Package: multivar (via r-universe)

October 9, 2026

Title Penalized Estimation of Multiple-Subject Vector Autoregressive Models

Version 1.4.0

Description Simulate, estimate, and forecast vector autoregressive (VAR) models for multiple-subject data using structured penalization. Decomposes dynamics into shared (common) and subject-specific (unique) components via adaptive LASSO with FISTA optimization. Supports cross-validation and extended BIC model selection and subgroup detection, and time-varying parameters.

Depends R (>= 3.5.0)

Imports methods, stats, utils, MASS, Rcpp (>= 1.0.3), Matrix, ggplot2, vars, reshape2, glmnet, igraph, viridis, scales

LinkingTo Rcpp, RcppArmadillo

License GPL (>= 2)

Encoding UTF-8

LazyData true

ByteCompile true

RoxygenNote 7.3.3

NeedsCompilation yes

Author Zachary Fisher [aut, cre], Christopher Crawford [aut], Younghoon Kim [ctb], Vladas Pipiras [ctb]

Maintainer Zachary Fisher <fish.zachary@gmail.com>

Repository https://cran.r-universe.dev

Date/Publication 2026-03-30 15:46:42 UTC

RemoteUrl https://github.com/cran/multivar

RemoteRef HEAD

RemoteSha 27b4b491e6d87a0c7f1967237a3faeb4e3e2b128

Contents

multivar-package . . . 3
build_matrix_spec . . . 3
build_scenario_table . . . 5
canonical.multivar . . . 5
compute_ebic . . . 6
constructModel . . . 7
cv.multivar . . . 11
cv_blocked . . . 12
cv_multivar . . . 13
dat_multivar_sim . . . 15
estimate_initial_coefs . . . 15
eval_multivar_performance . . . 17
expand_range . . . 17
extract_multivar_hyperparams . . . 18
get_common_cols . . . 19
get_common_tvp_cols . . . 19
get_dynamics_helpers . . . 20
get_n_periods . . . 21
get_period_rows . . . 22
get_subgrp_cols . . . 22
get_subgrp_cols_for_subject . . . 23
get_subject_rows . . . 23
get_unique_cols . . . 24
get_unique_tvp_cols . . . 24
is_tvp . . . 25
lambda_grid . . . 25
maity_debiasing . . . 27
multivar-class . . . 27
multivar_fit_methods . . . 30
multivar_sim . . . 31
multivar_sim_breaks . . . 32
multivar_sim_growth . . . 34
multivar_sim_subgroups . . . 35
multivar_sim_tvp . . . 38
plot_cv_lambda_grid . . . 39
plot_results . . . 40
plot_sim . . . 41
plot_transition_mat . . . 43
pretrained_var_stage1_glmnet . . . 44
print.matrix_spec . . . 45
print_dynamics . . . 46
print_dynamics_tvp . . . 47
print_edge_prevalence . . . 48
recover_intercepts . . . 48
select_by_ebic . . . 50
show.multivar . . . 51

summary_multivar 51
validate_matrix_spec 52
var_sim_tvp 52

Index **54**

`multivar-package` *multivar: Penalized Estimation of Multiple-Subject Vector Autoregressive Models*

Description

Simulate, estimate, and forecast vector autoregressive (VAR) models for multiple-subject data using structured penalization. Decomposes dynamics into shared (common) and subject-specific (unique) components via adaptive LASSO with FISTA optimization. Supports cross-validation and extended BIC model selection and subgroup detection, and time-varying parameters.

multivar is an R package for simulating, estimating and forecasting stationary Vector Autoregressive (VAR) models for multiple subject data using the penalized multi-VAR framework.

Author(s)

Maintainer: Zachary Fisher <fish.zachary@gmail.com>

Authors:

- Christopher Crawford

Other contributors:

- Younghoon Kim [contributor]
- Vladas Pipiras [contributor]

`build_matrix_spec` *Build Design Matrix Specification*

Description

Creates a specification object that defines the column and row structure of the design matrix A. This is the single source of truth for all downstream functions that need to know column indices.

Usage

```
build_matrix_spec(
  k,
  d,
  n,
  tvp = FALSE,
  common_effects = TRUE,
  subgroup = FALSE,
  common_tvp_effects = FALSE,
  breaks = NULL,
  subgroup_membership = NULL
)
```

Arguments

k	Integer. Number of subjects.
d	Integer. Number of variables.
n	Integer vector. Timepoints per subject (length k).
tvp	Logical. Whether time-varying parameters are used.
common_effects	Logical. Whether common effects are included.
subgroup	Logical. Whether subgroup structure is present.
common_tvp_effects	Logical. Whether common TVP effects are included.
breaks	List. Period definitions for TVP models (length k).
subgroup_membership	Integer vector. Subgroup assignments (length k).

Details

The design matrix A has the following column structure depending on model type:

k=1, non-TVP:

`[d columns]`

k=1, TVP, common_effects=TRUE:

`[d common | d*P unique_tvp]`

k=1, TVP, common_effects=FALSE:

`[d*P unique_tvp]`

k>1, non-TVP, no subgroup:

`[d common | d unique_1 | ... | d unique_k]`

k>1, non-TVP, with subgroup:

[d common | d subgrp_1 | ... | d subgrp_S | d unique_1 | ... | d unique_k]

k>1, TVP, common_effects=TRUE, common_tvp_effects=TRUE:

[d common | d unique_1 | ... | d unique_k | d*P common_tvp | d*P unique_tvp_1 | ... | d*P unique_tvp_k]

Value

A list with class "matrix_spec" containing:

- params: Model parameters (k, d, n, flags)
- cols: Column index ranges for each effect type
- rows: Row index ranges for subjects and periods

---

build_scenario_table *Reconstruct scenario-level multipliers used to build W*

Description

This matches the logic in est_base_weight_mat(): we expand ratios_unique, and (optionally) ratios_subgroup and ratios_unique_tvp into a list of S scenarios, where S = dim(W)[3].

Usage

```
build_scenario_table(object)
```

Arguments

object	multivar object built using ConstructModel.

---

canonical.multivar *Canonical VAR Fitting Function for multivar*

Description

Canonical VAR Fitting Function for multivar

Usage

```
canonical.multivar(object)
```

Arguments

object	multivar object built using ConstructModel.

Details

A function to fit a canonical VAR model to each individual dataset.

Value

A list of results.

See Also

constructModel,

Examples

```
# example 1 (run)
sim1  <- multivar_sim(
  k = 2,  # individuals
  d = 5,  # number of variables
  n = 20, # number of timepoints
  prop_fill_com = 0.1, # proportion of paths common
  prop_fill_ind = 0.05, # proportion of paths unique
  lb = 0.1,  # lower bound on coefficient magnitude
  ub = 0.5,  # upper bound on coefficient magnitude
  sigma = diag(5) # noise
)

model1 <- constructModel(data = sim1$data, weightest = "ols")
fit1 <- canonical.multivar(model1)
```

---

compute_ebic *Compute EBIC for an array of fitted coefficient matrices*

Description

Core computation of the Extended Bayesian Information Criterion (EBIC) for each scenario in a beta array. Used internally by `cv.multivar` when `selection = "ebic"` and by `select_by_ebic` for post-hoc reselection.

Usage

```
compute_ebic(beta_array, Z, Y, d, gamma = 0.5, spec = NULL)
```

Arguments

`beta_array`	3D array (d x p x n_scenarios) of fitted coefficients.
`Z`	Design matrix (transposed), p x n.
`Y`	Response matrix (transposed), d x n.
`d`	Number of response variables.

`gamma` EBIC tuning parameter. `gamma = 0` gives standard BIC; `gamma = 0.5` (default) is a moderate EBIC; `gamma = 1` is the most conservative (strongest sparsity preference).

`spec` Optional matrix_spec object. When provided, degrees of freedom are counted per coefficient position within each block (common, unique, TVP), following Chen & Chen (2008). Each d x d position that has any nonzero entry counts as one degree of freedom, regardless of how many subjects share that position. This avoids over-penalizing shared structure. When NULL, falls back to raw nonzero counting.

Details

With `spec = NULL` (legacy behavior), EBIC counts every nonzero entry in B as one degree of freedom:

$$\text{EBIC}_i = \sum_{j=1}^{d} n \log(\text{RSS}_{ij}/n) + k_i \log(n) + 2\gamma\, k_i \log(p)$$

With a `spec` object, degrees of freedom are counted per position: for each d x d coefficient position (i,j), the position contributes 1 df if any block (common, any unique, any TVP) has a nonzero entry at that position. The eBIC extension uses `p = d` (predictors per equation) instead of the full design matrix width, and employs the Chen & Chen (2008) `lchoose(p, df)` formulation per equation.

Value

Numeric vector of EBIC values, one per scenario.

`constructModel` *Construct an object of class multivar*

Description

Construct an object of class multivar

Usage

```
constructModel(
  data = NULL,
  lag = 1,
  horizon = 0,
  t1 = NULL,
  t2 = NULL,
  lambda1 = NULL,
  nlambda1 = 30,
  n_ratios_subgroup = 30,
  depth = NULL,
  tol = 1e-04,
  window = 1,
```

```
  standardize = TRUE,
  weightest = "maity",
  canonical = FALSE,
  threshold = FALSE,
  lassotype = "adaptive",
  intercept = FALSE,
  W = NULL,
  ratios_unique = NULL,
  ratios_subgroup = NULL,
  ratios_unique_tvp = NULL,
  cv = "blocked",
  nfolds = 10,
  lamadapt = FALSE,
  subgroup_membership = NULL,
  subgroup = FALSE,
  B = NULL,
  pendiag = TRUE,
  tvp = FALSE,
  inittvpcoefs = list(),
  breaks = list(),
  lambda_choice = "lambda.min",
  common_effects = TRUE,
  common_tvp_effects = NULL,
  save_beta = TRUE,
  ncores = 1,
  max_grid_size = NULL,
  eps = 0.001,
  warmstart = TRUE,
  stopping_crit = "absolute",
  selection = "cv",
  ebic_gamma = 0.5,
  weight_type = "standard",
  maity_opts = list()
)
```

Arguments

data	List. A list (length = k) of T by d multivariate time series
lag	Numeric. The VAR order. Default is 1.
horizon	Numeric. Desired forecast horizon. Default is 1.
t1	Numeric. Index of time series in which to start cross validation. If NULL, default is floor(nrow(n)/3) where nk is the time series length for individual k.
t2	Numeric. Index of times series in which to end cross validation. If NULL, default is floor(2*nrow(n)/3) where nk is the time series length for individual k.
lambda1	Matrix. Regularization parameter grid. Default is NULL (auto-generated).
nlambda1	Numeric. Number of lambda1 values to search over. Default is 30.

`n_ratios_subgroup`
: Numeric. Number of ratios_subgroup values to search over. Default is 30.

`depth`
: Numeric. Depth of lambda1 grid construction (lambda_min = lambda_max / depth). Default is 10000.

`tol`
: Numeric. Optimization tolerance (default 1e-4).

`window`
: Numeric. Size of rolling window.

`standardize`
: Logical. Default is true. Whether to standardize the individual data. Note, if intercept = TRUE and standardize = TRUE, the data is scaled but not de-meaned.

`weightest`
: Character. How to estimate initial coefficients for adaptive weights. Default is "maity". Other options include "lasso", "alasso" (adaptive LASSO: two-stage procedure using LASSO pilot then adaptive reweighting for sparser estimates), "ridge", "ols", and "multivar". The "multivar" option fits a standard lasso multivar model first to get structured initial estimates. Note: for k=1 TVP models, "multivar" works well with common_effects=TRUE but not with common_effects=FALSE (use "lasso" instead). Only used when lassotype = "adaptive" (ignored for standard LASSO).

`canonical`
: Logical. Default is false. If true, individual datasets are fit to a VAR(1) model.

`threshold`
: Logical. Default is false. If true, and canonical is true, individual transition matrices are thresholded based on significance.

`lassotype`
: Character. Default is "adaptive". Choices are "standard" or "adaptive" lasso.

`intercept`
: Logical. Default is FALSE.

`W`
: Matrix. Default is NULL.

`ratios_unique`
: Numeric vector. Penalty ratio for unique effects. Default is NULL.

`ratios_subgroup`
: Numeric vector. Penalty ratio for subgroup effects. Default is NULL.

`ratios_unique_tvp`
: Numeric vector. Default is NULL.

`cv`
: Character. Default is "blocked" for blocked folds cross-validation. "rolling" is also available for rolling window cross-validation. If "blocked" is selected the nfolds argument should be specified.

`nfolds`
: Numeric. The number of folds for use with "blocked" cross-validation.

`lamadapt`
: Logical. Should the lambdas be calculated adaptively. Default is FALSE.

`subgroup_membership`
: Numeric. Vector of subgroup assignments.

`subgroup`
: Logical. Internal argument whether to run subgrouping algorithm.

`B`
: Matrix. Default is NULL.

`pendiag`
: Logical. Logical indicating whether autoregressive parameters should be penalized. Default is TRUE.

`tvp`
: Logical. Default is FALSE.

`inittvpcoefs`
: List.

`breaks`
: List. A list of length K indicating structural breaks in the time series.

`lambda_choice` Character. Which lambda to use for initial coefficient estimation: "lambda.min" (default) or "lambda.1se". lambda.min provides better coefficient recovery, especially for small samples and TVP models; lambda.1se may be too conservative, causing all-zero initial estimates.

`common_effects` Logical. Whether to include common effects in TVP models. Only applies when tvp = TRUE. Default is TRUE (include common effects). When FALSE, the model becomes Total = Unique + TVP instead of Total = Common + Unique + TVP. This can be useful when you expect no shared dynamics across subjects.

`common_tvp_effects`
Logical. Whether to include time-varying common effects in TVP models. Default is NULL, which automatically sets to TRUE when tvp = TRUE and FALSE when tvp = FALSE. Only meaningful when tvp = TRUE.

`save_beta` Logical. Whether to retain the full beta coefficient array in the cv.multivar result. Default is TRUE. Set to FALSE to reduce memory usage when only the best-model coefficients (in mats) are needed.

`ncores` Numeric. Number of cores for parallel cross-validation. Default is 1.

`max_grid_size` Numeric. Maximum number of hyperparameter combinations. If the full grid exceeds this, dimensions are coarsened proportionally. Default is NULL (no limit).

`eps` Numeric. FISTA convergence tolerance. Default is 1e-3. Smaller values yield more precise solutions but increase computation time.

`warmstart` Logical. Whether to use the previous lambda's solution as the starting point for the next lambda in the FISTA solver. Default is TRUE. Reduces computation time with negligible effect on accuracy.

`stopping_crit` Character. FISTA convergence criterion. One of "absolute" (default), "relative", or "objective". "absolute" checks max|B_new - B_old| < eps; "relative" normalizes by max|B_old|; "objective" checks relative change in the objective function.

`selection` Character. Model selection criterion. `"cv"` (default) uses cross-validated MSFE. `"ebic"` skips CV folds entirely, fits once on the full data, and selects by Extended BIC. EBIC is much faster and can improve structure recovery when n » p.

`ebic_gamma` Numeric. EBIC tuning parameter, used when `selection = "ebic"`. `0` gives standard BIC; `0.5` (default) is moderate EBIC; `1` is most conservative.

`weight_type` Character. Adaptive weight function type. `"standard"` (default) uses `1/|coef|^gamma` with Inf replaced by 1e10. `"bounded"` uses `1/(1+|coef|/tau)^gamma` where tau is the median of nonzero |coefs|, producing weights in [0,1] with no infinities.

`maity_opts` List. Options for Maity et al. (2022) MrLasso initial estimation. Supports `eta` (scale parameter for re-descending loss, default 2.0) and `thresh_const` (threshold multiplier, default 1.0). Only used when `weightest = "maity"`.

Examples

```
sim  <- multivar_sim(
  k = 2,  # individuals
```

```
  d = 3,  # number of variables
  n = 20, # number of timepoints
  prop_fill_com = 0.1, # proportion of paths common
  prop_fill_ind = 0.1, # proportion of paths unique
  lb = 0.1,  # lower bound on coefficient magnitude
  ub = 0.9,  # upper bound on coefficient magnitude
  sigma = diag(3) # noise
)

plot_sim(sim, plot_type = "common")

model <- constructModel(data = sim$data, weightest = "ols")
```

`cv.multivar` *Cross Validation for multivar*

Description

Cross Validation for multivar

Usage

```
cv.multivar(object)
```

Arguments

`object`	multivar object built using `ConstructModel`.

Details

The main function of the multivar package. Performs cross validation to select penalty parameters over a training sample and evaluates them over a test set.

Value

An object of class `multivar.results`.

Examples

```
# example 1 (run)
sim1  <- multivar_sim(
  k = 2,  # individuals
  d = 5,  # number of variables
  n = 20, # number of timepoints
  prop_fill_com = 0.1, # proportion of paths common
  prop_fill_ind = 0.05, # proportion of paths unique
  lb = 0.1,  # lower bound on coefficient magnitude
  ub = 0.5,  # upper bound on coefficient magnitude
  sigma = diag(5) # noise
```

```
)

model1 <- constructModel(data = sim1$data)
fit1 <- multivar::cv.multivar(model1)
```

`cv_blocked` *Blocked Cross-Validation for multivar*

Description

Performs k-fold blocked cross-validation for time series data.

Usage

```
cv_blocked(
  B,
  Z,
  Y,
  W,
  Ak,
  k,
  d,
  lambda1,
  t1,
  t2,
  eps,
  intercept = FALSE,
  cv,
  nfolds,
  tvp = FALSE,
  breaks = NULL,
  spec = NULL,
  ncores = 1,
  warmstart = FALSE,
  stopping_crit = 0L
)
```

Arguments

`B`	Initial coefficient array
`Z`	Design matrix (transposed, d x n)
`Y`	Response matrix (transposed, d x n)
`W`	Weight array for adaptive LASSO
`Ak`	List of design matrices per subject (used to build row spec if spec not provided)

k	Number of subjects
d	Number of variables
lambda1	Lambda grid matrix
t1	Start indices (unused, kept for API compatibility)
t2	End indices (unused, kept for API compatibility)
eps	Convergence tolerance
intercept	Whether model includes intercepts
cv	CV method (unused here, always "blocked")
nfolds	Number of CV folds
tvp	Whether time-varying parameters are used
breaks	List of period indices per subject (required if tvp=TRUE)
spec	Optional matrix_spec object. If not provided, a minimal spec is built from Ak and breaks.
ncores	Numeric. Number of cores for parallel computation. Default is 1.
warmstart	Logical. Whether to use warm starts in the FISTA solver. Default is FALSE.
stopping_crit	Integer. Stopping criterion flag passed to the C++ solver. Default is 0.

Value

List with beta coefficients and MSFE matrix

`cv_multivar` *Cross-Validation Dispatcher for multivar*

Description

Routes to the appropriate CV method (blocked or rolling).

Usage

```
cv_multivar(
  B,
  Z,
  Y,
  W,
  Ak,
  bk,
  k,
  d,
  lambda1,
  t1,
  t2,
  eps,
```

```
  intercept = FALSE,
  cv,
  nfolds,
  tvp = FALSE,
  breaks = NULL,
  spec = NULL,
  ncores = 1,
  warmstart = FALSE,
  stopping_crit = 0L
)
```

Arguments

B	Initial coefficient array
Z	Design matrix (transposed)
Y	Response matrix (transposed)
W	Weight array
Ak	List of design matrices per subject
bk	List of response matrices per subject
k	Number of subjects
d	Number of variables
lambda1	Lambda grid
t1	Start indices
t2	End indices
eps	Convergence tolerance
intercept	Whether model includes intercepts
cv	CV method: "blocked" or "rolling"
nfolds	Number of CV folds
tvp	Whether time-varying parameters are used
breaks	List of period indices per subject
spec	Optional matrix_spec object for row/column indices
ncores	Numeric. Number of cores for parallel computation. Default is 1.
warmstart	Logical. Whether to use warm starts in the FISTA solver. Default is FALSE.
stopping_crit	Integer. Stopping criterion flag passed to the C++ solver. Default is 0.

Value

List with beta coefficients and MSFE matrix

dat_multivar_sim *Simulated multi-VAR data.*

Description

This dataset contains multivariate time series data for k = 9 individuals with $d = 10$ variables collected at $t = 100$ equidistant time points. The data was generated such that each individual's VAR(1) transition matrix has 20 percent nonzero entries. This means, for example, each individual has 20 nonzero directed relationships in their data generating model. The position of non-zero elements in each individual's transition matrix was selected randomly given the following constraints: 2/3 of each individual's paths are shared by all individuals, and 1/3 are unique to each individual. For each individual, coefficient values between U(0,1, 0.9) were randomly drawn until stability conditions for the VAR model were satisfied.

Usage

```
dat_multivar_sim
```

Format

A list containing

mat_com a common effects transition matrix

mat_ind_unique a list of unique (individual-specific) effect matrices

mat_ind_final a list of total (common + individual-specific) effect matrices

data a list of multivariate time series for all subjects ...

estimate_initial_coefs *Estimate initial coefficients for adaptive weights*

Description

'estimate_initial_coefs' returns consistent initial estimates to be used for calculating adaptive weights in adaptive LASSO regularization.

Usage

```
estimate_initial_coefs(
  Ak,
  bk,
  d,
  k,
  lassotype,
  weightest,
```

```
  subgroup_membership,
  subgroup,
  nlambda1,
  tvp,
  breaks,
  intercept,
  nfolds,
  lambda_choice = "lambda.min",
  common_effects = TRUE,
  common_tvp_effects = TRUE,
  maity_opts = list()
)
```

Arguments

Ak	List. A list (length = k) of lagged (T-lag-horizon) by d multivariate time series matrices.
bk	List. A list (length = k) of (T-lag-horizon) by d multivariate time series matrices.
d	Numeric vector. The number of variables for each dataset.
k	Numeric. The number of subjects.
lassotype	Character. Type of LASSO penalty: "standard" or "adaptive".
weightest	Character. How to estimate initial coefficients: "lasso", "ridge", or "ols".
subgroup_membership	Numeric vector. Vector of subgroup assignments for each subject.
subgroup	Logical. Whether to run subgrouping algorithm.
nlambda1	Numeric. Not used (kept for API compatibility).
tvp	Logical. Whether to estimate time-varying parameters.
breaks	List. A list of length k indicating structural breaks in the time series.
intercept	Logical. Whether to include intercepts in the model.
nfolds	Numeric. The number of folds for cross-validation.
lambda_choice	Character. Which lambda to use from cv.glmnet: "lambda.min" or "lambda.1se".
common_effects	Logical. Whether to include common effects in TVP models.
common_tvp_effects	Logical. Whether to include common TVP effects (k>1 only).
maity_opts	List. Options for Maity et al. (2022) MrLasso initial estimation.

Value

A list containing:

- common_effects: d x d matrix of common effects
- subgroup_effects: list of d x d matrices per subgroup (if subgroup=TRUE)
- unique_effects: list of d x d matrices per subject
- tvp_effects: list of lists of d x d matrices per subject per period (if tvp=TRUE)
- common_tvp_effects: list of d x d matrices per period (if tvp=TRUE and k>1)
- total_effects: list of d x d matrices per subject

`eval_multivar_performance` *Summarize multivar performance against simulation truth (robust to missing parts)*

Description

Compare estimated transition matrices to simulation truth, but only for components that actually exist in both ‘sim_obj‘ and ‘fit_obj‘. For example, if ‘fit_obj$mats‘ only contains ‘total‘, only "total" rows are returned.

Usage

```
eval_multivar_performance(
  sim_obj,
  fit_obj,
  eps = 1e-08,
  reduced.output = TRUE,
  averages.only = TRUE,
  label = NULL,
  intercept = FALSE
)
```

Arguments

`sim_obj`	Output of multivar_sim()
`fit_obj`	Output of cv.multivar() / cv.fused() (or similar list with $mats)
`eps`	Small positive constant to stabilize relative errors
`reduced.output`	If TRUE, keep a reduced column set
`averages.only`	If TRUE, return only summary rows (unique_mean, total_mean) and the single ’common’ row (if present)
`label`	Optional label for the fitted model; by default, uses the deparsed name of ‘fit_obj‘
`intercept`	Logical; if TRUE, evaluates intercepts separately from dynamics

`expand_range` *Expand Column Range to Indices*

Description

Converts a c(start, end) range to a full sequence.

Usage

```
expand_range(range)
```

Arguments

range	Integer vector c(start, end) or NULL

Value

Integer vector or NULL

`extract_multivar_hyperparams` *Extract hyperparameter grid search results*

Description

Creates a tidy dataframe with all tested hyperparameter combinations and their cross-validation error (MSFE). Each row represents one (lambda1, ratio) combination.

Usage

```
extract_multivar_hyperparams(object, fit)
```

Arguments

object	multivar model object used to call cv_multivar(...)
fit	result returned by cv_multivar(...); MSFE is assumed to be fit[[2]]

Details

The function averages MSFE across cross-validation folds and returns one row per hyperparameter combination. The dataframe can be used for:

- Finding best hyperparameters: `df[which.min(df$MSFE), ]`
- Plotting MSFE surface
- Filtering/analyzing hyperparameter performance

Value

A data frame with columns:

ratio_index Index of ratio (1 to n_ratios)

lambda1_index Index of lambda1 (1 to n_lambda)

slice_index Flattened index using C++ ordering

lambda1_value Actual lambda1 penalty value

ratio_value Actual ratio value

lambda2_value Computed lambda2 = lambda1 * ratio

MSFE Mean squared forecast error averaged across CV folds

get_common_cols *Get Common Effect Columns*

Description

Get Common Effect Columns

Usage

```
get_common_cols(spec)
```

Arguments

spec	A matrix_spec object

Value

Integer vector c(start, end) or NULL

get_common_tvp_cols *Get Common TVP Columns for a Period*

Description

Get Common TVP Columns for a Period

Usage

```
get_common_tvp_cols(spec, period)
```

Arguments

spec	A matrix_spec object
period	Integer. Period index (1 to P)

Value

Integer vector c(start, end) or NULL

`get_dynamics_helpers` *Helper Functions for Accessing multivar Results*

Description

Simplified functions for extracting dynamics matrices from fitted multivar models. These handle both TVP and non-TVP models transparently.

Usage

```
get_dynamics(
  fit,
  subject = NULL,
  period = NULL,
  type = c("total", "common", "unique")
)

get_common_effects(fit, period = NULL)

get_unique_effects(fit, subject = NULL, period = NULL)

get_total_effects(fit, subject = NULL, period = NULL)

get_dynamics_all_subjects(
  fit,
  period = NULL,
  type = c("total", "common", "unique")
)
```

Arguments

`fit`	A fitted multivar object (output from cv.multivar())
`subject`	Subject index (1 to K). If NULL, returns all subjects.
`period`	Period index (for TVP models). If NULL, returns all periods.
`type`	Type of effects: "total", "common", or "unique"

Details

These helper functions simplify access to dynamics matrices by handling the different structures of TVP and non-TVP models automatically.

For non-TVP models: - ‘fit$mats$common‘ is a single matrix (shared across all subjects) - ‘fit$mats$unique‘ is a list of K matrices (one per subject) - ‘fit$mats$total‘ is a list of K matrices (common + unique per subject)

For TVP models: - ‘fit$mats$common‘ is a single matrix (time-invariant, shared across all subjects) - ‘fit$mats$tvp‘ is a list of K lists, each containing P period matrices (time-varying per subject) - ‘fit$mats$total‘ is a list of K lists of P matrices (common + tvp per subject per period)

The helper functions hide this complexity from the user.

Examples

```
## Not run:
# Non-TVP model
fit <- cv.multivar(object)
get_dynamics(fit, subject = 1)             # Subject 1 total effects
get_common_effects(fit)                    # Common effects
get_dynamics_all_subjects(fit)             # All subjects

# TVP model
fit_tvp <- cv.multivar(object_tvp)
get_dynamics(fit_tvp, subject = 1, period = 2)    # Subject 1, Period 2
get_common_effects(fit_tvp, period = 2)           # Common for Period 2
get_dynamics_all_subjects(fit_tvp, period = 2)    # All subjects, Period 2

## End(Not run)
```

`get_n_periods` *Get Number of Periods in TVP Model*

Description

Returns the number of time-varying periods in a TVP model. Returns 1 for non-TVP models.

Usage

```
get_n_periods(fit)
```

Arguments

`fit`	A fitted multivar object

Value

Integer number of periods

Examples

```
## Not run:
n_periods <- get_n_periods(fit)

## End(Not run)
```

get_period_rows *Get Row Range for a Subject's Period*

Description

Get Row Range for a Subject's Period

Usage

```
get_period_rows(spec, subject, period)
```

Arguments

spec	A matrix_spec object
subject	Integer. Subject index (1 to k)
period	Integer. Period index (1 to P_i)

Value

Integer vector c(start, end) or NULL

get_subgrp_cols *Get Subgroup Effect Columns*

Description

Get Subgroup Effect Columns

Usage

```
get_subgrp_cols(spec, subgroup)
```

Arguments

spec	A matrix_spec object
subgroup	Integer. Subgroup index (1 to S)

Value

Integer vector c(start, end) or NULL

get_subgrp_cols_for_subject *Get Subgroup Columns for a Subject*

Description

Get Subgroup Columns for a Subject

Usage

```
get_subgrp_cols_for_subject(spec, subject)
```

Arguments

spec	A matrix_spec object
subject	Integer. Subject index (1 to k)

Value

Integer vector c(start, end) or NULL

get_subject_rows *Get Row Range for a Subject*

Description

Get Row Range for a Subject

Usage

```
get_subject_rows(spec, subject)
```

Arguments

spec	A matrix_spec object
subject	Integer. Subject index (1 to k)

Value

Integer vector c(start, end)

get_unique_cols *Get Unique Effect Columns for a Subject*

Description

Get Unique Effect Columns for a Subject

Usage

```
get_unique_cols(spec, subject)
```

Arguments

spec	A matrix_spec object
subject	Integer. Subject index (1 to k)

Value

Integer vector c(start, end) or NULL (for k=1 TVP)

get_unique_tvp_cols *Get Unique TVP Columns for a Subject and Period*

Description

Get Unique TVP Columns for a Subject and Period

Usage

```
get_unique_tvp_cols(spec, subject, period)
```

Arguments

spec	A matrix_spec object
subject	Integer. Subject index (1 to k)
period	Integer. Period index (1 to P_i)

Value

Integer vector c(start, end) or NULL

is_tvp *Check if Model is Time-Varying*

Description

Returns TRUE if the model has time-varying parameters (TVP).

Usage

```
is_tvp(fit)
```

Arguments

fit	A fitted multivar object

Value

Logical indicating if model is TVP

Examples

```
## Not run:
if (is_tvp(fit)) {
  cat("Model has time-varying parameters\n")
}

## End(Not run)
```

lambda_grid *Build the lambda1 path (one column per penalty scenario)*

Description

Construct the sequence (path) of lambda1 values that will be used for cv.

Usage

```
lambda_grid(
  depth = 1000,
  nlam = 30,
  Y,
  Z,
  W = NULL,
  tol = 1e-04,
  intercept = FALSE,
  lamadapt = FALSE,
  k = NULL
)
```

Arguments

depth	numeric. Default is 1000. Controls how wide the lambda1 path is. The top of the path is lambda_max (the largest penalty we think we need to zero everything out). The bottom is lambda_max / depth. Typical depth ~ 1000.
nlam	integer. Default is 30. Number of lambda1 values along the path (rows of the output).
Y	numeric outcome matrix. Matrix of outcomes.
Z	numeric matrix. Matrix of predictors.
W	numeric array of penalty weights. Shape: [n_outcomes , n_predictors , n_scenarios]. Interpretation: - W[,,i] is the penalty weight layout for scenario i. Different scenarios correspond to different relative penalization of global vs subgroup vs individual vs time-varying effects.
tol	numeric scalar. Default 1e-4. Tolerance used in the adaptive branch (lamadapt = TRUE) when binary-searching for the scenario-specific max lambda. If lamadapt = FALSE, tol is not used.
intercept	logical. Default FALSE. Whether the model includes an intercept column. Intercepts are usually unpenalized. In lamadapt = TRUE mode, we drop the first column of the fitted coefficient array when checking if everything is (approximately) zero, so we don't falsely call the intercept "nonzero".
lamadapt	logical. Default FALSE. If FALSE: We compute one proxy lambda_max from data (max(Y and then generate the same log-spaced path for every scenario. If TRUE: For each scenario i, we refine its own lambda_max using weighted penalized fits with W[,,i], using a binary search. That is closer to "adaptive lasso" logic: strong signals get downweighted and don't need as huge a lambda to kill them; weak ones get upweighted and do.
k	integer or NULL. Number of individuals. Only needed if lamadapt = TRUE, because we pass it to the solver (wlasso) inside the scenario-specific binary search. Ignored when lamadapt = FALSE. Notes on the returned object:

Details

In this package, lambda1 is the main sparsity / regularization strength applied to the common effects. Laer, we scaled these into the unique effects via ratios_unique. Importantly, we don't just pick a single lambda1 — we build a path of candidate lambda1 values, from very large (forces almost all penalized coefficients to zero) down to much smaller (less shrinkage).

This path is defined by a decreasing set of values, generated from lambda_max down to lambda_max / depth, so'depth' is used to control that span and the default value is 1,000.

Note: the models have multiple penalty scenarios as alluded to earlier. We call these scenarios, where each scenario encodes a different way to penalize the common and unique effects. Those scenarios live along the 3rd dimension of the array W are typically of length length(ratios_unique)*legnth(lambda1), such that

dim(W) = [n_outcomes , n_predictors , n_scenarios]

We build one lambda1 path per scenario. That's why the return value is a matrix with:

rows = number of lambda1 values along the path columns = number of scenarios

- cv.multivar() stores this matrix in object@lambda1. - During CV, for each scenario column i and each row r in that column, the solver fits the model at that lambda1 and scores prediction error.

- We also have object@ratios_unique, where each ratio says how strong to penalize deviation blocks (individual/subgroup/time-varying) relative to the common effects. Conceptually this means lambda2 = ratio * lambda1.

- We use the ingredients: * lambda1 grid (here) * ratios_unique (constructModel) to construct the lambda2 grid..

Some notes on the arguments:

Value

lambda1 matrix, dimension [nlam x n_scenarios].

Column i is the lambda1 path for scenario i. Row r is a particular lambda1 magnitude along that path

Later: - cv.multivar() stores this in object@lambda1, and cross-validates across [row r, column i]. - Combined with object@ratios_unique, you can reconstruct lambda2 via lambda2 = ratio * lambda1 and plot CV error surfaces.

`maity_debiasing` *Maity et al. (2022) MrLasso-style initial estimation*

Description

Provides debiased, robustly-aggregated, analytically-thresholded initial estimates for adaptive LASSO weights. Replaces the default median-based decomposition when `weightest = "maity"` in `constructModel`.

References

Maity, S., Sun, Y., & Banerjee, M. (2022). Meta-analysis of heterogeneous data: integrative sparse regression in high-dimensions. *JMLR*, 23, 1-50.

`multivar-class` *multivar object class*

Description

An object class to be used with cv.multivar

Details

To construct an object of class multivar, use the function `constructModel`

Slots

`k` Numeric. The number of subjects (or groupings) in the dataset.

`n` Numeric Vector. Vector containing the number of timepoints for each dataset.

`d` Numeric Vector. Vector containing the number of variables for each dataset.

`Ak` List. A list (length = k) of lagged (T-lag-horizon) by d multivariate time series.

`bk` List. A list (length = k) of (T-lag-horizon) by d multivariate time series.

`Ak_orig` List. A list (length = k) of lagged (T-lag-horizon) by d unscaled multivariate time series.

`bk_orig` List A list (length = k) of (T-lag-horizon) by d unscaled multivariate time series.

`Hk` List. A list (length = k) of (horizon) by d multivariate time series.

`A` Matrix. A matrix containing the lagged ((T-lag-horizon)k) by (d+dk) multivariate time series.

`b` Matrix. A matrix containing the non-lagged ((T-lag-horizon)k) by (d) multivariate time series.

`H` Matrix. A matrix containing the non-lagged (horizon k) by d multivariate time series.

`data_means` List. A list (length = k) of column means for each group's data, used for intercept recovery when intercept=TRUE.

`lag` Numeric. The VAR order. Currently only lag 1 is supported.

`horizon` Numeric. Forecast horizon.

`t1` Numeric vector. Index of time series in which to start cross validation for individual k.

`t2` Numeric vector. Index of time series in which to end cross validation for individual k.

`t1k` Numeric vector. Index of time series in which to start cross validation for individual k.

`t2k` Numeric vector. Index of time series in which to end cross validation for individual k.

`ntk` Numeric. Number of usable timepoints (rows of b) per individual k

`ndk` Numeric. Number of variables (cols of b) per individual k

`lambda1` Numeric vector. Regularization parameter grid.

`nlambda1` Numeric. Number of lambda1 values to search over. Default is 30.

`n_ratios_subgroup` Numeric. Number of ratios_subgroup values to search over. Default is 30.

`gamma` Numeric. Power for adaptive weighting (default 1). Controls how strongly initial estimates influence the penalty: weight = 1/|coef|^gamma.

`tol` Numeric. Convergence tolerance.

`depth` Numeric. Depth of grid construction. Default is 1000.

`window` Numeric. Size of rolling window.

`standardize` Logical. Default is true. Whether to standardize the individual data.

`weightest` Character. How to estimate initial coefficients for adaptive weights. Default is "lasso". Other options include "ridge" and "ols".

`canonical` Logical. Default is false. If true, individual datasets are fit to a VAR(1) model.

`threshold` Logical. Default is false. If true, and canonical is true, individual transition matrices are thresholded based on significance.

`lassotype` Character. Default is "adaptive". Choices are "standard" or "adaptive" lasso.

`intercept` Logical. Default is FALSE.

`W` Matrix. Default is NULL.

`ratios_unique` Numeric vector. Penalty ratio for unique effects. Default is NULL.

`ratios_subgroup` Numeric vector. Penalty ratio for subgroup effects. Default is NULL.

`ratios_unique_tvp` Numeric vector. Default is NULL.

`ratios_common_tvp` Numeric vector. Penalty ratio for common TVP effects. Default is NULL.

`cv` Character. Default is "blocked" for k-folds blocked cross-validation. rolling window cross-validation also available using "rolling". If "blocked" is selected the nfolds argument should be specified.

`nfolds` Numeric. The number of folds for use with "blocked" cross-validation.

`lamadapt` Logical. Should the lambdas be calculated adaptively. Default is FALSE.

`subgroup_membership` Numeric. Vector of subgroup assignments.

`subgroup` Logical. Argument whether to run subgrouping algorithm.

`B` Matrix. Default is NULL.

`initcoefs` List. A list of initial consistent estimates for the total, subgroup, unique and common effects.

`pendiag` Logical. Logical indicating where autoregressive paramaters should be penalized. Default is true.

`tvp` Logical.

`inittvpcoefs` List. A list of initial tvp estimates.

`breaks` List. A list of length K indicating structural breaks in the time series.

`lambda_choice` Character. Which lambda to use for initial coefficient estimation ("lambda.min" or "lambda.1se").

`common_effects` Logical. Whether to include common effects in TVP models. Only applies when tvp = TRUE.

`common_tvp_effects` Logical. Whether to include common TVP effects (shared time-varying patterns) in TVP models. Only applies when tvp = TRUE.

`save_beta` Logical. Whether to retain the full beta array in the cv.multivar result. Default is TRUE.

`ncores` Numeric. Number of cores for parallel computation. Default is 1.

`spec` List. Design matrix specification object created by `build_matrix_spec`. Single source of truth for column/row indices.

`eps` Numeric. FISTA convergence tolerance. Default is 1e-3.

`warmstart` Logical. Whether to use warm starts in the FISTA solver. Default is TRUE.

`stopping_crit` Character. FISTA convergence criterion. One of `"absolute"`, `"relative"`, or `"objective"`.

`selection` Character. Model selection criterion: `"cv"` (default) for cross-validated MSFE, or `"ebic"` for Extended BIC.

`ebic_gamma` Numeric. EBIC tuning parameter, used when `selection = "ebic"`. Default is 0.5.

`weight_type` Character. Adaptive weight function type: `"standard"` (default) uses `1/|coef|^gamma`, `"bounded"` uses `1/(1+|coef|/tau)^gamma`.

`maity_opts` List. Options for Maity et al. (2022) MrLasso initial estimation when `weightest = "maity"`. Supports `eta` (scale parameter for re-descending loss, default 2.0) and `thresh_const` (threshold multiplier, default 1.0).

See Also

constructModel

`multivar_fit_methods` *S3 Methods for multivar_fit Objects*

Description

Standard S3 methods for fitted multivar models.

Usage

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

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

## S3 method for class 'multivar_fit'
plot(x, type = c("dynamics", "prevalence"), ...)

## S3 method for class 'multivar_fit'
coef(
  object,
  type = c("total", "common", "unique"),
  subject = NULL,
  period = NULL,
  ...
)
```

Arguments

`x`	A multivar_fit object (output from cv.multivar()).
`...`	Additional arguments (unused).
`object`	A multivar_fit object (output from cv.multivar()).
`type`	Character; one of ‘"total"‘, ‘"common"‘, ‘"unique"‘.
`subject`	Optional integer subject index.
`period`	Optional integer period index (TVP only).

Functions

- `print(multivar_fit)`: Print a fitted multivar model.
- `summary(multivar_fit)`: Summarize a fitted multivar model.
- `plot(multivar_fit)`: Plot fitted dynamics or prevalence.
- `coef(multivar_fit)`: Extract coefficient matrices.

multivar_sim *Simulate multivar data.*

Description

Simulate multivar data.

Usage

```
multivar_sim(
  k,
  d,
  n,
  prop_fill_com,
  prop_fill_ind,
  lb,
  ub,
  sigma,
  unique_overlap = FALSE,
  mat_common = NULL,
  mat_unique = NULL,
  mat_total = NULL,
  diag = FALSE,
  intercept = NULL,
  standardize_output = FALSE
)
```

Arguments

k	Integer. The number of individuals (or datasets) to be generated.
d	Integer. The number of variables per dataset. For now this will be constant across individuals.
n	Integer. The time series length.
prop_fill_com	Numeric. The proportion of nonzero paths in the common transition matrix.
prop_fill_ind	Numeric. The proportion of nonzero unique (not in the common transition matrix or transition matrix of other individuals) paths in each individual transition matrix.
lb	Numeric. The lower bound for individual elements of the transition matrices.
ub	Numeric. The upper bound for individual elements of the transition matrices.
sigma	Matrix. The (population) innovation covariance matrix.
unique_overlap	Logical. Default is FALSE. Whether the unique portion should be completely unique (no overlap) or randomly chosen.
mat_common	Matrix. A common effects transition matrix (if known).
mat_unique	List. A list of unique effects transition matrix (if known).

mat_total	List. A list of total effects transition matrix (if known).
diag	Logical. Default is FALSE. Should diagonal elements be filled first for common elements.
intercept	List. Default is NULL. A list of length K containing numeric vectors of length d representing the intercept values.
standardize_output	Logical. Default is FALSE. If TRUE, scales each subject's data to have sample variance of 1 for each variable, and returns the transformed transition matrices and innovation covariances in standardized space.

Examples

```
k <- 3
d <- 10
n <- 20
prop_fill_com <- .1
prop_fill_ind <- .05
lb <- 0.1
ub <- 0.5
sigma <- diag(d)
data <- multivar_sim(k, d, n, prop_fill_com, prop_fill_ind, lb, ub,sigma)$data
```

`multivar_sim_breaks` *Simulate multivar data with time-varying dynamics according to a latent growth curve.*

Description

Simulate multivar data with time-varying dynamics according to a latent growth curve.

Usage

```
multivar_sim_breaks(
  k,
  d,
  n,
  prop_fill_com,
  prop_fill_ind,
  prop_fill_com_growth,
  prop_fill_ind_growth,
  offset = 0,
  lb,
  ub,
  lb_int,
  ub_int,
  lb_slope,
  ub_slope,
```

```
  sigma,
  diag = FALSE,
  iters = 1,
  intercept = NULL
)
```

Arguments

k	Integer. The number of individuals (or datasets) to be generated.
d	Integer. The number of variables per dataset. For now this will be constant across individuals.
n	Integer. The time series length.
prop_fill_com	Numeric. The proportion of nonzero paths in the common transition matrix.
prop_fill_ind	Numeric. The proportion of nonzero unique (not in the common transition matrix or transition matrix of other individuals) paths in each individual transition matrix.
prop_fill_com_growth	Numeric. The proportion of nonzero time-varying paths in the common transition matrix.
prop_fill_ind_growth	Numeric. The proportion of nonzero time-varying unique (not in the common transition matrix or transition matrix of other individuals) paths in each individual transition matrix.
offset	Numeric. An offset dictating when dynamics should transition from time-invariant to time-varying.
lb	Numeric. The upper bound for individual elements of the transition matrices.
ub	Numeric. The lower bound for individual elements of the transition matrices.
lb_int	Numeric. The upper bound for individual elements of the intercepts.
ub_int	Numeric. The lower bound for individual elements of the intercepts.
lb_slope	Numeric. The upper bound for individual elements of the slopes.
ub_slope	Numeric. The lower bound for individual elements of the slopes.
sigma	Matrix. The (population) innovation covariance matrix.
diag	Logical. Default is FALSE. Should diagonal elements be filled first for common elements.
iters	Integer. Default is 1. Number of iterations
intercept	List. Default is NULL. A list of length K containing numeric vectors of length d representing the intercept values. If NULL, intercepts are set to 0.

`multivar_sim_growth` *Simulate multivar data with time-varying dynamics according to a latent growth curve.*

Description

Simulate multivar data with time-varying dynamics according to a latent growth curve.

Usage

```
multivar_sim_growth(
  k,
  d,
  n,
  prop_fill_com,
  prop_fill_ind,
  prop_fill_com_growth,
  prop_fill_ind_growth,
  offset = 0,
  lb,
  ub,
  lb_int,
  ub_int,
  lb_slope,
  ub_slope,
  lb_quad = 0,
  ub_quad = 0,
  sigma,
  diag = FALSE,
  iters = 1,
  type = "linear",
  extreme_cutoff = 10,
  intercept = NULL
)
```

Arguments

`k`	Integer. The number of individuals (or datasets) to be generated.
`d`	Integer. The number of variables per dataset. For now this will be constant across individuals.
`n`	Integer. The time series length.
`prop_fill_com`	Numeric. The proportion of nonzero paths in the common transition matrix.
`prop_fill_ind`	Numeric. The proportion of nonzero unique (not in the common transition matrix or transition matrix of other individuals) paths in each individual transition matrix.

prop_fill_com_growth
: Numeric. The proportion of nonzero time-varying paths in the common transition matrix.

prop_fill_ind_growth
: Numeric. The proportion of nonzero time-varying unique (not in the common transition matrix or transition matrix of other individuals) paths in each individual transition matrix.

offset
: Numeric. An offset dictating when dynamics should transition from time-invariant to time-varying.

lb
: Numeric. The upper bound for individual elements of the transition matrices.

ub
: Numeric. The lower bound for individual elements of the transition matrices.

lb_int
: Numeric. The upper bound for individual elements of the transition matrices.

ub_int
: Numeric. The lower bound for individual elements of the transition matrices.

lb_slope
: Numeric. The upper bound for linear growth coefficients.

ub_slope
: Numeric. The lower bound for linear growth coefficients.

lb_quad
: Numeric. The upper bound for quadratic growth coefficients.

ub_quad
: Numeric. The lower bound for quadratic growth coefficients.

sigma
: Matrix. The (population) innovation covariance matrix.

diag
: Logical. Default is FALSE. Should diagonal elements be filled first for common elements.

iters
: Integer. Default is 1. Number of iterations.

type
: Character. Default is linear.

extreme_cutoff
: Numericr. Default is 10.

intercept
: List. Default is NULL. A list of length K containing numeric vectors of length d representing the intercept values. If NULL, intercepts are set to 0.

`multivar_sim_subgroups` *Simulate multivar data with explicit subgroup structure*

Description

Generates VAR data with a hierarchical structure: common effects (shared by all), subgroup effects (shared within groups), and unique effects (individual-specific). Importantly, these three components occupy NON-OVERLAPPING positions in the coefficient matrices, ensuring clean identifiability.

Usage

```
multivar_sim_subgroups(
  k,
  d,
  n,
  subgroup,
  p_com = d/d^2,
  p_sub = 0.05,
  p_ind = 0.05,
  sigma = NULL,
  lb = 0,
  ub = 1,
  intercept = NULL,
  max_attempts = 100,
  max_eig_threshold = 0.99
)
```

Arguments

k	Total number of subjects (sum across all subgroups)
d	Number of variables
n	Number of timepoints
subgroup	Integer vector of length k indicating subgroup membership. For example, c(1,1,1,2,2,2) indicates 3 subjects in group 1 and 3 in group 2.
p_com	Proportion of positions allocated to common effects. Sampled from diagonal positions. Default: d/d^2 (i.e., diagonal only).
p_sub	Proportion of positions allocated to subgroup effects (per group). Sampled from off-diagonal positions excluding common. Default: 0.05.
p_ind	Proportion of positions allocated to unique effects (per individual). Sampled from remaining positions. Default: 0.05.
sigma	Innovation covariance matrix. If NULL, uses identity matrix.
lb	Lower bound for coefficient values. Default: 0.
ub	Upper bound for coefficient values. Default: 1.
intercept	Optional list of intercept vectors (one per subject). If NULL, no intercepts are used. Default: NULL.
max_attempts	Maximum number of attempts to generate stationary dynamics. Default: 100.
max_eig_threshold	Maximum absolute eigenvalue allowed for stationarity. Default: 0.99.

Details

The function generates a three-level hierarchical structure where:

Common effects Shared by all subjects, typically on diagonal positions

Subgroup effects Shared within subgroups, on distinct off-diagonal positions

Unique effects Individual-specific, on remaining positions

The key feature is that these three components occupy NON-OVERLAPPING positions in the coefficient matrices, which ensures identifiability of the decomposition:

$$total[i] = common + subgroup[g] + unique[i]$$

The function repeatedly generates matrices until all subjects have stationary dynamics (maximum absolute eigenvalue < max_eig_threshold).

Value

A list with the following components:

data List of k data matrices (each n x d)

subgroup Vector of subgroup memberships

mat_com Common effect matrix (d x d)

mat_sub_unique List of k subgroup effect matrices (replicated within groups)

mat_ind_unique List of k unique effect matrices

mat_ind_final List of k total effect matrices

intercept List of intercept vectors (if provided)

Examples

```
## Not run:
# Simulate 6 subjects in 2 subgroups
sim <- multivar_sim_subgroups(
  k = 6,
  d = 4,
  n = 100,
  subgroup = c(1, 1, 1, 2, 2, 2),
  p_com = 0.2,
  p_sub = 0.1,
  p_ind = 0.05
)

# Fit model
object <- constructModel(sim$data, subgroup_membership = sim$subgroup)
fit <- cv.multivar(object)

# Evaluate including subgroup effects
perf <- eval_multivar_performance(sim, fit)
print(perf)

## End(Not run)
```

`multivar_sim_tvp` *Simulate multivar TVP data (piecewise or continuous)*

Description

Simulate multivar TVP data (piecewise or continuous)

Usage

```
multivar_sim_tvp(
  k,
  d,
  n,
  sigma,
  phi_common_list,
  phi_unique_list = NULL,
  T_per_period = NULL,
  mat_tvp = NULL,
  type = c("piecewise", "continuous"),
  burn = 500,
  intercept = NULL
)
```

Arguments

`k`	Integer. Number of subjects.
`d`	Integer. Number of variables per subject.
`n`	Integer. Total time series length per subject.
`sigma`	Matrix. Innovation covariance matrix (d x d).
`phi_common_list`	List of d x d common transition matrices. For piecewise: one per period. For continuous: one base matrix.
`phi_unique_list`	List of lists. phi_unique_list[[subject]][[period]] gives the unique effects matrix for that subject/period. For continuous: phi_unique_list[[subject]] is one matrix.
`T_per_period`	Integer vector. Observations per period (piecewise only). If NULL, periods are equal length.
`mat_tvp`	Logical matrix (d x d). Which elements are time-varying (continuous only).
`type`	Character. Either "piecewise" or "continuous".
`burn`	Integer. Burn-in period to discard. Default 500.
`intercept`	Intercept term. Can be: - NULL (zeros for all) - List of vectors: intercept[[subject]] is length d (constant over periods) - List of lists: intercept[[subject]][[period]] is length d (period-specific)

Value

List with components:

data	List of T x d data matrices, one per subject
mat_com	Common transition matrix (piecewise: list per period; continuous: list per time-point)
mat_ind_unique	List of unique effects (subject x period)
mat_ind_final	List of total effects (subject x period)
T_per_period	Observations per period
breaks	Break points for period changes

Examples

```
# Piecewise TVP with 2 subjects, 2 periods
d <- 3
phi_com1 <- matrix(0, d, d); phi_com1[1,2] <- 0.3
phi_com2 <- matrix(0, d, d); phi_com2[1,2] <- 0.3; phi_com2[2,3] <- 0.2

phi_uniq <- list(
  list(matrix(0, d, d), matrix(0, d, d)),  # subject 1: no unique effects
  list(matrix(0, d, d), matrix(0, d, d))   # subject 2: no unique effects
)
phi_uniq[[2]][[1]][3,1] <- 0.2  # subject 2, period 1 has unique edge

sim <- multivar_sim_tvp(
  k = 2, d = d, n = 200, sigma = diag(d),
  phi_common_list = list(phi_com1, phi_com2),
  phi_unique_list = phi_uniq,
  T_per_period = c(100, 100),
  type = "piecewise"
)
```

---

plot_cv_lambda_grid *Plot CV error over (lambda1, lambda2)*

Description

Plot CV error over (lambda1, lambda2)

Usage

```
plot_cv_lambda_grid(fit, object = NULL, log_scale = TRUE, show_best = TRUE)
```

Arguments

fit	result from cv.multivar(...)
object	OPTIONAL; only used if fit does NOT contain hyperparams or obj
log_scale	logical; plot on log10 scale?
show_best	logical; mark best pair?

---

plot_results *Plot fitted multivar model results*

Description

Creates heatmap visualizations of transition matrices from fitted multivar models. Supports plotting common effects, unique (subject-specific) effects, total effects, subgroup effects, and time-varying parameter (TVP) effects.

Usage

```
plot_results(
  x,
 plot_type = c("common", "unique", "total", "subgrp", "tvp", "common_tvp", "tvp_total"),
  facet_ncol = 3,
  subjects = "all",
  periods = "all",
  ub = 1,
  lb = -1,
  palette = "default",
  show_zeros = FALSE,
  ...
)
```

Arguments

x	Object returned by `cv.multivar()` or related fitting functions
plot_type	Character. Type of effects to plot: • "common" - Common effects shared across all subjects • "unique" - Subject-specific unique effects • "total" - Total effects (common + unique for each subject) • "subgrp" - Subgroup effects (if model includes subgroups) • "tvp" - Time-varying parameter effects (unique TVP deviations) • "common_tvp" - Common time-varying effects across subjects • "tvp_total" - Total effects for TVP models by period
facet_ncol	Numeric. Number of columns when faceting multiple matrices
subjects	Character "all" or numeric vector. Which subjects to plot for "unique", "total", or "tvp" types. Default is "all"

periods	Character "all" or numeric vector. Which time periods to plot for TVP models. Default is "all"
ub	Numeric. Upper bound for color scale. Default is 1
lb	Numeric. Lower bound for color scale. Default is -1
palette	Character. Color palette: "default", "viridis", or "greyscale"
show_zeros	Logical. If FALSE (default), zero values shown as white/NA
...	Additional arguments passed to plotting engine

Value

A ggplot2 object that can be further customized

See Also

plot_sim, plot_transition_mat

Examples

```
## Not run:
sim <- multivar_sim(k = 3, d = 5, n = 50, prop_fill_com = 0.2,
                    prop_fill_ind = 0.1, sigma = diag(5))
model <- constructModel(sim$data)
fit <- cv.multivar(model)

# Plot common effects
plot_results(fit, plot_type = "common")

# Plot unique effects for first 2 subjects
plot_results(fit, plot_type = "unique", subjects = 1:2)

# For TVP models
fit_tvp <- cv.multivar(constructModel(data, tvp = TRUE, breaks = list(c(50, 100))))
plot_results(fit_tvp, plot_type = "tvp")  # TVP deviations by period
plot_results(fit_tvp, plot_type = "tvp_total")  # Total effects by period

# Customize the plot
p <- plot_results(fit, plot_type = "total")
p + ggplot2::ggtitle("My Custom Title") + ggplot2::theme_minimal()

## End(Not run)
```

---

plot_sim *Plot simulated multivar ground truth*

Description

Creates heatmap visualizations of true transition matrices from multivar simulations. Supports plotting common effects, unique (subject-specific) effects, total effects, and subgroup effects.

Usage

```
plot_sim(
  x,
  plot_type = c("common", "unique", "total", "subgrp"),
  facet_ncol = 3,
  subjects = "all",
  ub = 1,
  lb = -1,
  palette = "default",
  show_zeros = FALSE,
  ...
)
```

Arguments

x	Object returned by `multivar_sim()` or `multivar_sim_subgroups()`
plot_type	Character. Type of effects to plot: • "common" - Common effects shared across all subjects • "unique" - Subject-specific unique effects • "total" - Total effects (common + unique for each subject) • "subgrp" - Subgroup effects (if simulation includes subgroups)
facet_ncol	Numeric. Number of columns when faceting multiple matrices
subjects	Character "all" or numeric vector. Which subjects to plot for "unique" or "total" types. Default is "all"
ub	Numeric. Upper bound for color scale. Default is 1
lb	Numeric. Lower bound for color scale. Default is -1
palette	Character. Color palette: "default", "viridis", or "greyscale"
show_zeros	Logical. If FALSE (default), zero values shown as white/NA
...	Additional arguments passed to plotting engine

Value

A ggplot2 object that can be further customized

See Also

plot_results, plot_transition_mat

Examples

```
## Not run:
# Simulate data
sim <- multivar_sim(k = 3, d = 5, n = 50,
                    prop_fill_com = 0.2, prop_fill_ind = 0.1,
                    lb = 0.1, ub = 0.9, sigma = diag(5))
```

```
# Plot true common effects
plot_sim(sim, plot_type = "common")

# Plot true unique effects for subjects 1-2
plot_sim(sim, plot_type = "unique", subjects = 1:2)

# Simulate with subgroups
sim_sub <- multivar_sim_subgroups(k = 4, d = 3, n = 40,
                                  subgroup = c(1, 1, 2, 2),
                                  p_com = 0.25, p_sub = 0.10, p_ind = 0.05,
                                  sigma = diag(3))

# Plot subgroup effects
plot_sim(sim_sub, plot_type = "subgrp")

## End(Not run)
```

`plot_transition_mat` *Plot arbitrary transition matrix*

Description

Creates a heatmap visualization of an arbitrary transition matrix. Useful for plotting custom matrices or comparing different estimates.

Usage

```
plot_transition_mat(
  x,
  title = NULL,
  subtitle = NULL,
  ub = 1,
  lb = -1,
  legend = TRUE,
  dimnames = TRUE,
  palette = "default",
  show_zeros = FALSE,
  ...
)
```

Arguments

`x`	Matrix. A transition matrix to plot
`title`	Character. Main title for the plot
`subtitle`	Character. Subtitle for the plot
`ub`	Numeric. Upper bound for color scale. Default is 1

lb	Numeric. Lower bound for color scale. Default is -1
legend	Logical. Should a legend be included? Default TRUE
dimnames	Logical. Should variable names be shown? Default TRUE
palette	Character. Color palette: "default", "viridis", or "greyscale"
show_zeros	Logical. If FALSE (default), zero values shown as white/NA
...	Additional arguments passed to plotting engine

Value

A ggplot2 object that can be further customized

See Also

plot_results, plot_sim

Examples

```
## Not run:
# Plot random matrix
plot_transition_mat(matrix(rnorm(25), 5, 5), title = "Random Matrix")

# Plot with custom bounds
mat <- matrix(runif(25, -0.5, 0.5), 5, 5)
plot_transition_mat(mat, title = "Small Coefficients", lb = -0.5, ub = 0.5)

# Without legend or dimension names
plot_transition_mat(mat, legend = FALSE, dimnames = FALSE)

## End(Not run)
```

---

pretrained_var_stage1_glmnet

Stage 1 (glmnet): common block only, with optional per-equation intercepts

---

Description

Fits the common block using cv.glmnet on a block-diagonal design. Returns A_pre (d x (d+1)); if include_intercept = FALSE, the first column is 0.

Usage

```
pretrained_var_stage1_glmnet(
  object,
  include_intercept = FALSE,
  standardize = FALSE,
  foldid = NULL,
  lambda = NULL,
  lambest = "min"
)
```

Arguments

object	multivar object with slots: A (N x p), b (N x d), d (length-1 or length-d), nfolds, etc.
include_intercept	logical; if TRUE, add one unpenalized intercept per outcome (default TRUE).
standardize	logical; passed to glmnet::cv.glmnet (default FALSE to mirror multivar defaults).
foldid	optional CV fold id. If length N, it will be replicated for each outcome (length N*d). If length N*d, used as-is. Otherwise ignored (glmnet will make its own).
lambda	optional numeric vector of lambdas to force glmnet to use (helps match multivar paths).
lambest	Character. Default is ‘"min"‘.

Value

list(A_pre, lambda_best, cvfit, include_intercept)

---

print.matrix_spec *Print method for matrix_spec*

Description

Print method for matrix_spec

Usage

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

Arguments

x	A matrix_spec object.
...	Additional arguments (ignored).

print_dynamics *Print dynamics matrix with clear formatting*

Description

Prints a dynamics matrix (or list of matrices) with clear row/column labels and formatting options. Handles matrices of different sizes gracefully. Can also accept a fit object directly (e.g., from cv.multivar()).

Usage

```
print_dynamics(
  dynamics,
  digits = 3,
  zero_char = ".",
  highlight_nonzero = TRUE,
  max_print = 15,
  time_labels = TRUE,
  subject_label = NULL,
  period_labels = NULL
)
```

Arguments

dynamics	A numeric matrix, a list of matrices (for multiple subjects), or a fit object from cv.multivar() (will automatically extract total effects)
digits	Number of decimal places to display (default: 3)
zero_char	Character to display for zero entries (default: ".")
highlight_nonzero	Logical; if TRUE, emphasize non-zero entries (default: TRUE)
max_print	Maximum number of rows/cols to print before truncating (default: 15)
time_labels	Logical; if TRUE, add subscripts like $V1_t$, $V1_{t-1}$ (default: TRUE)
subject_label	Optional label for the subject/matrix (e.g., "Subject 1")
period_labels	Optional character vector of period labels for TVP models

Value

Invisibly returns the input (for chaining)

Examples

```
# Simple 4x4 matrix
phi <- matrix(c(0.3, 0, 0.4, 0, 0.4, 0.5, 0, 0.3, 0, 0.3, 0.2, 0, 0, 0, 0, 0.4), 4, 4)
print_dynamics(phi)

# List of matrices
```

```
phi_list <- list(phi, phi * 0.8)
print_dynamics(phi_list)

# From fit object (automatic extraction)
# fit <- cv.multivar(object)
# print_dynamics(fit)  # Automatically extracts and prints total effects

# TVP fit object with period labels
# fit_tvp <- cv.multivar(object_tvp)
# print_dynamics(fit_tvp, period_labels = c("Pre", "During", "Post"))
```

`print_dynamics_tvp` *Print time-varying dynamics (TVP models)*

Description

Prints dynamics for TVP models where dynamics vary across time periods. This is a specialized wrapper around print_dynamics() for TVP structures.

Usage

```
print_dynamics_tvp(dynamics_tvp, period_labels = NULL, ...)
```

Arguments

`dynamics_tvp`	A list of matrices (one per period), or a list of lists (for multiple subjects, each containing a list of period-specific matrices)
`period_labels`	Optional character vector of period labels (e.g., c("Pre", "During", "Post"))
`...`	Additional arguments passed to print_dynamics()

Value

Invisibly returns the input (for chaining)

Examples

```
# Three periods with different dynamics
phi_period1 <- matrix(c(0.3, 0, 0.4, 0, 0.4, 0.5, 0, 0.3, 0, 0.3, 0.2, 0, 0, 0, 0, 0.4), 4, 4)
phi_period2 <- matrix(c(0.4, 0.3, 0, 0.2, 0, 0.6, 0.4, 0, 0.3, 0, 0.3, 0, 0, 0.4, 0, 0.5), 4, 4)
phi_period3 <- matrix(c(0.5, 0.2, 0, 0, 0, 0.4, 0.5, 0, 0.5, 0, 0.4, 0.2, 0, 0.5, 0, 0.3), 4, 4)

# For single subject
print_dynamics_tvp(list(phi_period1, phi_period2, phi_period3))

# With custom period labels
print_dynamics_tvp(list(phi_period1, phi_period2, phi_period3),
                   period_labels = c("Baseline", "Treatment", "Follow-up"))
```

print_edge_prevalence *Print edge prevalence across subjects*

Description

For models with multiple subjects (K>1), displays a matrix showing the proportion/percentage of subjects that have each edge (non-zero entry).

Usage

```
print_edge_prevalence(
  fit,
  type = c("proportion", "count"),
  digits = 2,
  zero_char = ".",
  threshold = 1e-10
)
```

Arguments

fit	A fitted multivar object (output from cv.multivar())
type	Character; "proportion" (default) or "count" to show counts instead
digits	Number of decimal places (default: 2)
zero_char	Character to display for edges present in 0 subjects (default: ".")
threshold	Threshold for considering an edge as present (default: 1e-10)

Value

Invisibly returns the prevalence matrix

Examples

```
# fit <- cv.multivar(object)  # K=2 or more subjects
# print_edge_prevalence(fit)
```

recover_intercepts *Recover Intercepts from Centered Data*

Description

After fitting the model on centered data, this function recovers the intercepts using the formula: c = mean(b) - Phi * mean(A). For multi-group models, it also decomposes intercepts into common and group-specific components.

Usage

```
recover_intercepts(
  mats,
  data_means,
  k,
  d,
  subgroup = FALSE,
  subgroup_membership = NULL,
  tvp = FALSE,
  breaks = NULL
)
```

Arguments

mats	List. The fitted coefficient matrices from breakup_transition, containing: - common: Common effects matrix (Psi) - unique: List of unique effects matrices (Psi^(k)) - total: List of total effects matrices (Phi^(k) = Psi + Psi^(k)) - subgrp: List of subgroup effects matrices (if subgroups)
data_means	List. For each group, a list containing mean_b and mean_A from original data
k	Numeric. Number of groups
d	Numeric. Number of variables
subgroup	Logical. Whether subgroup structure is present
subgroup_membership	Numeric vector. Subgroup assignments for each subject
tvp	Logical. Whether time-varying parameters are present
breaks	List. Time period definitions for TVP models

Details

For each group k, the total intercept is recovered using: $c_total^(k) = mean(b^(k)) - Phi^(k) * mean(A^(k))$ where $b^{(k)}$ are the outcomes (Y_t) and $A^{(k)}$ are the predictors (Y_{t-1}). This is the same approach used by glmnet for intercept recovery.

For multi-group models (k > 1), intercepts are decomposed as: $c = (1/K) * sum_k(c_total^(k))$ # Common intercept $c^(k) = c_total^(k) - c$ # Group-specific intercept

This decomposition ensures the constraint: sum_k(c^(k)) = 0

Note: This formula c = mean(b) - Phi * mean(A) is different from the steady-state formula $c = (I - Phi) * mean(Y)$, which assumes mean(b) = mean(A) = mean(Y). In finite samples they give similar but not identical results. The formula used here is preferred as it matches standard practice in penalized regression (e.g., glmnet).

For TVP models, this function recovers time-varying intercepts c_t for each period. When data_means contains per-period means (mean_b_t, mean_A_t), time-specific intercepts are computed. Otherwise, falls back to base intercepts.

Value

List containing: - intercepts_total: List of total intercepts for each group (c_total^(k)) - intercept_common: Common intercept (c) - intercepts_unique: List of group-specific intercepts (c^(k)) - intercepts_subgrp: List of subgroup-specific intercepts (if applicable) - intercepts_tvp: List of time-varying intercepts for TVP models (if applicable)

`select_by_ebic` *Reselect best model using (E)BIC instead of MSFE*

Description

Takes the output of `cv.multivar()` and reselects the best penalty scenario using the Extended Bayesian Information Criterion (EBIC) or standard BIC. This can improve structure recovery (fewer false positives) compared to prediction-based cross-validation, especially when n » p.

Usage

```
select_by_ebic(fit, gamma = 0.5)
```

Arguments

`fit`	Output of `cv.multivar()`.
`gamma`	EBIC tuning parameter. `gamma = 0` gives standard BIC; `gamma = 0.5` (default) is a moderate EBIC; `gamma = 1` is the most conservative (strongest sparsity preference).

Value

A modified copy of `fit` with:

`mats`	Transition matrices from the EBIC-selected scenario
`ebic`	Named list: `values` (EBIC for all scenarios), `best_idx` (selected scenario index), `gamma` (tuning value)

The original MSFE results are preserved.

show.multivar *Default show method for an object of class multivar*

Description

Default show method for an object of class multivar

Usage

```
## S4 method for signature 'multivar'
show(object)
```

Arguments

object	multivar object created from ConstructModel

Value

Displays the following information about the multivar object:

- To do.

See Also

constructModel

summary_multivar *Summary for multivar fit objects*

Description

Prints a summary of a fitted multivar model, including dataset information, model specifications, and key results.

Usage

```
summary_multivar(fit)
```

Arguments

fit	A fitted multivar object (output from cv.multivar())

Value

Invisibly returns the input object

Examples

```
# fit <- cv.multivar(object)
# summary_multivar(fit)
```

`validate_matrix_spec` *Validate Matrix Specification*

Description

Checks internal consistency of a matrix_spec object.

Usage

```
validate_matrix_spec(spec)
```

Arguments

`spec`	A matrix_spec object

Value

TRUE if valid, otherwise throws an error

`var_sim_tvp` *Simulate a time-varying VAR model (piecewise or continuous)*

Description

Simulate a time-varying VAR model (piecewise or continuous)

Usage

```
var_sim_tvp(
  n,
  sigma,
  phi_list,
  T_per_period = NULL,
  mat_tvp = NULL,
  type = c("piecewise", "continuous"),
  burn = 500,
  intercept = 0
)
```

Arguments

n	Integer. Total time series length.
sigma	Matrix. Innovation covariance matrix (d x d).
phi_list	List of d x d transition matrices. For piecewise: one per period. For continuous: one base matrix.
T_per_period	Integer vector. Observations per period (piecewise only). If NULL, periods are equal length.
mat_tvp	Logical matrix (d x d). Which elements are time-varying (continuous only).
type	Character. Either "piecewise" or "continuous".
burn	Integer. Burn-in period to discard. Default 500.
intercept	Intercept term. Can be: - Scalar (same for all variables and periods) - Vector of length d (different per variable, constant over periods) - List of vectors (different per period, for piecewise TVP)

Value

List with components:

data	List containing the T x d data matrix
mat_ind_final	List containing the true transition matrices (list of period matrices)
T_per_period	Observations per period
breaks	Break points for period changes
intercept	Intercept values used

Examples

```
# Piecewise TVP
phi1 <- matrix(c(0.3, 0, 0, 0.3), 2, 2)
phi2 <- matrix(c(0.3, 0.2, 0, 0.3), 2, 2)
sim <- var_sim_tvp(n = 200, sigma = diag(2), phi_list = list(phi1, phi2),
                   T_per_period = c(100, 100), type = "piecewise")

# Piecewise with period-specific intercepts
sim <- var_sim_tvp(n = 200, sigma = diag(2), phi_list = list(phi1, phi2),
                   T_per_period = c(100, 100), type = "piecewise",
                   intercept = list(c(1, 0), c(2, 1)))
```

Index

∗ **datasets**
dat_multivar_sim, 15
∗ **multivar**
multivar_sim, 31
multivar_sim_breaks, 32
multivar_sim_growth, 34
∗ **simulate**
multivar_sim, 31
multivar_sim_breaks, 32
multivar_sim_growth, 34
∗ **var**
multivar_sim, 31
multivar_sim_breaks, 32
multivar_sim_growth, 34

build_matrix_spec, 3, *29*
build_scenario_table, 5

canonical.multivar, 5
canonical.multivar,multivar-method *(*canonical.multivar*)*, 5
coef.multivar_fit *(*multivar_fit_methods*)*, 30
compute_ebic, 6
constructModel, *6*, 7, *27*, *30*, *51*
cv.multivar, 11
cv.multivar,multivar-method *(*cv.multivar*)*, 11
cv_blocked, 12
cv_multivar, 13

dat_multivar_sim, 15

estimate_initial_coefs, 15
eval_multivar_performance, 17
expand_range, 17
extract_multivar_hyperparams, 18

get_common_cols, 19
get_common_effects *(*get_dynamics_helpers*)*, 20
get_common_tvp_cols, 19
get_dynamics *(*get_dynamics_helpers*)*, 20
get_dynamics_all_subjects *(*get_dynamics_helpers*)*, 20
get_dynamics_helpers, 20
get_n_periods, 21
get_period_rows, 22
get_subgrp_cols, 22
get_subgrp_cols_for_subject, 23
get_subject_rows, 23
get_total_effects *(*get_dynamics_helpers*)*, 20
get_unique_cols, 24
get_unique_effects *(*get_dynamics_helpers*)*, 20
get_unique_tvp_cols, 24

is_tvp, 25

lambda_grid, 25

maity_debiasing, 27
multivar-class, 27
multivar-package, 3
multivar_fit_methods, 30
multivar_sim, 31
multivar_sim_breaks, 32
multivar_sim_growth, 34
multivar_sim_subgroups, 35
multivar_sim_tvp, 38

plot.multivar_fit *(*multivar_fit_methods*)*, 30
plot_cv_lambda_grid, 39
plot_results, 40, *42*, *44*
plot_sim, 41, *41*, *44*
plot_transition_mat, *41*, *42*, 43
pretrained_var_stage1_glmnet, 44
print.matrix_spec, 45
print.multivar_fit *(*multivar_fit_methods*)*, 30

print_dynamics, 46
print_dynamics_tvp, 47
print_edge_prevalence, 48

recover_intercepts, 48

select_by_ebic, 50
show,multivar-method *(*show.multivar*)*, 51
show.multivar, 51
summary.multivar_fit
*(*multivar_fit_methods*)*, 30
summary_multivar, 51

validate_matrix_spec, 52
var_sim_tvp, 52