Package 'matSPACE'

November 17, 2026

Title Sparse Partial Correlation Estimation for Matrix-Variate Data

Version 0.3.0

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. Columns are reweighted and residual variances are re-estimated across outer iterations until convergence. A lasso penalty sequence is generated automatically and the penalty is selected by BIC; the whole penalty path is fitted in compiled code from the Gram matrix of the data.

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 2026-11-17 11:00:02 UTC

Contents

lambda.bound 2
matSPACE 3
space 5

Index **7**

`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 of the column-centered data (centered as in `space()`, by the column means over all `n * p` rows); the sequence descends geometrically to `lambda_max * eps`.

Usage

```
lambda.bound(dt, eps = 1e-04, K = 80)
```

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()`, or as `lambda_V` / `lambda_U` of `matSPACE()`.

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 the space() model 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, and reconstructs the corresponding precision matrices. 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 = 80,
  f_type_V = "variance",
  f_type_U = "variance",
  iter = 500,
  outer_tol = 1e-08,
  sf = 1,
  eps = 1e-04
)
```

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. Pass a single value to fit at that one penalty directly, without searching a path.
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. Pass a single value to fit at that one penalty directly, without searching a path.
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 for the V fit, as f_type of space(): "variance" (default), "equal" or "degree".
f_type_U	column weighting scheme for the U fit, as for f_type_V.
iter	maximum number of outer iterations at each lambda, as in space().

outer_tol	convergence tolerance of the outer iterations, as in space(); 0 always runs iter outer iterations.
sf	scale factor of the log-RSS term of the BIC (see Details).
eps	ratio of the smallest to the largest generated penalty, passed to lambda.bound().

Details

Each path is fitted in compiled code from the Gram matrix of the column-centered data, with the same outer reweighting loop as space() at every lambda (each lambda is fitted from a cold start). Each fit is scored by

$$BIC = \sum_j \{N\, sf\, \log(RSS_j) + \log(N)\, deg_j/2\},$$

where RSS_j is the residual sum of squares of variable j, deg_j its number of nonzero partial correlations, and N = n * p for V (n * q for U).

When a dimension is large relative to the number of stacked rows (e.g. p > n * q for U), the outer reweighting may not converge at the smallest penalties; a larger eps (a shorter path) or a smaller iter keeps the fit fast. The outer_iter and cd_capped components of the path fits report convergence.

Value

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.
BIC_V, BIC_U	the corresponding minimum BIC values.

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, bic (a numeric vector of length length(lambda)) and lambda. Each element of fits is the fit at one lambda: a list with ParCor (partial correlations), sig (residual precisions), f (weights), rss and deg (the per-variable residual sums of squares and degrees behind the BIC), outer_iter (outer iterations used), total_iter (coordinate updates) and cd_capped (shooting runs that hit the sweep limit; 0 when the coordinate descent converged). Useful for plotting BIC (or the number of nonzero edges) 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)
fit$V
fit$U
path = attr(fit, "path")
plot(path$V$lambda, path$V$bic, 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 at a single lasso penalty, using an L1-penalized (lasso) SPACE-style shooting algorithm with per-column reweighting and residual precision (`sig`) re-estimation across outer iterations. The outer iterations stop when the partial correlations, residual precisions and weights have converged (`outer_tol`) or after `iter` iterations. `matSPACE()` runs the same algorithm over a whole penalty path.

Usage

```
space(
  data,
  lam,
  sig = NULL,
  f_type = "variance",
  iter = 500,
  beta_init = NULL,
  sig_init = NULL,
  outer_tol = 1e-08
)
```

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`	"variance" | "equal" | "degree": column weights, proportional to the estimated residual precisions, uniform, or proportional to the node degrees.
`iter`	maximum number of outer iterations (at least 2 for "variance" and "degree").
`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 = 1 / sample variance, 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

1 / sample variance 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.

`outer_tol` convergence tolerance of the outer iterations: they stop once the partial correlations (absolute change) and the residual precisions and weights (relative change) all change by less than `outer_tol`. Use 0 to always run exactly `iter` outer iterations.

Value

A list with components:

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

`beta` q x q matrix of the estimated partial correlations with a zero diagonal, as returned by the shooting algorithm.

`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 coordinate-descent updates across all outer iterations.

`outer_iter` number of outer iterations used.

`E` list of n residual matrices (p x q each) of the last shooting run (column-centered, weighted data).

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)
fit$ParCor
```

Index

lambda.bound, 2
lambda.bound(), *3*, *4*

matSPACE, 3
matSPACE(), *2*, *5*

space, 5
space(), *2–4*