Package: rags2ridges (via r-universe)

October 29, 2026

Type Package

Title Ridge Estimation of Precision Matrices from High-Dimensional Data

Version 2.2.9

Maintainer Carel F.W. Peeters <carel.peeters@wur.nl>

Description Proper L2-penalized maximum likelihood estimators for precision matrices and supporting functions to employ these estimators in a graphical modeling setting. For details, see Peeters, Bilgrau, & van Wieringen (2022) <doi:10.18637/jss.v102.i04> and associated publications.

Depends R (>= 2.15.1)

Imports igraph, stats, methods, expm, reshape, ggplot2, Hmisc, fdrtool, snowfall, sfsmisc, utils, grDevices, graphics, gRbase, RBGL, graph, Rcpp, RSpectra

LinkingTo Rcpp, RcppArmadillo

KeepSource yes

License GPL (>= 2)

URL https://cfwp.github.io/rags2ridges/, https://github.com/CFWP/rags2ridges

Suggests KEGGgraph, testthat, knitr, rmarkdown

RoxygenNote 7.2.3

Encoding UTF-8

VignetteBuilder knitr

NeedsCompilation yes

Author Carel F.W. Peeters [aut, cre, cph] (ORCID: <https://orcid.org/0000-0001-5766-9969>), Anders Ellern Bilgrau [aut, cph] (ORCID: <https://orcid.org/0000-0001-9875-2902>), Wessel N. van Wieringen [aut] (ORCID: <https://orcid.org/0000-0002-5100-9123>)

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

Date/Publication 2025-12-01 16:50:09 UTC

RemoteUrl https://github.com/cran/rags2ridges

RemoteRef HEAD

RemoteSha a207db84778948f67026b7d8289b062beaceebaf

Contents

rags2ridges-package . . . 3
ADdata . . . 6
adjacentMat . . . 7
CNplot . . . 8
Communities . . . 11
conditionNumberPlot . . . 13
covML . . . 15
covMLknown . . . 16
createS . . . 17
default.penalty . . . 20
default.target . . . 22
default.target.fused . . . 24
DiffGraph . . . 25
edgeHeat . . . 27
evaluateS . . . 29
evaluateSfit . . . 30
fullMontyS . . . 32
fused.test . . . 34
GGMblockNullPenalty . . . 35
GGMblockTest . . . 37
GGMmutualInfo . . . 39
GGMnetworkStats . . . 40
GGMnetworkStats.fused . . . 42
GGMpathStats . . . 43
GGMpathStats.fused . . . 47
hist.ptest . . . 48
is.Xlist . . . 49
isSymmetricPD . . . 50
kegg.target . . . 51
KLdiv . . . 53
KLdiv.fused . . . 54
loss . . . 56
momentS . . . 57
NLL . . . 58
optPenalty.aLOOCV . . . 59
optPenalty.fused.grid . . . 62
optPenalty.kCV . . . 67
optPenalty.kCVauto . . . 70
optPenalty.LOOCV . . . 72
optPenalty.LOOCVauto . . . 74

pcor 76
pooledS 77
print.optPenaltyFusedGrid 78
print.ptest 80
pruneMatrix 81
ridgeP 82
ridgeP.fused 84
ridgePathS 86
ridgeS 88
rmvnormal 89
sparsify 90
sparsify.fused 92
symm 93
Ugraph 94
Union 97

Index **99**

`rags2ridges-package` *Ridge estimation for high-dimensional precision matrices*

Description

Package contains proper L2-penalized ML estimators for the precision matrix as well as supporting functions to employ these estimators in a (integrative or meta-analytic) graphical modeling setting.

Details

The main function of the package is `ridgeP` which enables archetypal and proper alternative ML ridge estimation of the precision matrix. The alternative ridge estimators can be found in van Wieringen and Peeters (2015) and encapsulate both target and non-target shrinkage for the multivariate normal precision matrix. The estimators are analytic and enable estimation in large p small n settings. Supporting functions to employ these estimators in a graphical modeling setting are also given. These supporting functions enable, a.o., the determination of the optimal value of the penalty parameter, the determination of the support of a shrunken precision estimate, as well as various visualization options.

The package has a modular setup. The *core module* (rags2ridges.R) contains the functionality stated above. The *fused module* (rags2ridgesFused.R) extends the functionality of the core module to the joint estimation of multiple precision matrices from (aggregated) high-dimensional data consisting of distinct classes. The result is a targeted fused ridge estimator that is of use when the precision matrices of the constituent classes are believed to chiefly share the same structure while potentially differing in a number of locations of interest. The fused module also contains supporting functions for integrative or meta-analytic Gaussian graphical modeling. The third module is the *miscellaneous module* (rags2RidgesMisc.R) which contains assorted hidden functions.

Function overview *core module*:

- Function for (proper) ridge estimation of the precision matrix

  - ridgeP
- Functions for penalty parameter selection
  - CNplot
  - optPenalty.aLOOCV
  - optPenalty.kCV
  - optPenalty.kCVauto
- Functions for loss/entropy/fit evaluation
  - evaluateSfit
  - KLdiv
  - loss
- Functions for block-independence testing
  - GGMblockNullPenalty
  - GGMblockTest
- Function for support determination
  - sparsify
- Functions for (network) visualization
  - edgeHeat
  - ridgePathS
  - Ugraph
- Functions for topology statistics
  - GGMmutualInfo
  - GGMnetworkStats
  - GGMpathStats
- Wrapper function
  - fullMontyS
- Support functions
  - adjacentMat
  - covML
  - covMLknown
  - default.target
  - evaluateS
  - pcor
  - symm

Function overview *fused module*:

- Function for targeted fused ridge estimation of multiple precision matrices
  - ridgeP.fused
- Function for fused penalty parameter selection
  - optPenalty.fused
- Functions for loss/entropy/fit evaluation

  - KLdiv.fused
  - NLL
- Function for testing the necessity of fusion
  - fused.test
- Function for support determination
  - sparsify.fused
- Functions for topology statistics
  - GGMnetworkStats.fused
  - GGMpathStats.fused
- Support functions
  - createS
  - default.penalty
  - default.target.fused
  - getKEGGPathway
  - isSymmetricPD
  - is.Xlist
  - kegg.target
  - plot.ptest
  - pooledS
  - print.optPenaltyFusedGrid
  - print.ptest
  - rmvnormal

Calls of interest to *miscellaneous module*:

- `rags2ridges:::.TwoCents()` ~~(Unsolicited advice)
- `rags2ridges:::.Brooke()` ~~(Endorsement)
- `rags2ridges:::.JayZScore()` ~~(The truth)
- `rags2ridges:::.theHoff()` ~~(Wish)
- `rags2ridges:::.rags2logo()` ~~(Warm welcome)

Author(s)

Carel F.W. Peeters, Anders Ellern Bilgrau, Wessel, N. van Wieringen
Maintainer: Carel F.W. Peeters <carel.peeters@wur.nl>

References

Peeters, C.F.W., Bilgrau, A.E., and van Wieringen, W.N. (2022). rags2ridges: A One-Stop-l2-Shop for Graphical Modeling of High-Dimensional Precision Matrices. Journal of Statistical Software, vol. 102(4): 1-32.

Bilgrau, A.E., Peeters, C.F.W., Eriksen, P.S., Boegsted, M., and van Wieringen, W.N. (2020). Targeted Fused Ridge Estimation of Inverse Covariance Matrices from Multiple High-Dimensional

Data Classes. Journal of Machine Learning Research, 21(26): 1-52. Also available as arXiv:1509.07982v2 [stat.ME].

Peeters, C.F.W., van de Wiel, M.A., & van Wieringen, W.N. (2020). The Spectral Condition Number Plot for Regularization Parameter Evaluation. Computational Statistics, 35: 629-646. Also available as arXiv:1608.04123 [stat.CO].

van Wieringen, W.N. & Peeters, C.F.W. (2016). Ridge Estimation of Inverse Covariance Matrices from High-Dimensional Data. Computational Statistics & Data Analysis, vol. 103: 284-303. Also available as arXiv:1403.0904v3 [stat.ME].

van Wieringen, W.N. & Peeters, C.F.W. (2015). Application of a New Ridge Estimator of the Inverse Covariance Matrix to the Reconstruction of Gene-Gene Interaction Networks. In: di Serio, C., Lio, P., Nonis, A., and Tagliaferri, R. (Eds.) 'Computational Intelligence Methods for Bioinformatics and Biostatistics'. Lecture Notes in Computer Science, vol. 8623. Springer, pp. 170-179.

`ADdata` *R-objects related to metabolomics data on patients with Alzheimer's Disease*

Description

`ADdata` contains 3 objects related to metabolomics data on patients with Alzheimer's Disease (AD).

Details

`ADmetabolites` is a `matrix` containing metabolic expressions of 230 metabolites (rows) on 127 samples (columns).

`sampleInfo` is a `data.frame` containing information on the samples. Information pertains to diagnosis: AD class 1 or AD class 2.

`variableInfo` is a `data.frame` containing information on the metabolic features. Information pertains to compound families: Amines, organic acids, lipids, and oxidative stress compounds.

See description.

Author(s)

Carel F.W. Peeters <carel.peeters@wur.nl>

Source

de Leeuw, F., Peeters, C.F.W., Kester, M.I., Harms, A.C., Struys, E., Hankemeijer, T., van Vlijmen, H.W.T., van Duijn, C.M., Scheltens, P., Demirkan, A., van de Wiel, M.A., van der Flier, W.M., and Teunissen, C.E. (2017). Blood-based metabolic signatures in Alzheimer's Disease. Alzheimer's & Dementia: Diagnosis, Assessment & Disease Monitoring, 8: 196-207.

Examples

```
data(ADdata)

## Look at sample information
sampleInfo

## Look at feature information
variableInfo
```

---

adjacentMat *Transform real matrix into an adjacency matrix*

Description

Function that transforms a real matrix into an adjacency matrix. Intended use: Turn sparsified precision matrix into an adjacency matrix for undirected graphical representation.

Usage

```
adjacentMat(M, diag = FALSE)
```

Arguments

M	(Possibly sparsified precision) matrix.
diag	A logical indicating if the diagonal elements should be retained.

Value

Function returns an adjacency matrix.

Author(s)

Carel F.W. Peeters <carel.peeters@wur.nl>, Wessel N. van Wieringen

See Also

ridgeP, covML, sparsify, edgeHeat, Ugraph

Examples

```
## Obtain some (high-dimensional) data
p = 25
n = 10
set.seed(333)
X = matrix(rnorm(n*p), nrow = n, ncol = p)
colnames(X)[1:25] = letters[1:25]
Cx <- covML(X)
```

```
## Obtain regularized precision matrix
P <- ridgeP(Cx, lambda = 10, type = "Alt")

## Obtain sparsified partial correlation matrix
PC0 <- sparsify(P, threshold = "localFDR", FDRcut = .8)

## Obtain adjacency matrix
adjacentMat(PC0$sparsePrecision)
```

`CNplot` *Visualize the spectral condition number against the regularization parameter*

Description

Function that visualizes the spectral condition number of the regularized precision matrix against the domain of the regularization parameter. The function can be used to heuristically determine an acceptable (minimal) value for the penalty parameter.

Usage

```
CNplot(
  S,
  lambdaMin,
  lambdaMax,
  step,
  type = "Alt",
  target = default.target(S, type = "DUPV"),
  norm = "2",
  Iaids = FALSE,
  vertical = FALSE,
  value = 1e-100,
  main = "",
  nOutput = FALSE,
  verbose = TRUE,
  suppressChecks = FALSE
)
```

Arguments

`S`	Sample covariance `matrix`.
`lambdaMin`	A `numeric` giving the minimum value for the penalty parameter.
`lambdaMax`	A `numeric` giving the maximum value for the penalty parameter.
`step`	An `integer` determining the number of steps in moving through the grid [`lambdaMin`, `lambdaMax`].

type	A character indicating the type of ridge estimator to be used. Must be one of: "Alt", "ArchI", "ArchII".
target	A target matrix (in precision terms) for Type I ridge estimators.
norm	A character indicating the norm under which the condition number is to be calculated/estimated. Must be one of: "1", "2".
Iaids	A logical indicating if the basic condition number plot should be amended with interpretational aids.
vertical	A logical indicating if output graph should come with a vertical line at a pre-specified value for the penalty parameter.
value	A numeric indicating a pre-specified value for the penalty parameter.
main	A character with which to specify the main title of the output graph.
nOutput	A logical indicating if numeric output should be returned.
verbose	A logical indicating if information on progress should be printed on screen.
suppressChecks	A logical indicating if the input checks should be suppressed.

Details

Under certain target choices the proposed ridge estimators (see ridgeP) are rotation equivariant, i.e., the eigenvectors of $\mathbf{S}$ are left intact. Such rotation equivariant situations help to understand the effect of the ridge penalty on the precision estimate: The effect can be understood in terms of shrinkage of the eigenvalues of the unpenalized precision estimate $\mathbf{S}^{-1}$. Maximum shrinkage implies that all eigenvalues are forced to be equal (in the rotation equivariant situation). The spectral condition number w.r.t. inversion (ratio of maximum to minimum eigenvalue) of the regularized precision matrix may function as a heuristic in determining the (minimal) value of the penalty parameter. A matrix with a high condition number is near-singular (the relative distance to the set of singular matrices equals the reciprocal of the condition number; Demmel, 1987) and its inversion is numerically unstable. Such a matrix is said to be ill-conditioned. Numerically, ill-conditioning will mean that small changes in the penalty parameter lead to dramatic changes in the condition number. From a numerical point of view one can thus track the domain of the penalty parameter for which the regularized precision matrix is ill-conditioned. When plotting the condition number against the (domain of the) penalty parameter, there is a point of relative stabilization (when working in the $p > n$ situation) that can be characterized by a leveling-off of the acceleration along the curve when plotting the condition number against the (chosen) domain of the penalty parameter. This suggest the following fast heuristic for determining the (minimal) value of the penalty parameter: The value of the penalty parameter for which the spectral condition number starts to stabilize may be termed an acceptable (minimal) value.

The function outputs a graph of the (spectral) matrix condition number over the domain [lambdaMin, lambdaMax]. When norm = "2" the spectral condition number is calculated. It is determined by exact calculation using the spectral decomposition. For most purposes this exact calculation is fast enough, especially when considering rotation equivariant situations (see ridgeP). For such situations the amenities for fast eigenvalue calculation as provided by RSpectra are used internally. When exact computation of the spectral condition number is deemed too costly one may approximate the computationally friendly L1-condition number. This approximation is accessed through the rcond function (Anderson et al. 1999).

When Iaids = TRUE the basic condition number plot is amended/enhanced with two additional plots (over the same domain of the penalty parameter as the basic plot): The approximate loss

in digits of accuracy (for the operation of inversion) and an approximation to the second-order derivative of the curvature in the basic plot. These interpretational aids can enhance interpretation of the basic condition number plot and may support choosing a value for the penalty parameter (see Peeters, van de Wiel, & van Wieringen, 2016). When `vertical = TRUE` a vertical line is added at the constant `value`. This option can be used to assess if the optimal penalty obtained by, e.g., the routines `optPenalty.LOOCV` or `optPenalty.aLOOCV`, has led to a precision estimate that is well-conditioned.

Value

The function returns a graph. If `nOutput = TRUE` the function also returns an object of class `list`:

`lambdas` A `numeric` vector representing all values of the penalty parameter for which the condition number was calculated. The values of the penalty parameter are log-equidistant.

`conditionNumbers` A `numeric` vector containing the condition number for each value of the penalty parameter given in `lambdas`.

Note

The condition number of a (regularized) covariance matrix is equivalent to the condition number of its corresponding inverse, the (regularized) precision matrix. Please note that the `target` argument (for Type I ridge estimators) is assumed to be specified in precision terms. In case one is interested in the condition number of a Type I estimator under a covariance target, say $\mathbf{\Gamma}$, then just specify `target = solve(`$\mathbf{\Gamma}$`)`.

Author(s)

Carel F.W. Peeters <carel.peeters@wur.nl>

References

Anderson, E, Bai, Z., ..., Sorenson, D. (1999). LAPACK Users' Guide (3rd ed.). Philadelphia, PA, USA: Society for Industrial and Applied Mathematics.

Demmel, J.W. (1987). On condition numbers and the distance to the nearest ill-posed problem. Numerische Mathematik, 51: 251–289.

Peeters, C.F.W., van de Wiel, M.A., & van Wieringen, W.N. (2020). The spectral condition number plot for regularization parameter evaluation. Computational Statistics, 35: 629–646.

See Also

`covML`, `ridgeP`, `optPenalty.LOOCV`, `optPenalty.aLOOCV`, `default.target`

Examples

```
## Obtain some (high-dimensional) data
p = 25
n = 10
set.seed(333)
```

```
X = matrix(rnorm(n*p), nrow = n, ncol = p)
colnames(X)[1:25] = letters[1:25]
Cx <- covML(X)

## Assess spectral condition number across grid of penalty parameter
CNplot(Cx, lambdaMin = .0001, lambdaMax = 50, step = 1000)

## Include interpretational aids
CNplot(Cx, lambdaMin = .0001, lambdaMax = 50, step = 1000, Iaids = TRUE)
```

`Communities` *Search and visualize community-structures*

Description

Function that searches for and visualizes community-structures in graphs.

Usage

```
Communities(
  P,
  graph = TRUE,
  lay = "layout_with_fr",
  coords = NULL,
  Vsize = 15,
  Vcex = 1,
  Vcolor = "orangered",
  VBcolor = "darkred",
  VLcolor = "black",
  main = ""
)
```

Arguments

`P`	Sparsified precision `matrix`
`graph`	A `logical` indicating if the results should be visualized.
`lay`	A `character` mimicking a call to `igraph` layout functions. Determines the placement of vertices.
`coords`	A `matrix` containing coordinates. Alternative to the lay-argument for determining the placement of vertices.
`Vsize`	A `numeric` determining the vertex size.
`Vcex`	A `numeric` determining the size of the vertex labels.
`Vcolor`	A `character` (scalar or vector) determining the vertex color.
`VBcolor`	A `character` determining the color of the vertex border.
`VLcolor`	A `character` determining the color of the vertex labels.
`main`	A `character` giving the main figure title.

Details

Communities in a network are groups of vertices (modules) that are densely connected within. Community search is performed by the Girvan-Newman algorithm (Newman and Girvan, 2004).

When `graph = TRUE` the community structure in the graph is visualized. The default layout is according to the Fruchterman-Reingold algorithm (1991). Most layout functions supported by `igraph` are supported (the function is partly a wrapper around certain `igraph` functions). The igraph layouts can be invoked by a `character` that mimicks a call to a `igraph` layout functions in the `lay` argument. When using `lay = NULL` one can specify the placement of vertices with the `coords` argument. The row dimension of this matrix should equal the number of vertices. The column dimension then should equal 2 (for 2D layouts) or 3 (for 3D layouts). The `coords` argument can also be viewed as a convenience argument as it enables one, e.g., to layout a graph according to the coordinates of a previous call to `Ugraph`. If both the the lay and the coords arguments are not `NULL`, the lay argument takes precedence. Communities are indicated by color markings.

Value

An object of class list:

`membership` `numeric` vector indicating, for each vertex, community membership.

`modularityscore` `numeric` scalar indicating the modularity value of the community structure.

When `graph = TRUE` the function also returns a graph.

Author(s)

Carel F.W. Peeters <carel.peeters@wur.nl>

References

Csardi, G. and Nepusz, T. (2006). The igraph software package for complex network research. InterJournal, Complex Systems 1695. http://igraph.sf.net

Fruchterman, T.M.J., and Reingold, E.M. (1991). Graph Drawing by Force-Directed Placement. Software: Practice & Experience, 21: 1129-1164.

Newman, M. and Girvan, M. (2004). Finding and evaluating community structure in networks. Physical Review E, 69: 026113.

See Also

`Ugraph`

Examples

```
## Obtain some (high-dimensional) data
p = 25
n = 10
set.seed(333)
X = matrix(rnorm(n*p), nrow = n, ncol = p)
colnames(X)[1:25] = letters[1:25]
```

```
## Obtain regularized precision under optimal penalty
OPT <- optPenalty.LOOCV(X, lambdaMin = .5, lambdaMax = 30, step = 100)

## Determine support regularized standardized precision under optimal penalty
PC0 <- sparsify(symm(OPT$optPrec), threshold = "localFDR")$sparseParCor

## Search and visualize communities
Commy <- Communities(PC0)
```

`conditionNumberPlot` *Visualize the spectral condition number against the regularization parameter*

Description

This function is now deprecated. Please use `CNplot` instead.

Usage

```
conditionNumberPlot(
  S,
  lambdaMin,
  lambdaMax,
  step,
  type = "Alt",
  target = default.target(S),
  norm = "2",
  digitLoss = FALSE,
  rlDist = FALSE,
  vertical = FALSE,
  value,
  main = TRUE,
  nOutput = FALSE,
  verbose = TRUE
)
```

Arguments

`S`	Sample covariance `matrix`.
`lambdaMin`	A `numeric` giving the minimum value for the penalty parameter.
`lambdaMax`	A `numeric` giving the maximum value for the penalty parameter.
`step`	An `integer` determining the number of steps in moving through the grid [`lambdaMin`, `lambdaMax`].
`type`	A `character` indicating the type of ridge estimator to be used. Must be one of: "Alt", "ArchI", "ArchII".
`target`	A target `matrix` (in precision terms) for Type I ridge estimators.

norm	A character indicating the norm under which the condition number is to be calculated/estimated. Must be one of: "1", "2".
digitLoss	A logical indicating if the approximate loss in digits of accuracy should also be visualized in the output graph.
rlDist	A logical indicating if the relative distance to the set of singular matrices should also be visualized in the output graph.
vertical	A logical indicating if output graph should come with a vertical line at a pre-specified value for the penalty parameter.
value	A numeric indicating a pre-specified value for the penalty parameter.
main	A logical indicating if output graph should contain type of estimator as main title.
nOutput	A logical indicating if numeric output should be returned.
verbose	A logical indicating if information on progress should be printed on screen.

Details

See CNplot.

Value

The function returns a graph. If nOutput = TRUE the function also returns an object of class list:

lambdas	A numeric vector representing all values of the penalty parameter for which the condition number was calculated.
conditionNumbers	A numeric vector containing the condition number for each value of the penalty parameter given in lambdas.

Author(s)

Carel F.W. Peeters <carel.peeters@wur.nl>

See Also

CNplot

Examples

```
## Obtain some (high-dimensional) data
p = 25
n = 10
set.seed(333)
X = matrix(rnorm(n*p), nrow = n, ncol = p)
colnames(X)[1:25] = letters[1:25]
Cx <- covML(X)

## Assess spectral condition number across grid of penalty parameter
conditionNumberPlot(Cx, lambdaMin = .0001, lambdaMax = 50, step = 1000)
```

covML *Maximum likelihood estimation of the covariance matrix*

Description

Function that gives the maximum likelihood estimate of the covariance matrix.

Usage

```
covML(Y, cor = FALSE)
```

Arguments

Y	Data matrix. Variables assumed to be represented by columns.
cor	A logical indicating if the correlation matrix should be returned

Details

The function gives the maximum likelihood (ML) estimate of the covariance matrix. The input matrix Y assumes that the variables are represented by the columns. Note that when the input data is standardized, the ML covariance matrix of the scaled data is computed. If a correlation matrix is desired, use cor = TRUE.

Value

Function returns the maximum likelihood estimate of the covariance matrix. In case cor = TRUE, the correlation matrix is returned.

Author(s)

Carel F.W. Peeters <carel.peeters@wur.nl>, Wessel N. van Wieringen

See Also

ridgeP

Examples

```
## Obtain some (high-dimensional) data
p = 25
n = 10
set.seed(333)
X = matrix(rnorm(n*p), nrow = n, ncol = p)
colnames(X)[1:25] = letters[1:25]

## Obtain ML estimate covariance matrix
Cx <- covML(X)

## Obtain correlation matrix
```

```
Cx <- covML(X, cor = TRUE)
```

---

covMLknown *Maximum likelihood estimation of the covariance matrix with assumptions on its structure*

Description

Function that performs maximum likelihood estimation of the covariance matrix, with various types of assumptions on its structure.

Usage

```
covMLknown(
  Y,
  covMat = NULL,
  corMat = NULL,
  corType = "none",
  varType = "none",
  nInit = 100
)
```

Arguments

Y	Data matrix. Variables assumed to be represented by columns.
covMat	A positive-definite covariance matrix. When specified, the to-be-estimated covariance matrix is assumed to be proportional to the specified covariance matrix. Hence, only a constant needs to estimated.
corMat	A positive-definite correlation matrix. When specified, the to-be-estimated covariance matrix is assumed to have this correlation structure. Hence, only the marginal variances need to be estimated.
corType	A character, either "none" (no structure on the correlation among variate assumed) or "equi" (variates are equi-correlated).
varType	A character, either "none" (no structure on the marginal variances of the variates assumed) or "common" (variates have equal marginal variances).
nInit	An integer specifying the maximum number of iterations for likelihood maximization when corType="equi" .

Details

The function gives the maximum likelihood estimate of the covariance matrix. The input matrix Y assumes that the variables are represented by the columns.

When simultaneously covMat=NULL, corMat=NULL, corType="none" and varType="none" the covML-function is invoked and the regular maximum likelihood estimate of the covariance matrix is returned.

Value

The maximum likelihood estimate of the covariance `matrix` under the specified assumptions on its structure.

Author(s)

Wessel N. van Wieringen, Carel F.W. Peeters <carel.peeters@wur.nl>

See Also

covML

Examples

```
## Obtain some data
p = 10
n = 100
set.seed(333)
X = matrix(rnorm(n*p), nrow = n, ncol = p)
colnames(X)[1:10] = letters[1:10]

## Obtain maximum likelihood estimate covariance matrix
Cx <- covMLknown(X, corType="equi", varType="common")
```

`createS` *Simulate sample covariances or datasets*

Description

Simulate data from a p-dimensional (zero-mean) gaussian graphical model (GGM) with a specified (or random) topology and return the sample covariance matrix or matrices. Can also return the original simulated data or underlying precision matrix.

Usage

```
createS(
  n,
  p,
  topology = "identity",
  dataset = FALSE,
  precision = FALSE,
  nonzero = 0.25,
  m = 1L,
  banded.n = 2L,
  invwishart = FALSE,
  nu = p + 1,
  Plist
)
```

Arguments

n	A numeric vector giving number of samples. If the length is larger than 1, the covariance matrices are returned as a list.
p	A numeric of length 1 giving the dimension of the samples/covariance.
topology	character. The topology to use for the simulations. See the details.
dataset	A logical value specifying whether the sample covariance or the simulated data itself should be returned.
precision	A logical value. If TRUE the constructed precision matrix is returned.
nonzero	A numeric of length 1 giving the value of the nonzero entries used in some topologies.
m	A integer giving the number of blocks (i.e. conditionally independent components) to create. If m is greater than 1, then the given topology is used on m blocks of approximately equal size.
banded.n	A integer of length one giving the number of bands. Only used if topology is one of "banded", "small-world", or "Watts-Strogatz".
invwishart	logical. If TRUE the constructed precision matrix is used as the scale matrix of an inverse Wishart distribution and class covariance matrices are drawn from this distribution.
nu	numeric greater than p + 1 giving the degrees of freedom in the inverse Wishart distribution. A large nu implies high class homogeneity. A small nu near p + 1 implies high class heterogeneity.
Plist	An optional list of numeric matrices giving the precision matrices to simulate from. Useful when random matrices have already been generated by setting precision = TRUE.

Details

The data is simulated from a zero-mean p-dimensional multivariate gaussian distribution with some precision matrix determined by the argument topology which defines the GGM. If precision is TRUE the population precision matrix is returned. This is useful to see what the actual would-be-used precision matrices are. The available values of topology are described below. Unless otherwise stated the diagonal entries are always one. If m is 2 or greater block diagonal precision matrices are constructed and used.

- "identity": uses the identity matrix (diag(p)) as precision matrix. Corresponds to no conditional dependencies.
- "star": simulate from a star topology. Within each block the first node is selected as the "hub". The off-diagonal entries $(1, j)$ and $(j, 1)$ values taper off with the value $1/(j + 1)$.
- "clique": simulate from clique topology where each block is a complete graph with off-diagonal elements equal to nonzero.
- "complete": alias for (and identical to) "clique".
- "chain": simulate from a chain topology where the precision matrix is a tridiagonal matrix with off-diagonal elements (in each block) given by argument nonzero.

- "banded": precision elements (i, j) are given by $1/(|i-j|+1)$ if $|i-j|$ is less than or equal to banded.n and zero otherwise.
- "scale-free": The non-zero pattern of each block is generated by a Barabassi random graph. Non-zero off-diagonal values are given by nonzero. Gives are very "hubby" network.
- "Barabassi": alias for "scale-free".
- "small-world": The non-zero pattern of each block is generated by a 1-dimensional Watts-Strogatz random graph with banded.n starting neighbors and 5% probability of rewiring. Non-zero off-diagonal values are given by nonzero. Gives are very "bandy" network.
- "Watts-Strogatz": alias for "small-world"
- "random-graph": The non-zero pattern of each block is generated by a Erdos-Renyi random graph where each edge is present with probability $1/p$. Non-zero off-diagonal values are given by nonzero.
- "Erdos-Renyi": alias for "random-graph"

When n has length greater than 1, the datasets are generated i.i.d. given the topology and number of blocks.

Arguments invwishart and nu allows for introducing class homogeneity. Large values of nu imply high class homogeneity. nu must be greater than p + 1. More precisely, if invwishart == TRUE then the constructed precision matrix is used as the scale parameter in an inverse Wishart distribution with nu degrees of freedom. Each class covariance is distributed according to this inverse Wishart and independent.

Value

The returned type is dependent on n and covariance. The function generally returns a list of numeric matrices with the same length as n. If covariance is FALSE the simulated datasets with size n[i] by p are given in the i entry of the output. If covariance is TRUE the p by p sample covariances of the datasets are given. When n has length 1 the list structure is dropped and the matrix is returned.

Author(s)

Anders E. Bilgrau, Carel F.W. Peeters <carel.peeters@wur.nl>, Wessel N. van Wieringen

Examples

```
## Generate some simple sample covariance matrices
createS(n = 10, p = 3)
createS(n = c(3, 4, 5), p = 3)
createS(n = c(32, 55), p = 7)

## Generate some datasets and not sample covariance matrices
createS(c(3, 4), p = 6, dataset = TRUE)

## Generate sample covariance matrices from other topologies:
A <- createS(2000, p = 4, topology = "star")
round(solve(A), 3)
B <- createS(2000, p = 4, topology = "banded", banded.n = 2)
```

```
round(solve(B), 3)
C <- createS(2000, p = 4, topology = "clique")  # The complete graph (as m = 1)
round(solve(C), 3)
D <- createS(2000, p = 4, topology = "chain")
round(solve(D), 3)

## Generate smaple covariance matrices from block topologies:
C3 <- createS(2000, p = 10, topology = "clique", m = 3)
round(solve(C3), 1)
C5 <- createS(2000, p = 10, topology = "clique", m = 5)
round(solve(C5), 1)

## Can also return the precision matrix to see what happens
## m = 2 blocks, each "banded" with 4 off-diagonal bands
round(createS(1, 12, "banded", m = 2, banded.n = 4, precision = TRUE), 2)

## Simulation using graph-games
round(createS(1, 10, "small-world", precision = TRUE), 2)
round(createS(1, 5, "scale-free", precision = TRUE), 2)
round(createS(1, 5, "random-graph", precision = TRUE), 2)

## Simulation using inverse Wishart distributed class covariance
## Low class homogeneity
createS(n = c(10,10), p = 5, "banded", invwishart = TRUE, nu = 10)
## Extremely high class homogeneity
createS(n = c(10,10), p = 5, "banded", invwishart = TRUE, nu = 1e10)

# The precision argument can again be used to see the actual realised class
# precision matrices used when invwishart = TRUE.

# The Plist argument is used to reuse old precision matrices or
# user-generated ones
P <- createS(n = 1, p = 5, "banded", precision = TRUE)
lapply(createS(n = c(1e5, 1e5), p = 5, Plist = list(P, P+1)), solve)
```

`default.penalty` *Construct commonly used penalty matrices*

Description

Function that constructs default or commonly use penalty matrices according to a (factorial) study design. The constructed penalty matrix can be used directly in `optPenalty.fused.auto` or serve as basis for modification.

Usage

```
default.penalty(
  G,
  df,
```

```
  type = c("Complete", "CartesianEqual", "CartesianUnequal", "TensorProd")
)
```

Arguments

G	A numeric giving the number of classes. Can also be a list of length G such as the usual argument Slist from other **rags2ridges** functions. Can be omitted if df is given.
df	A data.frame with G rows and factors in the columns. Note, the columns has to be of type factor. Can be omitted when G is given and type == "Complete". The factors can be ordered.
type	A character giving the type of fused penalty graph to construct. Should be 'Complete' (default), 'CartesianEqual', or 'CartesianUnequal' or 'TensorProd' or an unique abbreviation hereof. See details.

Details

The type gives a number of common choices for the penalty matrix:

- 'Complete' is the complete penalty graph with equal penalties.
- 'CartesianEqual' corresponds to a penalizing along each "direction" of factors with a common penalty. The choice is named Cartesian as it is the Cartesian graph product of the complete penalty graphs for the individual factors.
- 'CartesianUnequal' corresponds to a penalizing each direction of factors with individual penalties.
- 'TensorProd' correspond to penalizing the "diagonals" only. It is equivalent to the graph tensor products of the complete graphs for each individual factor.

Value

Returns a G by G character matrix which specify the class of penalty graphs to be used. The output is suitable as input for the penalty matrix used in optPenalty.fused.auto.

Author(s)

Anders E. Bilgrau, Carel F.W. Peeters <carel.peeters@wur.nl>, Wessel N. van Wieringen

References

Bilgrau, A.E., Peeters, C.F.W., Eriksen, P.S., Boegsted, M., and van Wieringen, W.N. (2020). Targeted Fused Ridge Estimation of Inverse Covariance Matrices from Multiple High-Dimensional Data Classes. Journal of Machine Learning Research, 21(26): 1-52.

See Also

ridgeP.fused, optPenalty.fused, default.target

Examples

```
# Handling one-way designs
default.penalty(2)
default.penalty(4)
Slist <- vector("list", 6)
default.penalty(Slist)   # The function uses only the length of the list
df0 <- expand.grid(Factor = c("lvl1", "lvl2"))
default.penalty(df0)

# A more elaborate example
df1 <- expand.grid(DS = c("DS1", "DS2", "DS3"), ER = c("ER+", "ER-"))

# Usage (various interface demonstrations)
default.penalty(6, df1, type = "Complete")
default.penalty(6, type = "CartesianEqual")  # GIVES WARNING
default.penalty(6, df1, type = "CartesianEqual")
default.penalty(Slist, df1, type = "CartesianEqual")
default.penalty(6, df1, type = "CartesianUnequal")
default.penalty(df1)

# A 2 by 2 by 2 design
df2 <- expand.grid(A = c("A1", "A2"), B = c("B1", "B2"), C = c("C1", "C3"))
default.penalty(df2)
default.penalty(df2, type = "CartesianEqual")
default.penalty(df2, type = "CartesianUnequal")
```

`default.target` *Generate a (data-driven) default target for usage in ridge-type shrinkage estimation*

Description

Function that generates a (data-driven) default target for usage in (type I) ridge shrinkage estimation of the precision matrix (see `ridgeP`). The target that is generated is to be understood in precision terms. Most options for target generation result in a target that implies a situation of rotation equivariant estimation (see `ridgeP`).

Usage

```
default.target(S, type = "DAIE", fraction = 1e-04, const)
```

Arguments

`S`	Sample covariance `matrix`.
`type`	A `character` determining the type of default target. Must be one of: "DAIE", "DIAES", "DUPV", "DAPV", "DCPV", "DEPV", "Null".

fraction	A numeric indicating the fraction of the largest eigenvalue below which an eigenvalue is considered zero.
const	A numeric constant representing the partial variance.

Details

The function can generate the following default target matrices:

- DAIE: Diagonal matrix with average of inverse nonzero eigenvalues of S as entries;
- DIAES: Diagonal matrix with inverse of average of eigenvalues of S as entries;
- DUPV: Diagonal matrix with unit partial variance as entries (identity matrix);
- DAPV: Diagonal matrix with average of inverse variances of S as entries;
- DCPV: Diagonal matrix with constant partial variance as entries. Allows one to use other constant than DAIE, DIAES, DUPV, DAPV, and in a sense Null;
- DEPV: Diagonal matrix with the inverse variances of S as entries;
- Null: Null matrix.

The targets DUPV, DCPV, and Null are not data-driven in the sense that the input matrix S only provides information on the size of the desired target. The targets DAIE, DIAES, DAPV, and DEPV are data-driven in the sense that the input matrix S provides the information for the diagonal entries. The argument fraction is only used when type = "DAIE". The argument const is only used when type = "DCPV". All types except DEPV and Null lead to rotation equivariant alternative and archetypal Type I ridge estimators. The target Null also leads to a rotation equivariant alternative Type II ridge estimator (see ridgeP). Note that the DIAES, DAPV, and DEPV targets amount to the identity matrix when the sample covariance matrix S is standardized to be the correlation matrix. The same goes, naturally, for the DCPV target when const is specified to be 1.

Value

Function returns a target matrix.

Author(s)

Carel F.W. Peeters <carel.peeters@wur.nl>, Wessel N. van Wieringen

References

van Wieringen, W.N. & Peeters, C.F.W. (2016). Ridge Estimation of Inverse Covariance Matrices from High-Dimensional Data, Computational Statistics & Data Analysis, vol. 103: 284-303. Also available as arXiv:1403.0904v3 [stat.ME].

See Also

ridgeP, covML

Examples

```
## Obtain some (high-dimensional) data
p = 25
n = 10
set.seed(333)
X = matrix(rnorm(n*p), nrow = n, ncol = p)
colnames(X)[1:25] = letters[1:25]
Cx <- covML(X)

## Obtain default diagonal target matrix
default.target(Cx)
```

`default.target.fused` *Generate data-driven targets for fused ridge estimation*

Description

Generates a list of (data-driven) targets to use in fused ridge estimation. Simply a wrapper for `default.target`.

Usage

```
default.target.fused(Slist, ns, type = "DAIE", by, ...)
```

Arguments

`Slist`	A `list` of length K of `numeric` covariance matrices of the same size for K classes.
`ns`	A `numeric` vector of sample sizes corresponding to the entries of `Slist`.
`type`	A `character` giving the choice of target to construct. See `default.target` for the available options. Default is `"DAIE"`.
`by`	A `character` vector with the same length as `Slist` specifying which groups should share target. For each unique entry of `by` a target is constructed. If omitted, the default is to assign a unique target to each class. If not given as a `character` coercion into one is attempted.
`...`	Arguments passed to `default.target`.

Value

A `list` of K covariance target matrices of the same size.

Author(s)

Anders E. Bilgrau, Carel F.W. Peeters <carel.peeters@wur.nl>, Wessel N. van Wieringen

See Also

default.target

Examples

```
# Make some toy data
ns <- c(3, 4)  # Two classes with sample size 3 and 4
Slist <- createS(ns, p = 3)  # Generate two 3-dimensional covariance matrices
Slist

# Different choices:
default.target.fused(Slist, ns)
default.target.fused(Slist, ns, by = seq_along(Slist)) # The same as before
default.target.fused(Slist, ns, type = "Null")
default.target.fused(Slist, ns, type = "DAPV")
default.target.fused(Slist, ns, type = "DAPV", by = rep(1, length(Slist)))


# Make some (more) toy data
ns <- c(3, 4, 6, 7)  # Two classes with sample size 3 and 4
Slist <- createS(ns, p = 2)  # Generate four 2-dimensional covariance matrices

# Use the same target in class 1 and 2, but another in class 3 and 4:
default.target.fused(Slist, ns, by = c("A", "A", "B", "B"))
```

DiffGraph *Visualize the differential graph*

Description

Function visualizing the differential graph, i.e., the network of edges that are unique for 2 class-specific graphs over the same vertices

Usage

```
DiffGraph(
  P1,
  P2,
  lay = "layout_with_fr",
  coords = NULL,
  Vsize = 15,
  Vcex = 1,
  Vcolor = "orangered",
  VBcolor = "darkred",
  VLcolor = "black",
  P1color = "red",
  P2color = "green",
  main = ""
)
```

Arguments

P1	Sparsified precision matrix for class 1.
P2	Sparsified precision matrix for class 2.
lay	A character mimicking a call to igraph layout functions. Determines the placement of vertices.
coords	A matrix containing coordinates. Alternative to the lay-argument for determining the placement of vertices.
Vsize	A numeric determining the vertex size.
Vcex	A numeric determining the size of the vertex labels.
Vcolor	A character (scalar or vector) determining the vertex color.
VBcolor	A character determining the color of the vertex border.
VLcolor	A character determining the color of the vertex labels.
P1color	A character determining the color of edges unique to P1.
P2color	A character determining the color of edges unique to P2.
main	A character giving the main figure title.

Details

Say you have 2 class-specific precision matrices that are estimated over the same variables/features. This function visualizes in a single graph the edges that are unique to the respective classes. Hence, it gives the differential graph. Edges unique to P1 are colored according to P1color. Edges unique to P2 are colored according to P2color. Dashed lines indicate negative precision elements while solid lines indicate positive precision elements.

The default layout is according to the Fruchterman-Reingold algorithm (1991). Most layout functions supported by igraph are supported (the function is partly a wrapper around certain igraph functions). The igraph layouts can be invoked by a character that mimicks a call to a igraph layout functions in the lay argument. When using lay = NULL one can specify the placement of vertices with the coords argument. The row dimension of this matrix should equal the number of vertices. The column dimension then should equal 2 (for 2D layouts) or 3 (for 3D layouts). The coords argument can also be viewed as a convenience argument as it enables one, e.g., to layout a graph according to the coordinates of a previous call to Ugraph. If both the the lay and the coords arguments are not NULL, the lay argument takes precedence.

Value

The function returns a graph.

Author(s)

Carel F.W. Peeters <carel.peeters@wur.nl>

References

Csardi, G. and Nepusz, T. (2006). The igraph software package for complex network research. InterJournal, Complex Systems 1695. http://igraph.sf.net

Fruchterman, T.M.J., and Reingold, E.M. (1991). Graph Drawing by Force-Directed Placement. Software: Practice & Experience, 21: 1129-1164.

See Also

Ugraph

Examples

```
## Obtain some (high-dimensional) data, class 1
p = 25
n = 10
set.seed(333)
X = matrix(rnorm(n*p), nrow = n, ncol = p)
colnames(X)[1:25] = letters[1:25]

## Obtain some (high-dimensional) data, class 2
set.seed(123456)
X2 = matrix(rnorm(n*p), nrow = n, ncol = p)
colnames(X2)[1:25] = letters[1:25]

## Obtain regularized precision under optimal penalty, classes 1 and 2
OPT  <- optPenalty.LOOCV(X, lambdaMin = .5, lambdaMax = 30, step = 100)
OPT2 <- optPenalty.LOOCV(X2, lambdaMin = .5, lambdaMax = 30, step = 100)

## Determine support regularized standardized precision under optimal penalty
PC0  <- sparsify(symm(OPT$optPrec), threshold = "localFDR")$sparseParCor
PC02 <- sparsify(symm(OPT2$optPrec), threshold = "localFDR")$sparseParCor

## Visualize differential graph
DiffGraph(PC0, PC02)
```

---

edgeHeat *Visualize (precision) matrix as a heatmap*

Description

Function that visualizes a (precision) matrix as a heatmap. May be used to assess visually the elements of a single (possibly sparsified precision) matrix. May also be used in assessing the performance of edge selection techniques.

Usage

```
edgeHeat(
  M,
  lowColor = "blue",
  highColor = "red",
  textsize = 10,
  diag = TRUE,
  legend = TRUE,
  main = ""
)
```

Arguments

M	(Possibly sparsified precision) matrix.
lowColor	A character that determines the color scale in the negative range.
highColor	A character that determines the color scale in the positive range.
textsize	A numeric scaling the text size of row and column labels.
diag	A logical determining if the diagonal elements of the matrix should be included in the color scaling. This argument is only used when M is a square matrix.
legend	A logical indicating whether a color legend should be included.
main	A character giving the main figure title.

Details

This function utilizes ggplot2 (Wickham, 2009) to visualize a matrix as a heatmap: a false color plot in which the individual matrix entries are represented by colors. lowColor determines the color scale for matrix entries in the negative range. highColor determines the color scale for matrix entries in the positive range. For the colors supported by the arguments lowColor and highColor, see https://www.nceas.ucsb.edu/sites/default/files/2020-04/colorPaletteCheatsheet.pdf. White entries in the plot represent the midscale value of 0. One can opt to set the diagonal entries to the midscale color of white when one is interested in (heatmapping) the off-diagonal elements only. To achieve this, set diag = FALSE. Naturally, the diag argument is only used when the input matrix M is a square matrix.

The intended use of the function is to visualize a, possibly sparsified, precision matrix as a heatmap. The function may also be used, in a graphical modeling setting, to assess the performance of edge selection techniques. However, the function is quite general, in the sense that it can represent any matrix as a heatmap.

Author(s)

Carel F.W. Peeters <carel.peeters@wur.nl>, Wessel N. van Wieringen

References

Wickham, H. (2009). ggplot2: elegant graphics for data analysis. New York: Springer.

See Also

covML, ridgeP, sparsify

Examples

```
## Obtain some (high-dimensional) data
p = 25
n = 10
set.seed(333)
X = matrix(rnorm(n*p), nrow = n, ncol = p)
colnames(X)[1:25] = letters[1:25]
```

```
Cx <- covML(X)

## Obtain regularized precision matrix
P <- ridgeP(Cx, lambda = 10, type = "Alt")

## Obtain sparsified partial correlation matrix
PC0 <- sparsify(P, threshold = "localFDR", FDRcut = .8)$sparseParCor

## Visualize sparsified partial correlation matrix as heatmap
edgeHeat(PC0)
```

`evaluateS` *Evaluate numerical properties square matrix*

Description

Function that evaluates various numerical properties of a square input matrix. The intended use is to evaluate the various numerical properties of what is assumed to be a covariance matrix. Another use is to evaluate the various numerical properties of a (regularized) precision matrix.

Usage

```
evaluateS(S, verbose = TRUE)
```

Arguments

`S`	Covariance or (regularized) precision `matrix`.
`verbose`	A `logical` indicating if output should be printed on screen.

Details

The function evaluates various numerical properties of a covariance or precision input matrix. The function assesses if the input matrix is symmetric, if all its eigenvalues are real, if all its eigenvalues are strictly positive, and if it is a diagonally dominant matrix. In addition, the function calculates the trace, the determinant, and the spectral condition number of the input matrix. See, e.g., Harville (1997) for more details on the mentioned (numerical) matrix properties.

Value

`symm`	A `logical` indicating if the matrix is symmetric.
`realEigen`	A `logical` indicating if the eigenvalues are real.
`posEigen`	A `logical` indicating if the eigenvalues are strictly positive.
`dd`	A `logical` indicating if the matrix is diagonally dominant.
`trace`	A `numerical` giving the value of the trace.
`det`	A `numerical` giving the value of the determinant.
`condNumber`	A `numerical` giving the value of the spectral condition number.

Author(s)

Wessel N. van Wieringen, Carel F.W. Peeters <carel.peeters@wur.nl>

References

Harville, D.A.(1997). Matrix algebra from a statistician's perspective. New York: Springer-Verlag.

See Also

covML, ridgeP

Examples

```
## Obtain some (high-dimensional) data
p = 25
n = 10
set.seed(333)
X = matrix(rnorm(n*p), nrow = n, ncol = p)
colnames(X)[1:25] = letters[1:25]
Cx <- covML(X)

## Evaluate numerical properties covariance matrix
## Obtain, e.g., value trace
Seval <- evaluateS(Cx); Seval
Seval$trace

## Evaluate numerical properties precision matrix after regularization
P <- ridgeP(Cx, lambda = 10, type = 'Alt')
Peval <- evaluateS(P); Peval
```

---

evaluateSfit *Visual inspection of the fit of a regularized precision matrix*

Description

Function aiding the visual inspection of the fit of an estimated (possibly regularized) precision matrix vis-a-vis the sample covariance matrix.

Usage

```
evaluateSfit(
  Phat,
  S,
  diag = FALSE,
  fileType = "pdf",
  nameExt = "",
  dir = getwd()
)
```

Arguments

Phat	(Regularized) estimate of the precision matrix.
S	Sample covariance matrix
diag	A logical determining if the diagonal elements should be retained for plotting.
fileType	A character determining the output file type. Must be one of: "pdf", "eps".
nameExt	A character determining the extension of default output names generated by the function.
dir	A character specifying the directory in which the visual output is stored.

Details

The function outputs various visualizations to aid the visual inspection of an estimated and possibly regularized precision matrix vis-a-vis the sample covariance matrix. The inverse of the estimated precision matrix P is taken to represent the estimated covariance matrix. The function then outputs a QQ-plot and a heatmap of the observed covariances against the estimated ones. The heatmap has the estimated covariances as lower-triangular elements and the observed covariances as the upper-triangular elements. The function outputs analogous plots for the estimated and observed correlations. In case the observed covariance matrix S is non-singular also a QQ-plot an a heatmap are generated for the estimated and observed partial correlations.

The function generates files with extension fileType under default output names. These files are stored in the directory dir (default is the working directory). To avoid overwriting of files when working in a single directory one may employ the argument nameExt. By using nameExt the default output names are extended with a character of choice.

Author(s)

Wessel N. van Wieringen, Carel F.W. Peeters <carel.peeters@wur.nl>

See Also

ridgeP, covML

Examples

```
## Not run:
## Obtain some (high-dimensional) data
p = 25
n = 10
set.seed(333)
X = matrix(rnorm(n*p), nrow = n, ncol = p)
colnames(X)[1:25] = letters[1:25]
Cx <- covML(X)

## Obtain regularized precision matrix
P <- ridgeP(Cx, lambda = 10, type = 'Alt')

## Evaluate visually fit of regularized precision matrix vis-a-vis sample covariance
evaluateSfit(P, Cx, diag = FALSE, fileType = "pdf", nameExt = "test")
```

```
## End(Not run)
```

`fullMontyS` *Wrapper function*

Description

Function that forms a wrapper around certain `rags2ridges` functionalities. More specifically, it (automatically) invokes functionalities to get from high-dimensional data to a penalized precision estimate, to the corresponding conditional independence graph and topology summaries.

Usage

```
fullMontyS(
  Y,
  lambdaMin,
  lambdaMax,
  target = default.target(covML(Y)),
  dir = getwd(),
  fileTypeFig = "pdf",
  FDRcut = 0.9,
  nOutput = TRUE,
  verbose = TRUE
)
```

Arguments

`Y`	Data `matrix`. Variables assumed to be represented by columns.
`lambdaMin`	A `numeric` giving the minimum value for the penalty parameter.
`lambdaMax`	A `numeric` giving the maximum value for the penalty parameter.
`target`	A target `matrix` (in precision terms) for Type I ridge estimators.
`dir`	A `character` specifying the directory in which the (visual) output is to be stored.
`fileTypeFig`	A `character` determining the file type of visual output. Must be one of: "pdf", "eps".
`FDRcut`	A `numeric` indicating the cut-off for partial correlation element selection based on local FDR thresholding.
`nOutput`	A `logical` indicating if numeric output should be returned.
`verbose`	A `logical` indicating if progress updates should be printed on screen.

Details

The wrapper always uses the alternative ridge precision estimator (see ridgeP) with target as the target matrix. The optimal value for the penalty parameter is determined by employing Brent's method to the calculation of a cross-validated negative log-likelihood score (see optPenalty.LOOCVauto). The support of the regularized precision matrix is determined by way of local FDR thresholding (see sparsify). The corresponding conditional independence graph is visualized using Ugraph with type = "fancy". This visualization as well as the calculation of network statistics (see GGMnetworkStats) is based on the standardization of the regularized and sparsified precision matrix to a partial correlation matrix.

Value

The function stores in the specified directory dir a condition number plot (either .pdf or .eps file), a visualization of the network (either .pdf or .eps file), and a file containing network statistics (.txt file). When nOutput = TRUE the function also returns an object of class list:

optLambda	A numeric giving the optimal value of the penalty parameter.
optPrec	A matrix representing the regularized precision matrix under the optimal value of the penalty parameter.
sparseParCor	A matrix representing the sparsified partial correlation matrix.
networkStats	A matrix giving the calculated network statistics.

Note

We consider this to be a preliminary version of an envisioned wrapper than will take better form with subsequent versions of rags2ridges.

Author(s)

Carel F.W. Peeters <carel.peeters@wur.nl>, Wessel N. van Wieringen

See Also

ridgeP, conditionNumberPlot, optPenalty.LOOCVauto, sparsify, Ugraph, GGMnetworkStats

Examples

```
## Not run:
## Obtain some (high-dimensional) data
p = 25
n = 10
set.seed(333)
X = matrix(rnorm(n*p), nrow = n, ncol = p)
colnames(X)[1:25] = letters[1:25]

## Employ the wrapper function
theWorks <- fullMontyS(X, lambdaMin = .5, lambdaMax = 30)
## End(Not run)
```

`fused.test` *Test the necessity of fusion*

Description

Function for testing the null hypothesis that all population precision matrices are equal and thus the necessity for the fusion penalty. Note, the test performed is conditional on the supplied penalties and targets.

Usage

```
fused.test(Ylist, Tlist, lambda, n.permutations = 100, verbose = FALSE, ...)
```

Arguments

`Ylist`	A `list` of length G of observations matrices for each class. Variables are assumed to correspond to the columns.
`Tlist`	A `list` of target matrices for each class. Should be same length as `Ylist`-
`lambda`	A non-negative, symmetric G by G `matrix` giving the ridge and fusion penalties.
`n.permutations`	The number of permutations to approximate the null distribution. Default is 100. Should be increased if sufficient computing power is available.
`verbose`	Print out extra progress information
`...`	Arguments passed to `ridgeP.fused`.

Details

The function computes the observed score statistic U_obs using the fused ridge estimator on the given data. Next, the score statistic is computed a number of times (given by `n.permutations`) under the null-hypothesis by effectively permuting the class labels of the data.

Value

Returns a `list` values containing the observed test statistic and the test statistic under the null distribution.

Author(s)

Anders Ellern Bilgrau, Carel F.W. Peeters <carel.peeters@wur.nl>, Wessel, N. van Wieringen

References

Bilgrau, A.E., Peeters, C.F.W., Eriksen, P.S., Boegsted, M., and van Wieringen, W.N. (2020). Targeted Fused Ridge Estimation of Inverse Covariance Matrices from Multiple High-Dimensional Data Classes. Journal of Machine Learning Research, 21(26): 1-52.

See Also

ridgeP.fused

Examples

```
ns <- c(10, 5, 23)
Ylist <- createS(ns, p = 15, topology = "banded", dataset = TRUE)

# Use the identity target matrix for each class
Tlist <- replicate(length(ns), diag(15), simplify = FALSE)

# Do the test
lm <- matrix(10, 3, 3)
diag(lm) <- 1
ft <- fused.test(Ylist, Tlist, lambda = lm,
                 n.permutations = 500)
print(ft)

# Summary spits out a bit more information
summary(ft)

# The returned object can alo be plotted via
hist(ft)
# or via the alias
plot(ft)

# Customization and parameters work a usual:
hist(ft, col = "steelblue", main = "Null distribution", add.extra = FALSE,
     xlab = "Score statistic", freq = FALSE)
```

GGMblockNullPenalty *Generate the distribution of the penalty parameter under the null hypothesis of block-independence*

Description

Function that serves as a precursor function to the block-independence test (see GGMblockTest). It generates an empirical distribution of the penalty parameter under the null hypothesis of block independence (in the regularized precision matrix).

Usage

```
GGMblockNullPenalty(
  Y,
  id,
  nPerm = 25,
  lambdaMin,
  lambdaMax,
```

```
  lambdaInit = (lambdaMin + lambdaMax)/2,
  target = default.target(covML(Y)),
  type = "Alt",
  ncpus = 1
)
```

Arguments

Y	Data matrix. Variables assumed to be represented by columns.
id	A numeric vector acting as an indicator variable for two blocks of the precision matrix. The blocks should be coded as 0 and 1.
nPerm	A numeric or integer determining the number of permutations.
lambdaMin	A numeric giving the minimum value for the penalty parameter.
lambdaMax	A numeric giving the maximum value for the penalty parameter.
lambdaInit	A numeric giving the initial value for the penalty parameter for starting optimization.
target	A target matrix (in precision terms) for Type I ridge estimators.
type	A character indicating the type of ridge estimator to be used. Must be one of: "Alt", "ArchI", "ArchII".
ncpus	A numeric or integer indicating the desired number of cpus to be used.

Details

This function can be viewed as a precursor to the function for the block-independence test (see GGMblockTest). The mentioned test evaluates the null hypothesis of block-independence against the alternative of block-dependence (presence of non-zero elements in the off-diagonal block) in the precision matrix using high-dimensional data. To accommodate the high-dimensionality the parameters of interest are estimated in a penalized manner (ridge-type penalization, see ridgeP). Penalization involves a degree of freedom (the penalty parameter) which needs to be fixed before testing. This function then generates an empirical distribution of this penalty parameter. Hereto the samples are permutated within block. The resulting permuted data sets represent the null hypothesis. To avoid the dependence on a single permutation, many permuted data sets are generated. For each permutation the optimal penalty parameter is determined by means of cross-validation (see optPenalty.LOOCVauto). The resulting optimal penalty parameters are returned. An estimate of the location (such as the median) is recommended for use in the block-independence test.

Value

A numeric vector, representing the distribution of the (LOOCV optimal) penalty parameter under the null hypothesis of block-independence.

Author(s)

Wessel N. van Wieringen, Carel F.W. Peeters <carel.peeters@wur.nl>

See Also

ridgeP, optPenalty.LOOCVauto, default.target, GGMblockTest

Examples

```
## Obtain some (high-dimensional) data
p = 15
n = 10
set.seed(333)
X = matrix(rnorm(n*p), nrow = n, ncol = p)
colnames(X)[1:15] = letters[1:15]
id <- c(rep(0, 10), rep(1, 5))

## Generate null distribution of the penalty parameter
lambda0dist <- GGMblockNullPenalty(X, id, 5, 0.001, 10)

## Location of null distribution
lambdaNull <- median(lambda0dist)
```

`GGMblockTest` *Test for block-indepedence*

Description

Function performing a test that evaluates the null hypothesis of block-independence against the alternative of block-dependence (presence of non-zero elements in the off-diagonal block) in the precision matrix using high-dimensional data. The mentioned test is a permutation-based test (see details).

Usage

```
GGMblockTest(
  Y,
  id,
  nPerm = 1000,
  lambda,
  target = default.target(covML(Y)),
  type = "Alt",
  lowCiThres = 0.1,
  ncpus = 1,
  verbose = TRUE
)
```

Arguments

`Y`	Data `matrix`. Variables assumed to be represented by columns.
`id`	A `numeric` vector acting as an indicator variable for two blocks of the precision matrix. The blocks should be coded as `0` and `1`.
`nPerm`	A `numeric` or `integer` determining the number of permutations.
`lambda`	A `numeric` representing the penalty parameter employed in the permutation test.

target	A target matrix (in precision terms) for Type I ridge estimators.
type	A character indicating the type of ridge estimator to be used. Must be one of: "Alt", "ArchI", "ArchII".
lowCiThres	A numeric taking a value between 0 and 1. Determines speed of efficient p-value calculation.
ncpus	A numeric or integer indicating the desired number of cpus to be used.
verbose	A logical indicating if information on progress and output should be printed on screen.

Details

The function performs a permutation test for the null hypothesis of block-independence against the alternative of block-dependence (presence of non-zero elements in the off-diagonal block) in the precision matrix using high-dimensional data. In the low-dimensional setting the common test statistic under multivariate normality (cf. Anderson, 2003) is:

$$\log(\|\hat{\mathbf{\Sigma}}_a\|) + \log(\|\hat{\mathbf{\Sigma}}_b\|) - \log(\|\hat{\mathbf{\Sigma}}\|),$$

where the $\hat{\mathbf{\Sigma}}_a$, $\hat{\mathbf{\Sigma}}_b$, $\hat{\mathbf{\Sigma}}$ are the estimates of the covariance matrix in the sub- and whole group(s), respectively.

To accommodate the high-dimensionality the parameters of interest are estimated in a penalized manner (ridge-type penalization, see ridgeP). Penalization involves a degree of freedom (the penalty parameter: lambda) which needs to be fixed before testing. To decide on the penalty parameter for testing we refer to the GGMblockNullPenalty function. With an informed choice on the penalty parameter at hand, the null hypothesis is evaluated by permutation. Hereto the samples are permutated within block. The resulting permuted data set represents the null hypothesis. Many permuted data sets are generated. For each permutation the test statistic is calculated. The observed test statistic is compared to the null distribution from the permutations.

The function implements an efficient permutation resampling algorithm (see van Wieringen et al., 2008, for details.): If the probability of a p-value being below lowCiThres is smaller than 0.001 (read: the test is unlikely to become significant), the permutation analysis is terminated and a p-value of unity (1) is reported.

When verbose = TRUE also graphical output is generated: A histogram of the null-distribution. Note that, when ncpus is larger than 1, functionalities from snowfall are imported.

Value

An object of class list:

statistic	A numeric representing the observed test statistic (i.e., likelihood ratio).
pvalue	A numeric giving the p-value for the block-independence test.
nulldist	A numeric vector representing the permutation null distribution for the test statistic.
nperm	A numeric indicating the number of permutations used for p-value calculation.
remark	A "character" that states whether the permutation algorithm was terminated prematurely or not.

Author(s)

Wessel N. van Wieringen, Carel F.W. Peeters <carel.peeters@wur.nl>

References

Anderson, T.W. (2003). An Introduction to Multivariate Statistical Analysis, 3rd Edition. John Wiley.

van Wieringen, W.N., van de Wiel, M.A., and van der Vaart, A.W. (2008). A Test for Partial Differential Expression. Journal of the American Statistical Association 103: 1039-1049.

See Also

ridgeP, optPenalty.LOOCVauto, default.target, GGMblockNullPenalty

Examples

```
## Obtain some (high-dimensional) data
p = 15
n = 10
set.seed(333)
X = matrix(rnorm(n*p), nrow = n, ncol = p)
colnames(X)[1:15] = letters[1:15]
id <- c(rep(0, 10), rep(1, 5))

## Generate null distribution of the penalty parameter
lambda0dist <- GGMblockNullPenalty(X, id, 5, 0.001, 10)

## Location of null distribution
lambdaNull <- median(lambda0dist)

## Perform test
testRes <- GGMblockTest(X, id, nPerm = 100, lambdaNull)
```

GGMmutualInfo *Mutual information between two sets of variates within a multivariate normal distribution*

Description

Function computing the mutual information between two exhaustive and mutually exclusive splits of a set of multivariate normal random variables.

Usage

```
GGMmutualInfo(S, split1)
```

Arguments

S	A positive-definite covariance matrix.
split1	A numeric, indicating the variates (by column number) forming the first split. The second split is automatically formed from its complement.

Value

A numeric, the mutual information between the variates forming split1 and those forming its complement.

Author(s)

Wessel N. van Wieringen, Carel F.W. Peeters <carel.peeters@wur.nl>

References

Cover, T.M., Thomas, J.A. (2012), Elements of information theory.

See Also

covML, ridgeP.

Examples

```
# create a covariance matrix
Sigma <- covML(matrix(rnorm(100), ncol=5))

# impulse response analysis
GGMmutualInfo(Sigma, c(1,2))
```

GGMnetworkStats *Gaussian graphical model network statistics*

Description

Function that calculates various network statistics from a sparse precision matrix. The sparse precision matrix is taken to represent the conditional indepence graph of a Gaussian graphical model.

Usage

```
GGMnetworkStats(sparseP, as.table = FALSE)
```

Arguments

sparseP	Sparse precision/partial correlation matrix.
as.table	A logical indicating if the output should be in tabular format.

Details

The function calculates various network statistics from a sparse matrix. The input matrix `P` is assumed to be a sparse precision or partial correlation matrix. The sparse matrix is taken to represent a conditional independence graph. In the Gaussian setting, conditional independence corresponds to zero entries in the (standardized) precision matrix. Each node in the graph represents a Gaussian variable, and each undirected edge represents conditional dependence in the sense of a nonzero corresponding precision entry.

The function calculates various measures of centrality: node degree, betweenness centrality, closeness centrality, and eigenvalue centrality. It also calculates the number of positive and the number of negative edges for each node. In addition, for each variate the mutual information (with all other variates), the variance, and the partial variance is represented. It is also indicated if the graph is chordal (i.e., triangulated). For more information on network measures, consult, e.g., Newman (2010).

Value

An object of class `list` when `as.table = FALSE`:

`degree`	A `numeric` vector with the node degree for each node.
`betweenness`	A `numeric` vector representing the betweenness centrality for each node.
`closeness`	A `numeric` vector representing the closeness centrality for each node.
`eigenCentrality`	A `numeric` vector representing the eigenvalue centrality for each node.
`nNeg`	An `integer` vector representing the number of negative edges for each node.
`nPos`	An `integer` vector representing the number of positive edges for each node.
`chordal`	A `logical` indicating if the implied graph is chordal.
`mutualInfo`	A `numeric` vector with the mutual information (with all other nodes) for each node.
`variance`	A `numeric` vector representing the variance of each node.
`partialVariance`	A `numeric` vector representing the partial variance of each node.

When `as.table = TRUE` the list items above (with the exception of `chordal`) are represented in tabular form as an object of class `matrix`.

Author(s)

Carel F.W. Peeters <carel.peeters@wur.nl>, Wessel N. van Wieringen

References

Newman, M.E.J. (2010). "Networks: an introduction", Oxford University Press.

See Also

`ridgeP`, `covML`, `sparsify`, `Ugraph`

Examples

```
## Obtain some (high-dimensional) data
p = 25
n = 10
set.seed(333)
X = matrix(rnorm(n*p), nrow = n, ncol = p)
colnames(X)[1:25] = letters[1:25]
Cx <- covML(X)

## Obtain sparsified partial correlation matrix
Pridge   <- ridgeP(Cx, 10, type = "Alt")
PCsparse <- sparsify(Pridge , threshold = "top")$sparseParCor

## Represent the graph and calculate GGM network statistics
Ugraph(PCsparse, "fancy")
## Not run: GGMnetworkStats(PCsparse)
```

`GGMnetworkStats.fused` *Gaussian graphical model network statistics*

Description

Compute various network statistics from a `list` sparse precision matrices. The sparse precision matrix is taken to represent the conditional independence graph of a Gaussian graphical model. This function is a simple wrapper for `GGMnetworkStats`.

Usage

```
GGMnetworkStats.fused(Plist)
```

Arguments

`Plist`	A `list` of sparse precision/partial correlation matrix.

Details

For details on the columns see `GGMnetworkStats`.

Value

A `data.frame` of the various network statistics for each class. The names of `Plist` is prefixed to column-names.

Author(s)

Anders E. Bilgrau, Carel F.W. Peeters <carel.peeters@wur.nl>, Wessel N. van Wieringen

See Also

GGMnetworkStats

Examples

```
## Create some "high-dimensional" data
set.seed(1)
p <- 10
ns <- c(5, 6)
Slist <- createS(ns, p)

## Obtain sparsified partial correlation matrix
Plist    <- ridgeP.fused(Slist, ns, lambda = c(5.2, 1.3), verbose = FALSE)
PCsparse <- sparsify.fused(Plist , threshold = "absValue", absValueCut = 0.2)
SPlist <- lapply(PCsparse, "[[", "sparsePrecision") # Get sparse precisions

## Calculate GGM network statistics in each class
## Not run: GGMnetworkStats.fused(SPlist)
```

GGMpathStats *Gaussian graphical model node pair path statistics*

Description

Function that calculates, for a specified node pair representing endpoints, path statistics from a sparse precision matrix. The sparse precision matrix is taken to represent the conditional independence graph of a Gaussian graphical model. The contribution to the observed covariance between the specified endpoints is calculated for each (heuristically) determined path between the endpoints.

Usage

```
GGMpathStats(
  P0,
  node1,
  node2,
  neiExpansions = 2,
  verbose = TRUE,
  graph = TRUE,
  nrPaths = 2,
  lay = "layout_in_circle",
  coords = NULL,
  nodecol = "skyblue",
  Vsize = 15,
  Vcex = 0.6,
  VBcolor = "darkblue",
  VLcolor = "black",
  all.edges = TRUE,
```

```
  prune = TRUE,
  legend = TRUE,
  scale = 1,
  Lcex = 0.8,
  PTcex = 2,
  main = ""
)
```

Arguments

P0	Sparse (possibly standardized) precision matrix.
node1	A numeric specifying an endpoint. The numeric should correspond to a row/column of the precision matrix and as such represents the corresponding variable.
node2	A numeric specifying a second endpoint. The numeric should correspond to a row/column of the precision matrix and as such represents the corresponding variable.
neiExpansions	A numeric determining how many times the neighborhood around the respective endpoints should be expanded in the search for shortest paths between the node pair.
verbose	A logical indicating if a summary of the results should be printed on screen.
graph	A logical indicating if the strongest paths should be visualized with a graph.
nrPaths	A numeric indicating the number of paths (with the highest contribution to the marginal covariance between the indicated node pair) to be visualized/highlighted.
lay	A character mimicking a call to igraph layout functions. Determines the placement of vertices.
coords	A matrix containing coordinates. Alternative to the lay-argument for determining the placement of vertices.
nodecol	A character determining the color of node1 and node2.
Vsize	A numeric determining the vertex size.
Vcex	A numeric determining the size of the vertex labels.
VBcolor	A character determining the color of the vertex borders.
VLcolor	A character determining the color of the vertex labels.
all.edges	A logical indicating if edges other than those implied by the nrPaths-paths between node1 and node2 should also be visualized.
prune	A logical determining if vertices of degree 0 should be removed.
legend	A logical indicating if the graph should come with a legend.
scale	A numeric representing a scale factor for visualizing strenght of edges. It is a relative scaling factor, in the sense that the edges implied by the nrPaths-paths between node1 and node2 have edge thickness that is twice this scaling factor (so it is a scaling factor vis-a-vis the unimplied edges).
Lcex	A numeric determining the size of the legend box.
PTcex	A numeric determining the size of the exemplary lines in the legend box.
main	A character giving the main figure title.

Details

The conditional independence graph (as implied by the sparse precision matrix) is undirected. In undirected graphs origin and destination are interchangeable and are both referred to as 'endpoints' of a path. The function searches for shortest paths between the specified endpoints `node1` and `node2`. It searches for shortest paths that visit nodes only once. The shortest paths between the provided endpoints are determined heuristically by the following procedure. The search is initiated by application of the `get.all.shortest.paths`-function from the `igraph`-package, which yields all shortest paths between the nodes. Next, the neighborhoods of the endpoints are defined (excluding the endpoints themselves). Then, the shortest paths are found between: (a) `node1` and node *Vs* in its neighborhood; (b) node *Vs* in the `node1`-neighborhood and node *Ve* in the `node2`-neighborhood; and (c) node *Ve* in the `node2`-neighborhood and `node2`. These paths are glued and new shortest path candidates are obtained (preserving only novel paths). In additional iterations (specified by `neiExpansions`) the `node1`- and `node2`-neighborhood are expanded by including their neighbors (still excluding the endpoints) and shortest paths are again searched as described above.

The contribution of a particular path to the observed covariance between the specified node pair is calculated in accordance with Theorem 1 of Jones and West (2005). As in Jones and West (2005), paths whose weights have an opposite sign to the marginal covariance (between endnodes of the path) are referred to as 'moderating paths' while paths whose weights have the same sign as the marginal covariance are referred to as 'mediating' paths. Such paths are visualized when `graph = TRUE`.

All arguments following the `graph` argument are only (potentially) used when `graph = TRUE`. When `graph = TRUE` the conditional independence graph is returned with the paths highlighted that have the highest contribution to the marginal covariance between the specified endpoints. The number of paths highlighted is indicated by `nrPaths`. The edges of mediating paths are represented in green while the edges of moderating paths are represented in red. When `all.edges = TRUE` the edges other than those implied by the `nrPaths`-paths between `node1` and node2 are also visualized (in lightgrey). When `all.edges = FALSE` only the mediating and moderating paths implied by `nrPaths` are visualized.

The default layout gives a circular placement of the vertices. Most layout functions supported by `igraph` are supported (the function is partly a wrapper around certain `igraph` functions). The igraph layouts can be invoked by a `character` that mimicks a call to a `igraph` layout functions in the `lay` argument. When using `lay = NULL` one can specify the placement of vertices with the `coords` argument. The row dimension of this matrix should equal the number of (pruned) vertices. The column dimension then should equal 2 (for 2D layouts) or 3 (for 3D layouts). The `coords` argument can also be viewed as a convenience argument as it enables one, e.g., to layout a graph according to the coordinates of a previous call to `Ugraph`. If both the the lay and the coords arguments are not `NULL`, the lay argument takes precedence

The arguments `Lcex` and `PTcex` are only used when `legend = TRUE`. If `prune = TRUE` the vertices of degree 0 (vertices not implicated by any edge) are removed. For the colors supported by the arguments `nodecol`, `Vcolor`, and `VBcolor`, see https://www.nceas.ucsb.edu/sites/default/files/2020-04/colorPaletteCheatsheet.pdf.

Value

An object of class list:

`pathStats` A `matrix` specifying the paths, their respective lengths, and their respective contributions to the marginal covariance between the endpoints.

paths	A list representing the respective paths as numeric vectors.
Identifier	A data.frame in which each numeric from paths is connected to an identifier such as a variable name.

Note

Eppstein (1998) describes a more sophisticated algorithm for finding the top k shortest paths in a graph.

Author(s)

Wessel N. van Wieringen, Carel F.W. Peeters <carel.peeters@wur.nl>

References

Eppstein, D. (1998). Finding the k Shortest Paths. SIAM Journal on computing 28: 652-673.

Jones, B., and West, M. (2005). Covariance Decomposition in Undirected Gaussian Graphical Models. Biometrika 92: 779-786.

See Also

ridgeP, optPenalty.LOOCVauto, sparsify

Examples

```
## Obtain some (high-dimensional) data
p <- 25
n <- 10
set.seed(333)
X <- matrix(rnorm(n*p), nrow = n, ncol = p)
colnames(X) <- letters[1:p]

## Obtain regularized precision under optimal penalty
OPT <- optPenalty.LOOCVauto(X, lambdaMin = .5, lambdaMax = 30)

## Determine support regularized standardized precision under optimal penalty
PC0 <- sparsify(OPT$optPrec, threshold = "localFDR")$sparseParCor

## Obtain information on mediating and moderating paths between nodes 14 and 23
pathStats <- GGMpathStats(PC0, 14, 23, verbose = TRUE, prune = FALSE)
pathStats
```

GGMpathStats.fused *Fused gaussian graphical model node pair path statistics*

Description

A simple wrapper for GGMpathStats.

Usage

```
GGMpathStats.fused(sparsePlist, ...)
```

Arguments

sparsePlist	A list of sparsified precision matrices.
...	Arguments passed to GGMpathStats.

Value

A list of path stats.

Note

The function currently fails if no paths are present in one of the groups.

Author(s)

Anders E. Bilgrau, Carel F.W. Peeters <carel.peeters@wur.nl>, Wessel N. van Wieringen

See Also

GGMpathStats

Examples

```
## Obtain some (high-dimensional) data
set.seed(1)
ns <- c(10, 11)
Slist <- createS(ns, p = 7, topology = "banded")
Tlist <- default.target.fused(Slist, ns)

## Obtain regularized precision and sparsify
Plist <- ridgeP.fused(Slist, ns, Tlist, lambda = c(1, 1.6))
sparsePlist <- sparsify.fused(Plist, threshold = "absValue", absValueCut = 0.20)
SPlist <- lapply(sparsePlist, "[[", "sparsePrecision")

## Obtain information on mediating and moderating paths between nodes 14 and 23
res <- GGMpathStats.fused(SPlist, node1 = 3, node2 = 4, graph = FALSE)
```

`hist.ptest` *Plot the results of a fusion test*

Description

Plot a histogram of the null distribution and the observed test statistic in a permutation type "fusion test".

Usage

```
## S3 method for class 'ptest'
hist(x, add.extra = TRUE, ...)

## S3 method for class 'ptest'
plot(x, add.extra = TRUE, ...)
```

Arguments

`x`	A `ptest` object (a list). Usually the output of `fused.test`.
`add.extra`	A logical. Add extra information to the plot.
`...`	Arguments passed to `plot`.

Details

`plot.ptest` is simply a wrapper for `hist.ptest`.

Value

Invisibly returns `x` with extra additions.

Author(s)

Anders Ellern Bilgrau, Carel F.W. Peeters <carel.peeters@wur.nl>, Wessel N. van Wieringen

References

Bilgrau, A.E., Peeters, C.F.W., Eriksen, P.S., Boegsted, M., and van Wieringen, W.N. (2020). Targeted Fused Ridge Estimation of Inverse Covariance Matrices from Multiple High-Dimensional Data Classes. Journal of Machine Learning Research, 21(26): 1-52.

See Also

`fused.test`, `print.ptest`

Examples

```
ns <- c(10, 5, 23)
Ylist <- createS(ns, p = 15, topology = "banded", dataset = TRUE)

# Use the identity target matrix for each class
Tlist <- replicate(length(ns), diag(15), simplify = FALSE)

# Do the test
lam <- matrix(10, 3, 3)
diag(lam) <- 1
ft <- fused.test(Ylist, Tlist, lambda = lam, n.permutations = 500)

# The returned object can alo be plotted via
hist(ft)
# or via the alias
plot(ft)
```

---

is.Xlist *Test if fused list-formats are correctly used*

Description

Function to check if the argument submits to the various `list`-formats used by the fused ridge estimator and related functions are correct. That is, it tests if generic fused list arguments (such as `Slist`, `Tlist`, `Plist`, `Ylist`) are properly formatted.

Usage

```
is.Xlist(Xlist, Ylist = FALSE, semi = FALSE)
```

Arguments

`Xlist`	A `list` of precision matrices of equal size (`Plist`), sample covariance matrices (`Slist`), data matrices (`Ylist`)
`Ylist`	`logical`. Is `Xlist` a `list` of data matrices with the same number of columns (`Ylist`).
`semi`	`logical`. Should the matrices in the list be tested to be positive semi definite or positive definite?

Value

Returns `TRUE` if all tests are passed, throws error if not.

Author(s)

Anders Bilgrau, Carel F.W. Peeters <carel.peeters@wur.nl>, Wessel N. van Wieringen

References

Bilgrau, A.E., Peeters, C.F.W., Eriksen, P.S., Boegsted, M., and van Wieringen, W.N. (2020). Targeted Fused Ridge Estimation of Inverse Covariance Matrices from Multiple High-Dimensional Data Classes. Journal of Machine Learning Research, 21(26): 1-52.

van Wieringen, W.N. & Peeters, C.F.W. (2016). Ridge Estimation of Inverse Covariance Matrices from High-Dimensional Data, Computational Statistics & Data Analysis, vol. 103: 284-303. Also available as arXiv:1403.0904v3 [stat.ME].

See Also

`ridgeP.fused`, `optPenalty.fused`

Examples

```
Slist <- createS(n = c(4, 6, 9), p = 10)
is.Xlist(Slist, semi = TRUE)
```

`isSymmetricPD` *Test for symmetric positive (semi-)definiteness*

Description

Function to test if a `matrix` is symmetric positive (semi)definite or not.

Usage

```
isSymmetricPD(M)

isSymmetricPSD(M, tol = 1e-04)
```

Arguments

`M`	A square symmetric matrix.
`tol`	A numeric giving the tolerance for determining positive semi-definiteness.

Details

Tests positive definiteness by Cholesky decomposition. Tests positive semi-definiteness by checking if all eigenvalues are larger than $-\epsilon|\lambda_1|$ where ϵ is the tolerance and λ_1 is the largest eigenvalue.

While `isSymmetricPSD` returns `TRUE` if the matrix is symmetric positive definite and `FASLE` if not. In practice, it tests if all eigenvalues are larger than -tol*|l| where l is the largest eigenvalue. More here.

Value

Returns a `logical` value. Returns `TRUE` if the `M` is symmetric positive (semi)definite and `FALSE` if not. If `M` is not even symmetric, the function throws an error.

Author(s)

Anders Ellern Bilgrau Carel F.W. Peeters <carel.peeters@wur.nl>, Wessel N. van Wieringen

See Also

isSymmetric

Examples

```
A <- matrix(rnorm(25), 5, 5)
## Not run:
isSymmetricPD(A)

## End(Not run)
B <- symm(A)
isSymmetricPD(B)

C <- crossprod(B)
isSymmetricPD(C)

isSymmetricPSD(C)
```

---

kegg.target *Construct target matrix from KEGG*

Description

Construct a target matrix by combining topology information from the Kyoto Encyclopedia of Genes and Genomes (KEGG) database and pilot data.

Usage

```
kegg.target(
  Y,
  kegg.id,
  method = "linreg",
  organism = "hsa",
  graph = getKEGGPathway(kegg.id)$graph
)
```

Arguments

Y	The complete observation matrix of observations with variables in columns. The column names should be on the form e.g. "hsa:3988" ("<organism>:<Entrez id>"). It can however also be just the Entrez id with or without the post-fixed "_at" and then the specified organism will be assumed.
kegg.id	A character giving the KEGG ID, e.g. "map04210", "map04064", or "map04115".

method	The method for estimating the non-zero entries moralized graph of the KEGG topology. Currently, only "linreg" is implemented.
organism	A character giving the organism, the default is "hsa" (homo-sapiens).
graph	A graphNEL object specifying the topology of the pathway. Can be used to avoid repeatedly downloading the information.

Details

The function estimates the precision matrix based on the topology given by the KEGG database. Requires a connection to the internet.

Value

Returns a target matrix with size depending on the kegg.id.

Note

It is currently nessesary to require("KEGGgraph") (or require("KEGGgraph")) due to a bug in **KEGGgraph**.

Author(s)

Anders Ellern Bilgrau, Carel F.W. Peeters <carel.peeters@wur.nl>, Wessel N. van Wieringen

References

https://www.genome.jp/kegg/

See Also

getKEGGPathway, default.target, and default.target.fused

Examples

```
## Not run:
if (require("KEGGgraph")) {
kegg.g <- getKEGGPathway("map04115")$graph

# Create some toy data with the correct names
Y <- createS(n = 10, p = numNodes(kegg.g), dataset = TRUE)
colnames(Y) <- nodes(kegg.g)

T <- kegg.target(Y, "map04115")
print(T[1:10, 1:10])
}

## End(Not run)
```

KLdiv *Kullback-Leibler divergence between two multivariate normal distributions*

Description

Function calculating the Kullback-Leibler divergence between two multivariate normal distributions.

Usage

```
KLdiv(Mtest, Mref, Stest, Sref, symmetric = FALSE)
```

Arguments

Mtest	A numeric mean vector for the approximating multivariate normal distribution.
Mref	A numeric mean vector for the true/reference multivariate normal distribution.
Stest	A covariance matrix for the approximating multivariate normal distribution.
Sref	A covariance matrix for the true/reference multivariate normal distribution.
symmetric	A logical indicating if the symmetric version of Kullback-Leibler divergence should be calculated.

Details

The Kullback-Leibler (KL) information (Kullback and Leibler, 1951; also known as relative entropy) is a measure of divergence between two probability distributions. Typically, one distribution is taken to represent the 'true' distribution and functions as the reference distribution while the other is taken to be an approximation of the true distribution. The criterion then measures the loss of information in approximating the reference distribution. The KL divergence between two p-dimensional multivariate normal distributions $\mathcal{N}_p^0(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0)$ and $\mathcal{N}_p^1(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)$ is given as

$$\mathrm{I}_{KL}(\mathcal{N}_p^0 \| \mathcal{N}_p^1) = \frac{1}{2}\left\{\mathrm{tr}(\boldsymbol{\Omega}_1\boldsymbol{\Sigma}_0) + (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0)^{\mathrm{T}}\boldsymbol{\Omega}_1(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0) - p - \ln|\boldsymbol{\Sigma}_0| + \ln|\boldsymbol{\Sigma}_1|\right\},$$

where $\boldsymbol{\Omega} = \boldsymbol{\Sigma}^{-1}$. The KL divergence is not a proper metric as $\mathrm{I}_{KL}(\mathcal{N}_p^0 \| \mathcal{N}_p^1) \neq \mathrm{I}_{KL}(\mathcal{N}_p^1 \| \mathcal{N}_p^0)$. When symmetric = TRUE the function calculates the symmetric KL divergence (also referred to as Jeffreys information), given as

$$\mathrm{I}_{KL}(\mathcal{N}_p^0 \| \mathcal{N}_p^1) + \mathrm{I}_{KL}(\mathcal{N}_p^1 \| \mathcal{N}_p^0).$$

Value

Function returns a numeric representing the (symmetric) Kullback-Leibler divergence.

Author(s)

Wessel N. van Wieringen, Carel F.W. Peeters <carel.peeters@wur.nl>

References

Kullback, S. and Leibler, R.A. (1951). On Information and Sufficiency. Annals of Mathematical Statistics 22: 79-86.

See Also

covML, ridgeP

Examples

```
## Define population
set.seed(333)
p = 25
n = 1000
X = matrix(rnorm(n*p), nrow = n, ncol = p)
colnames(X)[1:25] = letters[1:25]
Cov0  <- covML(X)
mean0 <- colMeans(X)

## Obtain sample from population
samples <- X[sample(nrow(X), 10),]
Cov1  <- covML(samples)
mean1 <- colMeans(samples)

## Regularize singular Cov1
P <- ridgeP(Cov1, 10)
CovR <- solve(P)

## Obtain KL divergence
KLdiv(mean1, mean0, CovR, Cov0)
```

KLdiv.fused *Fused Kullback-Leibler divergence for sets of distributions*

Description

Function calculating the Kullback-Leibler divergence between two sets of multivariate normal distributions. In other words, it calculates a weigthed mean of Kullback-Leibler divergences between multiple paired normal distributions.

Usage

```
KLdiv.fused(MtestList, MrefList, StestList, SrefList, ns, symmetric = FALSE)
```

Arguments

MtestList	A list of mean vectors of the approximating multivariate normal distribution for each class. Assumed to be zero vectors if not supplied.
MrefList	A list of mean vectors of the reference multivariate normal distribution for each class. Assumed to be zero vectors if not supplied.
StestList	A list of covariance matrices of the approximating multivariate normal distribtuion for each class. Usually a list of sample covariance matrices.
SrefList	A list of covariance matrices of the references multivariate normal distribtuion for each class. Usually a list of the population or reference covariance matrices.
ns	a numeric of the same length as the previous arguments giving the sample sizes. Used as weights in the weighted mean.
symmetric	a logical indicating if original symmetric version of KL divergence should be calculated.

Value

Function returns a numeric representing the (optionally symmetric) fused Kullback-Leibler divergence.

Author(s)

Anders Ellern Bilgrau, Wessel N. van Wieringen, Carel F.W. Peeters <carel.peeters@wur.nl>

See Also

KLdiv

Examples

```
# Create some toy data
n <- c(40, 60, 80)
p <- 10
Stest <- replicate(length(n), diag(p), simplify = FALSE)
Sref <- createS(n, p = p)

KLdiv.fused(StestList = Stest, SrefList = Sref, ns = n, symmetric = FALSE)
KLdiv.fused(StestList = Stest, SrefList = Sref, ns = n, symmetric = TRUE)
```

`loss` *Evaluate regularized precision under various loss functions*

Description

Function that evaluates an estimated and possibly regularized precision matrix under various loss functions. The loss functions are formulated in precision terms. This function may be used to estimate the risk (vis-a-vis, say, the true precision matrix) of the various ridge estimators employed.

Usage

```
loss(E, T, precision = TRUE, type = c("frobenius", "quadratic"))
```

Arguments

`E`	Estimated (possibly regularized) precision `matrix`.
`T`	True (population) covariance or precision `matrix`.
`precision`	A `logical` indicating if T is a precision matrix.
`type`	A `character` indicating which loss function is to be used. Must be one of: "frobenius", "quadratic".

Details

Let $\boldsymbol{\Omega}$ denote a generic $(p \times p)$ population precision matrix and let $\hat{\boldsymbol{\Omega}}(\lambda)$ denote a generic ridge estimator of the precision matrix under generic regularization parameter λ (see also `ridgeP`). The function then considers the following loss functions:

1. Squared Frobenius loss, given by:

$$L_F[\hat{\boldsymbol{\Omega}}(\lambda), \boldsymbol{\Omega}] = \|\hat{\boldsymbol{\Omega}}(\lambda) - \boldsymbol{\Omega}\|_F^2;$$

2. Quadratic loss, given by:

$$L_Q[\hat{\boldsymbol{\Omega}}(\lambda), \boldsymbol{\Omega}] = \|\hat{\boldsymbol{\Omega}}(\lambda)\boldsymbol{\Omega}^{-1} - \mathbf{I}_p\|_F^2.$$

The argument `T` is considered to be the true precision matrix when `precision = TRUE`. If `precision = FALSE` the argument `T` is considered to represent the true covariance matrix. This statement is needed so that the loss is properly evaluated over the precision, i.e., depending on the value of the `logical` argument `precision` inversions are employed where needed.

The function can be employed to assess the risk of a certain ridge precision estimator (see also `ridgeP`). The risk $\mathcal{R}_f$ of the estimator $\hat{\boldsymbol{\Omega}}(\lambda)$ given a loss function L_f, with $f \in \{F, Q\}$ can be defined as the expected loss:

$$\mathcal{R}_f[\hat{\boldsymbol{\Omega}}(\lambda)] = \mathrm{E}\{L_f[\hat{\boldsymbol{\Omega}}(\lambda), \boldsymbol{\Omega}]\},$$

which can be approximated by the mean or median of losses over repeated simulation runs.

Value

Function returns a `numeric` representing the loss under the chosen loss function.

Author(s)

Carel F.W. Peeters <carel.peeters@wur.nl>, Wessel N. van Wieringen

References

van Wieringen, W.N. & Peeters, C.F.W. (2016). Ridge Estimation of Inverse Covariance Matrices from High-Dimensional Data, Computational Statistics & Data Analysis, vol. 103: 284-303. Also available as arXiv:1403.0904v3 [stat.ME].

See Also

covML, ridgeP

Examples

```
## Define population covariance
set.seed(333)
p = 25
n = 1000
X = matrix(rnorm(n*p), nrow = n, ncol = p)
colnames(X)[1:25] = letters[1:25]
Truecov <- covML(X)

## Obtain sample
samples <- X[sample(nrow(X), 10), ]
Cxx <- covML(samples)

## Obtain regularized precision
P <- ridgeP(Cxx, 10, type = "Alt")

## Evaluate estimated precision against population
## precision under Frobenius loss
loss(P, Truecov, precision = FALSE, type = "frobenius")
```

---

momentS *Moments of the sample covariance matrix.*

Description

Calculates the moments of the sample covariance matrix. It assumes that the summands (the outer products of the samples' random data vector) that constitute the sample covariance matrix follow a Wishart-distribution with scale parameter $\mathbf{\Sigma}$ and shape parameter ν. The latter is equal to the number of summands in the sample covariance estimate.

Usage

```
momentS(Sigma, shape, moment = 1)
```

Arguments

Sigma	Positive-definite matrix, the scale parameter $\mathbf{\Sigma}$ of the Wishart distribution.
shape	A numeric, the shape parameter ν of the Wishart distribution. Should exceed the number of variates (number of rows or columns of Sigma).
moment	An integer. Should be in the set $\{-4, -3, -2, -1, 0, 1, 2, 3, 4\}$ (only those are explicitly specified in Lesac, Massam, 2004).

Value

The r-th moment of a sample covariance matrix: $E(\mathbf{S}^r)$.

Author(s)

Wessel N. van Wieringen.

References

Lesac, G., Massam, H. (2004), "All invariant moments of the Wishart distribution", *Scandinavian Journal of Statistics*, 31(2), 295-318.

Examples

```
# create scale parameter
Sigma <- matrix(c(1, 0.5, 0, 0.5, 1, 0, 0, 0, 1), byrow=TRUE, ncol=3)

# evaluate expectation of the square of a sample covariance matrix
# that is assumed to Wishart-distributed random variable with the
# above scale parameter Sigma and shape parameter equal to 40.
momentS(Sigma, 40, 2)
```

NLL *Evaluate the (penalized) (fused) likelihood*

Description

Functions that evaluate the (penalized) (fused) likelihood.

Usage

```
NLL(S, P)

PNLL(S, P, T, lambda)

NLL.fused(Slist, Plist, ns)

PNLL.fused(Slist, Plist, ns, Tlist, lambda)
```

Arguments

S, Slist	A (list of) positive semi definite sample covariance matrices.
P, Plist	A (list of) positive definite precision matrices.
T, Tlist	A (list of) positive definite target matrices.
lambda	A numeric penalty parameter. For the .fused functions, this is a penalty matrix.
ns	A numeric of sample sizes.

Value

A single number.

Author(s)

Anders Ellern Bilgrau, Carel F.W. Peeters <cf.peeters@vumc.nl>, Wessel N. van Wieringen

See Also

ridgeP, ridgeP.fused

Examples

```
ns <- c(4,5)
Slist <- createS(n = ns, p = 5)
Plist <- list(diag(5), diag(2,5))
Tlist <- list(diag(5), diag(5))

NLL(Slist[[1]], Plist[[1]])
PNLL(Slist[[1]], Plist[[1]], Tlist[[1]], lambda = 1)
NLL.fused(Slist, Plist, ns)
PNLL.fused(Slist, Plist, ns, Tlist, lambda = diag(2))
```

---

optPenalty.aLOOCV *Select optimal penalty parameter by approximate leave-one-out cross-validation*

Description

Function that selects the optimal penalty parameter for the ridgeP call by usage of approximate leave-one-out cross-validation. Its output includes (a.o.) the precision matrix under the optimal value of the penalty parameter.

Usage

```
optPenalty.aLOOCV(
  Y,
  lambdaMin,
  lambdaMax,
  step,
  type = "Alt",
  cor = FALSE,
  target = default.target(covML(Y)),
  output = "light",
  graph = TRUE,
  verbose = TRUE
)
```

Arguments

Y	Data matrix. Variables assumed to be represented by columns.
lambdaMin	A numeric giving the minimum value for the penalty parameter.
lambdaMax	A numeric giving the maximum value for the penalty parameter.
step	An integer determining the number of steps in moving through the grid [lambdaMin, lambdaMax].
type	A character indicating the type of ridge estimator to be used. Must be one of: "Alt", "ArchI", "ArchII".
cor	A logical indicating if the evaluation of the approximate LOOCV score should be performed on the correlation scale.
target	A target matrix (in precision terms) for Type I ridge estimators.
output	A character indicating if the output is either heavy or light. Must be one of: "all", "light".
graph	A logical indicating if the grid search for the optimal penalty parameter should be visualized.
verbose	A logical indicating if information on progress should be printed on screen.

Details

The function calculates an approximate leave-one-out cross-validated (aLOOCV) negative log-likelihood score (using a regularized ridge estimator for the precision matrix) for each value of the penalty parameter contained in the search grid. The utilized aLOOCV score was proposed by Lian (2011) and Vujacic et al. (2014). The aLOOCV negative log-likeliho od score is computationally more efficient than its non-approximate counterpart (see optPenalty.LOOCV). For details on the aLOOCV negative log-likelihood score see Lian (2011) and Vujacic et al (2014). For scalar matrix targets (see default.target) the complete solution path of the alternative Type I and II ridge estimators (see ridgeP) depends on only 1 eigendecomposition and 1 matrix inversion, making the determination of the optimal penalty value particularly efficient (see van Wieringen and Peeters, 2015).

The value of the penalty parameter that achieves the lowest aLOOCV negative log-likelihood score is deemed optimal. The penalty parameter must be positive such that lambdaMin must be a positive

scalar. The maximum allowable value of `lambdaMax` depends on the type of ridge estimator employed. For details on the type of ridge estimator one may use (one of: "Alt", "ArchI", "ArchII") see `ridgeP`. The ouput consists of an object of class list (see below). When `output = "light"` (default) only the `optLambda` and `optPrec` elements of the list are given.

Value

An object of class list:

`optLambda`	A `numeric` giving the optimal value of the penalty parameter.
`optPrec`	A `matrix` representing the precision matrix of the chosen type (see `ridgeS`) under the optimal value of the penalty parameter.
`lambdas`	A `numeric` vector representing all values of the penalty parameter for which approximate cross-validation was performed; Only given when `output = "all"`.
`aLOOCVs`	A `numeric` vector representing the approximate cross-validated negative log-likelihoods for each value of the penalty parameter given in `lambdas`; Only given when `output = "all"`.

Note

When `cor = TRUE` correlation matrices are used in the computation of the approximate (cross-validated) negative log-likelihood score, i.e., the sample covariance matrix is a matrix on the correlation scale. When performing evaluation on the correlation scale the data are assumed to be standardized. If `cor = TRUE` and one wishes to used the default target specification one may consider using `target = default.target(covML(Y, cor = TRUE))`. This gives a default target under the assumption of standardized data.

Author(s)

Carel F.W. Peeters <carel.peeters@wur.nl>, Wessel N. van Wieringen

References

Lian, H. (2011). Shrinkage tuning parameter selection in precision matrices estimation. Journal of Statistical Planning and Inference, 141: 2839-2848.

van Wieringen, W.N. & Peeters, C.F.W. (2016). Ridge Estimation of Inverse Covariance Matrices from High-Dimensional Data, Computational Statistics & Data Analysis, vol. 103: 284-303. Also available as arXiv:1403.0904v3 [stat.ME].

Vujacic, I., Abbruzzo, A., and Wit, E.C. (2014). A computationally fast alternative to cross-validation in penalized Gaussian graphical models. arXiv: 1309.6216v2 [stat.ME].

See Also

`ridgeP`, `optPenalty.LOOCV`, `optPenalty.LOOCVauto`, `default.target`, `covML`

Examples

```
## Obtain some (high-dimensional) data
p = 25
n = 10
set.seed(333)
X = matrix(rnorm(n*p), nrow = n, ncol = p)
colnames(X)[1:25] = letters[1:25]

## Obtain regularized precision under optimal penalty
OPT  <- optPenalty.aLOOCV(X, lambdaMin = .001, lambdaMax = 30, step = 400); OPT
OPT$optLambda # Optimal penalty
OPT$optPrec   # Regularized precision under optimal penalty

## Another example with standardized data
X <- scale(X, center = TRUE, scale = TRUE)
OPT  <- optPenalty.aLOOCV(X, lambdaMin = .001, lambdaMax = 30,
                          step = 400, cor = TRUE,
                          target = default.target(covML(X, cor = TRUE))); OPT
OPT$optLambda # Optimal penalty
OPT$optPrec   # Regularized precision under optimal penalty
```

`optPenalty.fused.grid` *Identify optimal ridge and fused ridge penalties*

Description

Functions to find the optimal ridge and fusion penalty parameters via leave-one-out cross validation. The functions support leave-one-out cross-validation (LOOCV), k-fold CV, and two forms of approximate LOOCV. Depending on the used function, general numerical optimization or a grid-based search is used.

Usage

```
optPenalty.fused.grid(
  Ylist,
  Tlist,
  lambdas = 10^seq(-5, 5, length.out = 15),
  lambdaFs = lambdas,
  cv.method = c("LOOCV", "aLOOCV", "sLOOCV", "kCV"),
  k = 10,
  verbose = TRUE,
  ...
)

optPenalty.fused.auto(
  Ylist,
  Tlist,
```

```
  lambda,
  cv.method = c("LOOCV", "aLOOCV", "sLOOCV", "kCV"),
  k = 10,
  verbose = TRUE,
  lambda.init,
  maxit.ridgeP.fused = 1000,
  optimizer = "optim",
  maxit.optimizer = 1000,
  debug = FALSE,
  optim.control = list(trace = verbose, maxit = maxit.optimizer),
  ...
)

optPenalty.fused(
  Ylist,
  Tlist,
  lambda = default.penalty(Ylist),
  cv.method = c("LOOCV", "aLOOCV", "sLOOCV", "kCV"),
  k = 10,
  grid = FALSE,
  ...
)
```

Arguments

`Ylist`	A `list` of G matrices of data with n_g samples in the rows and p variables in the columns corresponding to G classes of data.
`Tlist`	A `list` of G of p.d. class target matrices of size p times p.
`lambdas`	A `numeric` vector of positive ridge penalties.
`lambdaFs`	A `numeric` vector of non-negative fusion penalties.
`cv.method`	`character` giving the cross-validation (CV) to use. The allowed values are `"LOOCV"`, `"aLOOCV"`, `"sLOOCV"`, `"kCV"` for leave-one-out cross validation (LOOCV), appproximate LOOCV, special LOOCV, and k-fold CV, respectively.
`k`	`integer` giving the number of approximately equally sized parts each class is partioned into for k-fold CV. Only use if `cv.method` is `"kCV"`.
`verbose`	`logical`. If `TRUE`, progress information is printed to the console.
`...`	For `optPenalty.fused`, arguments are passed to `optPenalty.fused.grid` or `optPenalty.fused.auto` depending on the value of `grid`. In `optPenalty.fused.grid`, arguments are passed to `ridgeP.fused`. In `optPenalty.fused.auto`, arguments are passed to the optimizer.
`lambda`	A symmetric `character` `matrix` encoding the class of penalty matrices to cross-validate over. The diagonal elements correspond to the class-specific ridge penalties whereas the off-diagonal elements correspond to the fusion penalties. The unique elements of lambda specify the penalties to determine by the method specified by `cv.method`. The penalties can be fixed if they are coercible to numeric values, such as e.g. `"0"`, `"2.71"` or `"3.14"`. Fusion between pairs can be

"left out"" using either of "", NA, "NA", or "0". See default.penalty for help on the construction hereof and more details. Unused and can be omitted if grid == TRUE.

lambda.init — A numeric penalty matrix of initial values passed to the optimizer. If omitted, the function selects a starting values using a common ridge penaltiy (determined by 1D optimization) and all fusion penalties to zero.

maxit.ridgeP.fused — A integer giving the maximum number of iterations allowed for each fused ridge fit.

optimizer — character. Either "optim" or "nlm" determining which optimizer to use.

maxit.optimizer — A integer giving the maximum number of iterations allowed in the optimization procedure.

debug — logical. If TRUE additional output from the optimizer is appended to the output as an attribute.

optim.control — A list of control arguments for optim.

grid — logical. Should a grid based search be used? Default is FALSE.

Details

optPenalty.fused.auto serves a utilizes optim for identifying the optimal fused parameters and works for general classes of penalty graphs.

optPenalty.fused.grid gives a grid-based evaluation of the (approximate) LOOCV loss.

Value

optPenalty.fused.auto returns a list:

Plist — A list of the precision estimates for the optimal parameters.

lambda — The estimated optimal fused penalty matrix.

lambda.unique — The unique entries of the lambda. A more concise overview of lambda

value — The value of the loss function in the estimated optimum.

optPenalty.fused.LOOCV returns a list:

ridge — A numeric vector of grid values for the ridge penalty

fusion — The numeric vector of grid values for the fusion penalty

fcvl — The numeric matrix of evaluations of the loss function

Author(s)

Anders Ellern Bilgrau, Carel F.W. Peeters <carel.peeters@wur.nl>, Wessel N. van Wieringen

References

Bilgrau, A.E., Peeters, C.F.W., Eriksen, P.S., Boegsted, M., and van Wieringen, W.N. (2020). Targeted Fused Ridge Estimation of Inverse Covariance Matrices from Multiple High-Dimensional Data Classes. Journal of Machine Learning Research, 21(26): 1-52.

See Also

See also `default.penalty`, `optPenalty.LOOCV`.

Examples

```
## Not run:
# Generate some (not so) high-dimensional data witn (not so) many samples
ns <- c(4, 5, 6)
Ylist <- createS(n = ns, p = 6, dataset = TRUE)
Slist <- lapply(Ylist, covML)
Tlist <- default.target.fused(Slist, ns, type = "DIAES")


# Grid-based
lambdas <- 10^seq(-5, 3, length.out = 7)
a <- optPenalty.fused.grid(Ylist, Tlist,
                           lambdas = lambdas,
                           cv.method = "LOOCV", maxit = 1000)
b <- optPenalty.fused.grid(Ylist, Tlist,
                           lambdas = lambdas,
                           cv.method = "aLOOCV", maxit = 1000)
c <- optPenalty.fused.grid(Ylist, Tlist,
                           lambdas = lambdas,
                           cv.method = "sLOOCV", maxit = 1000)
d <- optPenalty.fused.grid(Ylist, Tlist,
                           lambdas = lambdas,
                           cv.method = "kCV", k = 2, maxit = 1000)

# Numerical optimization (uses the default "optim" optimizer with method "BFGS")
aa <- optPenalty.fused.auto(Ylist, Tlist, cv.method = "LOOCV", method = "BFGS")
print(aa)
bb <- optPenalty.fused.auto(Ylist, Tlist, cv.method = "aLOOCV", method = "BFGS")
print(bb)
cc <- optPenalty.fused.auto(Ylist, Tlist, cv.method = "sLOOCV", method = "BFGS")
print(cc)
dd <- optPenalty.fused.auto(Ylist, Tlist, cv.method = "kCV", k=3, method="BFGS")
print(dd)


#
# Plot the results
#

# LOOCV
# Get minimums and plot
amin  <- log(expand.grid(a$lambda, a$lambdaF))[which.min(a$fcvl), ]
```

```
aamin <- c(log(aa$lambda[1,1]), log(aa$lambda[1,2]))

# Plot
filled.contour(log(a$lambda), log(a$lambdaF), log(a$fcvl), color = heat.colors,
               plot.axes = {points(amin[1], amin[2], pch = 16);
                            points(aamin[1], aamin[2], pch = 16, col = "purple");
                            axis(1); axis(2)},
               xlab = "lambda", ylab = "lambdaF", main = "LOOCV")

# Approximate LOOCV
# Get minimums and plot
bmin <- log(expand.grid(b$lambda, b$lambdaF))[which.min(b$fcvl), ]
bbmin <- c(log(bb$lambda[1,1]), log(unique(bb$lambda[1,2])))

filled.contour(log(b$lambda), log(b$lambdaF), log(b$fcvl), color = heat.colors,
               plot.axes = {points(bmin[1], bmin[2], pch = 16);
                            points(bbmin[1], bbmin[2], pch = 16, col ="purple");
                            axis(1); axis(2)},
               xlab = "lambda", ylab = "lambdaF", main = "Approximate LOOCV")


#
# Arbitrary penalty graphs
#

# Generate some new high-dimensional data and a 2 by 2 factorial design
ns <- c(6, 5, 3, 2)
df <- expand.grid(Factor1 = LETTERS[1:2], Factor2 = letters[3:4])
Ylist <- createS(n = ns, p = 4, dataset = TRUE)
Tlist <- lapply(lapply(Ylist, covML), default.target, type = "Null")

# Construct penalty matrix
lambda <- default.penalty(df, type = "CartesianUnequal")

# Find optimal parameters,
# Using optim with method "Nelder-Mead" with "special" LOOCV
ans1 <- optPenalty.fused(Ylist, Tlist, lambda = lambda,
                         cv.method = "sLOOCV", verbose = FALSE)
print(ans1$lambda.unique)

# By approximate LOOCV using optim with method "BFGS"
ans2 <- optPenalty.fused(Ylist, Tlist, lambda = lambda,
                         cv.method = "aLOOCV", verbose = FALSE,
                         method = "BFGS")
print(ans2$lambda.unique)

# By LOOCV using nlm
lambda.init <- matrix(1, 4, 4)
lambda.init[cbind(1:4,4:1)] <- 0
ans3 <- optPenalty.fused(Ylist, Tlist, lambda = lambda,
                         lambda.init = lambda.init,
                         cv.method = "LOOCV", verbose = FALSE,
                         optimizer = "nlm")
```

```
print(ans3$lambda.unique)

# Quite different results!


#
# Arbitrary penalty graphs with fixed penalties!
#

# Generate some new high-dimensional data and a 2 by 2 factorial design
ns <- c(6, 5, 5, 5)
df <- expand.grid(DS = LETTERS[1:2], ER = letters[3:4])
Ylist <- createS(n = ns, p = 4, dataset = TRUE)
Tlist <- lapply(lapply(Ylist, covML), default.target, type = "Null")

lambda <- default.penalty(df, type = "Tensor")
print(lambda)  # Say we want to penalize the pair (1,2) with strength 2.1;
lambda[2,1] <- lambda[1,2] <- 2.1
print(lambda)

# Specifiying starting values is also possible:
init <- diag(length(ns))
init[2,1] <- init[1,2] <- 2.1

res <- optPenalty.fused(Ylist, Tlist, lambda = lambda, lambda.init = init,
                        cv.method = "aLOOCV", optimizer = "nlm")
print(res)

## End(Not run)
```

`optPenalty.kCV` *Select optimal penalty parameter by K-fold cross-validation*

Description

Function that selects the optimal penalty parameter for the `ridgeP` call by usage of K-fold cross-validation. Its output includes (a.o.) the precision matrix under the optimal value of the penalty parameter.

Usage

```
optPenalty.kCV(
  Y,
  lambdaMin,
  lambdaMax,
  step,
  fold = nrow(Y),
  cor = FALSE,
```

```
  target = default.target(covML(Y)),
  type = "Alt",
  output = "light",
  graph = TRUE,
  verbose = TRUE
)
```

Arguments

Y	Data matrix. Variables assumed to be represented by columns.
lambdaMin	A numeric giving the minimum value for the penalty parameter.
lambdaMax	A numeric giving the maximum value for the penalty parameter.
step	An integer determining the number of steps in moving through the grid [lambdaMin, lambdaMax].
fold	A numeric or integer specifying the number of folds to apply in the cross-validation.
cor	A logical indicating if the evaluation of the LOOCV score should be performed on the correlation scale.
target	A target matrix (in precision terms) for Type I ridge estimators.
type	A character indicating the type of ridge estimator to be used. Must be one of: "Alt", "ArchI", "ArchII".
output	A character indicating if the output is either heavy or light. Must be one of: "all", "light".
graph	A logical indicating if the grid search for the optimal penalty parameter should be visualized.
verbose	A logical indicating if information on progress should be printed on screen.

Details

The function calculates a cross-validated negative log-likelihood score (using a regularized ridge estimator for the precision matrix) for each value of the penalty parameter contained in the search grid by way of K-fold cross-validation. The value of the penalty parameter that achieves the lowest cross-validated negative log-likelihood score is deemed optimal. The penalty parameter must be positive such that lambdaMin must be a positive scalar. The maximum allowable value of lambdaMax depends on the type of ridge estimator employed. For details on the type of ridge estimator one may use (one of: "Alt", "ArchI", "ArchII") see ridgeP. The ouput consists of an object of class list (see below). When output = "light" (default) only the optLambda and optPrec elements of the list are given.

Value

An object of class list:

optLambda	A numeric giving the optimal value of the penalty parameter.
optPrec	A matrix representing the precision matrix of the chosen type (see ridgeP) under the optimal value of the penalty parameter.

`lambdas` A `numeric` vector representing all values of the penalty parameter for which cross-validation was performed; Only given when `output = "all"`.

`LLs` A `numeric` vector representing the mean of cross-validated negative log-likelihoods for each value of the penalty parameter given in `lambdas`; Only given when `output = "all"`.

Note

When `cor = TRUE` correlation matrices are used in the computation of the (cross-validated) negative log-likelihood score, i.e., the K-fold sample covariance matrix is a matrix on the correlation scale. When performing evaluation on the correlation scale the data are assumed to be standardized. If `cor = TRUE` and one wishes to used the default target specification one may consider using `target = default.target(covML(Y, cor = TRUE))`. This gives a default target under the assumption of standardized data.

Under the default setting of the fold-argument, `fold = nrow(Y)`, one performes leave-one-out cross-validation.

Author(s)

Carel F.W. Peeters <carel.peeters@wur.nl>, Wessel N. van Wieringen

See Also

`ridgeP`, `optPenalty.kCVauto`, `optPenalty.aLOOCV`, `default.target`, `covML`

Examples

```
## Obtain some (high-dimensional) data
p = 25
n = 10
set.seed(333)
X = matrix(rnorm(n*p), nrow = n, ncol = p)
colnames(X)[1:25] = letters[1:25]

## Obtain regularized precision under optimal penalty using K = n
OPT  <- optPenalty.kCV(X, lambdaMin = .5, lambdaMax = 30, step = 100); OPT
OPT$optLambda # Optimal penalty
OPT$optPrec   # Regularized precision under optimal penalty

## Another example with standardized data
X <- scale(X, center = TRUE, scale = TRUE)
OPT  <- optPenalty.kCV(X, lambdaMin = .5, lambdaMax = 30, step = 100, cor = TRUE,
                       target = default.target(covML(X, cor = TRUE))); OPT
OPT$optLambda # Optimal penalty
OPT$optPrec   # Regularized precision under optimal penalty

## Another example using K = 5
OPT  <- optPenalty.kCV(X, lambdaMin = .5, lambdaMax = 30, step = 100, fold = 5); OPT
OPT$optLambda # Optimal penalty
OPT$optPrec   # Regularized precision under optimal penalty
```

`optPenalty.kCVauto` *Automatic search for optimal penalty parameter*

Description

Function that performs an 'automatic' search for the optimal penalty parameter for the `ridgeP` call by employing Brent's method to the calculation of a cross-validated negative log-likelihood score.

Usage

```
optPenalty.kCVauto(
  Y,
  lambdaMin,
  lambdaMax,
  lambdaInit = (lambdaMin + lambdaMax)/2,
  fold = nrow(Y),
  cor = FALSE,
  target = default.target(covML(Y)),
  type = "Alt"
)
```

Arguments

`Y`	Data `matrix`. Variables assumed to be represented by columns.
`lambdaMin`	A `numeric` giving the minimum value for the penalty parameter.
`lambdaMax`	A `numeric` giving the maximum value for the penalty parameter.
`lambdaInit`	A `numeric` giving the initial (starting) value for the penalty parameter.
`fold`	A `numeric` or `integer` specifying the number of folds to apply in the cross-validation.
`cor`	A `logical` indicating if the evaluation of the LOOCV score should be performed on the correlation scale.
`target`	A target `matrix` (in precision terms) for Type I ridge estimators.
`type`	A `character` indicating the type of ridge estimator to be used. Must be one of: "Alt", "ArchI", "ArchII".

Details

The function determines the optimal value of the penalty parameter by application of the Brent algorithm (1971) to the K-fold cross-validated negative log-likelihood score (using a regularized ridge estimator for the precision matrix). The search for the optimal value is automatic in the sense that in order to invoke the root-finding abilities of the Brent method, only a minimum value and a maximum value for the penalty parameter need to be specified as well as a starting penalty value. The value at which the K-fold cross-validated negative log-likelihood score is minimized is deemed optimal. The function employs the Brent algorithm as implemented in the optim function.

Value

An object of class `list`:

`optLambda`	A `numeric` giving the optimal value for the penalty parameter.
`optPrec`	A `matrix` representing the precision matrix of the chosen type (see `ridgeP`) under the optimal value of the penalty parameter.

Note

When `cor = TRUE` correlation matrices are used in the computation of the (cross-validated) negative log-likelihood score, i.e., the K-fold sample covariance matrix is a matrix on the correlation scale. When performing evaluation on the correlation scale the data are assumed to be standardized. If `cor = TRUE` and one wishes to used the default target specification one may consider using `target = default.target(covML(Y, cor = TRUE))`. This gives a default target under the assumption of standardized data.

Under the default setting of the fold-argument, `fold = nrow(Y)`, one performes leave-one-out cross-validation.

Author(s)

Wessel N. van Wieringen, Carel F.W. Peeters <carel.peeters@wur.nl>

References

Brent, R.P. (1971). An Algorithm with Guaranteed Convergence for Finding a Zero of a Function. Computer Journal 14: 422-425.

See Also

`GGMblockNullPenalty`, `GGMblockTest`, `ridgeP`, `optPenalty.aLOOCV`, `optPenalty.kCV`, `default.target`, `covML`

Examples

```
## Obtain some (high-dimensional) data
p = 25
n = 10
set.seed(333)
X = matrix(rnorm(n*p), nrow = n, ncol = p)
colnames(X)[1:25] = letters[1:25]

## Obtain regularized precision under optimal penalty using K = n
OPT <- optPenalty.kCVauto(X, lambdaMin = .001, lambdaMax = 30); OPT
OPT$optLambda # Optimal penalty
OPT$optPrec   # Regularized precision under optimal penalty

## Another example with standardized data
X <- scale(X, center = TRUE, scale = TRUE)
OPT <- optPenalty.kCVauto(X, lambdaMin = .001, lambdaMax = 30, cor = TRUE,
                          target = default.target(covML(X, cor = TRUE))); OPT
```

```
OPT$optLambda # Optimal penalty
OPT$optPrec   # Regularized precision under optimal penalty

## Another example using K = 5
OPT <- optPenalty.kCVauto(X, lambdaMin = .001, lambdaMax = 30, fold = 5); OPT
OPT$optLambda # Optimal penalty
OPT$optPrec   # Regularized precision under optimal penalty
```

`optPenalty.LOOCV` *Select optimal penalty parameter by leave-one-out cross-validation*

Description

This function is now deprecated. Please use `optPenalty.kCV` instead.

Usage

```
optPenalty.LOOCV(
  Y,
  lambdaMin,
  lambdaMax,
  step,
  type = "Alt",
  cor = FALSE,
  target = default.target(covML(Y)),
  output = "light",
  graph = TRUE,
  verbose = TRUE
)
```

Arguments

`Y`	Data `matrix`. Variables assumed to be represented by columns.
`lambdaMin`	A `numeric` giving the minimum value for the penalty parameter.
`lambdaMax`	A `numeric` giving the maximum value for the penalty parameter.
`step`	An `integer` determining the number of steps in moving through the grid [`lambdaMin`, `lambdaMax`].
`type`	A `character` indicating the type of ridge estimator to be used. Must be one of: "Alt", "ArchI", "ArchII".
`cor`	A `logical` indicating if the evaluation of the LOOCV score should be performed on the correlation scale.
`target`	A target `matrix` (in precision terms) for Type I ridge estimators.
`output`	A `character` indicating if the output is either heavy or light. Must be one of: "all", "light".
`graph`	A `logical` indicating if the grid search for the optimal penalty parameter should be visualized.
`verbose`	A `logical` indicating if information on progress should be printed on screen.

Details

Function that selects the optimal penalty parameter for the ridgeP call by usage of leave-one-out cross-validation. Its output includes (a.o.) the precision matrix under the optimal value of the penalty parameter.

The function calculates a cross-validated negative log-likelihood score (using a regularized ridge estimator for the precision matrix) for each value of the penalty parameter contained in the search grid by way of leave-one-out cross-validation. The value of the penalty parameter that achieves the lowest cross-validated negative log-likelihood score is deemed optimal. The penalty parameter must be positive such that `lambdaMin` must be a positive scalar. The maximum allowable value of `lambdaMax` depends on the type of ridge estimator employed. For details on the type of ridge estimator one may use (one of: "Alt", "ArchI", "ArchII") see ridgeP. The ouput consists of an object of class list (see below). When `output = "light"` (default) only the `optLambda` and `optPrec` elements of the list are given.

Value

An object of class list:

`optLambda`	A `numeric` giving the optimal value of the penalty parameter.
`optPrec`	A `matrix` representing the precision matrix of the chosen type (see ridgeP) under the optimal value of the penalty parameter.
`lambdas`	A `numeric` vector representing all values of the penalty parameter for which cross-validation was performed; Only given when `output = "all"`.
`LLs`	A `numeric` vector representing the mean of cross-validated negative log-likelihoods for each value of the penalty parameter given in `lambdas`; Only given when `output = "all"`.

Note

When `cor = TRUE` correlation matrices are used in the computation of the (cross-validated) negative log-likelihood score, i.e., the leave-one-out sample covariance matrix is a matrix on the correlation scale. When performing evaluation on the correlation scale the data are assumed to be standardized. If `cor = TRUE` and one wishes to used the default target specification one may consider using `target = default.target(covML(Y, cor = TRUE))`. This gives a default target under the assumption of standardized data.

Author(s)

Carel F.W. Peeters <carel.peeters@wur.nl>, Wessel N. van Wieringen

See Also

ridgeP, optPenalty.LOOCVauto, optPenalty.aLOOCV, default.target, covML

Examples

```
## Obtain some (high-dimensional) data
p = 25
n = 10
set.seed(333)
X = matrix(rnorm(n*p), nrow = n, ncol = p)
colnames(X)[1:25] = letters[1:25]

## Obtain regularized precision under optimal penalty
OPT  <- optPenalty.LOOCV(X, lambdaMin = .5, lambdaMax = 30, step = 100); OPT
OPT$optLambda # Optimal penalty
OPT$optPrec   # Regularized precision under optimal penalty

## Another example with standardized data
X <- scale(X, center = TRUE, scale = TRUE)
OPT  <- optPenalty.LOOCV(X, lambdaMin = .5, lambdaMax = 30, step = 100, cor = TRUE,
                         target = default.target(covML(X, cor = TRUE))); OPT
OPT$optLambda # Optimal penalty
OPT$optPrec   # Regularized precision under optimal penalty
```

`optPenalty.LOOCVauto` *Automatic search for optimal penalty parameter*

Description

This function is now deprecated. Please use `optPenalty.kCVauto` instead.

Usage

```
optPenalty.LOOCVauto(
  Y,
  lambdaMin,
  lambdaMax,
  lambdaInit = (lambdaMin + lambdaMax)/2,
  cor = FALSE,
  target = default.target(covML(Y)),
  type = "Alt"
)
```

Arguments

`Y`	Data `matrix`. Variables assumed to be represented by columns.
`lambdaMin`	A `numeric` giving the minimum value for the penalty parameter.
`lambdaMax`	A `numeric` giving the maximum value for the penalty parameter.
`lambdaInit`	A `numeric` giving the initial (starting) value for the penalty parameter.

cor	A logical indicating if the evaluation of the LOOCV score should be performed on the correlation scale.
target	A target matrix (in precision terms) for Type I ridge estimators.
type	A character indicating the type of ridge estimator to be used. Must be one of: "Alt", "ArchI", "ArchII".

Details

Function that performs an 'automatic' search for the optimal penalty parameter for the ridgeP call by employing Brent's method to the calculation of a cross-validated negative log-likelihood score.

The function determines the optimal value of the penalty parameter by application of the Brent algorithm (1971) to the (leave-one-out) cross-validated negative log-likelihood score (using a regularized ridge estimator for the precision matrix). The search for the optimal value is automatic in the sense that in order to invoke the root-finding abilities of the Brent method, only a minimum value and a maximum value for the penalty parameter need to be specified as well as a starting penalty value. The value at which the (leave-one-out) cross-validated negative log-likelihood score is minimized is deemed optimal. The function employs the Brent algorithm as implemented in the optim function.

Value

An object of class list:

optLambda	A numeric giving the optimal value for the penalty parameter.
optPrec	A matrix representing the precision matrix of the chosen type (see ridgeP) under the optimal value of the penalty parameter.

Note

When cor = TRUE correlation matrices are used in the computation of the (cross-validated) negative log-likelihood score, i.e., the leave-one-out sample covariance matrix is a matrix on the correlation scale. When performing evaluation on the correlation scale the data are assumed to be standardized. If cor = TRUE and one wishes to used the default target specification one may consider using target = default.target(covML(Y, cor = TRUE)). This gives a default target under the assumption of standardized data.

Author(s)

Wessel N. van Wieringen, Carel F.W. Peeters <carel.peeters@wur.nl>

References

Brent, R.P. (1971). An Algorithm with Guaranteed Convergence for Finding a Zero of a Function. Computer Journal 14: 422-425.

See Also

GGMblockNullPenalty, GGMblockTest, ridgeP, optPenalty.aLOOCV, optPenalty.LOOCV, default.target, covML

Examples

```
## Obtain some (high-dimensional) data
p = 25
n = 10
set.seed(333)
X = matrix(rnorm(n*p), nrow = n, ncol = p)
colnames(X)[1:25] = letters[1:25]

## Obtain regularized precision under optimal penalty
OPT <- optPenalty.LOOCVauto(X, lambdaMin = .001, lambdaMax = 30); OPT
OPT$optLambda # Optimal penalty
OPT$optPrec   # Regularized precision under optimal penalty

## Another example with standardized data
X <- scale(X, center = TRUE, scale = TRUE)
OPT <- optPenalty.LOOCVauto(X, lambdaMin = .001, lambdaMax = 30, cor = TRUE,
                            target = default.target(covML(X, cor = TRUE))); OPT
OPT$optLambda # Optimal penalty
OPT$optPrec   # Regularized precision under optimal penalty
```

---

pcor *Compute partial correlation matrix or standardized precision matrix*

Description

Function computing the partial correlation matrix or standardized precision matrix from an input precision matrix.

Usage

```
pcor(P, pc = TRUE)
```

Arguments

P	(Possibly regularized) precision matrix.
pc	A logical indicating if the partial correlation matrix should be computed.

Details

The function assumes that the input matrix is a precision matrix. If pc = FALSE the standardized precision matrix, rather than the partial correlation matrix, is given as the output value. The standardized precision matrix is equal to the partial correlation matrix up to the sign of off-diagonal entries.

Value

A partial correlation matrix or a standardized precision matrix.

Author(s)

Carel F.W. Peeters <carel.peeters@wur.nl>, Wessel N. van Wieringen

See Also

ridgeP, covML

Examples

```
## Obtain some (high-dimensional) data
p = 25
n = 10
set.seed(333)
X = matrix(rnorm(n*p), nrow = n, ncol = p)
colnames(X)[1:25] = letters[1:25]
Cx <- covML(X)

## Obtain regularized precision matrix
P <- ridgeP(Cx, lambda = 10, type = "Alt")

## Obtain partial correlation matrix
pcor(P)
```

---

pooledS *Compute the pooled covariance or precision matrix estimate*

Description

Compute the pooled covariance or precision matrix estimate from a `list` of covariance matrices or precision matrices.

Usage

```
pooledS(Slist, ns, subset = rep(TRUE, length(ns)), mle = TRUE)

pooledP(Plist, ns, subset = rep(TRUE, length(ns)), mle = TRUE)
```

Arguments

`Slist`	A `list` of length G of `numeric` covariance matrices of the same size.
`ns`	A `numeric` vector for length G giving the sample sizes in the corresponding entries of `Slist`
`subset`	`logical` vector of the same length as `Slist` giving the classes to pool. Default is all classes.
`mle`	`logical`. If `TRUE`, the (biased) MLE is given. If `FALSE`, the biased corrected estimate is given. Default is `TRUE`.
`Plist`	A `list` of length G of invertible `numeric` precision matrices of the same size.

Details

When `mle` is `FALSE` the given covariance/precision matrices is assumed to have been computed using the denominator `ns[i] - 1`. Hence, the sum of all `ns` minus G is used a the denominator of the pooled estimate. Conversely, when `mle` is `TRUE` the total sum of the sample sizes `ns` is used as the denominator in the pooled estimate.

The function `pooledP` is equivalent to a wrapper for `pooledS`. That is, it inverts all the precision matrices in `Plist`, applies `pooledS`, and inverts the resulting matrix.

Value

`pooledS` returns the pooled covariance matrix, that is a `numeric` matrix with the same size as the elements of `Slist`. Similarly, `pooledP` returns the pooled precision matrix, i.e. a `numeric` matrix with the same size as the elements of `Plist`.

Author(s)

Anders Ellern Bilgrau, Carel F.W. Peeters <carel.peeters@wur.nl>, Wessel N. van Wieringen

Examples

```
ns <- c(4, 6, 8)
Slist <- createS(ns, p = 6)

pooledS(Slist, ns)
pooledS(Slist, ns, mle = FALSE)

# Pool the first two classes only, leave out the remaning
pooledS(Slist, ns, subset = c(TRUE, TRUE, FALSE))
pooledS(Slist, ns, subset = ns > 5) # Pool studies with sample size > 5

# Pooled precision matrices
ns <- c(7, 8, 9)
Plist <- lapply(createS(ns, p = 6), solve)
pooledS(Plist, ns)
```

---

print.optPenaltyFusedGrid

Print and plot functions for fused grid-based cross-validation

---

Description

Print and plot functions for the output from `optPenalty.fused.grid` which performs a grid based cross-validation (CV) search to find optimal penalty parameters. Currently, only the complete penalty graph is supported.

Usage

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

## S3 method for class 'optPenaltyFusedGrid'
plot(
  x,
  add.text = TRUE,
  add.contour = TRUE,
  col = rainbow(100, end = 0.8),
  ...
)
```

Arguments

x	A optPenaltyFusedGrid-object print or plot. Usually the output of optPenalty.fused.grid.
...	Arguments passed on. In print.optPenaltyFusedGrid the arguments are passed to print.matrix. In plot.optPenaltyFusedGrid are passed to the standard plot function.
add.text	A logical value controlling if the text should be added to the plot.
add.contour	A logical value controlling if the contour lines should be added to the plot.
col	A character vector of colours used in the image plot.

Value

Invisibly returns the object (x).

Author(s)

Anders Ellern Bilgrau, Carel F.W. Peeters <carel.peeters@wur.nl>, Wessel N. van Wieringen

References

Bilgrau, A.E., Peeters, C.F.W., Eriksen, P.S., Boegsted, M., and van Wieringen, W.N. (2020). Targeted Fused Ridge Estimation of Inverse Covariance Matrices from Multiple High-Dimensional Data Classes. Journal of Machine Learning Research, 21(26): 1-52.

See Also

optPenalty.fused.grid

print.ptest *Print and summarize fusion test*

Description

Print and summary functions for the fusion test performed by fused.test.

Usage

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

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

Arguments

x, object	The object to print or summarize. Usually the output of fused.test.
digits	An integer controlling the number of printed digits.
...	Arguments passed on. In summary.ptest the arguments are passed to print.ptest. In print.ptest are passed to the standard summary function.

Value

Invisibly returns the object.

Author(s)

Anders Ellern Bilgrau, Carel F.W. Peeters <carel.peeters@wur.nl>, Wessel N. van Wieringen

References

Bilgrau, A.E., Peeters, C.F.W., Eriksen, P.S., Boegsted, M., and van Wieringen, W.N. (2020). Targeted Fused Ridge Estimation of Inverse Covariance Matrices from Multiple High-Dimensional Data Classes. Journal of Machine Learning Research, 21(26): 1-52.

See Also

fused.test, hist.ptest

Examples

```
ns <- c(10, 5, 23)
Ylist <- createS(ns, p = 15, topology = "banded", dataset = TRUE)

# Use the identity target matrix for each class
Tlist <- replicate(length(ns), diag(15), simplify = FALSE)
```

```
# Do the test
lam <- matrix(10, 3, 3)
diag(lam) <- 1
ft <- fused.test(Ylist, Tlist, lambda = lam, n.permutations = 500)
```

---

pruneMatrix *Prune square matrix to those variables having nonzero entries*

Description

Convenience function that prunes a square matrix to those variables (features) having nonzero row (column) entries (i.e., to features implied in graphical connections).

Usage

```
pruneMatrix(M)
```

Arguments

M	(Possibly sparsified) square matrix.

Value

A pruned matrix.

Author(s)

Carel F.W. Peeters <carel.peeters@wur.nl>

Examples

```
## Obtain some (high-dimensional) data
p = 25
n = 10
set.seed(333)
X = matrix(rnorm(n*p), nrow = n, ncol = p)
colnames(X)[1:25] = letters[1:25]

## Obtain regularized precision under optimal penalty
OPT <- optPenalty.LOOCV(X, lambdaMin = .5, lambdaMax = 30, step = 100)

## Determine support regularized standardized precision under optimal penalty
PC0 <- sparsify(symm(OPT$optPrec), threshold = "localFDR")$sparseParCor

## Prune sparsified partial correlation matrix
PC0P <- pruneMatrix(PC0)
```

`ridgeP` *Ridge estimation for high-dimensional precision matrices*

Description

Function that calculates various Ridge estimators for high-dimensional precision matrices.

Usage

```
ridgeP(S, lambda, type = "Alt", target = default.target(S))
```

Arguments

`S`	Sample covariance `matrix`.
`lambda`	A `numeric` representing the value of the penalty parameter.
`type`	A `character` indicating the type of ridge estimator to be used. Must be one of: "Alt", "ArchI", "ArchII".
`target`	A target `matrix` (in precision terms) for Type I ridge estimators.

Details

The function can calculate various ridge estimators for high-dimensional precision matrices. Current (well-known) ridge estimators can be roughly divided in two archetypes. The first archetypal form employs a convex combination of $\mathbf{S}$ and a positive definite (p.d.) target matrix $\mathbf{T}$: $\hat{\mathbf{\Omega}}^{\mathrm{I}}(\lambda_{\mathrm{I}}) = [(1-\lambda_{\mathrm{I}})\mathbf{S}+\lambda_{\mathrm{I}}\mathbf{T}]^{-1}$, with $\lambda_{\mathrm{I}} \in (0, 1]$. A common target choice is for $\mathbf{T}$ to be diagonal with $(\mathbf{T})_{jj} = (\mathbf{S})_{jj}$ for $j = 1, \ldots, p$. The second archetypal form can be given as $\hat{\mathbf{\Omega}}^{\mathrm{II}}(\lambda_{\mathrm{II}}) = (\mathbf{S}+\lambda_{\mathrm{II}}\mathbf{I}_p)^{-1}$ with $\lambda_{\mathrm{II}} \in (0, \infty)$. Viewed from a penalized estimation perspective, the two archetypes utilize penalties that do not coincide with the matrix-analogue of the common ridge penalty. van Wieringen and Peeters (2015) derive analytic expressions for alternative Type I and Type II ridge precision estimators based on a proper L2-penalty. Their alternative Type I estimator (target shrinkage) takes the form

$$\hat{\mathbf{\Omega}}^{\mathrm{I}a}(\lambda_a) = \left\{ \left[\lambda_a \mathbf{I}_p + \frac{1}{4}(\mathbf{S} - \lambda_a \mathbf{T})^2 \right]^{1/2} + \frac{1}{2}(\mathbf{S} - \lambda_a \mathbf{T}) \right\}^{-1},$$

while their alternative Type II estimator can be given as a special case of the former:

$$\hat{\mathbf{\Omega}}^{\mathrm{II}a}(\lambda_a) = \left\{ \left[\lambda_a \mathbf{I}_p + \frac{1}{4}\mathbf{S}^2 \right]^{1/2} + \frac{1}{2}\mathbf{S} \right\}^{-1}.$$

These alternative estimators were shown to be superior to the archetypes in terms of risk under various loss functions (van Wieringen and Peeters, 2015).

The `lambda` parameter in `ridgeP` generically indicates the penalty parameter. It must be chosen in accordance with the type of ridge estimator employed. The domains for the penalty parameter in the archetypal estimators are given above. The domain for `lambda` in the alternative estimators is $(0, \infty)$. The `type` parameter specifies the type of ridge estimator. Specifying `type = "ArchI"` leads to usage of the archetypal I estimator while specifying `type = "ArchII"` leads to usage of

the archetypal II estimator. In the latter situation the argument `target` remains unused. Specifying `type = "Alt"` enables usage of the alternative ridge estimators: when `type = "Alt"` and the `target` matrix is p.d. one obtains the alternative Type I estimator; when `type = "Alt"` and the `target` matrix is specified to be the null-matrix one obtains the alternative Type II estimator.

The Type I estimators thus employ target shrinkage. The default target for both the archetype and alternative is `default.target(S)`. When `target` is not the null-matrix it is expected to be p.d. for the alternative type I estimator. The target is always expected to be p.d. in case of the archetypal I estimator. The archetypal Type I ridge estimator is rotation equivariant when the target is of the form $\mu\mathbf{I}_p$ with $\mu \in (0, \infty)$. The archetypal Type II estimator is rotation equivariant by definition. When the target is of the form $\varphi\mathbf{I}_p$ with $\varphi \in [0, \infty)$, then the alternative ridge estimator is rotation equivariant. Its analytic computation is then particularly speedy as the (relatively) expensive matrix square root can then be circumvented.

Value

Function returns a regularized precision `matrix`.

Author(s)

Carel F.W. Peeters <carel.peeters@wur.nl>, Anders E. Bilgrau

References

van Wieringen, W.N. & Peeters, C.F.W. (2016). Ridge Estimation of Inverse Covariance Matrices from High-Dimensional Data, Computational Statistics & Data Analysis, vol. 103: 284-303. Also available as arXiv:1403.0904v3 [stat.ME].

van Wieringen, W.N. & Peeters, C.F.W. (2015). Application of a New Ridge Estimator of the Inverse Covariance Matrix to the Reconstruction of Gene-Gene Interaction Networks. In: di Serio, C., Lio, P., Nonis, A., and Tagliaferri, R. (Eds.) 'Computational Intelligence Methods for Bioinformatics and Biostatistics'. Lecture Notes in Computer Science, vol. 8623. Springer, pp. 170-179.

See Also

`default.target`

Examples

```
set.seed(333)

## Obtain some (high-dimensional) data
p <- 25
n <- 10
X <- matrix(rnorm(n*p), nrow = n, ncol = p)
colnames(X)[1:p] = letters[1:p]
Cx <- covML(X)

## Obtain regularized precision matrix
P <- ridgeP(Cx, lambda = 10, type = "Alt")
summary(P)
print(P)
```

```
plot(P)
```

`ridgeP.fused` *Fused ridge estimation*

Description

Performs fused ridge estimation of multiple precision matrices in cases where multiple classes of data is present for given a penalty matrix.

Usage

```
ridgeP.fused(
  Slist,
  ns,
  Tlist = default.target.fused(Slist, ns),
  lambda,
  Plist,
  maxit = 100L,
  verbose = TRUE,
  relative = TRUE,
  eps = sqrt(.Machine$double.eps)
)
```

Arguments

`Slist`	A `list` of length G of covariance matrices, i.e. square, symmetric `numeric` matrices of the same size. The gth matrix should correspond to the gth class.
`ns`	A `numeric` vector of sample sizes on which the matrices in `Slist` are based. I.e. `ns[g]` correspond to `Slist[[g]]`.
`Tlist`	A `list` of length G of `numeric` p.d. target matrices corresponding to the matrices in `Slist`. If not supplied, the default is given by `default.target`.
`lambda`	The G by G penalty matrix. That is, a symmetric, non-negative `numeric matrix` of size G times G giving the class- and pair-specific penalties. The diagonal entries are the class specific ridge penalties. I.e. `lambda[i, i]` is the ridge penalty for class i. The off-diagonal entries are the pair-specific fusion penalties. I.e. `lambda[i, j]` is the fusion penalty applied on the pair of classes i and j. Alternatively, can be supplied as a `numeric` of length 1 or 2. If a single number, a diagonal penalty with lambda in the diagonal is used If supplied as a `numeric` vector of two numbers, the first is used as a common ridge penalty and the second as a common fusion penalty.
`Plist`	An optional `list` of initial precision matrices for the fused ridge algorithm the same size as `Slist`. Can be omitted. Default is the nonfused ridge precision estimate using the pooled covariance matrix corresponding to setting all fusion penalties to zero.

maxit	A single integer giving the maximum number of allowed iterations. Can be set to Inf. If maxit is hit, a warning is given.
verbose	logical. Set to TRUE for extra output.
relative	logical indicating if the convergence criterion should be on a relative scale.
eps	A single positive numeric giving the convergence threshold.

Details

Performs a coordinate ascent to find the maximum likelihood of the fused likelihood problem for a given ridge penalty $lambda$ and fused penalty matrix $Lambda_f$.

Value

Returns a list as Slist with precision estimates of the corresponding classes.

Note

For extreme fusion penalties in lambda the algorithm is quite sensitive to the initial values given in Plist.

Author(s)

Anders Ellern Bilgrau, Carel F.W. Peeters <carel.peeters@wur.nl>, Wessel N. van Wieringen

References

Bilgrau, A.E., Peeters, C.F.W., Eriksen, P.S., Boegsted, M., and van Wieringen, W.N. (2020). Targeted Fused Ridge Estimation of Inverse Covariance Matrices from Multiple High-Dimensional Data Classes. Journal of Machine Learning Research, 21(26): 1-52.

See Also

default.penalty
ridgeP for the regular ridge estimate

Examples

```
# Create some (not at all high-dimensional) data on three classes
p <- 5  # Dimension
ns <- c(4, 6, 8)  # Sample sizes (K = 3 classes)
Slist <- createS(ns, p = p)
str(Slist, max.level = 2)  # The structure of Slist

#
# Estimate the precisions (using the complete penalty graph)
#

res1 <- ridgeP.fused(Slist, ns, lambda = c(1.3, 2.1))
print(res1)
```

```
# The same using the penalty matrix (the diagnal is ignored)
mylambda  <- matrix(c(1.3, 2.1, 2.1,
                      2.1, 1.3, 2.1,
                      2.1, 2.1, 1.3), 3, 3, byrow = TRUE)
res2 <- ridgeP.fused(Slist, ns, lambda = mylambda)
stopifnot(all.equal(res1, res2))


#
# Estimate the precisions (using a non-complete penalty graph)
#

# Say we only want to shrink pairs (1,2) and (2,3) and not (1,3)
mylambda[1,3] <- mylambda[3,1] <- 0
print(mylambda)
res3 <- ridgeP.fused(Slist, ns, lambda = mylambda)
# which similar to, but not the same as res1 and res2.


#
# Using other custom target matrices
#

# Construct a custom target list
myTlist <- list(diag(p), matrix(1, p, p), matrix(0, p, p))
res4 <- ridgeP.fused(Slist, ns, Tlist = myTlist, lambda = c(1.3, 2.1))
print(res4)

# Alternative, see ?default.target.fused
myTlist2 <- default.target.fused(Slist, ns, type = "Null")  # For the null target
res5 <- ridgeP.fused(Slist, ns, Tlist = myTlist2, lambda = c(1.3, 2.1))
print(res5)
```

`ridgePathS` *Visualize the regularization path*

Description

Function that visualizes the regularization paths of the nonredundant elements of a regularized precision matrix against the (range of the) penalty parameter.

Usage

```
ridgePathS(
  S,
  lambdaMin,
  lambdaMax,
  step,
  type = "Alt",
```

```
  target = default.target(S),
  plotType = "pcor",
  diag = FALSE,
  vertical = FALSE,
  value,
  verbose = TRUE
)
```

Arguments

S	Sample covariance matrix.
lambdaMin	A numeric giving the minimum value for the penalty parameter.
lambdaMax	A numeric giving the maximum value for the penalty parameter.
step	An integer determining the number of steps in moving through the grid [lambdaMin, lambdaMax].
type	A character indicating the type of ridge estimator to be used. Must be one of: "Alt", "ArchI", "ArchII".
target	A target matrix (in precision terms) for Type I ridge estimators.
plotType	A character indicating the type of element for which a visualization of the regularization paths is desired. Must be one of: "pcor", "cor", "cov", "prec".
diag	A logical indicating if the diagonal elements should be retained for visualization.
vertical	A logical indicating if output graph should come with a vertical line at a pre-specified value for the penalty parameter.
value	A numeric indicating a pre-specified value for the penalty parameter.
verbose	A logical indicating if information on progress should be printed on screen.

Details

The function visualizes the regularization path of the individual elements of a regularized precision matrix against the penalty parameter. The range of the penalty parameter is given by [lambdaMin,lambdaMax]. The penalty parameter must be positive such that lambdaMin must be a positive scalar. The maximum allowable value of lambdaMax depends on the type of ridge estimator employed. For details on the type of ridge estimator one may use (one of: "Alt", "ArchI", "ArchII") see ridgeP.

Regularization paths may be visualized for (partial) correlations, covariances and precision elements. The type of element for which a visualization of the regularization paths is desired can be indicated by the argument plotType. When vertical = TRUE a vertical line is added at the constant value. This option can be used to assess whereabouts the optimal penalty obtained by, e.g., the routines optPenalty.LOOCV or optPenalty.aLOOCV, finds itself along the regularization path.

Author(s)

Wessel N. van Wieringen, Carel F.W. Peeters <carel.peeters@wur.nl>

See Also

ridgeP, covML, optPenalty.LOOCV, optPenalty.aLOOCV, default.target

Examples

```
## Obtain some (high-dimensional) data
p = 25
n = 10
set.seed(333)
X = matrix(rnorm(n*p), nrow = n, ncol = p)
colnames(X)[1:25] = letters[1:25]
Cx <- covML(X)

## Visualize regularization paths
ridgePathS(Cx, .001, 50, 200, plotType = "pcor")
```

---

ridgeS *Ridge estimation for high-dimensional precision matrices*

Description

This function is now deprecated. Please use `ridgeP` instead.

Usage

```
ridgeS(S, lambda, type = "Alt", target = default.target(S))
```

Arguments

`S`	Sample covariance `matrix`.
`lambda`	A `numeric` representing the value of the penalty parameter.
`type`	A `character` indicating the type of ridge estimator to be used. Must be one of: "Alt", "ArchI", "ArchII".
`target`	A target `matrix` (in precision terms) for Type I ridge estimators.

Details

See `ridgeP`.

Value

Function returns a regularized precision `matrix`.

Author(s)

Carel F.W. Peeters <carel.peeters@wur.nl>, Wessel N. van Wieringen

See Also

`ridgeP`

Examples

```
## Obtain some (high-dimensional) data
p = 25
n = 10
set.seed(333)
X = matrix(rnorm(n*p), nrow = n, ncol = p)
colnames(X)[1:25] = letters[1:25]
Cx <- covML(X)

## Obtain regularized precision matrix
ridgeS(Cx, lambda = 10, type = "Alt")
```

`rmvnormal` *Multivariate Gaussian simulation*

Description

Fast simulation from multivariate Gaussian probability distribution.

Usage

```
rmvnormal(n, mu, sigma)
```

Arguments

`n`	An `integer` giving the number of observations to be simulated.
`mu`	A `numeric` vector of dimension p giving the means of normal distribution.
`sigma`	A variance-covariance `matrix` of dimension p times p.

Details

The `rmvnormal` function is copied from the `GMCM`-package. It is similar to `rmvnorm` from the `mvtnorm`-package.

Value

Returns a n by p matrix of observations from a multivariate normal distribution with the given mean `mu` and covariance

Author(s)

Anders Ellern Bilgrau

Examples

```
rmvnormal(n = 10, mu = 1:4, sigma = diag(4))
```

`sparsify` *Determine the support of a partial correlation/precision matrix*

Description

Function that determines the support of a partial correlation/precision matrix by thresholding and sparsifies it accordingly.

Usage

```
sparsify(
  P,
  threshold = c("absValue", "connected", "localFDR", "top"),
  absValueCut = 0.25,
  FDRcut = 0.9,
  top = 10,
  output = "heavy",
  verbose = TRUE
)
```

Arguments

`P`	(Possibly regularized) precision `matrix`.
`threshold`	A `character` signifying type of sparsification by thresholding. Must be one of: "absValue", "connected", "localFDR", "top".
`absValueCut`	A `numeric` giving the cut-off for partial correlation element selection based on absolute value thresholding.
`FDRcut`	A `numeric` giving the cut-off for partial correlation element selection based on local false discovery rate (FDR) thresholding.
`top`	A `numeric` specifying the exact number of partial correlation elements to retain based on absolute value.
`output`	A `character` specifying the type of output required. Must be one of: "heavy", "light".
`verbose`	A `logical` indicating if intermediate output should be printed on screen.

Details

The function transforms the possibly regularized input precision matrix to a partial correlation matrix. Subsequently, the support of this partial correlation matrix is determined. Support determination is performed either by simple thresholding on the absolute values of matrix entries (`threshold = "absValue"`) or by usage of local FDR (`threshold = "localFDR"`). A third option is to retain a prespecified number of matrix entries based on absolute values. For example, one could wish to retain those entries representing the ten strongest absolute partial correlations (`threshold = "top"`). As a variation on this theme, a fourth option (`threshold = "connected"`) retains the top edges such that the resulting graph is connected (this may result in dense graphs in practice). The argument

absValueCut is only used when threshold = "absValue". The argument top is only used when threshold = "top". The argument FDRcut is only used when threshold = "localFDR".

The function is to some extent a wrapper around certain fdrtool functions when threshold = "localFDR". In that case a mixture model is fitted to the nonredundant partial correlations by fdrtool. The decision to retain elements is then based on the argument FDRcut. Elements with a posterior probability $\geq$ FDRcut (equalling 1 - local FDR) are retained. See Schaefer and Strimmer (2005) for further details on usage of local FDR in graphical modeling.

Value

If the input P is a standardized precision (or partial correlation) matrix the function returns a sparsified precision (or partial correlation) matrix whenever output = "heavy". If the input P is an unstandardized precision matrix the function returns an object of class list whenever output = "heavy":

sparseParCor	A matrix representing the sparsified partial correlation matrix.
sparsePrecision	A matrix representing the sparsified precision matrix.

When output = "light", only the (matrix) positions of the zero and non-zero elements are returned in an object of class list:

zeros	A matrix representing the row and column positions of zero entries.
nonzeros	A matrix representing the row and column positions of non-zero entries.

Author(s)

Carel F.W. Peeters <carel.peeters@wur.nl>, Wessel N. van Wieringen

References

Schaefer, J., and Strimmer, K. (2005). A shrinkage approach to large-scale covariance estimation and implications for functional genomics. Statistical Applications in Genetics and Molecular Biology, 4:32.

See Also

ridgeP, optPenalty.aLOOCV, optPenalty.LOOCV

Examples

```
## Obtain some (high-dimensional) data
p = 25
n = 10
set.seed(333)
X = matrix(rnorm(n*p), nrow = n, ncol = p)
colnames(X)[1:25] = letters[1:25]

## Obtain regularized precision under optimal penalty
OPT <- optPenalty.LOOCV(X, lambdaMin = .5, lambdaMax = 30, step = 100)
```

```
## Determine support regularized (standardized) precision under optimal penalty
sparsify(OPT$optPrec, threshold = "localFDR")
```

`sparsify.fused` *Determine support of multiple partial correlation/precision matrices*

Description

A simple wrapper for `sparsify` which determines the support of a `list` of partial correlation/precision matrix by various methods and returns the sparsified matrices.

Usage

```
sparsify.fused(Plist, ...)
```

Arguments

`Plist`	A `list` of `numeric` precision matrices.
`...`	Arguments passed to `sparsify`.

Value

A `list` of the same length as `Plist` with the output from `sparsify`.

Author(s)

Anders Ellern Bilgrau, Wessel N. van Wierigen, Carel F.W. Peeters <carel.peeters@wur.nl>

See Also

`sparsify`

Examples

```
ns <- c(10, 11)
Ylist <- createS(ns, p = 16, dataset = TRUE)
Slist <- lapply(Ylist, covML)
Tlist <- default.target.fused(Slist, ns)

# Obtain regularized precision under optimal penalty
opt <- optPenalty.fused.auto(Ylist, Tlist, cv.method = "aLOOCV",
                             maxit.ridgeP.fused = 1500)
# Use the optimal penalties
Plist <- ridgeP.fused(Slist, ns, lambda = opt$lambda, maxit = 1000)

# Determine support regularized (standardized) precision under optimal penalty
res <- sparsify.fused(Plist, threshold = "top", verbose = FALSE)
round(res[[1]]$sparsePrecision, 1)
round(res[[2]]$sparsePrecision, 1)
```

symm *Symmetrize matrix*

Description

Function that symmetrizes matrices.

Usage

```
symm(M)
```

Arguments

M	(In numeric ideality symmetric) square matrix.

Details

Large objects that are symmetric sometimes fail to be recognized as such by R due to rounding under machine precision. This function symmetrizes for computational purposes matrices that are symmetric in numeric ideality.

Value

A symmetric matrix.

Author(s)

Carel F.W. Peeters <carel.peeters@wur.nl>, Wessel N. van Wieringen

Examples

```
## Obtain some (high-dimensional) data
p = 25
n = 10
set.seed(333)
X = matrix(rnorm(n*p), nrow = n, ncol = p)
colnames(X)[1:25] = letters[1:25]

## Obtain regularized precision under optimal penalty
OPT <- optPenalty.LOOCV(X, 10, 30, 10, target = diag(diag(1/covML(X))))

## Check symmetry
## OPT$optPrec is symmetric by definition
## But is not recognized as such due to rounding peculiarities
isSymmetric(OPT$optPrec)

## Symmetrize
symm(OPT$optPrec)
```

Ugraph *Visualize undirected graph*

Description

Function that visualizes the sparsified precision matrix as an undirected graph.

Usage

```
Ugraph(
  M,
  type = c("plain", "fancy", "weighted"),
  lay = "layout_in_circle",
  coords = NULL,
  Vsize = 15,
  Vcex = 1,
  Vcolor = "orangered",
  VBcolor = "darkred",
  VLcolor = "black",
  prune = FALSE,
  legend = FALSE,
  label = "",
  Lcex = 1.3,
  PTcex = 4,
  cut = 0.5,
  scale = 10,
  pEcolor = "black",
  nEcolor = "grey",
  main = ""
)
```

Arguments

M	(Possibly sparsified) precision matrix
type	A character indicating the type of graph to be produced. Must be one of: "plain", "fancy", "weighted".
lay	A character mimicking a call to igraph layout functions. Determines the placement of vertices.
coords	A matrix containing coordinates. Alternative to the lay-argument for determining the placement of vertices.
Vsize	A numeric determining the vertex size.
Vcex	A numeric determining the size of the vertex labels.
Vcolor	A character (scalar or vector) determining the vertex color.
VBcolor	A character determining the color of the vertex border.
VLcolor	A character determining the color of the vertex labels.

prune	A logical determining if vertices of degree 0 should be removed.
legend	A logical indicating if the graph should come with a legend.
label	A character giving a name to the legend label.
Lcex	A numeric determining the size of the legend box.
PTcex	A numeric determining the size of the exemplary vertex in the legend box.
cut	A numeric indicating the cut-off for indicating strong edges when type = "fancy".
scale	A numeric representing a scale factor for visualizing strength of edges when type = "weighted".
pEcolor	A character determining the color of the edges tied to positive precision elements. Only when type = "weighted".
nEcolor	A character determining the color of the edges tied to negative precision elements. Only when type = "weighted".
main	A character giving the main figure title.

Details

The intended use of this function is to visualize a sparsified precision/partial correlation matrix as an undirected graph. When type = "plain" a plain undirected graph is given representing the conditional (in)dependencies exemplified by the sparsified precision.

When type = "fancy" a more elaborate graph is given in which dashed lines indicate negative partial correlations while solid lines indicate positive partial correlations, and in which grey lines indicate strong edges. Strong edges are deemed such by setting cut. If a the absolute value of a precision element $\geq$ cut the corresponding edge is deemed strong and colored grey in the graph. The argument cut is thus only used when type = "fancy".

When type = "weighted" an undirected graph is given in which edge thickness represents the strength of the partial correlations. The nEcolor colored edges then represent negative partial correlations while pEcolor colored edges represent positive partial correlations. (Relative) edge thickness in this type of graph can be set by the argument scale. The arguments scale, nEcolor, and pEcolor are thus only used when type = "weighted".

The default layout gives a circular placement of the vertices. Most layout functions supported by igraph are supported (the function is partly a wrapper around certain igraph functions). The igraph layouts can be invoked by a character that mimicks a call to a igraph layout functions in the lay argument. When using lay = NULL one can specify the placement of vertices with the coords argument. The row dimension of this matrix should equal the number of (pruned) vertices. The column dimension then should equal 2 (for 2D layouts) or 3 (for 3D layouts). The coords argument can also be viewed as a convenience argument as it enables one, e.g., to layout a graph according to the coordinates of a previous call to Ugraph. If both the the lay and the coords arguments are not NULL, the lay argument takes precedence

The legend allows one to specify the kind of variable the vertices represent, such as, e.g., mRNA transcripts. The arguments label, Lcex, and PTcex are only used when legend = TRUE.

If prune = TRUE the vertices of degree 0 (vertices not implicated by any edge) are removed. For the colors supported by the arguments Vcolor, VBcolor, VLcolor, pEcolor, and nEcolor see https://www.nceas.ucsb.edu/sites/default/files/2020-04/colorPaletteCheatsheet.pdf.

Value

The function returns a graph. The function also returns a `matrix` object containing the coordinates of the vertices in the given graph.

Author(s)

Carel F.W. Peeters <carel.peeters@wur.nl>

References

Csardi, G. and Nepusz, T. (2006). The igraph software package for complex network research. InterJournal, Complex Systems 1695. http://igraph.sf.net

van Wieringen, W.N. & Peeters, C.F.W. (2016). Ridge Estimation of Inverse Covariance Matrices from High-Dimensional Data, Computational Statistics & Data Analysis, vol. 103: 284-303. Also available as arXiv:1403.0904v3 [stat.ME].

van Wieringen, W.N. & Peeters, C.F.W. (2015). Application of a New Ridge Estimator of the Inverse Covariance Matrix to the Reconstruction of Gene-Gene Interaction Networks. In: di Serio, C., Lio, P., Nonis, A., and Tagliaferri, R. (Eds.) 'Computational Intelligence Methods for Bioinformatics and Biostatistics'. Lecture Notes in Computer Science, vol. 8623. Springer, pp. 170-179.

See Also

`ridgeP`, `optPenalty.LOOCV`, `optPenalty.aLOOCV`, `sparsify`

Examples

```
## Obtain some (high-dimensional) data
p = 25
n = 10
set.seed(333)
X = matrix(rnorm(n*p), nrow = n, ncol = p)
colnames(X)[1:25] = letters[1:25]

## Obtain regularized precision under optimal penalty
OPT <- optPenalty.LOOCV(X, lambdaMin = .5, lambdaMax = 30, step = 100)

## Determine support regularized standardized precision under optimal penalty
PC0 <- sparsify(symm(OPT$optPrec), threshold = "localFDR")$sparseParCor

## Obtain graphical representation
Ugraph(PC0, type = "fancy", cut = 0.07)

## Obtain graphical representation with Fruchterman-Reingold layout
Ugraph(PC0, type = "fancy", lay = "layout_with_fr", cut = 0.07)

## Add pruning
Ugraph(PC0, type = "fancy", lay = "layout_with_fr",
       cut = 0.07, prune = TRUE)

## Obtain graph and its coordinates
```

```
Coordinates <- Ugraph(PC0, type = "fancy", lay = "layout_with_fr",
                      cut = 0.07, prune = TRUE)
Coordinates
```

`Union` *Subset 2 square matrices to union of variables having nonzero entries*

Description

Convenience function that subsets 2 square matrices (over the same features) to the union of features that have nonzero row (column) entries (i.e., features implied in graphical connections).

Usage

```
Union(M1, M2)
```

Arguments

`M1`	(Possibly sparsified) square `matrix`.
`M2`	(Possibly sparsified) square `matrix` over the same features as `M1`.

Details

Say you have 2 class-specific precision matrices that are estimated over the same variables/features. For various reasons (such as, e.g., the desire to visualize pruned class-specific networks in the same coordinates) one may want to prune these matrices to those features that are implied in graphical connections in at least 1 class.

Value

An object of class list:

`M1subset`	A pruned `matrix` for class 1.
`M2subset`	A pruned `matrix` for class 2.

Author(s)

Carel F.W. Peeters <carel.peeters@wur.nl>

See Also

`Ugraph`

Examples

```
## Invoke data
data(ADdata)

## Subset
ADclass1 <- ADmetabolites[, sampleInfo$ApoEClass == "Class 1"]
ADclass2 <- ADmetabolites[, sampleInfo$ApoEClass == "Class 2"]

## Transpose data
ADclass1 <- t(ADclass1)
ADclass2 <- t(ADclass2)

## Correlations for subsets
rAD1 <- cor(ADclass1)
rAD2 <- cor(ADclass2)

## Simple precision estimates
P1 <- ridgeP(rAD1, 2)
P2 <- ridgeP(rAD2, 2)
Plist = list(P1 = P1, P2 = P2)

## Threshold matrices
Mats <- sparsify.fused(Plist, threshold = "top", top = 20)

## Prune sparsified partial correlation matrices
## To union of features implied by edge
MatsPrune <- Union(Mats$P1$sparseParCor, Mats$P2$sparseParCor)
```

Index

∗ **datasets**
ADdata, 6

ADdata, 6
adjacentMat, *4*, 7
ADmetabolites *(*ADdata*)*, 6

CNplot, *4*, 8, *14*
Communities, 11
conditionNumberPlot, 13, *33*
covML, *4*, *7*, *10*, 15, *17*, *23*, *28*, *30, 31*, *40, 41*, *54*, *57*, *61*, *69*, *71*, *73*, *75*, *77*, *87*
covMLknown, *4*, 16
createS, *5*, 17

default.penalty, *5*, 20, *64, 65*, *85*
default.target, *4*, *10*, *21*, 22, *24, 25*, *36*, *39*, *52*, *60, 61*, *69*, *71*, *73*, *75*, *83, 84*, *87*
default.target.fused, *5*, 24, *52*
DiffGraph, 25

edgeHeat, *4*, *7*, 27
evaluateS, *4*, 29
evaluateSfit, *4*, 30

fullMontyS, *4*, 32
fused.test, *5*, 34, *48*, *80*

getKEGGPathway, *5*, *52*
GGMblockNullPenalty, *4*, 35, *38, 39*, *71*, *75*
GGMblockTest, *4*, *35, 36*, 37, *71*, *75*
GGMmutualInfo, *4*, 39
GGMnetworkStats, *4*, *33*, 40, *42, 43*
GGMnetworkStats.fused, *5*, 42
GGMpathStats, *4*, 43, *47*
GGMpathStats.fused, *5*, 47

hist.ptest, 48, *80*

igraph, *11, 12*, *26*, *44, 45*, *94, 95*
is.Xlist, *5*, 49
isSymmetric, *51*
isSymmetricPD, *5*, 50
isSymmetricPSD *(*isSymmetricPD*)*, 50

kegg.target, *5*, 51
KLdiv, *4*, 53, *55*
KLdiv.fused, *5*, 54

loss, *4*, 56

momentS, 57

NLL, *5*, 58

optim, *64*
optPenalty.aLOOCV, *4*, *10*, 59, *69*, *71*, *73*, *75*, *87*, *91*, *96*
optPenalty.fused, *4*, *21*, *50*
optPenalty.fused *(*optPenalty.fused.grid*)*, 62
optPenalty.fused.auto, *20, 21*
optPenalty.fused.grid, 62, *78, 79*
optPenalty.kCV, *4*, 67, *71*
optPenalty.kCVauto, *4*, *69*, 70
optPenalty.LOOCV, *10*, *60, 61*, 72, *75*, *87*, *91*, *96*
optPenalty.LOOCVauto, *33*, *36*, *39*, *46*, *61*, *73*, 74

pcor, *4*, 76
plot.optPenaltyFusedGrid *(*print.optPenaltyFusedGrid*)*, 78
plot.ptest, *5*
plot.ptest *(*hist.ptest*)*, 48
PNLL *(*NLL*)*, 58
pooledP *(*pooledS*)*, 77
pooledS, *5*, 77
print.optPenaltyFusedGrid, *5*, 78
print.ptest, *5*, *48*, 80
pruneMatrix, 81

rags2ridges *(*rags2ridges-package*)*, 3
rags2ridges-package, 3
rcond, *9*
ridgeP, *3, 4*, *7*, *9, 10*, *15*, *22, 23*, *28*, *30, 31*, *33*, *36*, *38–41*, *46*, *54*, *56, 57*, *59–61*, *67–71*, *73*, *75*, *77*, 82, *85*, *87, 88*, *91*, *96*
ridgeP.fused, *4*, *21*, *34, 35*, *50*, *59*, 84
ridgePathS, *4*, 86
ridgeS, *61*, 88
rmvnormal, *5*, 89

sampleInfo *(*ADdata*)*, 6
sparsify, *4*, *7*, *28*, *33*, *41*, *46*, 90, *92*, *96*
sparsify.fused, *5*, 92
summary.ptest *(*print.ptest*)*, 80
symm, *4*, 93

Ugraph, *4*, *7*, *12*, *27*, *33*, *41*, 94, *97*
Union, 97

variableInfo *(*ADdata*)*, 6