- **viridis_option**: Option for viridis string. See option of `ggplot2::scale_colour_viridis_d`. Choose one of c("A", "B", "C", "D", "E"). By default, "D".
- **NROW**: nrow of `ggplot2::facet_wrap()`
- **NCOL**: ncol of `ggplot2::facet_wrap()`
- **...**: additional option for `ggplot2::geom_path()`
- **ci_fill**: color of CI
Value

A ggplot object
A ggplot layer

autplot.summary.bvharsp

Plot the Heatmap of SSVS Coefficients

Description

Draw heatmap for SSVS prior coefficients.

Usage

## S3 method for class 'summary.bvharsp'
autplot(object, ...)

Arguments

object summary.bvharsp object
...
Other arguments passed on the ggplot2::geom_tile().

Value

A ggplot object

autplot.summary.normaliw

Density Plot for Minnesota Prior VAR Model

Description

This function draws density plot for coefficient matrices of Minnesota prior VAR model.

Usage

## S3 method for class 'summary.normaliw'
autplot(
    object,
    type = c("trace", "dens", "area"),
    pars = character(),
    regex_pars = character(),
    ...
)
**Arguments**

- **object**
  - summary.normaliw object

- **type**
  - The type of the plot. Trace plot ("trace"), kernel density plot ("dens"), and interval estimates plot ("area").

- **pars**
  - Parameter names to draw.

- **regex_pars**
  - Regular expression parameter names to draw.

- **...**
  - Other options for each `bayesplot::mcmc_trace()`, `bayesplot::mcmc_dens()`, and `bayesplot::mcmc_areas()`.

**Value**

A ggplot object

---

**BIC.varlse**

*Bayesian Information Criterion of Multivariate Time Series Model*

**Description**

Compute BIC of VAR(p), VHAR, BVAR(p), and BVHAR

**Usage**

```r
## S3 method for class 'varlse'
BIC(object, ...)
```

```r
## S3 method for class 'vharlse'
BIC(object, ...)
```

```r
## S3 method for class 'bvarmn'
BIC(object, ...)
```

```r
## S3 method for class 'bvarflat'
BIC(object, ...)
```

```r
## S3 method for class 'bvharmn'
BIC(object, ...)
```

**Arguments**

- **object**
  - Model fit

- **...**
  - not used
Details

Let \( \hat{\Sigma}_e \) be the MLE and let \( \tilde{\Sigma}_e \) be the unbiased estimator (covmat) for \( \Sigma_e \). Note that

\[
\tilde{\Sigma}_e = \frac{s - k}{n} \hat{\Sigma}_e
\]

Then

\[
BIC(p) = \log \det \Sigma_e + \frac{\log s}{s} \text{(number of freely estimated parameters)}
\]

where the number of freely estimated parameters is \( pm^2 \).

Value

BIC value.

References


Description

**[Experimental]** This function sets lower and upper bounds for `set_bvar()`, `set_bvhar()`, or `set_weight_bvhar()`.

Usage

```r
bound_bvhar(
  init_spec = set_bvhar(),
  lower_spec = set_bvhar(),
  upper_spec = set_bvhar()
)
```

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

knit_print.boundbvaremp(x, ...)
```
**Arguments**

- `init_spec`: Initial Bayes model specification
- `lower_spec`: Lower bound Bayes model specification
- `upper_spec`: Upper bound Bayes model specification
- `x`: boundbvharemp object
- `digits`: Digit option to print
- `...`: Not used

**Value**

boundbvharemp class

---

### bvar_flat

*Fitting Bayesian VAR(p) of Flat Prior*

**Description**

This function fits BVAR(p) with flat prior.

**Usage**

```r
bvar_flat(y, p, bayes_spec = set_bvar_flat(), include_mean = TRUE)
```

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

```r
knit_print.bvarflat(x, ...)
```

**Arguments**

- `y`: Time series data of which columns indicate the variables
- `p`: VAR lag
- `bayes_spec`: A BVAR model specification by `set_bvar_flat()`.
- `include_mean`: Add constant term (Default: TRUE) or not (FALSE)
- `x`: bvarflat object
- `digits`: Digit option to print
- `...`: Not used
Details

Ghosh et al. (2018) gives flat prior for residual matrix in BVAR. Under this setting, there are many models such as hierarchical or non-hierarchical. This function chooses the most simple non-hierarchical matrix normal prior in Section 3.1.

\[ A \mid \Sigma_e \sim MN(0, U^{-1}, \Sigma_e) \]

where \( U \): precision matrix (MN: matrix normal).

\[ p(\Sigma_e) \propto 1 \]

Value

`bvar_flat()` returns an object `bvarflat` class. It is a list with the following components:

- `coefficients`  Posterior Mean matrix of Matrix Normal distribution
- `fitted.values` Fitted values
- `residuals` Residuals
- `mn_prec` Posterior precision matrix of Matrix Normal distribution
- `iw_scale` Posterior scale matrix of posterior inverse-wishart distribution
- `iw_shape` Posterior shape of inverse-wishart distribution
- `df` Numer of Coefficients: mp + 1 or mp
- `p` Lag of VAR
- `m` Dimension of the time series
- `obs` Sample size used when training = totobs - p
- `totobs` Total number of the observation
- `process` Process string in the bayes_spec: "BVAR_Flat"
- `spec` Model specification (bvhspec)
- `type` include constant term ("const") or not ("none")
- `call` Matched call
- `prior_mean` Prior mean matrix of Matrix Normal distribution: zero matrix
- `prior_precision` Prior precision matrix of Matrix Normal distribution: \( U^{-1} \)
- `y0` \( Y_0 \)
- `design` \( X_0 \)
- `y` Raw input (matrix)

References


bvar_minnesota

See Also

- `set_bvar_flat()` to specify the hyperparameters of BVAR flat prior.
- `coef.bvarflat()`, `residuals.bvarflat()`, and `fitted.bvarflat()`
- `predict.bvarflat()` to forecast the BVHAR process

bvar_minnesota  Fitting Bayesian VAR(p) of Minnesota Prior

Description

This function fits BVAR(p) with Minnesota prior.

Usage

```r
bvar_minnesota(y, p = 1, bayes_spec = set_bvar(), include_mean = TRUE)
## S3 method for class 'bvarmn'
print(x, digits = max(3L, getOption("digits") - 3L), ...)
knit_print.bvarmn(x, ...)
```

Arguments

- `y`  Time series data of which columns indicate the variables
- `p`  VAR lag (Default: 1)
- `bayes_spec`  A BVAR model specification by `set_bvar()`.
- `include_mean`  Add constant term (Default: TRUE) or not (FALSE)
- `x`  bvarmn object
- `digits`  digit option to print
- `...`  not used

Details

Minnesota prior gives prior to parameters $A$ (VAR matrices) and $\Sigma_e$ (residual covariance).

\[
A \mid \Sigma_e \sim MN(A_0, \Omega_0, \Sigma_e)
\]

\[
\Sigma_e \sim IW(S_0, \alpha_0)
\]

(MN: matrix normal, IW: inverse-wishart)
Value

`bvar_minnesota()` returns an object `bvarmn` class. It is a list with the following components:

- **coefficients**: Posterior Mean matrix of Matrix Normal distribution
- **fitted.values**: Fitted values
- **residuals**: Residuals
- **mn_prec**: Posterior precision matrix of Matrix Normal distribution
- **iw_scale**: Posterior scale matrix of posterior inverse-Wishart distribution
- **iw_shape**: Posterior shape of inverse-Wishart distribution \((\alpha_{0} - \text{obs} + 2)\). \(\alpha_{0}\) : nrow(Dummy observation) - \(k\)

- **df**: Number of Coefficients: \(mp + 1\) or \(mp\)
- **p**: Lag of VAR
- **m**: Dimension of the time series
- **obs**: Sample size used when training = totobs - \(p\)
- **totobs**: Total number of the observation
- **call**: Matched call
- **process**: Process string in the `bayes_spec`: "BVAR_Minnesota"
- **spec**: Model specification (bvharspec)
- **type**: include constant term ("const") or not ("none")
- **prior_mean**: Prior mean matrix of Matrix Normal distribution: \(A_{0}\)
- **prior_precision**: Prior precision matrix of Matrix Normal distribution: \(\Omega^{-1}_{0}\)
- **prior_scale**: Prior scale matrix of inverse-Wishart distribution: \(S_{0}\)
- **prior_shape**: Prior shape of inverse-Wishart distribution: \(\alpha_{0}\)
- **y0**: \(Y_{0}\)
- **design**: \(X_{0}\)
- **y**: Raw input (matrix)

It is also `normaliw` and `bvharmod` class.

References


See Also

- `set_bvar()` to specify the hyperparameters of Minnesota prior.
- `coef.bvarmn()`, `residuals.bvarmn()`, and `fitted.bvarmn()`
- `summary.normaliw()` to summarize BVAR model
- `predict.bvarmn()` to forecast the BVAR process

Examples

```r
# Perform the function using etf_vix dataset
fit <- bvar_minnesota(y = etf_vix[,1:3], p = 2)
class(fit)

# Extract coef, fitted values, and residuals
coef(fit)
head(residuals(fit))
head(fitted(fit))
```

Description

This function fits hierarchical BVAR(p) with general Minnesota prior.

Usage

```r
bvar_niwhm(
  y,
  p,
  num_iter = 1000,
  num_burn = floor(num_iter/2),
  thinning = 1,
  bayes_spec = set_bvar(sigma = set_psi(), lambda = set_lambda()),
  scale_variance = 0.05,
  include_mean = TRUE,
  parallel = list(),
  verbose = FALSE
)
```

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

knit_print.bvarhm(x, ...)
Arguments

- **y**: Time series data of which columns indicate the variables
- **p**: VAR lag
- **num_iter**: MCMC iteration number
- **num_burn**: Number of burn-in (warm-up). Half of the iteration is the default choice.
- **thinning**: Thinning every thinning-th iteration
- **bayes_spec**: A BVAR model specification by `set_ssvs()`.
- **scale_variance**: Proposal distribution scaling constant to adjust an acceptance rate
- **include_mean**: Add constant term (Default: `TRUE`) or not (`FALSE`)
- **parallel**: List the same argument of `optimParallel::optimParallel()`. By default, this is empty, and the function does not execute parallel computation.
- **verbose**: Print the progress bar in the console. By default, `FALSE`.
- **x**: `bvarhm` object
- **digits**: digit option to print
- **...**: not used

Details

SSVS prior gives prior to parameters $\alpha = \text{vec}(A)$ (VAR coefficient) and $\Sigma_c^{-1} = \Psi \Psi^T$ (residual covariance).

$$
\alpha_j \mid \gamma_j \sim (1 - \gamma_j)N(0, \kappa^2_{0j}) + \gamma_j N(0, \kappa^2_{1j})
$$

$$
\gamma_j \sim \text{Bernoulli}(q_j)
$$

and for upper triangular matrix $\Psi$, $\psi_{ij}$

$$
\psi_{jj}^2 \sim \text{Gamma}(\text{shape} = a_j, \text{rate} = b_j)
$$

$$
\psi_{ij} \mid w_{ij} \sim (1 - w_{ij})N(0, \kappa^2_{0,ij}) + w_{ij} N(0, \kappa^2_{1,ij})
$$

$$
w_{ij} \sim \text{Bernoulli}(q_{ij})
$$

Gibbs sampler is used for the estimation. See `ssvs_bvar_algo` how it works.

Value

`bvar_niwhm` returns an object named `bvarhm` class. It is a list with the following components:

- **coefficients**: Coefficient Matrix
- **p**: Lag of VAR
- **m**: Dimension of the data
- **obs**: Sample size used when training = totobs - p
- **totobs**: Total number of the observation
- **call**: Matched call
**type**  include constant term ("const") or not ("none")

**y0**  $Y_0$

**design**  $X_0$

**y**  Raw input

### References


---

**bvar_ssvs**  

*Fitting Bayesian VAR(p) of SSVS Prior*

### Description

This function fits BVAR(p) with stochastic search variable selection (SSVS) prior.

### Usage

```r
bvar_ssvs(
  y,
  p,
  num_iter = 1000,
  num_burn = floor(num_iter/2),
  thinning = 1,
  bayes_spec = choose_ssvs(y = y, ord = p, type = "VAR", param = c(0.1, 10), include_mean = include_mean, gamma_param = c(0.01, 0.01), mean_non = 0, sd_non = 0.1),
  init_spec = init_ssvs(type = "auto"),
  include_mean = TRUE,
  minnesota = FALSE,
  verbose = FALSE
)
```

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

```r
knit_print.bvarssvs(x, ...)
```
Arguments

- `y`: Time series data of which columns indicate the variables
- `p`: VAR lag
- `num_iter`: MCMC iteration number
- `num_burn`: Number of burn-in (warm-up). Half of the iteration is the default choice.
- `thinning`: Thinning every thinning-th iteration
- `bayes_spec`: A SSVS model specification by `set_ssvs()`. By default, use a default semiautomatic approach `choose_ssvs()`.
- `init_spec`: SSVS initialization specification by `init_ssvs()`. By default, use OLS for coefficient and cholesky factor while 1 for dummies.
- `include_mean`: Add constant term (Default: TRUE) or not (FALSE)
- `minnesota`: Apply cross-variable shrinkage structure (Minnesota-way). By default, FALSE.
- `verbose`: Print the progress bar in the console. By default, FALSE.
- `x`: bvarssvs object
- `digits`: digit option to print
- `...`: not used

Details

SSVS prior gives prior to parameters $\alpha = \text{vec}(A)$ (VAR coefficient) and $\Sigma_e^{-1} = \Psi\Psi^T$ (residual covariance).

$$
\alpha_j \mid \gamma_j \sim (1 - \gamma_j)N(0, \kappa_{0,j}^2) + \gamma_j N(0, \kappa_{1,j}^2)
$$

$$
\gamma_j \sim \text{Bernoulli}(q_j)
$$

and for upper triangular matrix $\Psi$,

$$
\psi_{jj}^2 \sim \text{Gamma}(\text{shape} = a_j, \text{rate} = b_j)
$$

$$
\psi_{ij} \mid w_{ij} \sim (1 - w_{ij})N(0, \kappa_{0,ij}^2) + w_{ij} N(0, \kappa_{1,ij}^2)
$$

$$
w_{ij} \sim \text{Bernoulli}(q_{ij})
$$

Gibbs sampler is used for the estimation. See `ssvs_bvar_algo` how it works.

Value

`bvar_ssvs` returns an object named `bvarssvs` class. It is a list with the following components:

- `alpha_record`: MCMC trace for vectorized coefficients (alpha $\alpha$) with `posterior::draws_df` format.
- `eta_record`: MCMC trace for upper triangular element of cholesky factor (eta $\eta$) with `posterior::draws_df` format.
- `psi_record`: MCMC trace for diagonal element of cholesky factor (psi $\psi$) with `posterior::draws_df` format.
**omega_record** MCMC trace for indicator variable for \( \eta \) (omega \( \omega \)) with `posterior::draws_df` format.

**gamma_record** MCMC trace for indicator variable for \( \alpha \) (gamma \( \gamma \)) with `posterior::draws_df` format.

**chol_record** MCMC trace for cholesky factor matrix \( \Psi \) with list format.

**ols_coef** OLS estimates for VAR coefficients.

**ols_cholesky** OLS estimates for cholesky factor coefficients

**coefficients** Posterior mean of VAR coefficients.

**omega_posterior** Posterior mean of omega

**pip** Posterior inclusion probability

**param** `posterior::draws_df` with every variable: alpha, eta, psi, omega, and gamma

**chol_posterior** Posterior mean of cholesky factor matrix

**covmat** Posterior mean of covariance matrix

**df** Number of Coefficients: \( mp + 1 \) or \( mp \)

**p** Lag of VAR

**m** Dimension of the data

**obs** Sample size used when training = `totobs - p`

**totobs** Total number of the observation

**call** Matched call

**process** Description of the model, e.g. "VAR_SSVS"

**type** include constant term ("const") or not ("none")

**spec** SSVS specification defined by `set_ssvs()`

**init** Initial specification defined by `init_ssvs()`

**iter** Total iterations

**burn** Burn-in

**thin** Thinning

**chain** The number of chains

**y0** \( Y_0 \)

**design** \( X_0 \)

**y** Raw input

References


See Also

- Vectorization formulation `var_vec_formulation`
- Gibbs sampler algorithm `ssvs_bvar_algo`

```

bvar_sv             Fitting Bayesian VAR-SV of Minnesota Belief

Description

[Experimental] This function fits VAR-SV with Minnesota belief.

Usage

bvar_sv(
  y,
  p,
  num_iter = 1000,
  num_burn = floor(num_iter/2),
  thinning = 1,
  bayes_spec = set_bvar(),
  include_mean = TRUE,
  verbose = FALSE,
  num_thread = 1
)

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

knit_print.bvarsv(x, ...)

Arguments

y          Time series data of which columns indicate the variables
p          VAR lag
num_iter   MCMC iteration number
num_burn   Number of burn-in (warm-up). Half of the iteration is the default choice.
thinning   Thinning every thinning-th iteration
bayes_spec A BVAR model specification by `set_bvar()`.
include_mean Add constant term (Default: TRUE) or not (FALSE)
verbose    Print the progress bar in the console. By default, FALSE.
num_thread [Experimental] Number of threads
            bvarsv object
digits     digit option to print
...         not used
```
Details

Cholesky stochastic volatility modeling for VAR based on

\[ \Sigma_t = L^T D_t^{-1} L \]

Value

bvar_sv() returns an object named bvar_sv class.

References


Description

This function fits BVHAR with Minnesota prior.

Usage

```r
bvhar_minnesota(
  y,
  har = c(5, 22),
  bayes_spec = set_bvhar(),
  include_mean = TRUE
)
```

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

knit_print.bvharmn(x, ...)

Arguments

- `y`: Time series data of which columns indicate the variables
- `har`: Numeric vector for weekly and monthly order. By default, c(5, 22).
- `bayes_spec`: A BVHAR model specification by set_bvhar() (default) or set_weight_bvhar().
- `include_mean`: Add constant term (Default: TRUE) or not (FALSE)
- `x`: bvarmn object
- `digits`: digit option to print
- `...`: not used
Details

Apply Minnesota prior to Vector HAR: \( \Phi \) (VHAR matrices) and \( \Sigma_e \) (residual covariance).

\[
\Phi \mid \Sigma_e \sim MN(M_0, \Omega_0, \Sigma_e) \\
\Sigma_e \sim IW(\Psi_0, \nu_0)
\]

(MN: matrix normal, IW: inverse-wishart)

There are two types of Minnesota priors for BVHAR:

- VAR-type Minnesota prior specified by `set_bvhar()`, so-called BVHAR-S model.
- VHAR-type Minnesota prior specified by `set_weight_bvhar()`, so-called BVHAR-L model.

Two types of Minnesota priors builds different dummy variables for \( Y_0 \). See `var_design_formulation`.

Value

`bvhar_minnesota()` returns an object `bvharmn` class. It is a list with the following components:

- **coefficients** Posterior Mean matrix of Matrix Normal distribution
- **fitted.values** Fitted values
- **residuals** Residuals
- **mn_prec** Posterior precision matrix of Matrix Normal distribution
- **iw_scale** Posterior scale matrix of posterior inverse-wishart distribution
- **iw_shape** Posterior shape of inverse-Wishart distribution \((\nu_0 - \text{obs} + 2)\). \(\nu_0\): nrow(Dummy observation) - \(k\)
- **df** Number of Coefficients: \(3m + 1\) or \(3m\)
- **p** 3, this element exists to run the other functions
- **week** Order for weekly term
- **month** Order for monthly term
- **m** Dimension of the time series
- **obs** Sample size used when training = \(\text{totobs} - 22\)
- **totobs** Total number of the observation
- **call** Matched call
- **process** Process string in the `bayes_spec`: "BVHAR_MN_VAR" (BVHAR-S) or "BVHAR_MN_VHAR" (BVHAR-L)
- **spec** Model specification (bvhar_spec)
- **type** include constant term ("const") or not ("none")
- **prior_mean** Prior mean matrix of Matrix Normal distribution: \(M_0\)
- **prior_precision** Prior precision matrix of Matrix Normal distribution: \(\Omega_0^{-1}\)
- **prior_scale** Prior scale matrix of inverse-Wishart distribution: \(\Psi_0\)
- **prior_shape** Prior shape of inverse-Wishart distribution: \(\nu_0\)
- **HARtrans** VHAR linear transformation matrix: \(C_{HAR}\)
 bvhar_ssvs

\[ y_0 \]
\[ \text{design} \quad X_0 \]
\[ y \quad \text{Raw input (matrix)} \]

It is also normaliw and bvharmod class.

References


See Also

- `set_bvhar()` to specify the hyperparameters of BVHAR-S
- `set_weight_bvhar()` to specify the hyperparameters of BVHAR-L
- `coef.bvharmn()`, `residuals.bvharmn()`, and `fitted.bvharmn()`
- `summary.normaliw()` to summarize BVHAR model
- `predict.bvharmn()` to forecast the BVHAR process

Examples

```r
# Perform the function using etf_vix dataset
fit <- bvhar_minnesota(y = etf_vix[,1:3])
class(fit)

# Extract coef, fitted values, and residuals
coef(fit)
head(residuals(fit))
head(fitted(fit))
```

---

**bvhar_ssvs**

Fitting Bayesian VHAR of SSVS Prior

Description

This function fits BVAR(p) with stochastic search variable selection (SSVS) prior.

Usage

```r
bvhar_ssvs(
  y,
  har = c(5, 22),
  num_iter = 1000,
  num_burn = floor(num_iter/2),
  thinning = 1,
  bayes_spec = choose_ssvs(y = y, ord = har, type = "VHAR", param = c(0.1, 10),
                          include_mean = include_mean, gamma_param = c(0.01, 0.01),
                          mean_non = 0, sd_non = 0.1),
  ...)
```
init_spec = init_ssvs(type = "auto"),
iclause.include_mean = TRUE,
minnesota = c("no", "short", "longrun"),
verbose = FALSE
)

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

knit_print.bvharssvs(x, ...)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
</table>
y| Time series data of which columns indicate the variables|
har| Numeric vector for weekly and monthly order. By default, c(5, 22).
num_iter| MCMC iteration number
num_burn| Number of warm-up (burn-in). Half of the iteration is the default choice.
thinning| Thinning every thinning-th iteration
bayes_spec| A SSVS model specification by set_ssvs(). By default, use a default semiautomatic approach choose_ssvs().
init_spec| SSVS initialization specification by init_ssvs(). By default, use OLS for coefficient and cholesky factor while 1 for dummies.
include_mean| Add constant term (Default: TRUE) or not (FALSE)
minnesota| Apply cross-variable shrinkage structure (Minnesota-way). Two type: "short" type and "longrun" type. By default, "no".
verbose| Print the progress bar in the console. By default, FALSE.
x| bvharssvs object
digits| digit option to print
...| not used

Details

SSVS prior gives prior to parameters $\alpha = \text{vec}(A)$ (VAR coefficient) and $\Sigma^{-1} = \Psi \Psi^T$ (residual covariance).

$$\begin{align*}
\alpha_j | \gamma_j &\sim (1 - \gamma_j)N(0, \kappa_{0j}^2) + \gamma_j N(0, \kappa_{1j}^2) \\
\gamma_j &\sim \text{Bernoulli}(q_j)
\end{align*}$$

and for upper triangular matrix $\Psi$,

$$\begin{align*}
\psi_{jj}^2 &\sim \text{Gamma}(\text{shape} = a_j, \text{rate} = b_j) \\
\psi_{ij} | w_{ij} &\sim (1 - w_{ij})N(0, \kappa_{0,ij}^2) + w_{ij} N(0, \kappa_{1,ij}^2) \\
w_{ij} &\sim \text{Bernoulli}(q_{ij})
\end{align*}$$

Gibbs sampler is used for the estimation. See ssvs_bvar_algo how it works.


Value

`bvhar_ssvs` returns an object named `bvhar_ssvs` class. It is a list with the following components:

- **phi_record**  MCMC trace for vectorized coefficients ($\phi$) with `posterior::draws_df` format.
- **eta_record** MCMC trace for upper triangular element of cholesky factor ($\eta$) with `posterior::draws_df` format.
- **psi_record** MCMC trace for diagonal element of cholesky factor ($\psi$) with `posterior::draws_df` format.
- **omega_record** MCMC trace for indicator variable for $\eta$ (omega $\omega$) with `posterior::draws_df` format.
- **gamma_record** MCMC trace for indicator variable for $\alpha$ (gamma $\gamma$) with `posterior::draws_df` format.
- **chol_record** MCMC trace for cholesky factor matrix $\Psi$ with `list` format.
- **ols_coef** OLS estimates for VAR coefficients.
- **ols_cholesky** OLS estimates for cholesky factor coefficients.
- **coefficients** Posterior mean of VAR coefficients.
- **omega_posterior** Posterior mean of omega.
- **pip** Posterior inclusion probability.
- **param** `posterior::draws_df` with every variable: alpha, eta, psi, omega, and gamma.
- **chol_posterior** Posterior mean of cholesky factor matrix.
- **covmat** Posterior mean of covariance matrix.
- **df** Numer of Coefficients: $3m + 1$ or $3m$
- **p** 3 (The number of terms. It contains this element for usage in other functions.)
- **week** Order for weekly term.
- **month** Order for monthly term.
- **m** Dimension of the data.
- **obs** Sample size used when training = `totsobs - p`.
- **totsobs** Total number of the observation.
- **call** Matched call.
- **process** Description of the model, e.g. "VHAR_SSVS"
- **type** Include constant term ("const") or not ("none").
- **spec** SSVS specification defined by `set_ssvs()`.
- **init** Initial specification defined by `init_ssvs()`.
- **iter** Total iterations.
- **burn** Burn-in.
- **thin** Thinning.
- **chain** The number of chains.
- **HARtrans** VHAR linear transformation matrix.
- **y0** $Y_0$
- **design** $X_0$
- **y** Raw input.
References


See Also

- Vectorization formulation *var_vec_formulation*
- Gibbs sampler algorithm *ssvs_bvar_algo*

---

**bvhar_sv**

*Fitting Bayesian VHAR-SV of Minnesota Belief*

Description

*[Experimental]* This function fits VHAR-SV with Minnesota belief.

Usage

```r
bvhar_sv(
  y,
  har = c(5, 22),
  num_iter = 1000,
  num_burn = floor(num_iter/2),
  thinning = 1,
  bayes_spec = set_bvhar(),
  include_mean = TRUE,
  verbose = FALSE,
  num_thread = 1
)
```

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

```r
knit_print.bvharsv(x, ...)
```

Arguments

- `y`: Time series data of which columns indicate the variables
- `har`: Numeric vector for weekly and monthly order. By default, `c(5, 22)`.
- `num_iter`: MCMC iteration number
- `num_burn`: Number of burn-in (warm-up). Half of the iteration is the default choice.
**choose_bayes**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>thinning</td>
<td>Thinning every thinning-th iteration</td>
</tr>
<tr>
<td>bayes_spec</td>
<td>A BVHAR model specification by <code>set_bvhar()</code> (default) or <code>set_weight_bvhar()</code></td>
</tr>
<tr>
<td>include_mean</td>
<td>Add constant term (Default: TRUE) or not (FALSE)</td>
</tr>
<tr>
<td>verbose</td>
<td>Print the progress bar in the console. By default, FALSE.</td>
</tr>
<tr>
<td>num_thread</td>
<td>[Experimental] Number of threads</td>
</tr>
<tr>
<td>x</td>
<td>bvarsv object</td>
</tr>
<tr>
<td>digits</td>
<td>digit option to print</td>
</tr>
<tr>
<td>...</td>
<td>not used</td>
</tr>
</tbody>
</table>

**Details**

Cholesky stochastic volatility modeling for VHAR based on

\[
\Sigma_t = L_t^T D_t^{-1} L_t
\]

**Value**

`bvhar_sv()` returns an object named `bvharsv` class.

**References**


---

**Description**

[Experimental] This function chooses the set of hyperparameters of Bayesian model using `stats::optim()` function.

**Usage**

```r
choose_bayes(
  bayes_bound = bound_bvhar(),
  ...,
  eps = 1e-04,
  y,
  order = c(5, 22),
  include_mean = TRUE,
  parallel = list()
)
```
choose_bvar

Arguments

bayes_bound  Empirical Bayes optimization bound specification defined by \texttt{bound_bvhar()}.  
...  Additional arguments for \texttt{stats::optim()}.  
eps  Hyperparameter \(\text{eps}\) is fixed. By default, \(1e^{-04}\).  
y  Time series data  
order  Order for BVAR or BVHAR. \(p\) of \texttt{bvar_minnesota()} or \(\text{har}\) of \texttt{bvhar_minnesota()}. By default, \(c(5, 22)\) for \(\text{har}\).  
include_mean  Add constant term (Default: \texttt{TRUE}) or not (\texttt{FALSE}).  
parallel  List the same argument of \texttt{optimParallel::optimParallel()}. By default, this is empty, and the function does not execute parallel computation.

Value

\texttt{bvharemp} class is a list that has

... Many components of \texttt{stats::optim()} or \texttt{optimParallel::optimParallel()}

\texttt{spec} Corresponding \texttt{bvhar}spec

\texttt{fit} Chosen Bayesian model

\texttt{ml} Marginal likelihood of the final model

References


See Also

- \texttt{bound_bvhar()} to define L-BFGS-B optimization bounds.
- Individual functions: \texttt{choose_bvar()}

Description

Instead of these functions, you can use \texttt{choose_bayes()}.  
Usage

```r
choose_bvar(
  bayes_spec = set_bvar(),
  lower = 0.01,
  upper = 10,
  ..., 
  eps = 1e-04,
  y,
  p,
  include_mean = TRUE,
  parallel = list()
)

choose_bvhar(
  bayes_spec = set_bvhar(),
  lower = 0.01,
  upper = 10,
  ..., 
  eps = 1e-04,
  y,
  har = c(5, 22), 
  include_mean = TRUE,
  parallel = list()
)
```

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

knit_print.bvharemp(x, ...)

Arguments

- `bayes_spec`: Initial Bayes model specification.
- `lower`: [Experimental] Lower bound. By default, .01.
- `...`: not used
- `eps`: Hyperparameter eps is fixed. By default, 1e-04.
- `y`: Time series data
- `p`: BVAR lag
- `include_mean`: Add constant term (Default: TRUE) or not (FALSE)
- `parallel`: List the same argument of `optimParallel::optimParallel()`. By default, this is empty, and the function does not execute parallel computation.
- `har`: Numeric vector for weekly and monthly order. By default, c(5, 22).
- `x`: bvharemp object
- `digits`: digit option to print
choose_ssvs

Details

Empirical Bayes method maximizes marginal likelihood and selects the set of hyperparameters. These functions implement "L-BFGS-B" method of stats::optim() to find the maximum of marginal likelihood.

If you want to set lower and upper option more carefully, deal with them like as in stats::optim() in order of set_bvar(), set_bvhar(), or set_weight_bvhar()’s argument (except eps). In other words, just arrange them in a vector.

Value

bvharemp class is a list that has

- stats::optim() or optimParallel::optimParallel()
- chosen bvhar spec set
- Bayesian model fit result with chosen specification

... Many components of stats::optim() or optimParallel::optimParallel()

spec Corresponding bvhar spec
fit Chosen Bayesian model
ml Marginal likelihood of the final model

References


choose_ssvs

Choose the Hyperparameters Set of SSVS-VAR using a Default Semi-automatic Approach

Description

[Experimental] This function chooses \((\tau_{0i}, \tau_{1i})\) and \((\kappa_{0i}, \kappa_{1i})\) using a default semiautomatic approach.
choose_ssvs

Usage

choose_ssvs(
  y,
  ord,
  type = c("VAR", "VHAR"),
  param = c(0.1, 10),
  include_mean = TRUE,
  gamma_param = c(0.01, 0.01),
  mean_non = 0,
  sd_non = 0.1
)

Arguments

y               Time series data of which columns indicate the variables.
ord             Order for VAR or VHAR.
type            Model type (Default: "VAR" or "VHAR").
param           Preselected constants $c_0 << c_1$. By default, 0.1 and 10 (See Details).
include_mean    Add constant term (Default: TRUE) or not (FALSE).
gamma_param     Parameters (shape, rate) for Gamma distribution. This is for the output.
mean_non        Prior mean of unrestricted coefficients. This is for the output.
sd_non          Standard deviance of unrestricted coefficients. This is for the output.

Details

Instead of using subjective values of $(\tau_{0i}, \tau_{1i})$, we can use

$$\tau_{ki} = c_k \text{VAR}(OLS)$$

It must be $c_0 << c_1$.

In case of $(\omega_{0ij}, \omega_{1ij})$,

$$\omega_{kij} = c_k = \text{VAR}(OLS)$$

similarly.

Value

ssvsinput object

References


choose_var  
*Choose the Best VAR based on Information Criteria*

**Description**
This function computes AIC, FPE, BIC, and HQ up to \( p = \text{lag}\_\text{max} \) of VAR model.

**Usage**
```r
choose_var(y, lag_max = 5, include_mean = TRUE, parallel = FALSE)
```

**Arguments**
- **y**: Time series data of which columns indicate the variables
- **lag_max**: Maximum Var lag to explore (default = 5)
- **include_mean**: Add constant term (Default: `TRUE`) or not (`FALSE`)
- **parallel**: Parallel computation using `foreach::foreach()`? By default, `FALSE`.

**Value**
Minimum order and information criteria values

---

coef.varlse  
*Coefficient Matrix of Multivariate Time Series Models*

**Description**
By defining `stats::coef()` for each model, this function returns coefficient matrix estimates.

**Usage**
```r
## S3 method for class 'varlse'
coef(object, ...)
## S3 method for class 'vharlse'
coef(object, ...)
## S3 method for class 'bvarmn'
coef(object, ...)
## S3 method for class 'bvarflat'
coef(object, ...)
## S3 method for class 'bvharmn'
coef(object, ...)
```
**compute_dic**

Deviance Information Criterion of Multivariate Time Series Model

## Description

Compute DIC of BVAR and BVHAR.

## Usage

```r
compute_dic(object, ...)  
## S3 method for class 'bvarmn'
compute_dic(object, n_iter = 100L, ...)
```

## Arguments

- `object`: Model fit
- `...`: not used
- `n_iter`: Number to sample

## Details

Deviance information criteria (DIC) is

\[-2 \log p(y \mid \hat{\theta}_{bayes}) + 2p_{DIC}\]

where \(p_{DIC}\) is the effective number of parameters defined by

\[p_{DIC} = 2(\log p(y \mid \hat{\theta}_{bayes}) - E_{post} \log p(y \mid \theta))\]

Random sampling from posterior distribution gives its computation, \(\theta_i \sim \theta \mid y, i = 1, \ldots, M\).
\[ p_{\text{computed}}^{\text{DIC}} = 2(\log p(y \mid \hat{\theta}_{\text{bayes}}) - \frac{1}{M} \sum_{i} \log p(y \mid \theta_i)) \]

Value
DIC value.

References

---

**compute_logml**

*Extracting Log of Marginal Likelihood*

**Description**
Compute log of marginal likelihood of Bayesian Fit

**Usage**
```
compute_logml(object, ...)
```

```
## S3 method for class 'bvarmn'
compute_logml(object, ...)
```

```
## S3 method for class 'bvharmn'
compute_logml(object, ...)
```

**Arguments**
- `object`      Model fit
- `...`         not used

**Details**
Closed form of Marginal Likelihood of BVAR can be derived by

\[
p(Y_0) = \pi^{-ms/2} \frac{\Gamma_m((\alpha_0 + s)/2)}{\Gamma_m(\alpha_0/2)} \det(\Omega_0)^{-m/2} \det(S_0)^{\alpha_0/2} \det(\hat{V})^{-m/2} \det(\hat{\Sigma}_e)^{-(\alpha_0+s)/2}
\]

Closed form of Marginal Likelihood of BVHAR can be derived by

\[
p(Y_0) = \pi^{-ms_0/2} \frac{\Gamma_m((d_0 + s)/2)}{\Gamma_m(d_0/2)} \det(P_0)^{-m/2} \det(U_0)^{d_0/2} \det(\hat{V}_{HAR})^{-m/2} \det(\hat{\Sigma}_e)^{-(d_0+s)/2}
\]
**Value**

log likelihood of Minnesota prior model.

**References**


---

**confusion**

Evaluate the Sparsity Estimation Based on Confusion Matrix

**Description**

This function computes FDR (false discovery rate) and FNR (false negative rate) for sparse element of the true coefficients given threshold.

**Usage**

```r
confusion(x, y, ...)  
## S3 method for class 'summary.bvharsp'
confusion(x, y, truth_thr = 0, ...)
```

**Arguments**

- `x`: summary.bvharsp object.
- `y`: True inclusion variable.
- `...`: not used
- `truth_thr`: Threshold value when using non-sparse true coefficient matrix. By default, 0 for sparse matrix.

**Details**

When using this function, the true coefficient matrix $\Phi$ should be sparse.

In this confusion matrix, positive (0) means sparsity. FP is false positive, and TP is true positive. FN is false negative, and FN is false negative.

**Value**

Confusion table as following.

<table>
<thead>
<tr>
<th>True-estimate</th>
<th>Positive (0)</th>
<th>Negative (1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Positive (0)</td>
<td>TP</td>
<td>FN</td>
</tr>
<tr>
<td>Negative (1)</td>
<td>FP</td>
<td>TN</td>
</tr>
</tbody>
</table>
**References**


---

**Description**

This function computes false discovery rate (FDR) for sparse element of the true coefficients given threshold.

**Usage**

```r
conf_fdr(x, y, ...)  
## S3 method for class 'summary.bvharsp'
conf_fdr(x, y, truth_thr = 0, ...)
```

**Arguments**

- `x`: summary.bvharsp object.
- `y`: True inclusion variable.
- `truth_thr`: Threshold value when using non-sparse true coefficient matrix. By default, 0 for sparse matrix.
- `...`: not used

**Details**

When using this function, the true coefficient matrix $\Phi$ should be sparse. False discovery rate (FDR) is computed by

$$FDR = \frac{FP}{TP + FP}$$

where TP is true positive, and FP is false positive.

**Value**

FDR value in confusion table

**References**


**See Also**

`confusion()`
Description

This function computes false negative rate (FNR) for sparse element of the true coefficients given threshold.

Usage

conf_fnr(x, y, ...)

## S3 method for class 'summary.bvhar.sp'
conf_fnr(x, y, truth_thr = 0, ...)

Arguments

x  summary.bvhar.sp object.
y  True inclusion variable.
... not used
truth_thr  Threshold value when using non-sparse true coefficient matrix. By default, 0 for sparse matrix.

Details

False negative rate (FNR) is computed by

\[ FNR = \frac{FN}{TP + FN} \]

where TP is true positive, and FN is false negative.

Value

FNR value in confusion table

References


See Also

confusion()
Evaluate the Sparsity Estimation Based on F1 Score

Description

This function computes F1 score for sparse element of the true coefficients given threshold.

Usage

conf_fscore(x, y, ...)

## S3 method for class 'summary.bvharsp'
conf_fscore(x, y, truth_thr = 0, ...)

Arguments

x  
summary.bvharsp object.

y  
True inclusion variable.

...  
not used

truth_thr  
Threshold value when using non-sparse true coefficient matrix. By default, 0 for sparse matrix.

Details

The F1 score is computed by

\[
F_1 = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}}
\]

Value

F1 score in confusion table

See Also

confusion()
conf_prec

Evaluate the Sparsity Estimation Based on Precision

Description
This function computes precision for sparse element of the true coefficients given threshold.

Usage
conf_prec(x, y, 
## S3 method for class 'summary.bvharsp'
conf_prec(x, y, truth_thr = 0, 

Arguments
x summary.bvharsp object.
y True inclusion variable.
... not used
truth_thr Threshold value when using non-sparse true coefficient matrix. By default, 0 for sparse matrix.

Details
If the element of the estimate \( \hat{\Phi} \) is smaller than some threshold, it is treated to be zero. Then the precision is computed by

\[
\text{precision} = \frac{TP}{TP + FP}
\]

where TP is true positive, and FP is false positive.

Value
Precision value in confusion table

References

See Also
confusion()
conf_recall

Evaluate the Sparsity Estimation Based on Recall

Description

This function computes recall for sparse element of the true coefficients given threshold.

Usage

conf_recall(x, y, ...)

## S3 method for class 'summary.bvharsp'
conf_recall(x, y, truth_thr = 0L, ...)

Arguments

x summary.bvharsp object.
y True inclusion variable.
... not used
truth_thr Threshold value when using non-sparse true coefficient matrix. By default, 0 for sparse matrix.

Details

Precision is computed by

\[ recall = \frac{TP}{TP + FN} \]

where TP is true positive, and FN is false negative.

Value

Recall value in confusion table

References


See Also

confusion()
**divide_ts**  
*Split a Time Series Dataset into Train-Test Set*

**Description**
Split a given time series dataset into train and test set for evaluation.

**Usage**
```
divide_ts(y, n_ahead)
```

**Arguments**
- `y`: Time series data of which columns indicate the variables
- `n_ahead`: step to evaluate

**Value**
List of two datasets, train and test.

---

**etf_vix**  
*CBOE ETF Volatility Index Dataset*

**Description**
Chicago Board Options Exchange (CBOE) Exchange Traded Funds (ETFs) volatility index from FRED.

**Usage**
```
etf_vix
```

**Format**
A data frame of 1006 row and 9 columns:
From 2012-01-09 to 2015-06-27, 33 missing observations were interpolated by `stats::approx()` with linear.

- **GVZCLS**: Gold ETF volatility index
- **VXFXICLS**: China ETF volatility index
- **OVXCLS**: Crude Oil ETF volatility index
- **VXEEMCLS**: Emerging Markets ETF volatility index
- **EVZCLS**: EuroCurrency ETF volatility index
- **VXSLVCLS**: Silver ETF volatility index
- **VXGDXCLS**: Gold Miners ETF volatility index
- **VXXLECLS**: Energy Sector ETF volatility index
- **VXEWZCLS**: Brazil ETF volatility index
Details

Copyright, 2016, Chicago Board Options Exchange, Inc.

Note that, in this data frame, dates column is removed. This dataset interpolated 36 missing observations (nontrading dates) using imputeTS::na_interpolation().

Source

Source: https://www.cboe.com


References


Chicago Board Options Exchange, CBOE China ETF Volatility Index (VXFXICLS), retrieved from FRED, Federal Reserve Bank of St. Louis; https://fred.stlouisfed.org/series/VXFXICLS, August 1, 2021.

Chicago Board Options Exchange, CBOE Crude Oil ETF Volatility Index (OVXCLS), retrieved from FRED, Federal Reserve Bank of St. Louis; https://fred.stlouisfed.org/series/OVXCLS, August 1, 2021.


Chicago Board Options Exchange, CBOE Silver ETF Volatility Index (VXSLVCLS), retrieved from FRED, Federal Reserve Bank of St. Louis; https://fred.stlouisfed.org/series/VXSLVCLS, August 1, 2021.


Chicago Board Options Exchange, CBOE Brazil ETF Volatility Index (VXEWZCLS), retrieved from FRED, Federal Reserve Bank of St. Louis; https://fred.stlouisfed.org/series/VXEWZCLS, August 2, 2021.
Description

By defining `stats::fitted()` for each model, this function returns fitted matrix.

Usage

```r
## S3 method for class 'varlse'
fitted(object, ...)

## S3 method for class 'vharlse'
fitted(object, ...)

## S3 method for class 'bvarmn'
fitted(object, ...)

## S3 method for class 'bvarflat'
fitted(object, ...)

## S3 method for class 'bvharmn'
fitted(object, ...)
```

Arguments

- `object`: Model object
- `...`: not used

Value

`matrix` object.

Description

This function conducts expanding window forecasting.

Usage

```r
forecast_expand(object, n_ahead, y_test)
```
**Arguments**

- `object`: Model object
- `n_ahead`: Step to forecast in rolling window scheme
- `y_test`: Test data to be compared. Use `divide_ts()` if you don’t have separate evaluation dataset.

**Details**

Expanding windows forecasting fixes the starting period. It moves the window ahead and forecast h-ahead in `y_test` set.

**Value**

`predbvhar_expand` class

**References**


**See Also**

See `ts_forecasting_cv` for out-of-sample forecasting methods.

---

**Description**

This function conducts rolling window forecasting.

**Usage**

```r
forecast_roll(object, n_ahead, y_test, roll_thread = 1, mod_thread = 1)
```

### S3 method for class 'bvharcv'

```r
print(x, digits = max(3L, getOption("digits") - 3L), ...)
```

```r
knit_print.bvharcv(x, ...)
```
Arguments

- **object**: Model object
- **nAhead**: Step to forecast in rolling window scheme
- **yTest**: Test data to be compared. Use `divide_ts()` if you don’t have separate evaluation dataset.
- **roll_thread**: [Experimental] Number of threads when rolling window
- **mod_thread**: [Experimental] Number of threads when fitting the models
- **x**: bvharcv object
- **digits**: digit option to print
- **...**: not used

Details

Rolling windows forecasting fixes window size. It moves the window ahead and forecast h-ahead in `y_test` set.

Value

- `predbvhar_roll` class

References


See Also

See `ts_forecasting_cv` for out-of-sample forecasting methods.

---

**FPE**

*Final Prediction Error Criterion*

Description

Generic function that computes FPE criterion.

Usage

`FPE(object, ...)`

Arguments

- **object**: Model fit
- **...**: not used

Value

FPE value.
Description

Compute FPE of VAR(p), VHAR, BVAR(p), and BVHAR

Usage

```r
## S3 method for class 'varlse'
FPE(object, ...)
```

```r
## S3 method for class 'vharlse'
FPE(object, ...)
```

Arguments

- `object` Model fit
- `...` not used

Details

Let \( \hat{\Sigma}_e \) be the MLE and let \( \tilde{\Sigma}_e \) be the unbiased estimator (covmat) for \( \Sigma_e \). Note that

\[
\tilde{\Sigma}_e = \frac{s - k}{n} \hat{\Sigma}_e
\]

Then

\[
FPE(p) = \left( \frac{s + k}{s - k} \right)^m \det \tilde{\Sigma}_e
\]

Value

FPE value.

References

fromse

**Evaluate the Estimation Based on Frobenius Norm**

Description

This function computes estimation error given estimated model and true coefficient.

Usage

```r
fromse(x, y, ...)
```

## S3 method for class 'bvhar.sp'
fromse(x, y, ...)

Arguments

- `x`: Estimated model.
- `y`: Coefficient matrix to be compared.
- `...`: not used

Details

Consider the Frobenius Norm $\| \cdot \|_F$. Let $\hat{\Phi}$ be nrow x k the estimates, and let $\Phi$ be the true coefficients matrix. Then the function computes estimation error by

$$MSE = 100 \frac{\| \hat{\Phi} - \Phi \|_F}{nrow \times k}$$

Value

Frobenius norm value

References

**geom_eval**  
*Adding Test Data Layer*

**Description**  
This function adds a layer of test dataset.

**Usage**  
```r  
geom_eval(data, colour = "red", ...)  
```

**Arguments**  
- **data**: Test data to draw, which has the same format with the train data.  
- **colour**: Color of the line (By default, "red").  
- **...**: Other arguments passed on the `ggplot2::geom_path()`.

**Value**  
A ggplot layer

---

**gg_loss**  
*Compare Lists of Models*

**Description**  
Draw plot of test error for given models

**Usage**  
```r  
gg_loss(  
  mod_list,  
  y,  
  type = c("mse", "mae", "mape", "mase"),  
  mean_line = FALSE,  
  line_param = list(),  
  mean_param = list(),  
  viridis = FALSE,  
  viridis_option = "D",  
  NROW = NULL,  
  NCOL = NULL,  
  ...  
)  
```
Arguments

- **mod_list**: Lists of forecast results (predbvar objects)
- **y**: Test data to be compared. Should be the same format with the train data and predict$forecast.
- **type**: Loss function to be used ("mse": MSE, "mae": MAE, mape: MAPE, "mase": MASE)
- **mean_line**: Whether to draw average loss. By default, FALSE.
- **line_param**: Parameter lists for ggplot2::geom_path().
- **mean_param**: Parameter lists for average loss with ggplot2::geom_hline().
- **viridis**: If TRUE, scale CI and forecast line using ggplot2::scale_fill_viridis_d() and ggplot2::scale_colour_viridis_d, respectively.
- **viridis_option**: Option for viridis string. See option of ggplot2::scale_colour_viridis_d. Choose one of c("A", "B", "C", "D", "E"). By default, "D".
- **nrow**: nrow of ggplot2::facet_wrap()
- **ncol**: ncol of ggplot2::facet_wrap()
- **...**: Additional options for geom_loss (inherit.aes and show.legend)

Value

A ggplot object

See Also

- **mse()** to compute MSE for given forecast result
- **mae()** to compute MAE for given forecast result
- **mape()** to compute MAPE for given forecast result
- **mase()** to compute MASE for given forecast result

Description

Generic function that computes HQ criterion.

Usage

HQ(object, ...)

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

HQ

Hannan-Quinn Criterion
Arguments

object Model fit
... not used

Details

The formula is

\[ HQ = -2 \log p(y \mid \hat{\theta}) + k \log \log(n) \]

which can be computed by \texttt{AIC(object, ..., k = 2 * log(log(nobs(object))))} with \texttt{stats::AIC()}.

Value

HQ value.

References


---

**Description**

Compute HQ of VAR(p), VHAR, BVAR(p), and BVHAR

**Usage**

```r
## S3 method for class 'varlse'
HQ(object, ...)

## S3 method for class 'vharlse'
HQ(object, ...)

## S3 method for class 'bvarmn'
HQ(object, ...)

## S3 method for class 'bvarflat'
HQ(object, ...)

## S3 method for class 'bharmn'
HQ(object, ...)
```
Arguments

object Model fit
... not used

Details

Let $\tilde{\Sigma}_e$ be the MLE and let $\hat{\Sigma}_e$ be the unbiased estimator (covmat) for $\Sigma_e$. Note that

$$\tilde{\Sigma}_e = \frac{s}{n} \frac{k}{n} \Sigma_e$$

Then

$$HQ(p) = \log \det \Sigma_e + \frac{2 \log \log s}{s}$$

(number of freely estimated parameters)

where the number of freely estimated parameters is $pm^2$.

Value

HQ value.

References


init_ssvs

Initial Parameters of Stochastic Search Variable Selection (SSVS) Model

Description

Set initial parameters before starting Gibbs sampler for SSVS.

Usage

```r
init_ssvs(
  init_coef,
  init_coef_dummy,
  init_chol,
  init_chol_dummy,
  type = c("user", "auto")
)
```
# S3 method for class 'ssvsinit'
print(x, digits = max(3L, getOption("digits") - 3L), ...)

knit_print.ssvsinit(x, ...)

Arguments

init_coef
Initial coefficient matrix. Initialize with an array or list for multiple chains.

init_coef_dummy
Initial indicator matrix (1-0) corresponding to each component of coefficient. Initialize with an array or list for multiple chains.

init_chol
Initial cholesky factor (upper triangular). Initialize with an array or list for multiple chains.

init_chol_dummy
Initial indicator matrix (1-0) corresponding to each component of cholesky factor. Initialize with an array or list for multiple chains.

type
[Experimental] Type to choose initial values. One of "user" (User-given) and "auto" (OLS for coefficients and 1 for dummy).

x
ssvsinit

digits
digit option to print

... not used

Details

Set SSVS initialization for the VAR model.

- init_coef: \((kp + 1) \times m\) \(A\) coefficient matrix.
- init_coef_dummy: \(kp \times m\) \(\Gamma\) dummy matrix to restrict the coefficients.
- init_chol: \(k \times k\) \(\Psi\) upper triangular cholesky factor, which \(\Psi\Psi^T = \Sigma_{\epsilon}^{-1}\).
- init_chol_dummy: \(k \times k\) \(\Omega\) upper triangular dummy matrix to restrict the cholesky factor.

Denote that init_chol and init_chol_dummy should be upper_triangular or the function gives error.

For parallel chain initialization, assign three-dimensional array or three-length list.

Value

ssvsinit object

References


**is.stable**  
*Stability of the process*

**Description**
Stability of the process

**Usage**

```r
is.stable(x, ...)
```

**Arguments**

- `x` object
- `...` not used

**Value**
logical class

---

**is.stable.varlse**  
*Stability of VAR Coefficient Matrix*

**Description**
Check the stability condition of VAR(p) coefficient matrix.

**Usage**

```r
# S3 method for class 'varlse'
is.stable(x, ...)

# S3 method for class 'vharlse'
is.stable(x, ...)

# S3 method for class 'bvarmn'
is.stable(x, ...)

# S3 method for class 'bvarflat'
is.stable(x, ...)

# S3 method for class 'bvharmn'
is.stable(x, ...)
```
is.varlse

Arguments

x Model fit
... not used

Details

VAR(p) is stable if

$$\det(I_m - Az) \neq 0$$

for $$|z| \leq 1$$.

Value

logical class

References


is.varlse See if the Object a class in this package

Description

This function returns TRUE if the input is the class defined by this package.

Usage

is.varlse(x)
is.vharlse(x)
is.bvarmn(x)
is.bvarflat(x)
is.bvharmn(x)
is.predbvhar(x)
is.bvharcv(x)
is.bvharspec(x)
is.bvharpriorspec(x)
logLik.varlse

is.bvharemp(x)

is.boundbvharemp(x)

is.ssvsinput(x)

is.ssvsinit(x)

is.bvharpriorspec(x)

Arguments

x Object

Value

logical class

Description

Compute log-likelihood function value of VAR(p), VHAR, BVAR(p), and BVHAR

Usage

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

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

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

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

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

Arguments

object Model fit

... not used
Details

Consider the response matrix $Y_0$. Let $n$ be the total number of sample, let $m$ be the dimension of the time series, let $p$ be the order of the model, and let $s = n - p$. Likelihood of VAR($p$) has

$$Y_0 \mid B, \Sigma_e \sim MN(X_0B, I_s, \Sigma_e)$$

where $X_0$ is the design matrix, and $MN$ is matrix normal distribution.

Then log-likelihood of vector autoregressive model family is specified by

$$\log p(Y_0 \mid B, \Sigma_e) = -\frac{sn}{2} \log 2\pi - \frac{s}{2} \log \det \Sigma_e - \frac{1}{2} \text{tr}((Y_0 - X_0B)\Sigma_e^{-1}(Y_0 - X_0B)^T)$$

In addition, recall that the OLS estimator for the matrix coefficient matrix is the same as MLE under the Gaussian assumption. MLE for $\Sigma_e$ has different denominator, $s$.

$$\hat{B} = \hat{B}^{LS} = \hat{B}^{ML} = (X_0^TX_0)^{-1}X_0^TY_0$$

$$\hat{\Sigma}_e = \frac{1}{s-k}(Y_0 - X_0\hat{B})^T(Y_0 - X_0\hat{B})$$

$$\hat{\Sigma}_e = \frac{1}{s}(Y_0 - X_0\hat{B})^T(Y_0 - X_0\hat{B}) = \frac{s-k}{s}\hat{\Sigma}_e$$

In case of VHAR, just consider the linear relationship.

While frequentist models use OLS and MLE for coefficient and covariance matrices, Bayesian models implement posterior means.

Value

A logLik object.

References


mae

Evaluate the Model Based on MAE (Mean Absolute Error)

Description

This function computes MAE given prediction result versus evaluation set.

Usage

mae(x, y, ...)

## S3 method for class 'predbvhar'
mae(x, y, ...)

## S3 method for class 'bvharcv'
mae(x, y, ...)

Arguments

x Forecasting object
y Test data to be compared. should be the same format with the train data.
... not used

Details

Let $e_t = y_t - \hat{y}_t$. MAE is defined by

$$MSE = mean(|e_t|)$$

Some researchers prefer MAE to MSE because it is less sensitive to outliers.

Value

MAE vector corresponsing to each variable.

References

Evaluate the Model Based on MAPE (Mean Absolute Percentage Error)

Description

This function computes MAPE given prediction result versus evaluation set.

Usage

```r
mape(x, y, ...)
```

## S3 method for class 'predbvhar'
```r
mape(x, y, ...)
```

## S3 method for class 'bvharcv'
```r
mape(x, y, ...)
```

Arguments

- `x` Forecasting object
- `y` Test data to be compared. should be the same format with the train data.
- `...` not used

Details

Let \( e_t = y_t - \hat{y}_t \). Percentage error is defined by \( p_t = 100 \frac{e_t}{Y_t} \) (100 can be omitted since comparison is the focus).

\[
MAPE = \text{mean}(|p_t|)
\]

Value

MAPE vector corresponding to each variable.

References

### Description

This function computes MASE given prediction result versus evaluation set.

### Usage

```r
mase(x, y, ...)
```

#### S3 method for class 'predbvharcv'

```r
mase(x, y, ...)
```

### Arguments

- `x`  
  Forecasting object

- `y`  
  Test data to be compared, should be the same format with the train data.

- `...`  
  not used

### Details

Let \( e_t = y_t - \hat{y}_t \). Scaled error is defined by

\[
q_t = \frac{e_t}{\sum_{i=2}^{n}|Y_i - Y_{i-1}|/(n-1)}
\]

so that the error can be free of the data scale. Then

\[
MASE = \text{mean}(|q_t|)
\]

Here, \( Y_i \) are the points in the sample, i.e. errors are scaled by the in-sample mean absolute error \((\text{mean}(|e_t|))\) from the naive random walk forecasting.

### Value

MASE vector corresponding to each variable.

### References

mrae  Evaluate the Model Based on MRAE (Mean Relative Absolute Error)

Description
This function computes MRAE given prediction result versus evaluation set.

Usage
mrae(x, pred_bench, y, ...)

## S3 method for class 'predvhar'
mrae(x, pred_bench, y, ...)

## S3 method for class 'bvharcv'
mrae(x, pred_bench, y, ...)

Arguments
x  Forecasting object to use
pred_bench  The same forecasting object from benchmark model
y  Test data to be compared. should be the same format with the train data.
...  not used

Details
Let $e_t = y_t - \hat{y}_t$. MRAE implements benchmark model as scaling method. Relative error is defined by

$$r_t = \frac{e_t}{e_t^*}$$

where $e_t^*$ is the error from the benchmark method. Then

$$MRAE = mean(|r_t|)$$

Value
MRAE vector corresponding to each variable.

References
mse

Evaluate the Model Based on MSE (Mean Square Error)

Description

This function computes MSE given prediction result versus evaluation set.

Usage

mse(x, y, ...)

## S3 method for class 'predbvhar'
mse(x, y, ...)

## S3 method for class 'bvharcv'
mse(x, y, ...)

Arguments

x             Forecasting object
y             Test data to be compared. should be the same format with the train data.
...           not used

Details

Let $e_t = y_t - \hat{y}_t$. Then

$$MSE = mean(e_t^2)$$

MSE is the most used accuracy measure.

Value

MSE vector corresponding to each variable.

References

Description

The realized measure of financial assets dataset provided by Oxford-man Institute of Quantitative Finance.

Usage

oxfordman_rv

oxfordman_rk

Format

oxfordman_long is the raw data frame of 53507 rows and 20 columns (You cannot call this dataset.):

date  Date - From 2012-01-09 to 2015-06-27
Symbol  Name of the Assets - See below for each name
nobs  Number of observations
by_ss  Bipower Variation (5-min Sub-sampled)
rsv  Realized Semi-variance (5-min)
rk_parzen  Realized Kernel Variance (Non-Flat Parzen)
rv10  Realized Variance (10-min)
rv5_ss  Realized Variance (5-min Sub-sampled)
rv5  Realized Variance (5-min)
rv10_ss  Realized Variance (10-min Sub-sampled)
rk_twoscale  Realized Kernel Variance (Two-Scale/Bartlett)
close_price  Closing (Last) Price
rsv_ss  Realized Semi-variance (5-min Sub-sampled)
rk_th2  Realized Kernel Variance (Tukey-Hanning(2))
open_time  Opening Time
medrv  Median Realized Variance (5-min)
open_price  Opening (First) Price
bv  Bipower Variation (5-min)
open_to_close  Open to Close Return
close_time  Closing Time
oxfordman_rv is a data frame that interpolates NA values of oxfordman_wide_rv. Also, it does not have date column for fitting. The number of rows is 905 and the number of columns is 30 (except date).

**date**  Date - From 2012-01-09 to 2015-06-27
AEX  AEX index
AORD  All Ordinaries
BFX  Bell 20 Index
BSESN  S&P BSE SENSEX
BVSP  BSEB BOVESPA Index
DJI  Dow Jones Industrial Average
FCHI  CAC 40
FTMIB  FTSE MIB
FTSE  FTSE 100
GDAXI  DAX
GSPTSE  S&P/TSX Composite index
HSI  HANG SENG Index
IBEX  IBEX 35 Index
IXIC  Nasdaq 100
KS11  Korea Composite Stock Price Index (KOSPI)
KSE  Karachi SE 100 Index
MXX  IPC Mexico
N225  Nikkei 225
NSEI  NIFTY 50
OMXC20  OMX Copenhagen 20 Index
OMXHPI  OMX Helsinki All Share Index
OMXSPI  OMX Stockholm All Share Index
OSEAX  Oslo Exchange All-share Index
RUT  Russel 2000
SMSI  Madrid General Index
SPX  S&P 500 Index
SSEC  Shanghai Composite Index
SSMI  Swiss Stock Market Index
STI  Straits Times Index (excluded because this index is NA in the period)
STOXX50E  EURO STOXX 50

oxfordman_rk is a data frame that interpolates NA values of oxfordman_wide_rk. Also, it does not have DATE column for fitting. The number of rows is 1826 and the number of columns is 31.
Details

- As a raw dataset, we have internal dataset of long format oxfordman_long. It contains every realized measure.
- Denote that non-trading dates are excluded in oxfordman_long, not in NA. So be careful when dealing this set directly.
- For analysis, we widened the data for 5-min realized volatility (rv5) and realized kernel variance (rk_parzen), respectively.
  - oxfordman_wide_rv
  - oxfordman_wide_rk
- oxford_rv and oxford_rk are the sets whose NA values interpolated using imputeTS::na_interpolation().
- First three datasets should be called using data() function: data(..., package = "bvhar").
- Only oxford_rv and oxford_rk is lazy loaded.

Source

Realized library of oxford-man had been discontinued, so the source could not be listed.

predict.varlse  Forecasting Multivariate Time Series

Description

Forecasts multivariate time series using given model.

Usage

```r
## S3 method for class 'varlse'
predict(object, n_ahead, level = 0.05, ...)

## S3 method for class 'vharlse'
predict(object, n_ahead, level = 0.05, ...)

## S3 method for class 'bvarmn'
predict(object, n_ahead, n_iter = 100L, level = 0.05, ...)

## S3 method for class 'bvharmn'
predict(object, n_ahead, n_iter = 100L, level = 0.05, ...)

## S3 method for class 'bvarflat'
predict(object, n_ahead, n_iter = 100L, level = 0.05, ...)

## S3 method for class 'bvarssvs'
predict(object, n_ahead, level = 0.05, ...)

## S3 method for class 'bvharssvs'
```
predict.varlse

predict(object, n_ahead, level = 0.05, ...)

```
## S3 method for class 'bvarsv'
predict(object, n_ahead, ...)

## S3 method for class 'bvharsv'
predict(object, n_ahead, ...)

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

knit_print.predbvhar(x, ...)
```

Arguments

- **object**: Model object
- **n_ahead**: step to forecast
- **level**: Specify alpha of confidence interval level 100(1 - alpha) percentage. By default, 0.05.
- **...**: not used
- **n_iter**: Number to sample residual matrix from inverse-wishart distribution. By default, 100.
- **x**: predbvhar object
- **digits**: digit option to print

Value

predbvhar class with the following components:

- **process**: object$process
- **forecast**: forecast matrix
- **se**: standard error matrix
- **lower**: lower confidence interval
- **upper**: upper confidence interval
- **lower_joint**: lower CI adjusted (Bonferroni)
- **upper_joint**: upper CI adjusted (Bonferroni)
- **y**: object$y

n-step ahead forecasting VAR(p)

See pp35 of Lütkepohl (2007). Consider h-step ahead forecasting (e.g. n + 1, ... n + h).
Let \( y_{(n)}^T = (y_n^T, ..., y_{n-p+1}^T, 1) \). Then one-step ahead (point) forecasting:

\[
y_{n+1}^T = y_{(n)}^T \hat{B}
\]
Recursively, let \( \hat{y}_{(n+1)}^T = (\hat{y}_{n+1}^T, y_n^T, \ldots, y_{n-p+2}^T, 1) \). Then two-step ahead (point) forecasting:

\[
\hat{y}_{n+2}^T = \hat{y}_{(n+1)}^T \hat{B}
\]

Similarly, h-step ahead (point) forecasting:

\[
\hat{y}_{n+h}^T = \hat{y}_{(n+h-1)}^T \hat{B}
\]

How about confident region? Confidence interval at h-period is

\[
y_{k,t}(h) \pm z(\alpha/2)\sigma_k(h)
\]

Joint forecast region of \( 100(1 - \alpha)\% \) can be computed by

\[
\{(y_k, y_{k,h}) \mid y_{k,n}(i) - z(\alpha/2h)\sigma_n(i) \leq y_{n,i} \leq y_{k,n}(i) + z(\alpha/2h)\sigma_k(i), i = 1, \ldots, h\}
\]

See the pp41 of Lütkepohl (2007).

To compute covariance matrix, it needs VMA representation:

\[
Y_t(h) = c + \sum_{i=h}^{\infty} W_i \epsilon_{t+h-i} = c + \sum_{i=0}^{\infty} W_{h+i} \epsilon_{t-i}
\]

Then

\[
\Sigma_y(h) = MSE[y(h)] = \sum_{i=0}^{h-1} W_i \Sigma \epsilon W_i^T = \Sigma_y(h - 1) + W_{h-1} \Sigma \epsilon W_{h-1}^T
\]

n-step ahead forecasting VHAR

Let \( T_{HAR} \) is VHAR linear transformation matrix (See var_design_formulation). Since VHAR is the linearly transformed VAR(22), let \( y_{(n)}^T = (y_n^T, y_{n-1}^T, \ldots, y_{n-21}^T, 1) \).

Then one-step ahead (point) forecasting:

\[
\hat{y}_{n+1}^T = y_{(n)}^T T_{HAR} \hat{\Phi}
\]

Recursively, let \( \hat{y}_{(n+1)}^T = (\hat{y}_{n+1}^T, y_n^T, \ldots, y_{n-20}^T, 1) \). Then two-step ahead (point) forecasting:

\[
\hat{y}_{n+2}^T = \hat{y}_{(n+1)}^T T_{HAR} \hat{\Phi}
\]

and h-step ahead (point) forecasting:

\[
\hat{y}_{n+h}^T = \hat{y}_{(n+h-1)}^T T_{HAR} \hat{\Phi}
\]
n-step ahead forecasting BVAR(p) with minnesota prior

Point forecasts are computed by posterior mean of the parameters. See Section 3 of Babírú et al. (2010).

Let \( \hat{B} \) be the posterior MN mean and let \( \hat{V} \) be the posterior MN precision.

Then predictive posterior for each step

\[
y_{n+1} | \Sigma_e, y \sim N(vec(y^T_{(n)} A), \Sigma_e \otimes (1 + y^T_{(n)} \hat{V}^{-1} y_{(n)}))
\]

\[
y_{n+2} | \Sigma_e, y \sim N(vec(y^T_{(n+1)} A), \Sigma_e \otimes (1 + y^T_{(n+1)} \hat{V}^{-1} y_{(n+1)}))
\]

and recursively,

\[
y_{n+h} | \Sigma_e, y \sim N(vec(y^T_{(n+h-1)} A), \Sigma_e \otimes (1 + y^T_{(n+h-1)} \hat{V}^{-1} y_{(n+h-1)}))
\]

See `bvar_predictive_density` how to generate the predictive distribution.

n-step ahead forecasting BVHAR

Let \( \hat{\Phi} \) be the posterior MN mean and let \( \hat{\Psi} \) be the posterior MN precision.

Then predictive posterior for each step

\[
y_{n+1} | \Sigma_e, y \sim N(vec(y^T_{(n)} \hat{\Phi}), \Sigma_e \otimes (1 + y^T_{(n)} \hat{\Psi}^{-1} \hat{\Phi} y_{(n)}))
\]

\[
y_{n+2} | \Sigma_e, y \sim N(vec(y^T_{(n+1)} \hat{\Phi}), \Sigma_e \otimes (1 + y^T_{(n+1)} \hat{\Psi}^{-1} \hat{\Phi} y_{(n+1)}))
\]

and recursively,

\[
y_{n+h} | \Sigma_e, y \sim N(vec(y^T_{(n+h-1)} \hat{\Phi}), \Sigma_e \otimes (1 + y^T_{(n+h-1)} \hat{\Psi}^{-1} \hat{\Phi} y_{(n+h-1)}))
\]

See `bvar_predictive_density` how to generate the predictive distribution.

n-step ahead forecasting VAR(p) with SSVS and Horseshoe

The process of the computing point estimate is the same. However, predictive interval is achieved from each Gibbs sampler sample.

\[
y_{n+1} | A, \Sigma_e, y \sim N(vec(y^T_{(n)} A), \Sigma_e)
\]

\[
y_{n+h} | A, \Sigma_e, y \sim N(vec(y^T_{(n+h-1)} A), \Sigma_e)
\]

n-step ahead forecasting VHAR with SSVS and Horseshoe

The process of the computing point estimate is the same. However, predictive interval is achieved from each Gibbs sampler sample.

\[
y_{n+1} | \Sigma_e, y \sim N(vec(y^T_{(n)} \hat{\Phi}), \Sigma_e \otimes (1 + y^T_{(n)} \hat{\Psi}^{-1} \hat{\Phi} y_{(n)}))
\]

\[
y_{n+h} | \Sigma_e, y \sim N(vec(y^T_{(n+h-1)} \hat{\Phi}), \Sigma_e \otimes (1 + y^T_{(n+h-1)} \hat{\Psi}^{-1} \hat{\Phi} y_{(n+h-1)}))
\]
References


---

`print.summary.ssvsmod` **Summarizing VAR and VHAR with SSVS Prior Model**

**Description**

Conduct variable selection.

**Usage**

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

knit_print.summary.ssvsmod(x, ...)

## S3 method for class 'ssvsmod'
summary(object, ...)
```

**Arguments**

- `x` summary.ssvsmod object
- `digits` digit option to print
- `object` ssvsmod object
Details

In each cell, variable selection can be done by giving threshold for each cell of coefficient:

\[ |\phi_i| \leq 3\tau_0 \]

and

\[ |\eta_{ij}| \leq 3\kappa_{0ij} \]

Value

summary.ssvsmod object

References


relmae

Evaluate the Model Based on RelMAE (Relative MAE)

Description

This function computes RelMAE given prediction result versus evaluation set.

Usage

relmae(x, pred_bench, y, ...)

## S3 method for class 'predbvar'
relmae(x, pred_bench, y, ...)

## S3 method for class 'bvharcv'
relmae(x, pred_bench, y, ...)

Arguments

x Forecasting object to use
pred_bench The same forecasting object from benchmark model
y Test data to be compared. should be the same format with the train data.
... not used
Details
Let $e_t = y_t - \hat{y}_t$. RelMAE implements MAE of benchmark model as relative measures. Let $MAE_b$ be the MAE of the benchmark model. Then

$$RelMAE = \frac{MAE}{MAE_b}$$

where $MAE$ is the MAE of our model.

Value
RelMAE vector corresponding to each variable.

References

Description
This function computes relative estimation error given estimated model and true coefficient.

Usage
relspne(x, y, ...)

## S3 method for class 'bvharsp'
relspne(x, y, ...)

Arguments
x Estimated model.
y Coefficient matrix to be compared.
... not used

Details
Let $\|\cdot\|_2$ be the spectral norm of a matrix, let $\hat{\Phi}$ be the estimates, and let $\Phi$ be the true coefficients matrix. Then the function computes relative estimation error by

$$\frac{\|\hat{\Phi} - \Phi\|_2}{\|\Phi\|_2}$$
Value

Spectral norm value

References


---

residuals.varlse  
*Residual Matrix from Multivariate Time Series Models*

Description

By defining `stats::residuals()` for each model, this function returns residual.

Usage

```r
## S3 method for class 'varlse'
residuals(object, ...)

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

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

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

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

Arguments

- `object`  
  Model object

- `...`  
  not used

Value

`matrix` object.
Evaluate the Model Based on RMAFE

**Description**

This function computes RMAFE (Mean Absolute Forecast Error Relative to the Benchmark)

**Usage**

```r
rmafe(x, pred_bench, y, ...)
```

## S3 method for class 'predbvhav'
```r
rmafe(x, pred_bench, y, ...)
```

## S3 method for class 'bvharcv'
```r
rmafe(x, pred_bench, y, ...)
```

**Arguments**

- `x` Forecasting object to use
- `pred_bench` The same forecasting object from benchmark model
- `y` Test data to be compared. should be the same format with the train data.
- `...` not used

**Details**

Let $e_t = y_t - \hat{y}_t$. RMAFE is the ratio of L1 norm of $e_t$ from forecasting object and from benchmark model.

$$RMAFE = \frac{\text{sum}(\|e_t\|)}{\text{sum}(\|e_t^{(b)}\|)}$$

where $e_t^{(b)}$ is the error from the benchmark model.

**Value**

RMAFE vector corresponding to each variable.

**References**


Description

This function computes RMAPE given prediction result versus evaluation set.

Usage

```r
rmape(x, pred_bench, y, ...)
```

## S3 method for class 'predbvar'
```r
rmape(x, pred_bench, y, ...)
```

## S3 method for class 'bvharcv'
```r
rmape(x, pred_bench, y, ...)
```

Arguments

- `x` Forecasting object to use
- `pred_bench` The same forecasting object from benchmark model
- `y` Test data to be compared. should be the same format with the train data.
- `...` not used

Details

RMAPE is the ratio of MAPE of given model and the benchmark one. Let $MAPE_b$ be the MAPE of the benchmark model. Then

$$RMAPE = \frac{\text{mean}(MAPE)}{\text{mean}(MAPE_b)}$$

where $MAPE$ is the MAPE of our model.

Value

RMAPE vector corresponding to each variable.

References

Evaluate the Model Based on RMASE (Relative MASE)

Description
This function computes RMASE given prediction result versus evaluation set.

Usage
```r
rmse(x, pred_bench, y, ...)
```

Arguments
- `x` Forecasting object to use
- `pred_bench` The same forecasting object from benchmark model
- `y` Test data to be compared. should be the same format with the train data.
- `...` not used

Details
RMASE is the ratio of MAPE of given model and the benchmark one. Let $MASE_b$ be the MAPE of the benchmark model. Then

$$RMASE = \frac{\text{mean}(MASE)}{\text{mean}(MASE_b)}$$

where $MASE$ is the MASE of our model.

Value
RMASE vector corresponding to each variable.

References
Description
This function computes RMSFE (Mean Squared Forecast Error Relative to the Benchmark).

Usage

```r
rmsfe(x, pred_bench, y, ...) # S3 method for class 'predbvhvar'
```

Arguments

- `x`: Forecasting object to use.
- `pred_bench`: The same forecasting object from benchmark model.
- `y`: Test data to be compared. should be the same format with the train data.
- `...`: not used

Details

Let $e_t = y_t - \hat{y}_t$. RMSFE is the ratio of L2 norm of $e_t$ from forecasting object and from benchmark model.

$$RMSFE = \frac{\sum \|e_t\|}{\sum \|e_{t}^{(b)}\|}$$

where $e_{t}^{(b)}$ is the error from the benchmark model.

Value

RMSFE vector corresponding to each variable.

References

Hyperparameters for Bayesian Models

Description

Set hyperparameters of Bayesian VAR and VHAR models.

Usage

```r
set_bvar(sigma, lambda = 0.1, delta, eps = 1e-04)
set_bvar_flat(U)
set_bvhar(sigma, lambda = 0.1, delta, eps = 1e-04)
set_weight_bvhar(sigma, lambda = 0.1, eps = 1e-04, daily, weekly, monthly)
## S3 method for class 'bvharspec'
print(x, digits = max(3L, getOption("digits") - 3L), ...)
knit_print.bvharspec(x, ...)
```

Arguments

- `sigma`: Standard error vector for each variable (Default: sd)
- `lambda`: Tightness of the prior around a random walk or white noise (Default: .1)
- `delta`: Persistence (Default: Litterman sets 1 = random walk prior, White noise prior = 0)
- `eps`: Very small number (Default: 1e-04)
- `U`: Positive definite matrix. By default, identity matrix of dimension ncol(X0)
- `daily`: Same as `delta` in VHAR type (Default: 1 as Litterman)
- `weekly`: Fill the second part in the first block (Default: 1)
- `monthly`: Fill the third part in the first block (Default: 1)
- `x`: `bvharspec` object
- `digits`: digit option to print
- `...`: not used

Details

- Missing arguments will be set to be default values in each model function mentioned above.
- `set_bvar()` sets hyperparameters for `bvar_minnesota()`.
- Each `delta` (vector), `lambda` (length of 1), `sigma` (vector), `eps` (vector) corresponds to $\delta_j, \lambda, \delta_j, \epsilon$.
- $\delta_i$ are related to the belief to random walk.
set_bvar

- If $\delta_i = 1$ for all $i$, random walk prior
- If $\delta_i = 0$ for all $i$, white noise prior

$\lambda$ controls the overall tightness of the prior around these two prior beliefs.

- If $\lambda = 0$, the posterior is equivalent to prior and the data do not influence the estimates.
- If $\lambda = \infty$, the posterior mean becomes OLS estimates (VAR).

$\sigma_i^2/\sigma_j^2$ in Minnesota moments explain the data scales.

- set_bvar_flat sets hyperparameters for bvar_flat().

- set_bvharp() sets hyperparameters for bvhar_minnesota() with VAR-type Minnesota prior, i.e. BVHAR-S model.

- set_weight_bvharp() sets hyperparameters for bvhar_minnesota() with VHAR-type Minnesota prior, i.e. BVHAR-L model.

Value

Every function returns bvhar spec class. It is the list of which the components are the same as the arguments provided. If the argument is not specified, NULL is assigned here. The default values mentioned above will be considered in each fitting function.

process  Model name: BVAR, BVHAR
prior  Prior name: Minnesota (Minnesota prior for BVAR), Hierarchical (Hierarchical prior for BVAR), MN_VAR (BVHAR-S), MN_VHAR (BVHAR-L), Flat (Flat prior for BVAR)
sigma  Vector value (or bvhar priorspec class) assigned for sigma
lambda  Value (or bvhar priorspec class) assigned for lambda
delta  Vector value assigned for delta
eps  Value assigned for epsilon

set_weight_bvharp() has different component with delta due to its different construction.

daily  Vector value assigned for daily weight
weekly  Vector value assigned for weekly weight
monthly  Vector value assigned for monthly weight

Note

By using set_psi() and set_lambda() each, hierarchical modeling is available.
References


See Also

• lambda hyperprior specification set_lambda()
• sigma hyperprior specification set_psi()

Examples

# Minnesota BVAR specification------------------------
bvar_spec <- set_bvar(
sigma = c(.03, .02, .01), # Sigma = diag(.03^2, .02^2, .01^2)
lambda = .2, # lambda = .2
delta = rep(.1, 3), # delta_1 = .1, delta_2 = .1, delta_3 = .1
eps = 1e-04 # eps = 1e-04
)
class(bvar_spec)
str(bvar_spec)
# Flat BVAR specification-------------------------
# 3-dim
# p = 5 with constant term
# U = 500 * I(mp + 1)
bvar_flat_spec <- set_bvar_flat(U = 500 * diag(16))
class(bvar_flat_spec)
str(bvar_flat_spec)
# BVHAR-S specification-----------------------
bvarh_var_spec <- set_bvhar(
sigma = c(.03, .02, .01), # Sigma = diag(.03^2, .02^2, .01^2)
lambda = .2, # lambda = .2
delta = rep(.1, 3), # delta_1 = .1, delta_2 = .1, delta_3 = .1
eps = 1e-04 # eps = 1e-04
)
class(bvarh_var_spec)
str(bvarh_var_spec)
# BVHAR-L specification---------------------------
bvarh_vhar_spec <- set_weight_bvhar(
sigma = c(.03, .02, .01), # Sigma = diag(.03^2, .02^2, .01^2)
lambda = .2, # lambda = .2
eps = 1e-04, # eps = 1e-04
)
```r
daily = rep(.2, 3), # daily1 = .2, daily2 = .2, daily3 = .2
weekly = rep(.1, 3), # weekly1 = .1, weekly2 = .1, weekly3 = .1
monthly = rep(.05, 3) # monthly1 = .05, monthly2 = .05, monthly3 = .05
}
class(bvhar_vhar_spec)
str(bvhar_vhar_spec)
```

---

### set_lambda

#### Description

Set hyperpriors of Bayesian VAR and VHAR models.

#### Usage

```r
set_lambda(mode = 0.2, sd = 0.4, lower = 1e-05, upper = 3)
set_psi(shape = 4e-04, scale = 4e-04, lower = 1e-05, upper = 3)
```

#### Arguments

- `mode`: Mode of Gamma distribution. By default, .2.
- `sd`: Standard deviation of Gamma distribution. By default, .4.
- `lower`: [Experimental] Lower bound for `stats::optim()`. By default, 1e-5.
- `shape`: Shape of Inverse Gamma distribution. By default, (0.02)^2.
- `scale`: Scale of Inverse Gamma distribution. By default, (0.02)^2.
- `x`: bvharpriorspec object
- `digits`: digit option to print
- `...`: not used

#### Details

In addition to Normal-IW priors `set_bvar()`, `set_bvhar()`, and `set_weight_bvhar()`, these functions give hierarchical structure to the model.

- `set_lambda()` specifies hyperprior for $\lambda$ (lambda), which is Gamma distribution.
- `set_psi()` specifies hyperprior for $\psi/(\nu_0 - k - 1) = \sigma^2$ (sigma), which is Inverse gamma distribution.
The following set of \((\text{mode}, \text{sd})\) are recommended by Sims and Zha (1998) for \text{set\_lambda}().

- \((\text{mode} = .2, \text{sd} = .4)\): default
- \((\text{mode} = 1, \text{sd} = 1)\)

Giannone et al. (2015) suggested data-based selection for \text{set\_psi}(). It chooses \((0.02)^2\) based on its empirical data set.

**Value**

\text{bvharpriorspec} object

**References**


**Examples**

```r
# Hierarchical BVAR specification------------------------
set_bvar(
  sigma = set_psi(shape = 4e-4, scale = 4e-4),
  lambda = set_lambda(mode = .2, sd = .4),
  delta = rep(1, 3),
  eps = 1e-04 # eps = 1e-04
)
```

**set\_ssvs**

*Stochastic Search Variable Selection (SSVS) Hyperparameter for Coefficients Matrix and Cholesky Factor*

**Description**

Set SSVS hyperparameters for VAR or VHAR coefficient matrix and Cholesky factor.

**Usage**

```r
set\_ssvs(
  coef\_spike = 0.1,
  coef\_slab = 5,
  coef\_mixture = 0.5,
  mean\_non = 0,
  sd\_non = 0.1,
  shape = 0.01,
  rate = 0.01,
  chol\_spike = 0.1,
  chol\_slab = 5,
  chol\_mixture = 0.5
)
```
## S3 method for class 'ssvsinput'
print(x, digits = max(3L, getOption("digits") - 3L), ...)

knit_print.ssvsinput(x, ...)

### Arguments

- **coef_spike**: Standard deviation for Spike normal distribution (See Details).
- **coef_slab**: Standard deviation for Slab normal distribution (See Details).
- **coef_mixture**: Bernoulli parameter for sparsity proportion (See Details).
- **mean_non**: Prior mean of unrestricted coefficients
- **sd_non**: Standard deviation for unrestricted coefficients
- **shape**: Gamma shape parameters for precision matrix (See Details).
- **rate**: Gamma rate parameters for precision matrix (See Details).
- **chol_spike**: Standard deviation for Spike normal distribution, in the cholesky factor (See Details).
- **chol_slab**: Standard deviation for Slab normal distribution, in the cholesky factor (See Details).
- **chol_mixture**: Bernoulli parameter for sparsity proportion, in the cholesky factor (See Details).
- **x**: ssvsinput
- **digits**: digit option to print
- **...**: not used

### Details

Let $\alpha$ be the vectorized coefficient, $\alpha = \text{vec}(A)$. Spike-slab prior is given using two normal distributions.

$$\alpha_j \mid \gamma_j \sim (1 - \gamma_j)N(0, \tau_{0j}^2) + \gamma_j N(0, \tau_{1j}^2)$$

As spike-slab prior itself suggests, set $\tau_{0j}$ small (point mass at zero: spike distribution) and set $\tau_{1j}$ large (symmetric by zero: slab distribution).

$\gamma_j$ is the proportion of the nonzero coefficients and it follows

$$\gamma_j \sim \text{Bernoulli}(p_j)$$

- **coef_spike**: $\tau_{0j}$
- **coef_slab**: $\tau_{1j}$
- **coef_mixture**: $p_j$
- **$j = 1, \ldots, mk$**: vectorized format corresponding to coefficient matrix
- **If one value is provided, model function will read it by replicated value.**
- **coef_non**: vectorized constant term is given prior Normal distribution with variance $cI$. Here, coef_non is $\sqrt{c}$.
Next for precision matrix $\Sigma^{-1}_e$, SSVS applies Cholesky decomposition.

$$\Sigma^{-1}_e = \Psi \Psi^T$$

where $\Psi = \{\psi_{ij}\}$ is upper triangular.

Diagonal components follow the gamma distribution.

$$\psi_{jj}^2 \sim \text{Gamma}(\text{shape} = a_j, \text{rate} = b_j)$$

For each row of off-diagonal (upper-triangular) components, we apply spike-slab prior again.

$$\psi_{ij} | w_{ij} \sim \left(1 - w_{ij}\right)N(0, \kappa_{0,ij}^2) + w_{ij}N(0, \kappa_{1,ij}^2)$$

$$w_{ij} \sim \text{Bernoulli}(q_{ij})$$

- **shape**: $a_j$
- **rate**: $b_j$
- **chol_spike**: $\kappa_{0,ij}$
- **chol_slab**: $\kappa_{1,ij}$
- **chol_mixture**: $q_{ij}$
- **$j = 1, \ldots, mk$**: vectorized format corresponding to coefficient matrix
- **$i = 1, \ldots, j-1$ and $j = 2, \ldots, m$**: $\eta = (\psi_{12}, \psi_{13}, \psi_{23}, \psi_{14}, \ldots, \psi_{34}, \ldots, \psi_{1m}, \ldots, \psi_{m-1,m})^T$
- **chol_** arguments can be one value for replication, vector, or upper triangular matrix.

**Value**

ssvsinput object

**References**


**Description**

This function samples one matrix IW matrix.

**Usage**

\[ \text{sim_iw(mat_scale, shape)} \]

**Arguments**

- mat_scale: Scale matrix
- shape: Shape

**Details**

Consider \( \Sigma \sim IW(\Psi, \nu) \).

1. Upper triangular Bartlett decomposition: k x k matrix \( Q = [q_{ij}] \) upper triangular with
   (a) \( q_{ii} \chi^2_{\nu-i+1} \)
   (b) \( q_{ij} \sim N(0, 1) \) with \( i < j \) (upper triangular)
2. Lower triangular Cholesky decomposition: \( \Psi = LL^T \)
3. \( A = L(Q^{-1})^T \)
4. \( \Sigma = AA^T \sim IW(\Psi, \nu) \)

**Value**

One k x k matrix following IW distribution

---

**sim_matgaussian**

*Generate Matrix Normal Random Matrix*

**Description**

This function samples one matrix gaussian matrix.

**Usage**

\[ \text{sim_matgaussian(mat_mean, mat_scale_u, mat_scale_v)} \]
Arguments

- `mat_mean`: Mean matrix
- `mat_scale_u`: First scale matrix
- `mat_scale_v`: Second scale matrix

Details

Consider an $n \times k$ matrix $Y_1, \ldots, Y_n \sim MN(M, U, V)$ where $M$ is $n \times k$, $U$ is $n \times n$, and $V$ is $k \times k$.

1. Lower triangular Cholesky decomposition: $U = PP^T$ and $V = LL^T$
2. Standard normal generation: $s \times m$ matrix $Z_i = [z_{ij} \sim N(0, 1)]$ in row-wise direction.
3. $Y_i = M + PZ_iL^T$

This function only generates one matrix, i.e. $Y_1$.

Value

One $n \times k$ matrix following MN distribution.

---

**sim_mncoef**

*Generate Minnesota BVAR Parameters*

**Description**

This function generates parameters of BVAR with Minnesota prior.

**Usage**

```
sim_mncoef(p, bayes_spec = set_bvar(), full = TRUE)
```

**Arguments**

- `p`: VAR lag
- `bayes_spec`: A BVAR model specification by `set_bvar()`.
- `full`: Generate variance matrix from IW (default: TRUE) or not (FALSE)?

**Details**

Implementing dummy observation constructions, Banbura et al. (2010) sets Normal-IW prior.

$$ A \mid \Sigma_e \sim MN(A_0, \Omega_0, \Sigma_e) $$

$$ \Sigma_e \sim IW(S_0, \alpha_0) $$

If `full = FALSE`, the result of $\Sigma_e$ is the same as input $(\text{diag}(\sigma))$. 
Value

List with the following component.

coefficients BVAR coefficient (MN)
covmat BVAR variance (IW or diagonal matrix of sigma of bayes_spec)

References


See Also

• set_bvar() to specify the hyperparameters of Minnesota prior.
• bvar_adding_dummy for dummy observations definition.

Examples

# Generate (A, Sigma)
# BVAR(p = 2)
# sigma: 1, 1, 1
# lambda: .1
# delta: .1, .1, .1
# epsilon: 1e-04
set.seed(1)
sim_mncoef(
    p = 2,
    bayes_spec = set_bvar(       
        sigma = rep(1, 3),
        lambda = .1,
        delta = rep(.1, 3),
        eps = 1e-04
    ),
    full = TRUE
)

Description

This function samples normal inverse-wishart matrices.
Usage

```r
sim_mniw(num_sim, mat_mean, mat_scale_u, mat_scale, shape)
```

Arguments

- `num_sim` Number to generate
- `mat_mean` Mean matrix of MN
- `mat_scale_u` First scale matrix of MN
- `mat_scale` Scale matrix of IW
- `shape` Shape of IW

Details

Consider \((Y_i, \Sigma_i) \sim MIW(M, U, \Psi, \nu)\).

1. Generate upper triangular factor of \(\Sigma_i = C_i C_i^T\) in the upper triangular Bartlett decomposition.
2. Standard normal generation: \(n \times k\) matrix \(Z_i = \{z_{ij} \sim N(0, 1)\}\) in row-wise direction.
3. Lower triangular Cholesky decomposition: \(U = PP^T\)
4. \(A_i = M + PZ_i C_i^T\)

Value

List of MN and IW matrices. Multiple samples are column-stacked.

---

**sim_mnormal**

Generate Multivariate Normal Random Vector

Description

This function samples \(n \times\) multi-dimensional normal random matrix.

Usage

```r
sim_mnormal(
  num_sim, 
  mu = rep(0, 5), 
  sig = diag(5), 
  method = c("eigen", "chol")
)
```

Arguments

- `num_sim` Number to generate process
- `mu` Mean vector
- `sig` Variance matrix
- `method` Method to compute \(\Sigma^{1/2}\). Choose between "eigen" (spectral decomposition) and "chol" (cholesky decomposition). By default, "eigen".
Details

Consider \( x_1, \ldots, x_n \sim N_m(\mu, \Sigma) \).

1. Lower triangular Cholesky decomposition: \( \Sigma = LL^T \)
2. Standard normal generation: \( Z_{i1}, Z_{in} \sim iid N(0, 1) \)
3. \( Z_i = (Z_{i1}, \ldots, Z_{in})^T \)
4. \( X_i = L Z_i + \mu \)

Value

T x k matrix

---

**sim_mnvhar_coef**  
*Generate Minnesota BVAR Parameters*

Description

This function generates parameters of BVAR with Minnesota prior.

Usage

```
sim_mnvhar_coef(bayes_spec = set_bvhar(), full = TRUE)
```

Arguments

- **bayes_spec** A BVHAR model specification by `set_bvhar()` (default) or `set_weight_bvhar()`.
- **full** Generate variance matrix from IW (default: TRUE) or not (FALSE)?

Details

Normal-IW family for vector HAR model:

\[
\Phi | \Sigma_e \sim MN(M_0, \Omega_0, \Sigma_e) \\
\Sigma_e \sim IW(\Psi_0, \nu_0)
\]

Value

List with the following component.

- **coefficients** BVHAR coefficient (MN)
- **covmat** BVHAR variance (IW or diagonal matrix of sigma of bayes_spec)

References

See Also

- `set_bvhar()` to specify the hyperparameters of VAR-type Minnesota prior.
- `set_weight_bvhar()` to specify the hyperparameters of HAR-type Minnesota prior.
- `bvar_adding_dummy` for dummy observations definition.

Examples

```r
# Generate (Phi, Sigma)
# BVHAR-S
# sigma: 1, 1, 1
# lambda: .1
# delta: .1, .1, .1
# epsilon: 1e-04
set.seed(1)
sim_mnvhar_coef(
  bayes_spec = set_bvhar(
    sigma = rep(1, 3),
    lambda = .1,
    delta = rep(.1, 3),
    eps = 1e-04
  ),
  full = TRUE
)
```

---

**sim_mvt**  
*Generate Multivariate t Random Vector*

**Description**

This function samples n x multi-dimensional t-random matrix.

**Usage**

```r
sim_mvt(num_sim, df, mu, sig, method = c("eigen", "chol"))
```

**Arguments**

- `num_sim` Number to generate process.
- `df` Degrees of freedom.
- `mu` Location vector
- `sig` Scale matrix.
- `method` Method to compute $\Sigma^{1/2}$. Choose between "eigen" (spectral decomposition) and "chol" (cholesky decomposition). By default, "eigen".

**Value**

T x k matrix
**sim_ssvs_var**

**Generate SSVS Parameters**

**Description**

This function generates parameters of VAR with SSVS prior.

**Usage**

```r
sim_ssvs_var(
  bayes_spec,
  p,
  dim_data = NULL,
  include_mean = TRUE,
  relax = FALSE,
  method = c("eigen", "chol")
)
```

```r
sim_ssvs_vhar(
  bayes_spec,
  har = c(5, 22),
  dim_data = NULL,
  include_mean = TRUE,
  relax = c("no", "minnesota", "longrun"),
  method = c("eigen", "chol")
)
```

**Arguments**

- **bayes_spec**: A SSVS model specification by `set_ssvs()`.
- **p**: VAR lag
- **dim_data**: Specify the dimension of the data if hyperparameters of `bayes_spec` have constant values.
- **include_mean**: Add constant term (Default: TRUE) or not (FALSE)
- **relax**: Only use off-diagonal terms of each coefficient matrices for restriction. In `sim_ssvs_var()` function, use TRUE or FALSE (default). In `sim_ssvs_vhar()` function, "no" (default), "minnesota" type, or "longrun" type.
- **method**: Method to compute $\Sigma^{1/2}$.
- **har**: Numeric vector for weekly and monthly order. By default, c(5, 22).

**Value**

List including coefficients.
VAR(p) with SSVS prior

Let $\alpha$ be the vectorized coefficient of $\text{VAR}(p)$.

$$(\alpha | \gamma)$$

$$(\gamma_i)$$

$$(\eta_j | \omega_j)$$

$$(\omega_{ij})$$

$$(\psi_{ii}^2)$$

VHAR with SSVS prior

Let $\phi$ be the vectorized coefficient of $\text{VHAR}$.

$$(\phi | \gamma)$$

$$(\gamma_i)$$

$$(\eta_j | \omega_j)$$

$$(\omega_{ij})$$

$$(\psi_{ii}^2)$$

References


**Description**

This function generates multivariate time series dataset that follows VAR(p).

**Usage**

```r
sim_var(
  num_sim,
  num_burn,
  var_coef,
  var_lag,
  sig_error = diag(ncol(var_coef)),
  init = matrix(0L, nrow = var_lag, ncol = ncol(var_coef)),
  method = c("eigen", "chol"),
  process = c("gaussian", "student"),
  t_param = 5
)
```

**Arguments**

- `num_sim`: Number to generated process
- `num_burn`: Number of burn-in
- `var_coef`: VAR coefficient. The format should be the same as the output of `coef.varlse()` from `var_lm()`
- `var_lag`: Lag of VAR
- `sig_error`: Variance matrix of the error term. By default, `diag(dim)`. 
- `init`: Initial y1, ..., yp matrix to simulate VAR model. Try `matrix(0L, nrow = var_lag, ncol = ncol(var_coef))`
- `method`: Method to compute $\Sigma^{1/2}$. Choose between "eigen" (spectral decomposition) and "chol" (cholosky decomposition). By default, "eigen".
- `process`: Process to generate error term. "gaussian": Normal distribution (default) or "student": Multivariate t-distribution.
- `t_param`: [Experimental] argument for MVT, e.g. DF: 5.

**Details**

1. Generate $\epsilon_1, \epsilon_n \sim N(0, \Sigma)$
2. For $i = 1, \ldots, n$,
   
   $$y_{p+i} = (y_{p+i-1}, \ldots, y_i^T, 1)^T B + \epsilon_i$$

3. Then the output is $(y_{p+1}, \ldots, y_{n+p})^T$

Initial values might be set to be zero vector or $(I_m - A_1 - \cdots - A_p)^{-1} c$. 
Value

T x k matrix

References


---

**sim_vhar**

*Generate Multivariate Time Series Process Following VAR(p)*

Description

This function generates multivariate time series dataset that follows VAR(p).

Usage

```r
sim_vhar(
  num_sim,
  num_burn,
  vhar_coef,
  week = 5L,
  month = 22L,
  sig_error = diag(ncol(vhar_coef)),
  init = matrix(0L, nrow = month, ncol = ncol(vhar_coef)),
  method = c("eigen", "chol"),
  process = c("gaussian", "student"),
  t_param = 5
)
```

Arguments

- **num_sim** Number to generated process
- **num_burn** Number of burn-in
- **vhar_coef** VAR coefficient. The format should be the same as the output of `coef.varlse()` from `var.lm()`
- **week** Weekly order of VHAR. By default, 5.
- **month** Weekly order of VHAR. By default, 22.
- **sig_error** Variance matrix of the error term. By default, `diag(dim)`.
- **init** Initial y1,...,yp matrix to simulate VAR model. Try matrix(0L, nrow = month, ncol = dim).
- **method** Method to compute $\Sigma^{1/2}$. Choose between "eigen" (spectral decomposition) and "chol" (cholesky decomposition). By default, "eigen".
- **process** Process to generate error term. "gaussian": Normal distribution (default) or "student": Multivariate t-distribution.
- **t_param** *[Experimental]* argument for MVT, e.g. DF: 5.
Details

Let $M$ be the month order, e.g. $M = 22$. 

1. Generate $\epsilon_1, \epsilon_n \sim N(0, \Sigma)$
2. For $i = 1, \ldots, n$,
   $$y_{M+i} = (y_{M+i-1}^T, \ldots, y_i^T, 1)^T C_{HAR}^T \Phi + \epsilon_i$$
3. Then the output is $(y_{M+1}, \ldots, y_{n+M})^T$
4. For $i = 1, \ldots, n$,
   $$y_{p+i} = (y_{p+i-1}^T, \ldots, y_i^T, 1)^T B + \epsilon_i$$
5. Then the output is $(y_{p+1}, \ldots, y_{n+p})^T$

Initial values might be set to be zero vector or $(I_m - A_1 - \cdots - A_p)^{-1} c$.

Value

$T \times k$ matrix

References


---

**split_coef**

*Splitting Coefficient Matrix into List*

**Description**

Split coefficients into matrix list.

**Usage**

```r
split_coef(object, ...)
```

### S3 method for class `bvharmod`

```r
split_coef(object, ...)
```

### S3 method for class `bvharirf`

```r
split_coef(object, ...)
```

**Arguments**

- **object**: `bvharmod` object
- **...**: not used

**Details**

Each result of `var_lm()`, `vhar_lm()`, `bvar_minnesota()`, `bvar_flat()`, and `bvhar_minnesota()` is a subclass of `bvharmod`. For example, `c("varlse", "bvharmod")`. 
spne

Evaluate the Estimation Based on Spectral Norm Error

Description

This function computes estimation error given estimated model and true coefficient.

Usage

spne(x, y, ...)

## S3 method for class 'bvhar.sp'
spne(x, y, ...)

Arguments

- **x**: Estimated model.
- **y**: Coefficient matrix to be compared.
- **...**: not used

Details

Let $\|\cdot\|_2$ be the spectral norm of a matrix, let $\hat{\Phi}$ be the estimates, and let $\Phi$ be the true coefficients matrix. Then the function computes estimation error by

$$\|\hat{\Phi} - \Phi\|_2$$

Value

Spectral norm value

References

stableroot

---

**Description**

Roots of characteristic polynomial

**Usage**

```r
stableroot(x, ...)
```

**Arguments**

- `x`: object
- `...`: not used

**Value**

Numeric vector.

---

stableroot.varlse

---

**Description**

Characteristic polynomial roots for VAR Coefficient Matrix

**Usage**

```r
## S3 method for class 'varlse'
stableroot(x, ...)

## S3 method for class 'vharlse'
stableroot(x, ...)

## S3 method for class 'bvarmn'
stableroot(x, ...)

## S3 method for class 'bvarflat'
stableroot(x, ...)

## S3 method for class 'bvharmn'
stableroot(x, ...)
```
Arguments

x  Model fit
... not used

Details

To know whether the process is stable or not, make characteristic polynomial.

\[ \det(I_m - A z) = 0 \]

where \( A \) is VAR(1) coefficient matrix representation.

Value

Numeric vector.

References


summarizing Bayesian Multivariate Time Series Model

Description

summary method for normaliw class.

Usage

```r
## S3 method for class 'normaliw'
summary(
  object,
  num_iter = 10000L,
  num_burn = floor(num_iter/2),
  thinning = 1L,
  ...
)
```

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

```r
knit_print.summary.normaliw(x, ...)
```
Arguments

object
num_iter
num_burn
thinning
... x
digits

Details

From Minnesota prior, set of coefficient matrices and residual covariance matrix have matrix Normal Inverse-Wishart distribution.

BVAR:

\[(A, \Sigma_e) \sim MNIW(\hat{A}, \hat{V}^{-1}, \hat{\Sigma}_e, \alpha_0 + n)\]

where \(\hat{V} = X^T X_e\) is the posterior precision of MN.

BVHAR:

\[(\Phi, \Sigma_e) \sim MNIW(\hat{\Phi}, \hat{V}_{H}^{-1}, \hat{\Sigma}_e, \nu + n)\]

where \(\hat{V}_{H} = X_+^T X_+\) is the posterior precision of MN.

Value

`summary.normaliw` class has the following components:

- names Variable names
- totobs Total number of observation
- obs Sample size used when training = totobs - p
- p Lag of VAR
- m Dimension of the data
- call Matched call
- spec Model specification (bvhspec)
- mn_mean MN Mean of posterior distribution (MN-IW)
- mn_prec MN Precision of posterior distribution (MN-IW)
- iw_scale IW scale of posterior distribution (MN-IW)
- iw_shape IW df of posterior distribution (MN-IW)
- iter Number of MCMC iterations
- burn Number of MCMC burn-in
- thin MCMC thinning
alpha_record (BVAR) and phi_record (BVHAR) MCMC record of coefficients vector
psi_record MCMC record of upper cholesky factor
omega_record MCMC record of diagonal of cholesky factor
eta_record MCMC record of upper part of cholesky factor
param MCMC record of every parameter
coefficients Posterior mean of coefficients
covmat Posterior mean of covariance

References


**summary.varlse**

*Summarizing Vector Autoregressive Model*

**Description**

summary method for varlse class.

**Usage**

```r
## S3 method for class 'varlse'
summary(object, ...)

## S3 method for class 'summary.varlse'
print(x, digits = max(3L,getOption("digits") - 3L), signif_code = TRUE, ...)

knit_print.summary.varlse(x, ...)
```

**Arguments**

- **object** varlse object
- **...** not used
- **x** summary.varlse object
- **digits** digit option to print
- **signif_code** Check significant rows (Default: TRUE)
Summary

The `summary.vharlse` function computes the following:

- **names**: Variable names
- **totobs**: Total number of observations
- **obs**: Sample size used when training = totobs - p
- **p**: Lag of VAR
- **coefficients**: Coefficient Matrix
- **call**: Matched call
- **process**: Process: VAR
- **covmat**: Covariance matrix of the residuals
- **corrmat**: Correlation matrix of the residuals
- **roots**: Roots of characteristic polynomials
- **is_stable**: Whether the process is stable or not based on roots
- **log_lik**: log-likelihood
- **ic**: Information criteria vector
  - AIC - AIC
  - BIC - BIC
  - HQ - HQ
  - FPE - FPE

References


**summary.vharlse**

*Summarizing Vector HAR Model*

**Description**

The `summary` method for `vharlse` class.

**Usage**

```r
## S3 method for class 'vharlse'
summary(object, ...)

## S3 method for class 'summary.vharlse'
print(x, digits = max(3L,getOption("digits") - 3L), signif_code = TRUE, ...)

knit_print.summary.vharlse(x, ...)
```
### Arguments

- **object**: `vharlse` object
- **...**: not used
- **x**: `summary.vharlse` object
- **digits**: digit option to print
- **signif_code**: Check significant rows (Default: TRUE)

### Value

`summary.vharlse` class additionally computes the following:

- **names**: Variable names
- **totobs**: Total number of the observation
- **obs**: Sample size used when training = `totobs - p`
- **p**: 3
- **week**: Order for weekly term
- **month**: Order for monthly term
- **coefficients**: Coefficient Matrix
- **call**: Matched call
- **process**: Process: VAR
- **covmat**: Covariance matrix of the residuals
- **corrmat**: Correlation matrix of the residuals
- **roots**: Roots of characteristic polynomials
- **is_stable**: Whether the process is stable or not based on roots
- **log_lk**: log-likelihood
- **ic**: Information criteria vector
  - AIC - AIC
  - BIC - BIC
  - HQ - HQ
  - FPE - FPE

### References


VARtoVMA

Convert VAR to VMA(infinite)

Description

Convert VAR process to infinite vector MA process

Usage

VARtoVMA(object, lag_max)

Arguments

object varlse object
lag_max Maximum lag for VMA

Details

Let VAR(p) be stable.

\[ Y_t = c + \sum_{j=0}^{\infty} W_j Z_{t-j} \]

For VAR coefficient \(B_1, B_2, \ldots, B_p\),

\[ I = (W_0 + W_1 L + W_2 L^2 + \cdots + 1)(I - B_1 L - B_2 L^2 - \cdots - B_p L^p) \]

Recursively,

\[ W_0 = I \]
\[ W_1 = W_0 B_1 (W_1^T = B_1^T W_0^T) \]
\[ W_2 = W_1 B_1 + W_0 B_2 (W_2^T = B_1^T W_1^T + B_2^T W_0^T) \]
\[ W_j = \sum_{k=1}^{\infty} W_{k-j} B_j (W_j^T = \sum_{k=1}^{\infty} B_j^T W_{k-j}^T) \]

Value

VMA coefficient of \(k(lag-max + 1) \times k\) dimension

References

This function fits VAR(p) using OLS method.

**Usage**

```r
var_lm(y, p = 1, include_mean = TRUE, method = c("nor", "chol", "qr"))
```

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

```r
knit_print.varlse(x, ...)
```

**Arguments**

- `y` Time series data of which columns indicate the variables
- `p` Lag of VAR (Default: 1)
- `include_mean` Add constant term (Default: TRUE) or not (FALSE)
- `method` Method to solve linear equation system. ("nor": normal equation (default), "chol": Cholesky, and "qr": HouseholderQR)
- `x` varlse object
- `digits` digit option to print
- `...` not used

**Details**

This package specifies VAR(p) model as

\[ Y_t = A_1 Y_{t-1} + \cdots + A_p Y_{t-p} + c + \epsilon_t \]

If `include_type = TRUE`, there is `c` term. Otherwise (`include_type = FALSE`), there is no `c` term.

The function estimates every coefficient matrix \( A_1, \ldots, A_p, c \).

- Response matrix, \( Y_0 \) in `var_design_formulation`
- Design matrix, \( X_0 \) in `var_design_formulation`
- Coefficient matrix is the form of \( A = [A_1, A_2, \ldots, A_p, c]^T \).

Then perform least squares to the following multivariate regression model

\[ Y_0 = X_0 A + \text{error} \]

which gives

\[ \hat{A} = (X_0^T X_0)^{-1} X_0^T Y_0 \]
Value

`var_lm()` returns an object named `varlse` class. It is a list with the following components:

- **coefficients**  Coefficient Matrix
- **fitted.values**  Fitted response values
- **residuals**  Residuals
- **covmat**  LS estimate for covariance matrix
- **df**  Number of Coefficients: \( mp + 1 \) or \( mp \)
- **p**  Lag of VAR
- **m**  Dimension of the data
- **obs**  Sample size used when training = \( \text{totobs} - p \)
- **totobs**  Total number of the observation
- **call**  Matched call
- **process**  Process: VAR
- **type**  include constant term ("const") or not ("none")
- **y0**  \( Y_0 \)
- **design**  \( X_0 \)
- **y**  Raw input

It is also a `bvharmod` class.

References


See Also

- `coef.varlse()`, `residuals.varlse()`, and `fitted.varlse()`
- `summary.varlse()` to summarize VAR model
- `predict.varlse()` to forecast the VAR process
- `var_design_formulation` for the model design

Examples

```r
# Perform the function using etf_vix dataset
fit <- var_lm(y = etf_vix, p = 2)
class(fit)
str(fit)

# Extract coef, fitted values, and residuals
coef(fit)
head(residuals(fit))
head(fitted(fit))
```
**VHARtoVMA**

*Convert VHAR to VMA(infinite)*

**Description**

Convert VHAR process to infinite vector MA process

**Usage**

VHARtoVMA(object, lag_max)

**Arguments**

- **object**: vharlse object
- **lag_max**: Maximum lag for VMA

**Details**

Let \( VAR(p) \) be stable and let \( VAR(p) \) be \( Y_0 = X_0B + Z \)

VHAR is \( VAR(22) \) with

\[
Y_0 = X_1B + Z = (X_0\tilde{T})\Phi + Z
\]

Observe that

\[
B = \tilde{T}\Phi
\]

**Value**

VMA coefficient of \( k(lag-max + 1) \times k \) dimension

**References**


---

**vhar_lm**

*Fitting Vector Heterogeneous Autoregressive Model*

**Description**

This function fits VHAR using OLS method.
vhar_lm

Usage

vhar_lm(
  y,
  har = c(5, 22),
  include_mean = TRUE,
  method = c("nor", "chol", "qr")
)

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

knit_print.vharlse(x, ...)

Arguments

y  Time series data of which columns indicate the variables
har Numeric vector for weekly and monthly order. By default, c(5, 22).
include_mean Add constant term (Default: TRUE) or not (FALSE)
method Method to solve linear equation system. ("nor": normal equation (default),
          "chol": Cholesky, and "qr": HouseholderQR)
x  vharlse object
digits digit option to print
... not used

Details

For VHAR model

\[ Y_t = \Phi^{(d)} Y_{t-1} + \Phi^{(w)} Y^{(w)}_{t-1} + \Phi^{(m)} Y^{(m)}_{t-1} + \epsilon_t \]

the function gives basic values.

Value

vhar_lm() returns an object named vharlse class. It is a list with the following components:

- coefficients Coefficient Matrix
- fitted.values Fitted response values
- residuals Residuals
- covmat LS estimate for covariance matrix
- df Numer of Coefficients: 3m + 1 or 3m
- p 3 (The number of terms. vharlse contains this element for usage in other functions.)
- week Order for weekly term
- month Order for monthly term
- m Dimension of the data
- obs Sample size used when training = totobs - 22
**totalobs**  Total number of the observation

**call**  Matched call

**process**  Process: VHAR

**type**  include constant term ("const") or not ("none")

**HARtrans**  VHAR linear transformation matrix: $C_{HAR}$

$y_0$  $Y_0$

**design**  $X_0$

$y$  Raw input

It is also a bvharmod class.

References


See Also

- `coef.vharlse()`, `residuals.vharlse()`, and `fitted.vharlse()`
- `summary.vharlse()` to summarize VHAR model
- `predict.vharlse()` to forecast the VHAR process

Examples

# Perform the function using etf_vix dataset
fit <- vhar_lm(y = etf_vix)
class(fit)
str(fit)

# Extract coef, fitted values, and residuals
coef(fit)
head(residuals(fit))
fitted(fit)
Index

* datasets
  
etf_vix, 43
  oxfordman, 64

AIC.bvarflat (AIC.varlse), 4
AIC.bvarmn (AIC.varlse), 4
AIC.bvharmn (AIC.varlse), 4
AIC.varlse, 4
analyze_ir (analyze_ir.varlse), 5
analyze_ir(), 7
analyze_ir.varlse, 5
autolayer.predbvhar
  (autoplot.predbvhar), 9
autoplot.bvharirf, 7
autoplot.bvharsp, 7
autoplot.normaliw, 8
autoplot.summary.bvharsp, 10
autoplot.summary.normaliw, 10

bayesplot::mcmc_areas(), 8, 11
bayesplot::mcmc_dens(), 8, 11
bayesplot::mcmc_trace(), 8, 11
BIC.bvarflat (BIC.varlse), 11
BIC.bvarmn (BIC.varlse), 11
BIC.bvharmn (BIC.varlse), 11
BIC.varlse, 11

bound_bvhar, 12
bound_bvharsp(), 30
bvar.adding.dummy, 87, 90
bvar.flat, 13
bvar.flat(), 59, 79, 95
bvar.minnesota, 15
bvar.minnesotar(), 30, 59, 78, 95
bvar.niwhm, 17
bvar.predictive_density, 69
bvar.ssvs, 19
bvar_sv, 22

bvharmn, 23
bvharmn(), 30, 59, 79, 95
bvar_ssvs, 25
bvar_sv, 28

choose_bayes, 29
choose_bayes(), 30
choose_bvar, 30
choose_bvar(), 30
choose_bvharr (choose_bvar), 30
choose_ssvs, 32
choose_ssvs(), 20, 26
choose_var, 34

class, 6, 13, 14, 16, 18, 20, 23, 24, 27, 29, 30,
32, 46, 47, 56, 67, 79, 99, 101, 102,
105, 107

coeff.bvvarflat (coeff.varlse), 34
coeff.bvvarflat(), 15
coeff.bvvarmn (coeff.varlse), 34
coeff.bvvarmn(), 17
coeff.bvvarmn(coeff.varlse), 34
coeff.bvvarmn(), 25
coeff.bvharrsp (coeff.varlse), 34
coeff.summary.bvharrsp (coeff.varlse), 34
coeff.varlse, 34
coeff.varlse(), 93, 94, 105
coeff.vharrse (coeff.varlse), 34
coeff.vharrse(), 108

compute_dic, 35
compute_logml, 36
conf.fdr, 38
conf_fnr, 39
conf.fscore, 40
conf.prec, 41
conf_recall, 42
confusion, 37
confusion(), 38–42

data(), 66

divide_ts, 43
divide_ts(), 46, 47
etf_vix, 43
fitted.bvarflat (fitted.varlse), 45
fitted.bvarflat(), 15
fitted.bvarmn (fitted.varlse), 45
fitted.bvarmn(), 17
fitted.bvharmn (fitted.varlse), 45
fitted.bvharmn(), 25
fitted.varlse, 45
fitted.varlse(), 105
fitted.vharlse (fitted.varlse), 45
fitted.vharlse(), 108
foreach::foreach(), 34
forecast_expand, 45
forecast_roll, 46
FPE, 47
FPE.varlse, 48
FPE.vharlse (FPE.varlse), 48
fromse, 49
geom_eval, 50
gg_loss, 50
ggplot2::facet_grid(), 9
ggplot2::facet_wrap(), 9, 51
ggplot2::geom_hline(), 51
ggplot2::geom_path(), 7, 9, 50, 51
ggplot2::geom_tile(), 10
ggplot2::scale_colour_viridis_d, 9, 51
ggplot2::scale_fill_viridis_d(), 9, 51
HQ, 51
HQ.bvarflat (HQ.varlse), 52
HQ.bvarmn (HQ.varlse), 52
HQ.bvharmn (HQ.varlse), 52
HQ.varlse, 52
HQ.vharlse (HQ.varlse), 52
init_ssvs, 53
init_ssvs(), 20, 21, 26, 27
is.boundbharemmp (is.varlse), 56
is.bvarflat (is.varlse), 56
is.bvarmn (is.varlse), 56
is.bvharmn (is.varlse), 56
is.bvharemp (is.varlse), 56
is.bvharmn (is.varlse), 56
is.bvharpriorspec (is.varlse), 56
is.bvharspec (is.varlse), 56
is.predbharm (is.varlse), 56
is.ssvsinit (is.varlse), 56
is.ssvsinput (is.varlse), 56
is.stable, 55
is.stable.bvarflat (is.stable.varlse),
   55
is.stable.bvarmn (is.stable.varlse), 55
is.stable.bvharmn (is.stable.varlse), 55
is.stable.varlse, 55
is.stable.vharlse (is.stable.varlse), 55
is.varlse, 56
is.vharlse (is.varlse), 56
knit_print.boundbharemmp (bound_bharm), 12
knit_print.bvarflat (bvar_flat), 13
knit_print.bvarhm (bvar_minnesota), 17
knit_print.bvarmn (bvar_minnesota), 15
knit_print.bvarssvs (bvar_ssvs), 19
knit_print.bvarsv (bvar_sv), 22
knit_print.bvharcv (forecast_roll), 46
knit_print.bvharemmp (choose_bvar), 30
knit_print.bvharirf (analyze_ir.varlse), 5
knit_print.bvharmn (bvharm_minnesota), 23
knit_print.bvharpriorspec (set_lambda), 81
knit_print.bvharspec (set_bvar), 78
knit_print.bvharsvs (bvharm_ssvs), 25
knit_print.bvharsv (bvharm_sv), 28
knit_print.bvharemmp (predict_varlse), 66
knit_print.ssvsinit (init_ssvs), 53
knit_print.ssvsinput (set_ssvs), 82
knit_print.summary.normaliw (summary.normaliw), 98
knit_print.summary.ssvsmod (print.summary.ssvsmod), 70
knit_print.summary.varlse (summary.varlse), 100
knit_print.summary.vharlse (summary.vharlse), 101
knit_print.varlse (var_lm), 104
knit_print.vharlse (vhar_lm), 106
list, 21, 27
logLik.bvarflat (logLik.varlse), 57
logLik.bvarmn (logLik.varlse), 57
logLik.bvharmn (logLik.varlse), 57
logLik.varlse, 57
logLik.vharlse (logLik.varlse), 57
mae, 59
mae(), 51
mape, 60
mape(), 51
mase, 61
mase(), 51
matrix, 35, 45, 73
mrae, 62
mse, 63
mse(), 51
optimParallel::optimParallel(), 18, 30–32
oxfordman, 64
oxfordman_rk (oxfordman), 64
oxfordman_rv (oxfordman), 64
posterior::draws_df, 20, 21, 27
predict.bvarflat (predict.varlse), 66
predict.bvarflat(), 15
predict.bvarmn (predict.varlse), 66
predict.bvarmn(), 17
predict.bvarsssvs (predict.varlse), 66
predict.bvarsrv (predict.varlse), 66
predict.bvharmonic, 25
predict.bvharsssvs (predict.varlse), 66
predict.bvharssvs (predict.varlse), 66
predict.varlse, 66
predict.varlse(), 105
predict.vharlse (predict.varlse), 66
predict.vharlse(), 108
print.boundbvharemp (bound_bvharemp), 12
print.bvarflat (bvar_flat), 13
print.bvarhm (bvar_niwhm), 17
print.bvarmm (bvar_minnesota), 15
print.bvarssvs (bvar_ssvs), 19
print.bvarsrv (bvar_sv), 22
print.bvharcv (forecast_roll), 46
print.bvharemp (choose_bvharemp), 30
print.bvhari (analyze_ir.varlse), 5
print.bvharmm (bvharm_minnesota), 23
print.bvharpriorspec (set_lambda), 81
print.bvharspec (set_bvar), 78
print.bvharssvs (bvar_ssvs), 25
print.bvharsv (bvharm_sv), 28
print.predbvharemp (predict.varlse), 66
print.ssvsinit (init_ssvs), 53
print.ssvsinput (set_ssvs), 82
print.summary.normaliw (summary.normaliw), 98
print.summary.ssvsmod, 70
print.summary.varlse (summary.varlse), 100
print.summary.vharlse (summary.vharlse), 101
print.varlse (var_lm), 104
print.vharlse (vhar_lm), 106
relmae, 71
relspne, 72
residuals.bvarflat (residuals.varlse), 73
residuals.bvarflat(), 15
residuals.bvarmn (residuals.varlse), 73
residuals.bvarmn(), 17
residuals.bvharmonic (residuals.varlse), 73
residuals.bvharmonic(), 25
residuals.varlse, 73
residuals.varlse(), 105
residuals.vharlse (residuals.varlse), 73
residuals.vharlse(), 108
rmse, 74
rmstde, 75
rmse(), 76
rmsfe, 77
set_bvar, 78
set_bvar(), 12, 15, 17, 22, 32, 81, 86, 87
set_bvar_flat (set_bvar), 78
set_bvar_flat(), 13, 15
set_bvharemp (set_bvar), 78
set_bvharemp(), 12, 23–25, 29, 32, 81, 89, 90
set_lambda, 81
set_lambda(), 79, 80
set_psi (set_lambda), 81
set_psi(), 79, 80
set_ssvs, 82
set_ssvs(), 18, 20, 21, 26, 27, 91
set_weight_bvharemp (set_bvharemp), 78
set_weight_bvharemp(), 12, 23–25, 29, 32, 81, 89, 90
sim_iw, 85
sim_matgaussian, 85
sim_mncoef, 86
INDEX

sim_mniw, 87
sim_mnormal, 88
sim_mnnvhar_coef, 89
sim_mvt, 90
sim_ssvs_var, 91
sim_ssvs_vhar(sim_ssvs_var), 91
sim_var, 93
sim_vhar, 94
split_coef, 95
spne, 96
ssvs_bvar_algo, 18, 20, 22, 26, 28
stableroot, 97
stableroot.bvarflat(stableroot.varlse), 97
stableroot.bvarnm(stableroot.varlse), 97
stableroot.bvharmn(stableroot.varlse), 97
stableroot.varlse, 97
stableroot.vharlse(stableroot.varlse), 97
stats::AIC(), 52
stats::approx(), 43
stats::coef(), 34
stats::fitted(), 45
stats::optim(), 29, 30, 32, 81
stats::residuals(), 73
summary.normaliw, 98
summary.normaliw(), 17, 25
summary.ssvsmod(print.summary.ssvsmod), 70
summary.varlse, 100
summary.varlse(), 105
summary.vharlse, 101
summary.vharlse(), 108
ts_forecasting_cv, 46, 47

var_design_formulation, 24, 59, 68, 104, 105
var_lm, 104
var_lm(), 59, 93–95
var_vec_formulation, 22, 28
VARtoVMA, 103
VARtoVMA(), 6
vhar_lm, 106
vhar_lm(), 59, 95
VHARtoVMA, 106
VHARtoVMA(), 6