Package 'pdglasso'

September 8, 2026

Type Package

Title Graphical Lasso for Coloured Gaussian Graphical Models for Paired Data

Version 2.0.1

Description Implements methods for coloured Gaussian graphical models with equality ``R"estrictions on ``CON"centration values (RCON models; Højsgaard and Lauritzen (2008) <doi:10.1111/j.1467-9868.2008.00666.x>) for paired data (pdRCON models). Provides an Alternating Direction Method of Multipliers (ADMM) algorithm for solving the penalized likelihood problem introduced by Ranciati and Roverato (2024) <doi:10.1007/s11222-024-10513-6>. Provides functions for computing maximum likelihood estimates and generating simulated pdRCON models and datasets.

License GPL (>= 3)

Imports Rcpp (>= 1.0.13), MASS

LinkingTo Rcpp, RcppEigen

Encoding UTF-8

LazyData true

Config/roxygen2/version 8.0.0

NeedsCompilation yes

Author Saverio Ranciati [aut] (ORCID: <https://orcid.org/0000-0001-7880-9465>), Alberto Roverato [aut] (ORCID: <https://orcid.org/0000-0001-7984-3593>), Anna Vesely [aut, cre] (ORCID: <https://orcid.org/0000-0001-6696-2390>), Peter Majcen [aut] (ORCID: <https://orcid.org/0009-0009-7020-7246>)

Maintainer Anna Vesely <anna.vesely2@unibo.it>

Repository CRAN

Date/Publication 2026-09-08 13:30:09 UTC

Contents

`pdglasso-package` *pdglasso: Graphical Lasso for Coloured Gaussian Graphical Models for Paired Data*

Description

Implements methods for coloured Gaussian graphical models with equality "R"estrictions on "CON"centration values (RCON models; Højsgaard and Lauritzen (2008) doi:10.1111/j.14679868.2008.00666.x) for paired data (pdRCON models). Provides an Alternating Direction Method of Multipliers (ADMM) algorithm for solving the penalized likelihood problem introduced by Ranciati and Roverato (2024) doi:10.1007/s11222024105136. Provides functions for computing maximum likelihood estimates and generating simulated pdRCON models and datasets.

Details

An RCON model for paired data (pdRCON model) is a coloured Gaussian Graphical Model (GGM) where the p variables are partitioned into a Left block L and a Right block R. The two blocks are not independent, every variable in the left block has an homologous variable in the right block and certain types of equality "R"estrictions on the entries of the "CON"centration matrix K are allowed. A pdRCON model is represented by a Coloured Graph for Paired Data (pdColG) with a vertex for every variable and where every vertex and edge is either *coloured* or *uncoloured*. More details on the equality constraints of pdRCON models, on submodel classes of interest and on the usage of the package are given in the following.

pdRCON models - terminology and relevant submodel classes

A pdRCON model is a Gaussian Graphical Model (GGM) with additional equality restrictions on the entries of the concentration matrix. In the paired data framework, there are three different types of equality restrictions of interest, identified by the names *vertex*, *inside-block edge* and *across-block edge*, respectively. Relevant submodel classes can be specified both by allowing different combinations of restriction types and by forcing different types of fully symmetric structures. In this package, different submodel classes are identified by the arguments `type` and `force.symm` and models are represented by coloured graphs for paired data encoded in the form of a pdColG matrix.

- Every variable in L has an homologous variable in R and the corresponding diagonal entries of K can be constrained to have equal value. Such entries of K are represented by *coloured vertices* of the independence graph whereas the unconstrained diagonal entries are represented by *uncoloured vertices*. Different types of submodel classes of interest may be obtained by (i) not allowing coloured vertices, (ii) allowing both coloured and uncoloured vertices and (iii) allowing only coloured vertices.
- For every pair of variables in L there exists an homologous pair of variables in R, thereby identifying a pair of homologous edges. If both edges are present in the graph the corresponding off-diagonal entries of K can be constrained to have equal value. These type of edges are referred to as *coloured inside-block edges*. Different types of submodel classes of interest may be obtained by (i) not allowing coloured inside-block edges, (ii) allowing both coloured and uncoloured inside-block edges and (iii) allowing only coloured inside-block edges.
- We say that two variables are *across-block* if one variable belongs to L and the other to R. For every pair of non-homologous across-block variables there exists an homologous pair across-block variables, thereby identifying a pair of homologous edges. If both edges are present in the graph the corresponding off-diagonal entries of K can be constrained to have equal value. These type of edges are referred to as *coloured across-block edges*. Different types of submodel classes of interest may be obtained by (i) not allowing coloured across-block edges, (ii) allowing both coloured and uncoloured across-block edges and (iii) allowing only coloured across-block edges, with the exception of edges joining a variable in L with its homologous in R.

A pair of homologous edges which are both present in the graph form what we call a *structural symmetry*. A structural symmetry can be either *uncoloured* or *coloured* and the latter is also called a *parametric symmetry*. Accordingly, a pair of homologous coloured vertices is called a *vertex symmetry*. It is worth remarking that, strictly speaking, a pair of homologous missing edges may be regarded as a parametric symmetry because the associated concentrations have the same, zero, value. However, we deem it convenient to use this terminology only to refer to present edges.

Use of the arguments type and force.symm to specify submodel classes

The functions of this package make it possible to specify different types of pdRCON submodel classes of interest through the arguments `type` and `force.symm` which can both take as value any subvector of the character vector `c("vertex", "inside.block.edge", "across.block.edge")`; note that the names of the components can be abbreviated down, up to the first letter only, and are not case-sensitive. The argument `type` cannot be `NULL` and:

- If `type` contains the string `"vertex"` then vertex symmetries are allowed and, if in addition also `force.symm` contains the string `"vertex"`, then all vertices are coloured.
- If `type` contains the string `"inside.block.edge"` then coloured inside-block symmetries are allowed and, if in addition also `force.symm` contains the string `"inside.block.edge"`, then only coloured edges are allowed inside blocks.
- If `type` contains the string `"across.block.edge"` then coloured across-block symmetries are allowed and, if in addition also `force.symm` contains the string `"across.block.edge"`, then only coloured edges are allowed across blocks, with the exception of edges joining a variable in L with its homologous in R.

Note that `force.symm` is a, possibly `NULL`, subvector of `type`. Elements of `force.symm` which are not elements of `type` are ignored.

Variables position and block structure of sample covariance matrices

The functions of this package assume that the positions occupied by variables follow certain rules. More specifically, the positions from 1 to $q = p/2$ correspond to the first group of variables, say L, whereas the positions from $q + 1$ to p are associated with the second group of variables, R. Furthermore, the variables of the first group are ordered in the same way as those of the second group, in the sense that for every $i = 1, \ldots, q$ the variable in position i is homologous to the variable in position $q + i$. Hence, for instance, the functions that receive in input a sample covariance matrix assume that the rows and columns of this matrix are ordered according to these rules, so that it can be partitioned into four $q \times q$ submatrices naturally associated with the inside- and across-block components. The same is true for the `pdColG` matrix described below.

Model representation through a pdColG matrix

Every pdRCON model is uniquely represented by a Coloured Graph for Paired Data (pdColG) implemented in the form of an object of class `pdColG` that is a $p \times p$ symmetric matrix where every entry is one of the values 0, 1 or 2, as follows:

- The diagonal entries of a pdColG matrix are all equal to either 1, for uncoloured vertices, or 2, for coloured vertices.
- The off-diagonal entries of a pdColG matrix are equal to 0 for missing edges and either 1 or 2 for present edges, where the value 2 is used to encode coloured edges.

Author(s)

Saverio Ranciati (ORCID: 0000-0001-7880-9465), Alberto Roverato (ORCID: 0000-0001-7984-3593), Anna Vesely (ORCID: 0000-0001-6696-2390), and Peter Majcen (ORCID: 0009-0009-7020-7246)
Maintainer: Anna Vesely anna.vesely2@unibo.it

References

Højsgaard, S. and Lauritzen, S. L. (2008). Graphical Gaussian models with edge and vertex symmetries. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, *70*(5), 1005-1027.

Ranciati, S. and Roverato, A., (2024). On the application of Gaussian graphical models to paired data problems. *Statistics and Computing*, *34*(6), 1-19.

Ranciati, S., Roverato, A. and Luati, A. (2021). Fused graphical lasso for brain networks with symmetries. *Journal of the Royal Statistical Society Series C: Applied Statistics*, *70*(5), 1299-1322.

Roverato, A. and Nguyen, D.N. (2024). Exploration of the search space of Gaussian graphical models for paired data. *Journal of Machine Learning Research*, *25*(92), 1-41.

admm.pdglasso *ADMM graphical lasso algorithm for pdRCON models*

Description

Estimate of a concentration matrix within the pdRCON submodel class identified by the arguments `type` and `force.symm`, based on the pdglasso method with penalty terms `lambda1` and `lambda2` (Ranciati and Roverato, 2024). Optimization is implemented by an ADMM algorithm (see Boyd *et. al.* 2011). The output is an object of class `ADMMoutput` and the user may then call the function `pdColG.get` to obtain the Coloured Graph for Paired Data (pdColG) representing the selected model.

Usage

```
admm.pdglasso(
  S,
  lambda1,
  lambda2,
  type = c("vertex", "inside.block.edge", "across.block.edge"),
  force.symm = NULL,
  X.init = NULL,
  rho1 = 1,
  rho2 = 1,
  varying.rho1 = TRUE,
  varying.rho2 = TRUE,
  max_iter = 5000,
  eps.abs = 1e-06,
  eps.rel = 1e-06,
  rcpp = TRUE,
  verbose = FALSE
)
```

Arguments

`S`	a sample covariance matrix with the block structure described in `pdglasso-package`.
`lambda1`, `lambda2`	two non-negative scalar penalties that encourage sparsity (`lambda1`) and parametric symmetries (`lambda2`) in the concentration matrix.
`type`, `force.symm`	two subvectors of `c("vertex", "inside.block.edge", "across.block.edge")` which identify the pdRCON submodel class of interest; see `pdglasso-package` for details.
`X.init`	a matrix of the same dimension as `S` to be used as starting point of the ADMM. If `NULL` a default value is used.
`rho1`, `rho2`	two positive scalars; tuning parameters of the outer and inner loop, respectively, of the ADMM.

`varying.rho1`, `varying.rho2`
: two logicals; if `TRUE` the parameters `rho1` and `rho2`, respectively, are updated iteratively to speed-up convergence.

`max_iter`
: an integer; maximum number of iterations for convergence.

`eps.abs`, `eps.rel`
: two positive scalars; the absolute and relative tolerance, respectively, for the computation of primal and dual feasibility tolerances of the ADMM.

`rcpp`
: a logical; if `TRUE`, computations are performed using the Rcpp (C++) implementation; if `FALSE`, a pure (slower) R implementation is used.

`verbose`
: a logical; if `TRUE` the pdRCON submodel class considered, as specified by the arguments `type` and `force.symm`, is returned as printed output on the console.

Value

A object of class `ADMMoutput` that is a list with the following components:

- `X` the estimated concentration matrix.
- `acronyms` a vector of strings identifying the pdRCON submodel class considered as identified by the arguments `type` and `force.symm`.
- `internal.par` a list of internal parameters passed to the function at the call, as well as convergence information.

References

Ranciati, S. and Roverato, A., (2024). On the application of Gaussian graphical models to paired data problems. *Statistics and Computing*, *34*(6), 1-19.

Boyd, S., Parikh, N., Chu, E., Borja, P. and Eckstein, J. (2011). Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine learning*, *3*(1), 1-122.

Examples

```
# computation of the sample covariance
S <- cov(toy_data$sample.data)

# model with all types of symmetries allowed and no full symmetry required
admm.out <- admm.pdglasso(S, lambda1=4, lambda2=0.7)
G <- pdColG.get(admm.out)
summary(G)

# model with no across-block symmetries allowed and full vertex-symmetry required
admm.out <- admm.pdglasso(S, lambda1=4, lambda2=0.7, type=c("v", "i"),
            force.symm = c("v"), verbose=TRUE)
G <- pdColG.get(admm.out)
summary(G)
```

bc_data *Breast Cancer dataset*

Description

The paired samples in this dataset refer to individuals with both tumor and healthy adjacent tissue measurements, in the form of a set of genes of the Hedgehog Pathway and their gene-level transcription estimates, i.e. transformed normalized counts. Please check Section 8 of Ranciati and Roverato (2024) for more information.

Usage

```
bc_data
```

Format

`bcdata`**:**
A named 114×178 matrix, where:

- each row refers to one of the 114 individuals in the dataset;
- each column is one of the $p = 89 \times 2 = 178$ genes of the Hedgehog Pathway, whose expressions were measured on healthy (first half) and adjacent tumor affected samples (second half).

compute.eBIC *Extended Bayesian Information Criterion (eBIC) for pdRCON models*

Description

This function computes the value of the eBIC for a pdRCON model from the output of `admm.pdglasso`; see Eq.1 of Feygel and Drton (2010).

Usage

```
compute.eBIC(S, admm.out, n, gamma.eBIC = 0.5, mle = TRUE)
```

Arguments

`S`	a sample covariance matrix with the block structure described in `pdglasso-package`.
`admm.out`	an object of class `ADMMoutput`, such as the output of a call to `admm.pdglasso`; see also `pdRCON.select`.
`n`	a positive integer; the sample size.
`gamma.eBIC`	a value between zero and one; the magnitude of the penalization term inside the criterion, where 0 makes eBIC equivalent to BIC and 0.5 is the value suggested by Foygel and Drton (2010).
`mle`	a logical; if `TRUE`, the MLE are used otherwise the pdglasso estimator is used; see the "Details" section below.

Details

Details on the specification of the argument mle

The extended Bayesian Information Criterion implemented in this function requires that the parameters are estimated by maximum likelihood. However, the computation of MLEs may considerably slow down the procedure and, more seriously, in the high-dimensional setting MLEs may even not exist. For this reason, we provide the option that the eBIC is, naively, computed by using the pdglasso estimate.

Value

A vector containing three elements:

- the value of the eBIC,
- the value of the log-likelihood,
- the number of parameters.

References

Foygel, R. and Drton, M. (2010). Extended Bayesian information criteria for Gaussian graphical models. *Advances in neural information processing systems*, 23.

Examples

```
S <- cov(toy_data$sample.data)
admm.out <- admm.pdglasso(S, lambda1=4, lambda2=0.7)
compute.eBIC(S, admm.out, n=60, gamma.eBIC=0.5)

compute.eBIC(S, admm.out, n=60, gamma.eBIC=0.5, mle=FALSE)
```

---

fMRI_parietal *fMRI dataset*

Description

This dataset contains residuals obtained through score-driven models on the time-series of brain activity at rest for an individual, measured on 10 regions of interest (ROI) of the brain in the parietal area (five on the left hemisphere and five paired regions on the right hemisphere). Please check Sections 2 and 7 of Ranciati and Roverato (2021) for more information.

Usage

```
fMRI_parietal
```

Format

`fMRI_parietal`:

A 404×10 dataframe with 404 observations referring to residuals from $p = 5 \times 2 = 10$ time-series after filtering through a score-driven model. The residuals are associated to 10 different regions of interest of the parietal area of the brain, five on the left empisphere e the paired homologous regions on the right hemisphere.

`GGM.simulate` *Random simulation of Gaussian graphical models (GGMs)*

Description

Randomly generates a GGM and, more specifically, the undirected graph `G` representing the model, a concentration matrix `K` adapted to `G`, that is a positive definite matrix whose zero pattern is specified by the missing edges of `G`, and a random sample from a multivariate normal distribution with zero mean vector and covariance matrix `Sigma`, that is the inverse of `K`.

Usage

```
GGM.simulate(
  p,
  concent.mat = TRUE,
  sample = TRUE,
  Sigma = NULL,
  sample.size = NULL,
  dens = 0.1
)
```

Arguments

`p`	an even integer; the number of variables of the generated model; see the "Details" section below.
`concent.mat`	a logical; if `TRUE` a concentration matrix `K` adapted to `G` is generated.
`sample`	a logical; if `TRUE` a sample from a normal distribution with zero mean vector and concentration matrix `K` is generated.
`Sigma`	a pxp positive definite matrix; the matrix argument of the function `rWishart`, which is used as starting point in the random generation of the concentration matrix, as described in the "Details" section of the function `pdRCON.simulate`. If `NULL` the identity matrix is used.
`sample.size`	a positive integer; the size of the randomly generated sample. If `NULL` then the sample size is set to 3xp.
`dens`	a value between zero and one used to specify the sparsity degree of the generated graph, as described in the "Details" section of the function `pdRCON.simulate`.

Details

A GGM is a pdRCON model with no parametric symmetries, and the purpose of this function is that of providing a simplified call to the function pdRCON.simulate, with the appropriate choice of arguments required to simulate a GGM. Note, however, that pdRCON models make sense only if the number of variables p is even, and this requirement is (unnecessarily) retained here.

Value

A list with the following components:

- G a randomly generated (symmetric) adjacency matrix of an undirected graph on p vertices.
- K a randomly generated concentration matrix adapted to G if concent.mat=TRUE, or NULL otherwise.
- sample.data a data frame randomly generated from a multivariate normal distribution with mean vector zero and concentration matrix K if both sample=TRUE and concent.mat=TRUE, or NULL otherwise.

Note that the variable are named V1,...,Vp.

Examples

```
# generates distribution and data from a GGM on 20 variables and graph density equal to 0.2
p <- 20
n <-100
set.seed(1234)
GenMod <- GGM.simulate(p, sample.size=n, dens=0.20)

# check graph sparsity degree
n.edges <- sum(GenMod$G[upper.tri(GenMod$G)])
n.edges/(p*(p-1)/2)

# check positive definiteness
min(eigen(GenMod$K)$values)

# computation of the partial correlation matrix
R <- -cov2cor(GenMod$K)
diag(R) <- 1
```

---

lams.max *Maximum theoretical values of lambda1 and lambda2*

Description

Computes the maximum theoretical values lambda1.max and lambda2.max of lambda1 and lambda2, respectively. For every lambda1 greater than lambda1.max the pdglasso estimator is a diagonal matrix whereas for every lambda2 greater than lambda2.max the pdglasso estimator is fully symmetric.

Usage

```
lams.max(S)
```

Arguments

S	a sample covariance matrix.

Value

A vector of two elements, `lambda1.max` and `lambda2.max`.

Examples

```
S <- cov(toy_data$sample.data)
lams.max(S)
```

---

pdColG.get *pdColG matrix from the output of a call to* admm.pdglasso

---

Description

This function returns the pdColG matrix representing the pdRCON model resulting from a call to admm.pdglasso; see also pdRCON.select.

Usage

```
pdColG.get(admm.out, th1 = NULL, th2 = NULL, model.dimension = FALSE)
```

Arguments

admm.out	an object of class ADMMoutput.
th1, th2	two positive scalars; the thresholds to identify missing edges (th1) and coloured edges (th2) in the graph, if NULL a default value is used; see the "Details" section below.
model.dimension	a logical, if TRUE the number of parameters of the model is returned.

Details

Details of the specification of the argument th1 and th2

Due to finite-precision arithmetic in computing, the fitted concentration matrix obtained from the ADMM has no exact zeros and no exact identical pairs of concentration values. Hence, th1 and th2 provide tolerances for values and differences, respectively, to be regarded as close enough to zero. NULL values are replaced by default thresholds automatically obtained from tolerance values used to check convergence of the ADMM. Note that the function plot.ADMMoutput allows one to visualize the role played by the default threshold as well as to facilitate the specification of suitable values for the personalized thresholds th1 and th2.

Value

Either an object of class pdColG, if model.dimension=FALSE, or a list with the following components if model.dimension=TRUE:

- pdColG an object of class pdColG, that is a matrix representing a coloured graph for paired data.
- n.par the number of parameters of the pdRCON model.

Examples

```
S <- cov(toy_data$sample.data)
admm.out <- admm.pdglasso(S, lambda1=4, lambda2=0.7)
pdColG.get(admm.out)
```

---

pdRCON.mle *Maximum likelihood estimate of a pdRCON model*

Description

Computes the maximum likelihood estimate of the concentration matrix of a pdRCON model.

Usage

```
pdRCON.mle(S, pdColG, eps.rel = 1e-06, eps.abs = 1e-06, max_iter = 5000)
```

Arguments

S	a sample covariance matrix with the block structure described in pdglasso-package.
pdColG	a pdColG matrix representing a coloured graph for paired data; see pdglasso-package for details.
eps.abs, eps.rel	two positive scalars; the absolute and relative tolerance, respectively, for the computation of primal and dual feasibility tolerances of the ADMM.
max_iter	an integer; maximum number of iterations for convergence.

Details

If the sample covariance matrix is not full-rank, then it is possible that the maximum likelihood estimate does not exist. The maximum likelihood estimate is computed by running the function admm.pdglasso with suitable penalties and, if it does not exist, then the ADMM algorithm fails to converge. In this case, a warning is produced and a NULL is returned.

Value

Either a matrix, that is the maximum likelihood estimate of $K = \Sigma^{-1}$ under the pdRCON model represented by pdColG, or NULL if the maximum likelihood estimate does not exist.

Examples

```
S <- var(toy_data$sample.data)
K.hat <- pdRCON.mle(S, toy_data$pdColG)
```

`pdRCON.select` *Selection and estimate of a pdRCON model according to eBIC*

Description

Performs a sequence of calls to `admm.pdglasso` on a sequence of values for `lambda1` and `lambda2`. More specifically, firstly a path of `lambda1` values, with `lambda2=0`, is considered to select a "best" `lambda1` value according to eBIC. Next, a path of `lambda2` values is considered, with `lambda1` fixed to its previously selected value, and the final model is selected according to eBIC. The user can then call `pdColG.get` to obtain the Coloured Graph for Paired Data (pdColG) representing the selected model.

Usage

```
pdRCON.select(
  S,
  n,
  lams = NULL,
  gamma.eBIC = 0.5,
  type = c("vertex", "inside.block.edge", "across.block.edge"),
  force.symm = NULL,
  X.init = NULL,
  rho1 = 1,
  rho2 = 1,
  varying.rho1 = TRUE,
  varying.rho2 = TRUE,
  max_iter = 5000,
  eps.abs = 1e-08,
  eps.rel = 1e-08,
  verbose = FALSE,
  mle.estimate = TRUE
)
```

Arguments

`S`	a sample covariance matrix with the block structure described in `pdglasso-package`.
`n`	the sample size.
`lams`	a 2x4 numeric matrix that specifies the path of lambda values, if `NULL` default values are used, as explained in the "Details" section below.
`gamma.eBIC`	a value between zero and one; the magnitude of the penalization term of the eBIC; see `compute.eBIC` for details.

`type`, `force.symm`
: two subvectors of `c("vertex", "inside.block.edge", "across.block.edge")` which identify the pdRCON submodel class of interest; see `pdglasso-package` for details.

`X.init`
: a matrix of the same dimension as `S` to be used as starting point of the ADMM. If `NULL` a default value is used.

`rho1`, `rho2`
: two positive scalars; tuning parameters of the outer and inner loop, respectively, of the ADMM.

`varying.rho1`, `varying.rho2`
: two logicals; if `TRUE` the parameters `rho1` and `rho2`, respectively, are updated iteratively to speed-up convergence.

`max_iter`
: an integer; maximum number of iterations for convergence.

`eps.abs`, `eps.rel`
: two positive scalars; the absolute and relative tolerance, respectively, for the computation of primal and dual feasibility tolerances of the ADMM.

`verbose`
: a logical; if `TRUE` both a visual update about the grid search over `lambda1` and `lambda2` and the pdRCON submodel class considered are returned as printed output on the console.

`mle.estimate`
: a logical; if `TRUE`, compute eBIC via the MLE, if `FALSE` the pdglasso estimator is used; see `compute.eBIC` for details.

Details

Details on the specification of the argument lams

The argument `lams` is a 2x4 matrix used to specified the grid of penalty terms. The first row of `lams` refers to `lambda1` and second row to `lambda2`. For each row of the `lams` matrix, entries are: the minimum and maximum value of the path (columns 1 and 2); the number of points of the path (column 3); `0` for linear spacing and `1` logarithmic spacing (column 4). If `lams=NULL` the default values are computed as follows: maximum lambda values are computed from `lams.max` and paths of 20 points are used form max.lams/20 to max.lams, with logarithm spacing.

Value

A list with the following components:

- `model` selected model; object of class `ADMMoutput` output of `admm.pdglasso`.
- `lambda.grid` the grid of values used for `lambda1` and `lambda2`.
- `best.lambdas` the selected values of `lambda1` and `lambda2` according to eBIC.
- `l1.path` a matrix containing the path values for `lambda1` as well as relevant quantities of the eBIC.
- `l2.path` a matrix containing the path values for `lambda2` as well as relevant quantities of the eBIC.
- `time.exec` total execution time for the called function.

A warning is produced if at least one run of the algorithm for the grid searches has resulted in non-convergence (status can be checked by inspecting `l1.path` and `l2.path`); see the "Details" section of the function `compute.eBIC`.

Examples

```
S <- cov(toy_data$sample.data)
sel.mod <- pdRCON.select(S, n=60, verbose=TRUE)
sel.mod$l1.path
sel.mod$l2.path
plot(sel.mod$model)
pdColG.get(sel.mod$model)
```

---

pdRCON.simulate *Random simulation of pdRCON models*

Description

Randomly generates a pdRCN model and, more specifically, the pdColG matrix representing the model, a concentration matrix K with the same zero pattern and equality constraints encoded by pdColG, and a random sample from a multivariate normal distribution with zero mean vector and covariance matrix Sigma, that is the inverse of K.

Usage

```
pdRCON.simulate(
  p,
  concent.mat = TRUE,
  sample = TRUE,
  Sigma = NULL,
  sample.size = NULL,
  type = c("vertex", "inside.block.edge", "across.block.edge"),
  force.symm = NULL,
  dens = 0.1,
  dens.vertex = NULL,
  dens.inside = NULL,
  dens.across = NULL,
  verbose = TRUE
)
```

Arguments

p	an even integer; the number of variables of the generated model.
concent.mat	a logical; if TRUE a concentration matrix K with the zero structure and symmetries encoded by pdColG is generated.
sample	a logical; if TRUE a sample from a normal distribution with zero mean vector and concentration matrix K is generated.
Sigma	a pxp positive definite matrix; the matrix argument of the rWishart function, which is used as starting point in the random generation of the concentration matrix, as described in the "Details" section below. If NULL the identity matrix is used.

`sample.size` a positive integer; the size of the randomly generated sample. If `NULL` then the sample size is set to 3xp.

`type`, `force.symm`
two subvectors of `c("vertex", "inside.block.edge", "across.block.edge")` which identify the pdRCON submodel class of interest; see `pdglasso-package` for details.

`dens`, `dens.vertex`, `dens.inside`, `dens.across`
four values between zero and one used to specify the sparsity degree of the generated graph, as explained in the "Details" section below. The default `dens.vertex=NULL` is equivalent to `dens.vertex=dens`, and similarly for `dens.inside` and `dens.across`.

`verbose` a logical; if `TRUE` the pdRCON submodel class considered, as specified by the arguments `type` and `force.symm`, is returned as printed output in the console.

Details

Details on the sparsity degree of the generated graph

The argument `dens.vertex` specifies the proportion of coloured vertices among the p vertices. This is used if the string "vertex" is a component of `type` but not of `force.symm`. The string "vertex" not being a component of `type` is equivalent to `dens.vertex=0` whereas the string "vertex" being a component of both `type` and `force.symm` is equivalent to `dens.vertex=1`.

The argument `dens.inside` specifies the proportion of coloured symmetric inside-block edges among the $q(q-1)$ inside-block edges, where $q = p/2$, . This is used if the string "inside.block.edge" is a component of `type`, otherwise it is equivalent to `dens.inside=0`. The overall density of inside-block edges is obtained by the sum of the densities of coloured and uncoloured inside-block edges. This is a value between `dens.inside` and `dens.inside+dens`. Furthermore, it is exactly equal to `dens` if "inside.block.edge" is not a component of `type` and to `dens.inside` if "inside.block.edge" is a component of both `type` and `force.symm`.

The argument `dens.across` specifies the proportion of coloured symmetric across-block edges among the potentially coloured $q(q-1)$ across-block edges. This is used if the string "across.block.edge" is a component of `type`, otherwise it is equivalent to `dens.across=0`. The overall density of across-block edges is obtained by the sum of the densities of coloured and uncoloured across-block edges. This is a value between `dens.across` and `dens.across+dens`. Furthermore, it is exactly equal to `dens` if "across.block.edge" is not a component of `type`.

The argument `dens` specifies the density of uncoloured edges. Note that the algorithm generates uncoloured edges first, which may be overwritten by coloured edges. For this reason the actual density of uncoloured edges is typically smaller than `dens`.

Details on the generating process of the concentration matrix

The concentration matrix is obtained by first generating a random Wishart matrix with matrix parameter `Sigma` and p degrees of freedom, which represents an initial unconstrained covariance matrix. This is inverted and adapted to a suitable coloured graph for paired data with sparsity degree according to the `dens.xxx` arguments.

Value

A list with the following components:

- pdColG a randomly generated a matrix of class pdColG representing a pdRCON model; see pdglasso-package for details.
- K a randomly generated concentration matrix with the zero structure and symmetries encoded by pdColG if concent.mat=TRUE, or NULL otherwise.
- sample.data a data frame randomly generated from a multivariate normal distribution with mean vector zero and concentration matrix K if both sample=TRUE and concent.mat=TRUE, or NULL otherwise.

Note that the variable in L are named L1,...,Lq and variables in R are are named R1,...,Rq where Li is homologous to Ri for every i=1,...,q.

Examples

```
# generates a pdRCON model on 10 variables in the form of a pdColG matrix

set.seed(123)
rpdColG <- pdRCON.simulate(10, concent=FALSE, sample=FALSE, dens=0.25)$pdColG
rpdColG

# generates a distribution from a pdRCON model on 20 variables, a concentration matrix
# for this model and a sample of size 50
# all vertices are coloured and no coloured across-block edge is allowed

set.seed(123)
GenMod <- pdRCON.simulate(20, type=c("v", "i"), force.symm=c("v"), sample.size=50, dens=0.20)

summary(GenMod$pdColG)

# computation of the partial correlation matrix
R <- -cov2cor(GenMod$K)
diag(R) <- 1
```

---

plot.ADMMoutput *Diagnostic plot for the output of the ADMM*

Description

A plot, with different panels, obtained from the output of admm.pdglasso, which is helpful to check the convergence of the ADMM. This function can also be used to visualize the default threshold used by pdColG.get as well to identify specific th1 and th2 threshold values to be used in place of the default one.

Usage

```
## S3 method for class 'ADMMoutput'
plot(
  x,
  y = NULL,
```

```
  add.default.th = TRUE,
  th1 = NULL,
  th2 = NULL,
  logarithm10 = TRUE,
  ...
)
```

Arguments

x	an object of class ADMMoutput such as the output of a call to the admm.pdglasso function; see also pdRCON.select.
y	not used. Included for compatibility with the generic plot() method.
add.default.th	a logical; if TRUE the default threshold used in pdColG.get is represented (in log10-scale) in the plot.
th1, th2	two scalars as in pdColG.get; if not NULL they are represented (in log10-scale) in the plot.
logarithm10	a logical; if TRUE the values of th1 and th2 are expected to be provided in log10-scale. This facilitate interaction with the plot whose y-axis is in log10 scale.
...	not used. Included for compatibility with the generic plot() method.

Value

A plot is produced with different panels depending on the model type. More specifically:

- a panel with the values of the off-diagonal concentrations is always provided,
- a panel with the differences of homologous diagonal concentration values is provided if vertex symmetries are allowed,
- a panel with the differences of homologous inside-block concentration values is provided if inside-block symmetries are allowed,
- a panel with the differences of homologous across-block concentration values is provided if across-block symmetries are allowed

According to the values of the arguments add.default.th, th1 and th2, horizontal dashed lines with the corresponding thresholds are included in the panels.

The function returns an invisible vector with the values of th1, th2 and of the default threshold. The quantities th1 and th2 can directly be given as input for the same arguments of pdColG.get. Consistently with the behavior of pdColG.get, a NULL value of th1 or th2 is replaced by the default threshold.

Examples

```
S <- cov(toy_data$sample.data)
mod.out <- admm.pdglasso(S, lambda1=4, lambda2=0.3)

# full plot with four panels
plot(mod.out)
```

```
# model without across-block symmetries
mod.out <- admm.pdglasso(S, lambda1=4, lambda2=0.3, type=c("v", "i"))
plot(mod.out)
```

---

plot.pdColG *Visual representation of a coloured graph for paired data*

Description

Heatmap-style graphical representation of an object of class pdColG, such as that returned by a call to pdColG.get.

Usage

```
## S3 method for class 'pdColG'
plot(x, y = NULL, uncol.sym = FALSE, col.sym = FALSE, ...)
```

Arguments

x	a matrix of class pdColG; see pdglasso-package for details.
y	not used. Included for compatibility with the generic plot() method.
uncol.sym	a logical; if TRUE, structural symmetries are highlighted in the heatmap.
col.sym	a logical; if TRUE, coloured structural symmetries are highlighted in the heatmap.
...	unused. Included for compatibility with the generic plot() method.

Value

A plot visualizing the coloured graph enconded in the input matrix 'x'. Each square of the heatmap can either be empty (white) for the diagonal and missing edges or represent a type of edge:

- a darkgrey square represents an edge in the graph;
- a deepskyblue2 square represents an uncoloured structural symmetry;
- a purple2 square represents either a vertex symmetry or a coloured structural symmetry.

Examples

```
S <- var(toy_data$sample.data)
mod.out <- admm.pdglasso(S, lambda1=4, lambda2=0.6)
G <- pdColG.get(mod.out)
plot(G)
plot(G, uncol.sym=TRUE)
plot(G, uncol.sym=TRUE, col.sym=TRUE)
```

summary.pdColG *Structural properties of a coloured graph for paired data*

Description

Summary statistics relative to the structural properties of an object of class pdColG, such as that raturned by a call to pdColG.get.

Usage

```
## S3 method for class 'pdColG'
summary(object, verbose = TRUE, ...)
```

Arguments

object	a matrix of class pdColG; see pdglasso-package for details.
verbose	a logical; if TRUE the summary is printed on the console.
...	not used. Included for compatibility with the generic summary() method.

Value

An invisible list with the following components:

- overall a list with the number of vertices, edges and free parameters of the associated pdRCON model.
- vertex a list with the number of coloured vertices.
- inside a list with the number of inside-block edges, the number of uncolored symmetric and coloured inside block edges.
- across a list with the number of across-block edges, the number of uncolored symmetric and coloured across block edges.

Examples

```
summary(toy_data$pdColG)

model.summary <- summary(toy_data$pdColG, verbose=FALSE)
model.summary
```

toy_data *Toy dataset generated by a call to the* pdRCON.simulate *function*

Description

Data simulated by the function pdRCON.simulate with parameters:

- p=20
- type=c("v", "i","a")
- force.symm=NULL
- dens=0.5
- Sigma a $p \times p$ matrix with unitary diagonal and 0.5 on off-diagonal elements.

Usage

```
toy_data
```

Format

toy_data**:**

A list with the following components:

- pdColG, a matrix of class pdColG; see pdglasso-package for details.
- K, a concentration matrix adapted to pdColG.
- sample.data, a data frame with 250 rows and 20 columns from a multivariate normal distribution with zero mean vector and concentration matrix K.

Index