Package 'pdglasso'

June 16, 2023

Type Package

Title Graphical Lasso for Coloured Gaussian Graphical Models for Paired Data

Version 0.1.0

Description This package deals with RCON models for paired data and implements an Alternating Directions Method of Multipliers (ADMM) algorithm to solve a penalized likelihood method. Also functions for the computation of maximum likelihood estimates and for the generation of simulated pdRCON models and data are provided.

License GPL (>= 2)

Encoding UTF-8

LazyData true

RoxygenNote 7.2.3

ByteCompile TRUE

Roxygen list(markdown = TRUE)

Url https://github.com/savranciati/pdglasso

Depends R (>= 2.10)

Imports MASS

R topics documented:

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

Description

This package deals with RCON models for paired data and implements an Alternating Directions Method of Multipliers (ADMM) algorithm to solve a penalized likelihood method. Also functions for the computation of maximum likelihood estimates and for the generation of simulated pdRCON models and data are provided.

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 Restrictions on the entries of the CONcentration 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, 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 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 with 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 symmetric 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 symmetric 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.

- We remark that every coloured edge belongs to a pair of coloured symmetric edges, either inside or across blocks. On the other hand, for an uncolored edge its homologous edge may or may not be present in the graph. In the case where an uncoloured edge and its homologous are both present we say that they form a pair of *uncoloured symmetric edges*, either inside or across blocks.

Use of the arguments `type` and `force.symm` to specify a submodel class

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 coloured 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 edge 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 edge 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 group components.

Model representation through the `pdColG` matrix

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

- The diagonal entries of the pdColG matrix are all equal to either 1, for uncoloured vertices, or 2, for coloured vertices.
- The off-diagonal entries of the 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) and Alberto Roverato (ORCID: 0000-0001-7984-3593)
Maintainer: Saverio Ranciati saverio.ranciati2@unibo.it

References

Ranciati, S., Roverato, A., 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. https://doi.org/10.1111/rssc.12514

Højsgaard, S., 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 https://doi.org/10.1111/j.1467-9868.2008.00666.x

---

`admm.pdglasso` *ADMM graphical lasso algorithm for coloured GGMs for paired data.*

Description

By providing a covariance matrix `S` and values for `lambda1` and `lambda2`, this function estimates a concentration matrix X within the family of pdRCON submodel class identified by the arguments `type` and `force.symm`, using an (adaptive) ADMM algorithm. The output is the matrix and a list of internal parameters used by the function, together with the specific call with the relevant pdRCON submodel class.

Usage

```
admm.pdglasso(
  S,
  lambda1 = 1,
  lambda2 = 1e-04,
  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,
  verbose = FALSE,
  print.type = TRUE
)
```

Arguments

`S`	a sample covariance (or correlation) matrix with the block structure described in `pdglasso-package`.
`lambda1`	a non-negative scalar (or vector) penalty that encourages sparsity in the concentration matrix.

`lambda2` a non-negative scalar (or vector) penalty that encourages equality constraints 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` (optional) a $p \times p$ initial guess for the concentration matrix and/or starting solution for the ADMM algorithm.

`rho1` a scalar; tuning parameter of the ADMM algorithm to be used for the outer loop. It must be strictly positive.

`rho2` a scalar; tuning parameter of the ADMM algorithm to be used for the inner loop. It must be strictly positive.

`varying.rho1` a logical; if `TRUE` the parameter rho1 is updated iteratively to speed-up convergence.

`varying.rho2` a logical; if `TRUE` the parameter rho2 is updated iteratively to speed-up convergence.

`max_iter` an integer; maximum number of iterations to be run in case the algorithm does not converge.

`eps.abs` a scalar; the absolute precision required for the computation of primal and dual residuals of the ADMM algorithm.

`eps.rel` a scalar; the relative precision required for the computation of primal and dual residuals of the ADMM algorithm.

`verbose` a logical; if `TRUE` the progress (and internal convergence of inner loop) is shown in the console while the algorithm is running.

`print.type` 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.

Value

A list with the following components:

- `X` the estimated concentration matrix under the pdRCON submodel class considered and the values of `lambda1` and `lambda2`.
- `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.

Examples

```
S <- cov(toy_data$sample.data)
admm.pdglasso(S)
```

`compute.eBIC` *Compute the extended Bayesian Information Criterion (eBIC).*

Description

The function computes the value of the eBIC for a given model and gamma value, for the purpose of model selection (see Eq.1 of Feygel & Drton, 2010).

Usage

```
compute.eBIC(S, mod, n, gamma.eBIC = 0.5, max_iter = 5000)
```

Arguments

`S`	a sample covariance (or correlation) matrix with the block structure described in `pdglasso-package`.
`mod`	a list, the output object of a call to `admm.pdglasso`
`n`	the sample size of the data used to compute the sample covariance matrix S.
`gamma.eBIC`	a parameter governing the magnitude of the penalization term inside the criterion; it ranges from 0 to 1, where 0 makes the eBIC equivalent to BIC, and 0.5 being the suggested default value.
`max_iter`	an integer; maximum number of iterations to be run in case the algorithm does not converge; passed to `pdRCON.mle`.

Value

A vector containing three elements:

- the value of the eBIC,
- the log-likelihood,
- and the estimated number of degrees of freedom.

References

Foygel, R., Drton, M. (2010). Extended Bayesian information criteria for Gaussian graphical models. *Advances in neural information processing systems*, 23. https://proceedings.neurips.cc/paper/2010/file/072b030ba126b2f4b2374f342be9ed44-Paper.pdf

Examples

```
S <- cov(toy_data$sample.data)
mod <- admm.pdglasso(S, lambda1=1, lambda2=0.5)
compute.eBIC(S,mod,n=60,gamma.eBIC=0.5)
```

max.lams *Compute maximum theoretical values for* lambda1 *and* lambda2.

Description

Computes the maximum values for lambda1 and lambda2 such that:

- if max of lambda1 is used, the estimated concentration matrix will be diagonal;
- if max of lambda2 is used, the estimated concentration matrix will be fully symmetric.

Usage

```
## S3 method for class 'lams'
max(S)
```

Arguments

S	a covariance matrix.

Value

a vector of two elements.

Examples

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

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

Description

This function returns a Coloured Graph for Paired Data (pdColG) from the output of a call to admm.pdglasso. We refer to pdglasso-package both for the description of the matrix encoding a coloured graph for paired and the available submodel classes.

Usage

```
pdColG.get(admm.out, th1 = NULL, th2 = NULL, print.summary = FALSE)
```

Arguments

admm.out	An object of list type, that is the output of a call to the admm.pdglasso function.
th1	(optional) A scalar, the threshold to identify edges in the graph; it must be non-negative.
th2	(optional) A scalar, the threshold to identify coloured edges in the graph; it must be non-negative.
print.summary	(optional) if TRUE provides summary statistics of the graph.

Value

a list with the following components:

- pdColG a matrix representing a coloured graph for paired data; see pdglasso-package for details.
- n.par the number of parameters of the pdRCON model represented by pdColg.

Examples

```
S <- cov(toy_data$sample.data)
mod.out <- admm.pdglasso(S)
pdColG.get(mod.out)
```

---

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

Description

This function returns some summary statistics relative to the structural properties of a coloured graph for paired data $\mathcal{G}$. We refer to pdglasso-package both for the description of the matrix encoding a coloured graph for paired and details on how the structural quantities are defined and computed.

Usage

```
pdColG.summarize(pdColG, print.summary = TRUE)
```

Arguments

pdColG	a matrix representing a coloured graph for paired data; see pdglasso-package for details.
print.summary	a logical (default TRUE) indicating whether a summary should be printed.

Value

An invisible list with the following components:

- overall a list with the number of vertices and edges of $\mathcal{G}$.
- vertex a list with the number of coloured vertices of $\mathcal{G}$.
- inside a list with the number of inside block edges, the number of uncolored symmetric and coloured inside block edges of $\mathcal{G}$.
- across a list with the number of across block edges, the number of uncolored symmetric and coloured across block edges of $\mathcal{G}$.

If print.summary=TRUE some summary statistics are also printed on the console

Examples

```
#
pdColG.summarize(toy_data$pdColG)
```

pdRCON.check *Check orders of magnitude of entries of the estimated concentration matrix X under the pdRCON submodel class considered.*

Description

This function produces different plots of values of X in log10 scale, depending on the submodel class identified by the acronym stored in the input object `mod.out`. The user might want to call this function to identify what values to pass as arguments `th1` and `th2` to a call to the function `pdColGet`.

Usage

```
pdRCON.check(mod.out)
```

Arguments

`mod.out`	An object of list type, that is the output of a call to the `admm.pdglasso` function. This can also be the output of a call to `pdRCON.fit`.

Value

Depending on the acronym stored inside the input object, one or more plots depicting:

- off-diagonal elements,
- vertices,
- inside block,
- across block.

Examples

```
S <- cov(toy_data$sample.data)
mod.out <- pdRCON.fit(S,n=60)
pdRCON.check(mod.out)
```

pdRCON.fit *Fit and select a coloured GGM for paired data according to eBIC criterion.*

Description

Performs a sequence of calls to `admm.pdglasso` providing two grids of values for `lambda1` and `lambda2`. First, a grid search conditional on `lambda2=0` is run to select the best `lambda1` value among the candidates (according to eBIC); conditional on the best `lambda1`, a similar search is performed for `lambda2`. The output is the select model, given by the estimated concenration matrix. The user may call `pdColG.get` to obtain the corresponding Coloured Graph for Paired Data (pdColG) from the selected model.

Usage

```
pdRCON.fit(
  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,
  progress = TRUE,
  print.type = TRUE
)
```

Arguments

S	a sample covariance (or correlation) matrix with the block structure described in pdglasso-package.
n	the sample size of the data used to compute the sample covariance matrix S.
lams	a 2x3 matrix; first row refers to lambda1 and second row to lambda2; for each row, values are (i) minimum value for the grid; (ii) maximum value for the grid; (iii) number of points in the grid; if NULL defaulta values are used, i.e. from max.lams to max.lams/10, and 10 grid points.
gamma.eBIC	the parameter for the eBIC computation. gamma=0 is equivalent to BIC.
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	(optional) a $p \times p$ initial guess for the concentration matrix and/or starting solution for the ADMM algorithm.
rho1	a scalar; tuning parameter of the ADMM algorithm to be used for the outer loop. It must be strictly positive.
rho2	a scalar; tuning parameter of the ADMM algorithm to be used for the inner loop. It must be strictly positive.
varying.rho1	a logical; if TRUE the parameter rho1 is updated iteratively to speed-up convergence.
varying.rho2	a logical; if TRUE the parameter rho2 is updated iteratively to speed-up convergence.
max_iter	an integer; maximum number of iterations to be run in case the algorithm does not converge.
eps.abs	a scalar; the absolute precision required for the computation of primal and dual residuals of the ADMM algorithm.

eps.rel	a scalar; the relative precision required for the computation of primal and dual residuals of the ADMM algorithm.
verbose	a logical; if TRUE the progress (and internal convergence of inner loop) is shown in the console while the algorithm is running.
progress	a logical value; if TRUE provides a visual update in the console about the grid search over lambda1 and lambda2
print.type	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.

Value

A list with the following components:

- model the final model.
- lambda.grid the grid of values used for lambda1 and lambda2.
- best.lambdas the selected values of lambda1 and lambda2 according to eBIC criterion.
- l1.path a matrix containing the grid values for lambda1 as well as quantities used in eBIC computation.
- l2.path a matrix containing the grid values for lambda2 as well as quantities used in eBIC computation.
- 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).

Examples

```
S <- cov(toy_data$sample.data)
pdRCON.fit(S,n=60)
```

---

pdRCON.mle *Maximum likelihood estimation*

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,
  verbose = FALSE
)
```

Arguments

S	a sample covariance matrix with the block structure described in pdglasso-package.
pdColG	pdColG a matrix representing a coloured graph for paired data; see pdglasso-package for details.
eps.rel	a scalar; the relative precision required for the computation of primal and dual residuals of the ADMM algorithm.
eps.abs	a scalar; the absolute precision required for the computation of primal and dual residuals of the ADMM algorithm.
max_iter	an integer; maximum number of iterations to be run in case the algorithm does not converge.
verbose	a logical; if TRUE the progress (and internal convergence of inner loop) is shown in the console while the algorithm is running.

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, 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.simulate *Random simulation of pdRCON models*

Description

Randomly generates a coloured graph for paired data $\mathcal{G}$, a concentration matrix K adapted to $\mathcal{G}$ and a random sample from a multivariate normal distribution with zero mean vector and covariance matrix $\Sigma = K^{-1}$.

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
)
```

Arguments

p	an even integer, that is the number of vertices of the generated coloured graph for paired data pdColG.
concent.mat	a logical (default TRUE) indicating whether a concentration matrix K adapted to pdColG should be generated.
sample	a logical (default TRUE) indicating whether a sample from a normal distribution with zero mean vector and concentration matrix K should be generated.
Sigma	a $p \times p$ positive definite matrix. This 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. The default NULL is equivalent to the identity matrix Sigma=diag(p).
sample.size	size of the randomly generated sample. The default NULL is equivalent to sample.size=3*p.
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 described in the details section below. The default dens.vertex=NULL is equivalent to dens.vertex=dens, and similarly for dens.inside and dens.across.

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 $q(q-1)/2$, with $q = p/2$, inside block edges. 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 $q(q-1)/2$ 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 representing a coloured graph for paired data on p vertices; see pdglasso-package for details.
- K a randomly generated concentration matrix adapted to pdColG.
- sample.data a randomly generated sample form a multivariate normal distribution with mean vector zero and concentration matrix K.

Note that the variable in L are 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)
pdRCON.model <- pdRCON.simulate(10, concent=FALSE, sample=FALSE, dens=0.25)$pdColG

# generates 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)
```

---

toy_data *Toy dataset generated through* pdRCON.simulate *function*

Description

Data simulated by the function pdRCON.simulate with a call: toy_data <- pdRCON.simulate(20, type=c("v", "i"

Usage

```
toy_data
```

Format

`toy_data`**:**

A list with the following components:

- `pdColG` a matrix representing a coloured graph for paired data; see `pdglasso-package` for details.
- `K`, a concentration matrix adapted to `pdColG`.
- `sample.data`, a data frame with 60 rows and 20 columns from a multivariate normal distribution with zero mean vector and concentration matrix `K`.

Index