Package ‘pdglasso’

April 11, 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

Roxygen list(markdown = TRUE)

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

Depends R (>= 2.10)

Imports MASS

R topics documented:

`admm.pdglasso` *Estimate a concentration matrix under the pdColG model using (adaptive) ADMM graphical lasso algorithm.*

Description

By providing a covariance matrix S and values for lambda_1 and lambda_2, this function estimates a concentration matrix X under the coloured graphical model for paired data, using the (adaptive) ADMM algorithm. The output is the matrix and a list of internal parameters used by the function, together with the specific call in terms of symmetries and penalties required by the user.

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 = 1000,
  eps.abs = 1e-12,
  eps.rel = 1e-12,
  verbose = FALSE,
  print.type = TRUE
)
```

Arguments

`S`	A $p \times p$ covariance (or correlation) matrix.
`lambda1`	A non-negative scalar (or vector) penalty that encourages sparsity in the concentration matrix. If a vector is provided, it should match the appropriate length, i.e.
`lambda2`	A non-negative scalar (or vector) penalty that encourages equality constraints in the concentration matrix. If a vector is provided, it should match the appropriate length, i.e.
`type`	A string or vector of strings for the type of equality constraints to be imposed; zero, one or more available options can be selected among: * "vertex", symmetries are imposed on the diagonal entries of the concentration matrix. * "inside.block.edge", symmetries are imposed between elements of the LL and RR block the concentration matrix. * "across.block.edge", symmetries are imposed between elements of the LR and RL block the concentration matrix. Shortened forms are accepted too, i.e. "V" or "vert" for "vertex".
`force.symm`	A string or vector of strings to impose forced symmetry on the corresponding block of the concentration matrix. Same options as "type".

`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 boolean value; if `TRUE` the parameter rho1 is updated iteratively to speed-up convergence.
`varying.rho2`	A boolean value; 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 boolean value; if `TRUE` the progress (and internal convergence of inner loop) is shown in the console while the algorithm is running.
`print.type`	A boolean value; if `TRUE` the acronym used for the model - which penalties - is returned as printed output in the console.

Value

A list, whose element are:

- `X`, the estimated concentration matrix under the pdglasso model; the model is identified by the values of lambda1 and lambda 2, together with the type of penalization imposed.
- `acronyms`, a vector of strings for the type of penalties and forced symmetries imposed when calling the function.
- `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) for a given model.*

Description

The function computes the value of the eBIC for a given model and gamma value, for the purpose of model selection.

Usage

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

Arguments

S	A $p \times p$ covariance (or correlation) matrix.
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 needed to compute the eBIC; ranges from 0 to 1, where 0 makes the eBIC equivalent to BIC.

Value

a vector containing three elements:

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

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)
```

---

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

Description

Description here.

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, containing:

- g, the graph in matrix form.
- dof, the degrees of freedom corresponding to the graph build under the pdglasso model provided.

Examples

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

`pdColG.summarize` *Summary statistics for coloured graphs for paired data*

Description

This function

Usage

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

Arguments

`pdColG`	a coloured graph for paired data.
`print.summary`	logical (default `TRUE`) indicating whether a summary should be printed.

Value

a list

Examples

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

`pdglasso` *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 `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 as of submodels of interest are given in the following.

pdRCON models - terminology and relevant submodels

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 submodels can be specified both by allowing different combinations of restriction types and by forcing different types of fully symmetric structures. In this package, different submodels 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 submodels 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 submodels 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 submodels 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 coloured edges always belong 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` for model type specification

The functions of this package make it possible to specify different types of pdRCON submodels 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.

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)

Maintainer: Alberto Roverato `<alberto.roverato@unipd.it>` (ORCID)

Authors:

- Saverio Ranciati `<saverio.ranciati2@unibo.it>` (ORCID)

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

Description

Performs a sequence of calls to `admm.pdglasso` providing two grids of values for lambda_1 and lambda_2. First, a grid search conditional on lambda_2=0 is run to select the best lambda_1 value among the candidates (according to eBIC); conditional on the best lambda_1, a similar search is performed for lambda_2. The output is the select model, given by the estimated concenration matrix and corresponding graph.

Usage

```
pdRCON.fit(
  S,
  n,
  n.l1 = 15,
  n.l2 = 15,
  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 = 1000,
  eps.abs = 1e-12,
  eps.rel = 1e-12,
  verbose = FALSE,
  print.type = TRUE
)
```

Arguments

`S`	A $p \times p$ covariance (or correlation) matrix.
`n`	the sample size of the data used to compute the sample covariance matrix S.
`n.l1`	the number of values in the grid of candidates for lambda_1.
`n.l2`	the number of values in the grid of candidates for lambda_2.
`gamma.eBIC`	the parameter for the eBIC computation. gamma=0 is equivalent to BIC.
`type`	A string or vector of strings for the type of equality constraints to be imposed; zero, one or more available options can be selected among: * "vertex", symmetries are imposed on the diagonal entries of the concentration matrix. * "inside.block.edge", symmetries are imposed between elements of the LL and RR block the concentration matrix. * "across.block.edge", symmetries are imposed between elements of the LR and RL block the concentration matrix. Shortened forms are accepted too, i.e. "V" or "vert" for "vertex".
`force.symm`	A string or vector of strings to impose forced symmetry on the corresponding block of the concentration matrix. Same options as "type".
`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 boolean value; if `TRUE` the parameter rho1 is updated iteratively to speed-up convergence.
`varying.rho2`	A boolean value; 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 boolean value; if `TRUE` the progress (and internal convergence of inner loop) is shown in the console while the algorithm is running.
`print.type`	A boolean value; if `TRUE` the acronym used for the model - which penalties - is returned as printed output in the console.

Value

a list:

- model, the final model;
- pdColG, the associated coloured graph;
- best.lambdas, the selected values of lambda_1 and lambda_2 according to eBIC criterion,
- l1.path, a matrix containing the grid values for lambda_1 as well as quantities used in eBIC computation;
- l2.path, a matrix containing the grid values for lambda_2 as well as quantities used in eBIC computation.

Examples

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

---

pdRCON.mle *Maximum likelihood estimate*

Description

Computes the m.l.e. of the concentration matrix of a colured graphical model for paired data.

Usage

```
pdRCON.mle(S, pdColG)
```

Arguments

S	a sample variance and covariance matrix.
pdColG	a coloured graph for paired data.

Value

the m.l.e. of the concentration matrix Σ^{-1}.

Examples

```
#
```

---

pdRCON.simulate *Title*

Description

Title

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`	number of variables.
`concent.mat`	a logical (default `TRUE`) indicating whether a concentration matrix should be generated.
`sample`	a logical (default `TRUE`) indicating whether a sample form a normal distribution with zero mean vector and the generated concentration matrix should be generated.
`Sigma`	x
`sample.size`	sample size with default value equal to $3p$.
`type`	x
`force.symm`	x
`dens`	x
`dens.vertex`	x
`dens.inside`	x
`dens.across`	x

Value

this function

Examples

```
#
```

---

toy_data — *Toy dataset generated through* `simul.pdColG` *function*

Description

Data simulated by using the function `simul.pdColG` with the following arguments:

- `p <- 20`
- `q <- p/2`
- `dens=0.3, type=c("v","i","a"), force=NULL`

Usage

```
toy_data
```

Format

`toy_data`:

A list containing three elements:

- pdColG, a $p \times p$ matrix describing the coloured graphical model,
- K, the concentration matrix associated to the model,
- sample.data, a data frame with 60 rows and 20 columns.

Index