Package ‘matSPACE’

September 12, 2026

Title Sparse Partial Correlation Estimation for Matrix-Variate Data

Version 0.1.0

Description Fits sparse partial correlation networks for matrix-variate data by extending the SPACE joint partial correlation estimation framework to a Kronecker-product covariance structure. All partial correlations are estimated simultaneously via an L1-penalized ('lasso') shooting algorithm within a single optimization framework, which preserves symmetry of the estimated network and avoids the tuning-parameter selection difficulties of separate node-wise regressions. Optional features include column reweighting, residual variance re-estimation across outer iterations, and automatic generation of a lasso penalty sequence for tuning.

License GPL (>= 3)

Encoding UTF-8

LinkingTo Rcpp

Imports Rcpp, stats

Config/roxygen2/version 8.1.0

URL https://github.com/kimhyew1/matSPACE

BugReports https://github.com/kimhyew1/matSPACE/issues

NeedsCompilation yes

Author Hyewon Kim [aut, cre],
Seongoh Park [aut]

Maintainer Hyewon Kim <kimhw4126@gmail.com>

Repository CRAN

Date/Publication 2026-09-12 13:40:09 UTC

Contents

---

lambda.bound *Generate a log-spaced lasso penalty sequence for* space()

Description

Computes a decreasing, log-spaced sequence of K candidate lasso penalties (lam values) for tuning space(), analogous to the lambda_max-based grids used in lasso path algorithms. The largest value, lambda_max, is the largest off-diagonal entry of a variance-rescaled Gram matrix — the smallest penalty above which every off-diagonal partial correlation coefficient is driven to zero; the sequence descends geometrically to lambda_max * eps.

Usage

```
lambda.bound(dt, eps = 1e-06, K = 30)
```

Arguments

dt	list of n matrices, each p x q, in the same format expected by the data argument of space().
eps	ratio of the smallest to the largest penalty in the returned sequence, i.e. lambda_min = lambda_max * eps.
K	number of penalty values to generate.

Value

A numeric vector of length K, decreasing geometrically from lambda_max to lambda_max * eps, suitable to pass one at a time as the lam argument of space() (e.g. selecting among the fits with a BIC-type criterion).

Examples

```
set.seed(1)
p <- 5; q <- 4; n <- 3
data <- replicate(n, matrix(rnorm(p * q), p, q), simplify = FALSE)
lambda.bound(data, K = 10)
```

matSPACE — *Estimate row/column precision matrices for Kronecker-structured (matrix-variate) SPACE, with identifiability resolved*

Description

Fits `space()` independently on the data (for the column precision `V`, q x q) and on the transposed data (for the row precision `U`, p x p), each over a lasso penalty path, selects the BIC-minimizing lambda for `U` and for `V` separately at every scaling factor in `sf_vec`, and reconstructs the corresponding precision matrices (via `precision_from_parcor()`). Because the Kronecker product `kronecker(V, U)` is invariant under `(V / c, U * c)` for any `c` > `0`, the pair is not separately identifiable from the data; the result is rescaled so that `V[1, 1]` == `1`, using `c` equal to the raw fitted `V[1, 1]` (`U` is multiplied by that same `c`), which preserves `kronecker(V, U)` exactly.

Usage

```
matSPACE(
  data,
  lambda_V = NULL,
  lambda_U = NULL,
  K = 30,
  f_type_V = "equal",
  f_type_U = "equal",
  sf_vec = c(1, 1.5)
)
```

Arguments

`data`	list of n matrices, each p x q; the matrix-variate observations, in the same format expected by the `data` argument of `space()`.
`lambda_V`	optional numeric vector of lasso penalties to use for the column (`V`, q x q) fit. If `NULL` (default), generated via `lambda.bound()` on `data` with `K` values.
`lambda_U`	optional numeric vector of lasso penalties to use for the row (`U`, p x p) fit. If `NULL` (default), generated via `lambda.bound()` on the transposed data with `K` values.
`K`	number of lambda values to generate with `lambda.bound()` when `lambda_V` or `lambda_U` is `NULL`. Ignored for whichever of the two is supplied directly.
`f_type_V`	column weighting scheme forwarded to `space()` for the `V` fit; see `compute_weight()`.
`f_type_U`	column weighting scheme forwarded to `space()` for the `U` fit; see `compute_weight()`.
`sf_vec`	numeric vector of BIC scaling factors (the `sf` argument of `compute_BIC()`). A separate BIC-minimizing lambda — and resulting identifiability-resolved `(U, V)` pair — is returned for each value.

Value

A named list, one element per value of `sf_vec` (named `sf_<value>`), each a list with components:

`V` — q x q precision matrix at the BIC-minimizing `lambda_V`, rescaled so `V[1, 1] == 1`.

`U` — p x p precision matrix at the BIC-minimizing `lambda_U`, rescaled by the same factor to preserve `kronecker(V, U)`.

`lambda_V`, `lambda_U` — the BIC-minimizing lambda for `V` and `U` at this `sf`.

`BIC_V`, `BIC_U` — the corresponding minimum BIC values.

`sf` — the scaling factor this element was selected at.

The full lambda paths behind this selection are attached as a `"path"` attribute (`attr(fit, "path")`), a list with `V` and `U` components, each with `fits` (the raw `space()` fit at every lambda tried), `bic` (a `length(lambda)` x `length(sf_vec)` matrix, named `sf_<value>`), and `lambda`. Useful for plotting BIC (or the number of nonzero edges, from `fits[[i]]$ParCor`) against lambda without refitting.

Examples

```
set.seed(1)
p <- 4; q <- 3; n <- 3
data <- replicate(n, matrix(rnorm(p * q), p, q), simplify = FALSE)
fit <- matSPACE(data, K = 5, sf_vec = c(1, 1.5))
fit$sf_1$V
fit$sf_1$U
path <- attr(fit, "path")
plot(path$V$lambda, path$V$bic[, "sf_1"], type = "b")
```

---

space — *Fit a matrix-variate SPACE sparse partial correlation model*

Description

Estimates a sparse network of partial correlations among the q columns of matrix-variate data using an L1-penalized (lasso) SPACE-style shooting algorithm, with optional per-column reweighting and residual variance (`sig`) re-estimation across outer iterations.

Usage

```
space(
  data,
  lam,
  sig = NULL,
  f_type = "equal",
  iter = 2,
  beta_init = NULL,
  sig_init = NULL
)
```

Arguments

`data` list of n matrices, each p × q. All matrices share the same p × q shape; n is the number of independent replicates and the q columns are the variables whose pairwise partial correlations are estimated.

`lam` lasso penalty applied to the off-diagonal partial correlation coefficients.

`sig` optional length-q sigma vector; NULL = iterative estimate

`f_type` "equal" | "variance" | "degree"

`iter` number of outer iterations

`beta_init` optional warm-start for beta: a q x q matrix/flat vector (row-major, i*q+j layout) forwarded to the first internal `space_shooting()` call instead of the default univariate soft-threshold initialization. Only meaningful for the first outer iteration, since that is the only one whose sigma_sr (sqrt(SIG/WEIGHT)) does not depend on lam — the first pass always starts from SIG = rep(1, q), WEIGHT = rep(1, q) regardless of lam, so a beta fit at a neighboring lambda is a valid warm start there. Requires the compiled `space_shooting()` routine in `src/matSPACE.cpp`, which accepts a `beta_init` argument.

`sig_init` optional length-q warm-start for the STARTING value of SIG when SIG.update = TRUE (i.e. sig = NULL). Unlike `sig`, this does not pin SIG constant — it is still re-estimated every outer iteration via estimate_sigma(); it only replaces the rep(1, q) cold start at i = 1 with a neighboring lambda's converged SIG, so f_type = "variance"/"degree" don't force a uniform-weight first pass before ever seeing a realistic weighting.

Value

A list with components:

`ParCor` q x q matrix of estimated partial correlations (diagonal = 1).

`beta` q x q matrix of estimated regression coefficients.

`sig` length-q vector of estimated column residual precisions (1/variance).

`f` length-q vector of final per-column weights.

`total_iter` total number of shooting iterations across all outer iterations.

`E` list of n residual matrices (p x q each).

Examples

```
set.seed(1)
p <- 5; q <- 4; n <- 3
data <- replicate(n, matrix(rnorm(p * q), p, q), simplify = FALSE)
fit <- space(data, lam = 0.1, iter = 1)
fit$ParCor
```

Index