Package ‘fake’

August 9, 2022

Title Flexible Data Simulation Using the Multivariate Normal Distribution

Version 1.0.0

Author Barbara Bodinier [aut, cre]

Maintainer Barbara Bodinier <b.bodinier@imperial.ac.uk>

Description
Simulation of data from Gaussian Graphical Models (B Bodinier, S Filippi, TH Nost, J Chiquet, M Chadeau-Hyam (2021) <arXiv:2106.02521>). By controlling the conditional independence structure between the variables, these multivariate simulation tools can be used to evaluate the performance of regression, dimensionality reduction or graphical models.

License GPL (>= 3)

Language en-GB

Encoding UTF-8

RoxygenNote 7.2.0

Imports huge, igraph, MASS, Rdpack, withr (>= 2.4.0)

Suggests testthat (>= 3.0.0),

Config/testthat/edition 3

RdMacros Rdpack

NeedsCompilation no

Repository CRAN

Date/Publication 2022-08-09 12:40:02 UTC

R topics documented:

`BlockDiagonal` *Block diagonal matrix*

Description

Generates a binary block diagonal matrix.

Usage

```
BlockDiagonal(pk)
```

Arguments

`pk`	vector encoding the grouping structure.

Value

A binary block diagonal matrix.

See Also

Other block matrix functions: `BlockMatrix()`, `BlockStructure()`

Examples

```
# Example 1
BlockDiagonal(pk = c(2, 3))

# Example 2
BlockDiagonal(pk = c(2, 3, 2))
```

BlockMatrix *Block matrix*

Description

Generates a symmetric block matrix of size (`sum(pk)` x `sum(pk)`). The sizes of the submatrices is defined based on pk. For each submatrix, all entries are equal to the submatrix (block) index.

Usage

```
BlockMatrix(pk)
```

Arguments

pk	vector encoding the grouping structure.

Value

A symmetric block matrix.

See Also

Other block matrix functions: `BlockDiagonal()`, `BlockStructure()`

Examples

```
# Example 1
BlockMatrix(pk = c(2, 3))

# Example 2
BlockMatrix(pk = c(2, 3, 2))
```

BlockStructure *Block structure*

Description

Generates a symmetric matrix of size (`length(pk)` x `length(pk)`) where entries correspond to block indices. This function can be used to visualise block indices of a matrix generated with `BlockMatrix`.

Usage

```
BlockStructure(pk)
```

Arguments

pk	vector encoding the grouping structure.

Value

A symmetric matrix of size `length(pk))`.

See Also

Other block matrix functions: `BlockDiagonal()`, `BlockMatrix()`

Examples

```
# Example 1
BlockMatrix(pk = c(2, 3))
BlockStructure(pk = c(2, 3))

# Example 2
BlockMatrix(pk = c(2, 3, 2))
BlockStructure(pk = c(2, 3, 2))
```

`Contrast` *Matrix contrast*

Description

Computes matrix contrast, defined as the number of unique truncated entries with a specified number of digits.

Usage

```
Contrast(mat, digits = 3)
```

Arguments

`mat`	input matrix.
`digits`	number of digits to use.

Value

A single number, the contrast of the input matrix.

References

Bodinier B, Filippi S, Nost TH, Chiquet J, Chadeau-Hyam M (2021). “Automated calibration for stability selection in penalised regression and graphical models: a multi-OMICs network application exploring the molecular response to tobacco smoking.” https://arxiv.org/abs/2106.02521.

Examples

```
# Example 1
mat <- matrix(c(0.1, 0.2, 0.2, 0.2), ncol = 2, byrow = TRUE)
Contrast(mat)

# Example 2
mat <- matrix(c(0.1, 0.2, 0.2, 0.3), ncol = 2, byrow = TRUE)
Contrast(mat)
```

Heatmap *Heatmap visualisation*

Description

Produces a heatmap for visualisation of matrix entries.

Usage

```
Heatmap(
  mat,
  col = c("ivory", "navajowhite", "tomato", "darkred"),
  resolution = 10000,
  bty = "o",
  axes = TRUE,
  cex.axis = 1,
  xlas = 2,
  ylas = 2,
  text = FALSE,
  cex = 1,
  legend = TRUE,
  legend_length = NULL,
  legend_range = NULL,
  ...
)
```

Arguments

`mat`	data matrix.
`col`	vector of colours.
`resolution`	number of different colours to use.
`bty`	character string indicating if the box around the plot should be drawn. Possible values include: `"o"` (default, the box is drawn), or `"n"` (no box).
`axes`	logical indicating if the row and column names of `mat` should be displayed.
`cex.axis`	font size for axes.

xlas	orientation of labels on the x-axis, as las in par.
ylas	orientation of labels on the y-axis, as las in par.
text	logical indicating if numbers should be displayed.
cex	font size for numbers. Only used if text=TRUE.
legend	logical indicating if the colour bar should be included.
legend_length	length of the colour bar.
legend_range	range of the colour bar.
...	additional arguments passed to formatC for number formatting. Only used if text=TRUE.

Value

A heatmap.

Examples

```
oldpar <- par(no.readonly = TRUE)
par(mar = c(3, 3, 1, 5))

# Data simulation
set.seed(1)
mat <- matrix(rnorm(200), ncol = 20)
rownames(mat) <- paste0("r", 1:nrow(mat))
colnames(mat) <- paste0("c", 1:ncol(mat))

# Generating heatmaps
Heatmap(mat = mat)
Heatmap(
  mat = mat,
  col = c("lightgrey", "blue", "black"),
  legend = FALSE
)

par(oldpar)
```

MakePositiveDefinite *Making positive definite matrix*

Description

Determines the diagonal entries of a symmetric matrix to make it is positive definite.

Usage

```
MakePositiveDefinite(
  omega,
  pd_strategy = "diagonally_dominant",
  ev_xx = NULL,
  scale = TRUE,
  u_list = c(1e-10, 1),
  tol = .Machine$double.eps^0.25
)
```

Arguments

`omega`	input matrix.
`pd_strategy`	method to ensure that the generated precision matrix is positive definite (and hence can be a covariance matrix). If `pd_strategy="diagonally_dominant"`, the precision matrix is made diagonally dominant by setting the diagonal entries to the sum of absolute values on the corresponding row and a constant u. If `pd_strategy="min_eigenvalue"`, diagonal entries are set to the sum of the absolute value of the smallest eigenvalue of the precision matrix with zeros on the diagonal and a constant u.
`ev_xx`	expected proportion of explained variance by the first Principal Component (PC1) of a Principal Component Analysis. This is the largest eigenvalue of the correlation (if `scale_ev=TRUE`) or covariance (if `scale_ev=FALSE`) matrix divided by the sum of eigenvalues. If `ev_xx=NULL` (the default), the constant u is chosen by maximising the contrast of the correlation matrix.
`scale`	logical indicating if the proportion of explained variance by PC1 should be computed from the correlation (`scale=TRUE`) or covariance (`scale=FALSE`) matrix.
`u_list`	vector with two numeric values defining the range of values to explore for constant u.
`tol`	accuracy for the search of parameter u as defined in `optimise`.

Details

Two strategies are implemented to ensure positive definiteness: by diagonally dominance or using eigendecomposition.

A diagonally dominant symmetric matrix with positive diagonal entries is positive definite. With `pd_strategy="diagonally_dominant"`, the diagonal entries of the matrix are defined to be strictly higher than the sum of entries on the corresponding row in absolute value, which ensures diagonally dominance. Let $\Omega*$ denote the input matrix with zeros on the diagonal and Ω be the output positive definite matrix. We have:

$\Omega_{ii} = \sum_{j=1}^{p} |\Omega_{ij}*| + u$, where $u > 0$ is a parameter.

A matrix is positive definite if all its eigenvalues are positive. With `pd_strategy="diagonally_dominant"`, diagonal entries of the matrix are defined to be higher than the absolute value of the smallest eigenvalue of the same matrix with a diagonal of zeros. Let λ_1 denote the smallest eigenvvalue of the input matrix $\Omega*$ with a diagonal of zeros, and v_1 be the corresponding eigenvector. Diagonal entries in the output matrix Ω are defined as:

$\Omega_{ii} = |\lambda_1| + u$, where $u > 0$ is a parameter.

It can be showed that Ω has stricly positive eigenvalues. Let λ and v denote any eigenpair of $\Omega*$:

$\Omega * v = \lambda v$

$\Omega * v + (|\lambda_1| + u)v = \lambda v + (|\lambda_1| + u)v$

$(\Omega * +(|\lambda_1| + u)I)v = (\lambda + |\lambda_1| + u)v$

$\Omega v = (\lambda + |\lambda_1| + u)v$

The eigenvalues of Ω are equal to the eigenvalues of $\Omega*$ plus $|\lambda_1|$. The smallest eigenvalue of Ω is $(\lambda_1 + |\lambda_1| + u) > 0$.

Considering the matrix to make positive definite is a precision matrix, its standardised inverse matrix is the correlation matrix. In both cases, the magnitude of correlations is controlled by the constant u.

If `ev_xx=NULL`, the constant u is chosen to maximise the `Contrast` of the corresponding correlation matrix.

If `ev_xx` is provided, the constant u is chosen to generate a correlation matrix with required proportion of explained variance by the first Principal Component, if possible. This proportion of explained variance is equal to the largest eigenvalue of the correlation matrix divided by the sum of its eigenvalues. If `scale=FALSE`, the covariance matrix is used instead of the correlation matrix for faster computations.

Value

A list with:

`omega`	positive definite matrix.
`u`	value of the constant u.

References

Bodinier B, Filippi S, Nost TH, Chiquet J, Chadeau-Hyam M (2021). “Automated calibration for stability selection in penalised regression and graphical models: a multi-OMICs network application exploring the molecular response to tobacco smoking.” https://arxiv.org/abs/2106.02521.

Examples

```
# Simulation of a symmetric matrix
p <- 5
set.seed(1)
omega <- matrix(rnorm(p * p), ncol = p)
omega <- omega + t(omega)
diag(omega) <- 0

# Diagonal dominance maximising contrast
omega_pd <- MakePositiveDefinite(omega,
  pd_strategy = "diagonally_dominant"
)
eigen(omega_pd$omega)$values # positive eigenvalues
```

```
# Diagonal dominance with specific proportion of explained variance by PC1
omega_pd <- MakePositiveDefinite(omega,
  pd_strategy = "diagonally_dominant",
  ev_xx = 0.55
)
lambda_inv <- eigen(cov2cor(solve(omega_pd$omega)))$values
max(lambda_inv) / sum(lambda_inv) # expected ev

# Version not scaled (using eigenvalues from the covariance)
omega_pd <- MakePositiveDefinite(omega,
  pd_strategy = "diagonally_dominant",
  ev_xx = 0.55, scale = FALSE
)
lambda_inv <- 1 / eigen(omega_pd$omega)$values
max(lambda_inv) / sum(lambda_inv) # expected ev

# Non-negative eigenvalues maximising contrast
omega_pd <- MakePositiveDefinite(omega,
  pd_strategy = "min_eigenvalue"
)
eigen(omega_pd$omega)$values # positive eigenvalues

# Non-negative eigenvalues with specific proportion of explained variance by PC1
omega_pd <- MakePositiveDefinite(omega,
  pd_strategy = "min_eigenvalue",
  ev_xx = 0.7
)
lambda_inv <- eigen(cov2cor(solve(omega_pd$omega)))$values
max(lambda_inv) / sum(lambda_inv)

# Version not scaled (using eigenvalues from the covariance)
omega_pd <- MakePositiveDefinite(omega,
  pd_strategy = "min_eigenvalue",
  ev_xx = 0.7, scale = FALSE
)
lambda_inv <- 1 / eigen(omega_pd$omega)$values
max(lambda_inv) / sum(lambda_inv)
```

`MatchingArguments` *Matching arguments*

Description

Returns a vector of overlapping character strings between `extra_args` and arguments from function `FUN`. If `FUN` is taking `...` as input, this function returns `extra_args`.

Usage

```
MatchingArguments(extra_args, FUN)
```

Arguments

extra_args	vector of character strings.
FUN	function.

Value

A vector of overlapping arguments.

Examples

```
MatchingArguments(
  extra_args = list(Sigma = 1, test = FALSE),
  FUN = MASS::mvrnorm
)
```

SimulateAdjacency *Simulation of undirected graph with block structure*

Description

Simulates the adjacency matrix of an unweighted, undirected graph with no self-loops. If `topology="random"`, different densities in diagonal (`nu_within`) compared to off-diagonal (`nu_between`) blocks can be used.

Usage

```
SimulateAdjacency(
  pk = 10,
  implementation = HugeAdjacency,
  topology = "random",
  nu_within = 0.1,
  nu_between = 0,
  ...
)
```

Arguments

pk	vector of the number of variables per group in the simulated data. The number of nodes in the simulated graph is `sum(pk)`. With multiple groups, the simulated (partial) correlation matrix has a block structure, where blocks arise from the integration of the `length(pk)` groups. This argument is only used if `sum(pk)` is equal to the number of rows/columns in `theta` is not provided.
implementation	function for simulation of the graph. By default, algorithms implemented in `huge.generator` are used. Alternatively, a user-defined function can be used. It must take `pk`, `topology` and `nu` as arguments and return a `(sum(pk)*(sum(pk)))` binary and symmetric matrix for which diagonal entries are all equal to zero. This function is only applied if `theta` is not provided.

`topology`	topology of the simulated graph. If using `implementation=HugeAdjacency`, possible values are listed for the argument graph of `huge.generator`. These are: "random", "hub", "cluster", "band" and "scale-free".
`nu_within`	expected density (number of edges over the number of node pairs) of within-group blocks in the graph. If `length(pk)=1`, this is the expected density of the graph. If `implementation=HugeAdjacency`, this argument is only used for `topology="random"` or `topology="cluster"` (see argument prob in `huge.generator`).
`nu_between`	expected density (number of edges over the number of node pairs) of between-group blocks in the graph. Similar to `nu_within`. By default, the same density is used for within and between blocks (`nu_within=nu_between`). Only used if `length(pk)>1`.
`...`	additional arguments passed to the graph simulation function provided in `implementation`.

Details

Random graphs are simulated using the Erdos-Renyi algorithm. Scale-free graphs are simulated using a preferential attachment algorithm. More details are provided in `huge.generator`.

Value

A symmetric adjacency matrix encoding an unweighted, undirected graph with no self-loops, and with different densities in diagonal compared to off-diagonal blocks.

References

Bodinier B, Filippi S, Nost TH, Chiquet J, Chadeau-Hyam M (2021). “Automated calibration for stability selection in penalised regression and graphical models: a multi-OMICs network application exploring the molecular response to tobacco smoking.” `https://arxiv.org/abs/2106.02521`.

Jiang H, Fei X, Liu H, Roeder K, Lafferty J, Wasserman L, Li X, Zhao T (2021). *huge: High-Dimensional Undirected Graph Estimation*. R package version 1.3.5, `https://CRAN.R-project.org/package=huge`.

See Also

Other simulation functions: `SimulateComponents()`, `SimulateGraphical()`, `SimulateRegression()`

Examples

```
# Simulation of a scale-free graph with 20 nodes
adjacency <- SimulateAdjacency(pk = 20, topology = "scale-free")
plot(adjacency)

# Simulation of a random graph with three connected components
adjacency <- SimulateAdjacency(
  pk = rep(10, 3),
  nu_within = 0.7, nu_between = 0
)
plot(adjacency)
```

```
# Simulation of a random graph with block structure
adjacency <- SimulateAdjacency(
  pk = rep(10, 3),
  nu_within = 0.7, nu_between = 0.03
)
plot(adjacency)

# User-defined function for graph simulation
CentralNode <- function(pk, hub = 1) {
  theta <- matrix(0, nrow = sum(pk), ncol = sum(pk))
  theta[hub, ] <- 1
  theta[, hub] <- 1
  diag(theta) <- 0
  return(theta)
}
simul <- SimulateAdjacency(pk = 10, implementation = CentralNode)
plot(simul) # star
simul <- SimulateAdjacency(pk = 10, implementation = CentralNode, hub = 2)
plot(simul) # variable 2 is the central node
```

SimulateComponents *Data simulation for sparse Principal Component Analysis*

Description

Simulates data with with independent groups of variables.

Usage

```
SimulateComponents(
  n = 100,
  pk = c(10, 10),
  adjacency = NULL,
  nu_within = 1,
  v_within = c(0.5, 1),
  v_sign = -1,
  continuous = TRUE,
  pd_strategy = "min_eigenvalue",
  ev_xx = 0.1,
  scale_ev = TRUE,
  u_list = c(1e-10, 1),
  tol = .Machine$double.eps^0.25,
  scale = TRUE,
  output_matrices = FALSE
)
```

Arguments

`n`	number of observations in the simulated data.
`pk`	vector of the number of variables per group in the simulated data. The number of nodes in the simulated graph is `sum(pk)`. With multiple groups, the simulated (partial) correlation matrix has a block structure, where blocks arise from the integration of the `length(pk)` groups. This argument is only used if `sum(pk)` is equal to the number of rows/columns in `theta` is not provided.
`adjacency`	optional binary and symmetric adjacency matrix encoding the conditional graph structure between observations. The clusters encoded in this argument must be in line with those indicated in `pk`. Edges in off-diagonal blocks are not allowed to ensure that the simulated orthogonal components are sparse. Corresponding entries in the precision matrix will be set to zero.
`nu_within`	expected density (number of edges over the number of node pairs) of within-group blocks in the graph. If `length(pk)=1`, this is the expected density of the graph. If `implementation=HugeAdjacency`, this argument is only used for `topology="random"` or `topology="cluster"` (see argument `prob` in `huge.generator`).
`v_within`	vector defining the (range of) nonzero entries in the diagonal blocks of the precision matrix. These values must be between -1 and 1 if `pd_strategy="min_eigenvalue"`. If `continuous=FALSE`, `v_within` is the set of possible precision values. If `continuous=TRUE`, `v_within` is the range of possible precision values.
`v_sign`	vector of possible signs for precision matrix entries. Possible inputs are: `-1` for positive partial correlations, `1` for negative partial correlations, or `c(-1, 1)` for both positive and negative partial correlations.
`continuous`	logical indicating whether to sample precision values from a uniform distribution between the minimum and maximum values in `v_within` (diagonal blocks) or `v_between` (off-diagonal blocks) (`continuous=TRUE`) or from proposed values in `v_within` (diagonal blocks) or `v_between` (off-diagonal blocks) (`continuous=FALSE`).
`pd_strategy`	method to ensure that the generated precision matrix is positive definite (and hence can be a covariance matrix). If `pd_strategy="diagonally_dominant"`, the precision matrix is made diagonally dominant by setting the diagonal entries to the sum of absolute values on the corresponding row and a constant u. If `pd_strategy="min_eigenvalue"`, diagonal entries are set to the sum of the absolute value of the smallest eigenvalue of the precision matrix with zeros on the diagonal and a constant u.
`ev_xx`	expected proportion of explained variance by the first Principal Component (PC1) of a Principal Component Analysis. This is the largest eigenvalue of the correlation (if `scale_ev=TRUE`) or covariance (if `scale_ev=FALSE`) matrix divided by the sum of eigenvalues. If `ev_xx=NULL` (the default), the constant u is chosen by maximising the contrast of the correlation matrix.
`scale_ev`	logical indicating if the proportion of explained variance by PC1 should be computed from the correlation (`scale_ev=TRUE`) or covariance (`scale_ev=FALSE`) matrix. If `scale_ev=TRUE`, the correlation matrix is used as parameter of the multivariate normal distribution.
`u_list`	vector with two numeric values defining the range of values to explore for constant u.

`tol`	accuracy for the search of parameter u as defined in `optimise`.
`scale`	logical indicating if the simulated data should be standardised using `scale`.
`output_matrices`	logical indicating if the true precision and (partial) correlation matrices should be included in the output.

Details

The data is simulated from a centered multivariate Normal distribution with a block-diagonal covariance matrix. Independence between variables from the different blocks ensures that sparse orthogonal components can be generated.

The block-diagonal partial correlation matrix is obtained using a graph structure encoding the conditional independence between variables. The orthogonal latent variables are obtained from eigendecomposition of the true correlation matrix. The sparse eigenvectors contain the weights of the linear combination of variables to construct the latent variable (loadings coefficients). The proportion of explained variance by each of the latent variable is computed from eigenvalues.

As latent variables are defined from the true correlation matrix, the number of sparse orthogonal components is not limited by the number of observations and is equal to `sum(pk)`.

Value

A list with:

`data`	simulated data with `n` observation and `sum(pk)` variables.
`loadings`	loadings coefficients of the orthogonal latent variables (principal components).
`theta`	support of the loadings coefficients.
`ev`	proportion of explained variance by each of the orthogonal latent variables.
`adjacency`	adjacency matrix of the simulated graph.
`omega`	simulated (true) precision matrix. Only returned if `output_matrices=TRUE`.
`phi`	simulated (true) partial correlation matrix. Only returned if `output_matrices=TRUE`.
`C`	simulated (true) correlation matrix. Only returned if `output_matrices=TRUE`.

References

Bodinier B, Filippi S, Nost TH, Chiquet J, Chadeau-Hyam M (2021). “Automated calibration for stability selection in penalised regression and graphical models: a multi-OMICs network application exploring the molecular response to tobacco smoking.” https://arxiv.org/abs/2106.02521.

See Also

`MakePositiveDefinite`

Other simulation functions: `SimulateAdjacency()`, `SimulateGraphical()`, `SimulateRegression()`

Examples

```
# Simulation of 3 components with high e.v.
set.seed(1)
simul <- SimulateComponents(pk = c(5, 3, 4), ev_xx = 0.4)
print(simul)
plot(simul)
plot(cumsum(simul$ev), ylim = c(0, 1), las = 1)

# Simulation of 3 components with moderate e.v.
set.seed(1)
simul <- SimulateComponents(pk = c(5, 3, 4), ev_xx = 0.25)
print(simul)
plot(simul)
plot(cumsum(simul$ev), ylim = c(0, 1), las = 1)

# Simulation of multiple components with low e.v.
pk <- sample(3:10, size = 5, replace = TRUE)
simul <- SimulateComponents(
  pk = pk,
  nu_within = 0.3, v_within = c(0.8, 0.5), v_sign = -1, ev_xx = 0.1
)
plot(simul)
plot(cumsum(simul$ev), ylim = c(0, 1), las = 1)
```

`SimulateGraphical` *Data simulation for Gaussian Graphical Modelling*

Description

Simulates data from a Gaussian Graphical Model (GGM).

Usage

```
SimulateGraphical(
  n = 100,
  pk = 10,
  theta = NULL,
  implementation = HugeAdjacency,
  topology = "random",
  nu_within = 0.1,
  nu_between = NULL,
  v_within = c(0.5, 1),
  v_between = c(0.1, 0.2),
  v_sign = c(-1, 1),
  continuous = TRUE,
  pd_strategy = "diagonally_dominant",
  ev_xx = NULL,
  scale_ev = TRUE,
  u_list = c(1e-10, 1),
  tol = .Machine$double.eps^0.25,
  scale = TRUE,
  output_matrices = FALSE,
  ...
)
```

Arguments

`n`	number of observations in the simulated data.
`pk`	vector of the number of variables per group in the simulated data. The number of nodes in the simulated graph is `sum(pk)`. With multiple groups, the simulated (partial) correlation matrix has a block structure, where blocks arise from the integration of the `length(pk)` groups. This argument is only used if `sum(pk)` is equal to the number of rows/columns in `theta` is not provided.
`theta`	optional binary and symmetric adjacency matrix encoding the conditional independence structure.
`implementation`	function for simulation of the graph. By default, algorithms implemented in `huge.generator` are used. Alternatively, a user-defined function can be used. It must take `pk`, `topology` and `nu` as arguments and return a `(sum(pk)*(sum(pk)))` binary and symmetric matrix for which diagonal entries are all equal to zero. This function is only applied if `theta` is not provided.
`topology`	topology of the simulated graph. If using `implementation=HugeAdjacency`, possible values are listed for the argument `graph` of `huge.generator`. These are: "random", "hub", "cluster", "band" and "scale-free".
`nu_within`	expected density (number of edges over the number of node pairs) of within-group blocks in the graph. If `length(pk)=1`, this is the expected density of the graph. If `implementation=HugeAdjacency`, this argument is only used for `topology="random"` or `topology="cluster"` (see argument `prob` in `huge.generator`).
`nu_between`	expected density (number of edges over the number of node pairs) of between-group blocks in the graph. Similar to `nu_within`. By default, the same density is used for within and between blocks (`nu_within=nu_between`). Only used if `length(pk)>1`.
`v_within`	vector defining the (range of) nonzero entries in the diagonal blocks of the precision matrix. These values must be between -1 and 1 if `pd_strategy="min_eigenvalue"`. If `continuous=FALSE`, `v_within` is the set of possible precision values. If `continuous=TRUE`, `v_within` is the range of possible precision values.
`v_between`	vector defining the (range of) nonzero entries in the off-diagonal blocks of the precision matrix. This argument is the same as `v_within` but for off-diagonal blocks. It is only used if `length(pk)>1`.
`v_sign`	vector of possible signs for precision matrix entries. Possible inputs are: `-1` for positive partial correlations, `1` for negative partial correlations, or `c(-1, 1)` for both positive and negative partial correlations.
`continuous`	logical indicating whether to sample precision values from a uniform distribution between the minimum and maximum values in `v_within` (diagonal blocks) or `v_between` (off-diagonal blocks) (`continuous=TRUE`) or from proposed values in `v_within` (diagonal blocks) or `v_between` (off-diagonal blocks) (`continuous=FALSE`).

`pd_strategy`	method to ensure that the generated precision matrix is positive definite (and hence can be a covariance matrix). If `pd_strategy="diagonally_dominant"`, the precision matrix is made diagonally dominant by setting the diagonal entries to the sum of absolute values on the corresponding row and a constant u. If `pd_strategy="min_eigenvalue"`, diagonal entries are set to the sum of the absolute value of the smallest eigenvalue of the precision matrix with zeros on the diagonal and a constant u.
`ev_xx`	expected proportion of explained variance by the first Principal Component (PC1) of a Principal Component Analysis. This is the largest eigenvalue of the correlation (if `scale_ev=TRUE`) or covariance (if `scale_ev=FALSE`) matrix divided by the sum of eigenvalues. If `ev_xx=NULL` (the default), the constant u is chosen by maximising the contrast of the correlation matrix.
`scale_ev`	logical indicating if the proportion of explained variance by PC1 should be computed from the correlation (`scale_ev=TRUE`) or covariance (`scale_ev=FALSE`) matrix. If `scale_ev=TRUE`, the correlation matrix is used as parameter of the multivariate normal distribution.
`u_list`	vector with two numeric values defining the range of values to explore for constant u.
`tol`	accuracy for the search of parameter u as defined in `optimise`.
`scale`	logical indicating if the simulated data should be standardised using `scale`.
`output_matrices`	logical indicating if the true precision and (partial) correlation matrices should be included in the output.
`...`	additional arguments passed to the graph simulation function provided in `implementation`.

Details

The simulation is done in two steps with (i) generation of a graph, and (ii) sampling from multivariate Normal distribution for which nonzero entries in the partial correlation matrix correspond to the edges of the simulated graph. This procedure ensures that the conditional independence structure between the variables corresponds to the simulated graph.

Step 1 is done using `SimulateAdjacency`.

In Step 2, the precision matrix (inverse of the covariance matrix) is simulated using `SimulatePrecision` so that (i) its nonzero entries correspond to edges in the graph simulated in Step 1, and (ii) it is positive definite (see `MakePositiveDefinite`). The inverse of the precision matrix is used as covariance matrix to simulate data from a multivariate Normal distribution.

The outputs of this function can be used to evaluate the ability of a graphical model to recover the conditional independence structure.

Value

A list with:

`data`	simulated data with `n` observation and `sum(pk)` variables.
`theta`	adjacency matrix of the simulated graph
`omega`	simulated (true) precision matrix. Only returned if `output_matrices=TRUE`.

`phi`	simulated (true) partial correlation matrix. Only returned if `output_matrices=TRUE`.
`sigma`	simulated (true) covariance matrix. Only returned if `output_matrices=TRUE`.
`u`	value of the constant `u` used for the simulation of `omega`. Only returned if `output_matrices=TRUE`.

References

Bodinier B, Filippi S, Nost TH, Chiquet J, Chadeau-Hyam M (2021). "Automated calibration for stability selection in penalised regression and graphical models: a multi-OMICs network application exploring the molecular response to tobacco smoking." https://arxiv.org/abs/2106.02521.

See Also

`SimulatePrecision`, `MakePositiveDefinite`, `Contrast`

Other simulation functions: `SimulateAdjacency()`, `SimulateComponents()`, `SimulateRegression()`

Examples

```
# Simulation of random graph with 50 nodes
set.seed(1)
simul <- SimulateGraphical(n = 100, pk = 50, topology = "random", nu_within = 0.05)
print(simul)
plot(simul)

# Simulation of scale-free graph with 20 nodes
set.seed(1)
simul <- SimulateGraphical(n = 100, pk = 20, topology = "scale-free")
plot(simul)

# Extracting true precision/correlation matrices
set.seed(1)
simul <- SimulateGraphical(
  n = 100, pk = 20,
  topology = "scale-free", output_matrices = TRUE
)
str(simul)

# Simulation of multi-block data
set.seed(1)
pk <- c(20, 30)
simul <- SimulateGraphical(
  n = 100, pk = pk,
  pd_strategy = "min_eigenvalue"
)
mycor <- cor(simul$data)
Heatmap(mycor,
  col = c("darkblue", "white", "firebrick3"),
  legend_range = c(-1, 1), legend_length = 50,
  legend = FALSE, axes = FALSE
)
```

```
for (i in 1:2) {
  axis(side = i, at = c(0.5, pk[1] - 0.5), labels = NA)
  axis(
    side = i, at = mean(c(0.5, pk[1] - 0.5)),
    labels = ifelse(i == 1, yes = "Group 1", no = "Group 2"),
    tick = FALSE, cex.axis = 1.5
  )
  axis(side = i, at = c(pk[1] + 0.5, sum(pk) - 0.5), labels = NA)
  axis(
    side = i, at = mean(c(pk[1] + 0.5, sum(pk) - 0.5)),
    labels = ifelse(i == 1, yes = "Group 2", no = "Group 1"),
    tick = FALSE, cex.axis = 1.5
  )
}

# User-defined function for graph simulation
CentralNode <- function(pk, hub = 1) {
  theta <- matrix(0, nrow = sum(pk), ncol = sum(pk))
  theta[hub, ] <- 1
  theta[, hub] <- 1
  diag(theta) <- 0
  return(theta)
}
simul <- SimulateGraphical(n = 100, pk = 10, implementation = CentralNode)
plot(simul) # star
simul <- SimulateGraphical(n = 100, pk = 10, implementation = CentralNode, hub = 2)
plot(simul) # variable 2 is the central node

# User-defined adjacency matrix
mytheta <- matrix(c(
  0, 1, 1, 0,
  1, 0, 0, 0,
  1, 0, 0, 1,
  0, 0, 1, 0
), ncol = 4, byrow = TRUE)
simul <- SimulateGraphical(n = 100, theta = mytheta)
plot(simul)

# User-defined adjacency and block structure
simul <- SimulateGraphical(n = 100, theta = mytheta, pk = c(2, 2))
mycor <- cor(simul$data)
Heatmap(mycor,
  col = c("darkblue", "white", "firebrick3"),
  legend_range = c(-1, 1), legend_length = 50, legend = FALSE
)
```

`SimulatePrecision` *Simulation of precision matrix*

Description

Simulates a sparse precision matrix from a binary adjacency matrix `theta` encoding conditional independence in a Gaussian Graphical Model.

Usage

```
SimulatePrecision(
  pk = NULL,
  theta,
  v_within = c(0.5, 1),
  v_between = c(0, 0.1),
  v_sign = c(-1, 1),
  continuous = TRUE,
  pd_strategy = "diagonally_dominant",
  ev_xx = NULL,
  scale = TRUE,
  u_list = c(1e-10, 1),
  tol = .Machine$double.eps^0.25
)
```

Arguments

`pk`	vector of the number of variables per group in the simulated data. The number of nodes in the simulated graph is `sum(pk)`. With multiple groups, the simulated (partial) correlation matrix has a block structure, where blocks arise from the integration of the `length(pk)` groups. This argument is only used if `sum(pk)` is equal to the number of rows/columns in `theta` is not provided.
`theta`	binary and symmetric adjacency matrix encoding the conditional independence structure.
`v_within`	vector defining the (range of) nonzero entries in the diagonal blocks of the precision matrix. These values must be between -1 and 1 if `pd_strategy="min_eigenvalue"`. If `continuous=FALSE`, `v_within` is the set of possible precision values. If `continuous=TRUE`, `v_within` is the range of possible precision values.
`v_between`	vector defining the (range of) nonzero entries in the off-diagonal blocks of the precision matrix. This argument is the same as `v_within` but for off-diagonal blocks. It is only used if `length(pk)>1`.
`v_sign`	vector of possible signs for precision matrix entries. Possible inputs are: `-1` for positive partial correlations, `1` for negative partial correlations, or `c(-1, 1)` for both positive and negative partial correlations.
`continuous`	logical indicating whether to sample precision values from a uniform distribution between the minimum and maximum values in `v_within` (diagonal blocks) or `v_between` (off-diagonal blocks) (`continuous=TRUE`) or from proposed values in `v_within` (diagonal blocks) or `v_between` (off-diagonal blocks) (`continuous=FALSE`).
`pd_strategy`	method to ensure that the generated precision matrix is positive definite (and hence can be a covariance matrix). If `pd_strategy="diagonally_dominant"`, the precision matrix is made diagonally dominant by setting the diagonal entries to the sum of absolute values on the corresponding row and a constant u. If

	`pd_strategy="min_eigenvalue"`, diagonal entries are set to the sum of the absolute value of the smallest eigenvalue of the precision matrix with zeros on the diagonal and a constant u.
`ev_xx`	expected proportion of explained variance by the first Principal Component (PC1) of a Principal Component Analysis. This is the largest eigenvalue of the correlation (if `scale_ev=TRUE`) or covariance (if `scale_ev=FALSE`) matrix divided by the sum of eigenvalues. If `ev_xx=NULL` (the default), the constant u is chosen by maximising the contrast of the correlation matrix.
`scale`	logical indicating if the proportion of explained variance by PC1 should be computed from the correlation (`scale=TRUE`) or covariance (`scale=FALSE`) matrix.
`u_list`	vector with two numeric values defining the range of values to explore for constant u.
`tol`	accuracy for the search of parameter u as defined in `optimise`.

Details

Entries that are equal to zero in the adjacency matrix `theta` are also equal to zero in the generated precision matrix. These zero entries indicate conditional independence between the corresponding pair of variables (see `SimulateGraphical`).

Argument pk can be specified to create groups of variables and allow for nonzero precision entries to be sampled from different distributions between two variables belonging to the same group or to different groups.

If `continuous=FALSE`, nonzero off-diagonal entries of the precision matrix are sampled from a discrete uniform distribution taking values in `v_within` (for entries in the diagonal block) or `v_between` (for entries in off-diagonal blocks). If `continuous=TRUE`, nonzero off-diagonal entries are sampled from a continuous uniform distribution taking values in the range given by `v_within` or `v_between`.

Diagonal entries of the precision matrix are defined to ensure positive definiteness as described in `MakePositiveDefinite`.

Value

A list with:

`omega`	true simulated precision matrix.
`u`	value of the constant u used to ensure that `omega` is positive definite.

References

Bodinier B, Filippi S, Nost TH, Chiquet J, Chadeau-Hyam M (2021). “Automated calibration for stability selection in penalised regression and graphical models: a multi-OMICs network application exploring the molecular response to tobacco smoking.” `https://arxiv.org/abs/2106.02521`.

See Also

`SimulateGraphical`, `MakePositiveDefinite`

Examples

```
# Simulation of an adjacency matrix
theta <- SimulateAdjacency(pk = c(5, 5), nu_within = 0.7)
print(theta)

# Simulation of a precision matrix maximising the contrast
simul <- SimulatePrecision(theta = theta)
print(simul$omega)

# Simulation of a precision matrix with specific ev by PC1
simul <- SimulatePrecision(
  theta = theta,
  pd_strategy = "min_eigenvalue",
  ev_xx = 0.3, scale = TRUE
)
print(simul$omega)
```

SimulateRegression *Data simulation for multivariate regression*

Description

Simulates data with outcome(s) and predictors, where only a subset of the predictors actually contributes to the definition of the outcome(s).

Usage

```
SimulateRegression(
  n = 100,
  pk = 10,
  N = 3,
  family = "gaussian",
  ev_xz = 0.8,
  adjacency_x = NULL,
  nu_within = 0.1,
  theta_xz = NULL,
  nu_xz = 0.2,
  theta_zy = NULL,
  nu_zy = 0.5,
  eta = NULL,
  eta_set = c(-1, 1),
  v_within = c(0.5, 1),
  v_sign = c(-1, 1),
  continuous = TRUE,
  pd_strategy = "diagonally_dominant",
  ev_xx = NULL,
  scale_ev = TRUE,
  u_list = c(1e-10, 1),
  tol = .Machine$double.eps^0.25
)
```

Arguments

n	number of observations in the simulated data.
pk	vector with the number of predictors in each independent block of variables in xdata. The number of independent blocks, which determines the maximum number of orthogonal latent variables that can be simulated, is given by length(pk).
N	number of classes of the categorical outcome. Only used if family="multinomial".
family	type of outcome. If family="gaussian", normally distributed outcomes are simulated. If family="binomial" or family="multinomial", binary outcome(s) are simulated from a multinomial distribution where the probability is defined from a linear combination of normally distributed outcomes.
ev_xz	vector of the expected proportions of explained variances for each of the orthogonal latent variables. It must contain values in]0,1[, and must be a vector of length length(pk) or a single value to generate latent variables with the same expected proportion of explained variance.
adjacency_x	optional matrix encoding the conditional independence structure between predictor variables in xdata. This argument must be a binary symmetric matrix of size sum(pk) with zeros on the diagonal.
nu_within	expected density (number of edges over the number of node pairs) of the conditional independence graph in the within-group blocks for predictors. For independent predictors, use nu_within=0. This argument is only used if adjancency_x is not provided.
theta_xz	optional binary matrix encoding the predictor variables from xdata (columns) contributing to the definition of the orthogonal latent outcomes from zdata (rows).
nu_xz	expected proportion of relevant predictors over the total number of predictors to be used for the simulation of the orthogonal latent outcomes. This argument is only used if theta_xz is not provided.
theta_zy	optional binary matrix encoding the latent variables from zdata (columns) contributing to the definition of the observed outcomes from ydata (rows). This argument must be a square matrix of size length(pk). If theta_zy is a diagonal matrix, each latent variable contributes to the definition of one observed outcome so that there is a one-to-one relationship between latent and observed outcomes (i.e. they are collinear). Nonzero off-diagonal elements in theta_zy introduce some correlation between the observed outcomes by construction from linear combinations implicating common latent outcomes. This argument is only used if eta is not provided.
nu_zy	probability for each of the off-diagonal elements in theta_zy to be a 1. If nu_zy=0, theta_zy is a diagonal matrix. This argument is only used if theta_zy is not provided.
eta	optional matrix of coefficients used in the linear combination of latent outcomes to generate observed outcomes.

`eta_set`	vector defining the range of values from which `eta` is sampled. This argument is only used if `eta` is not provided.
`v_within`	vector defining the (range of) nonzero entries in the diagonal blocks of the precision matrix. These values must be between -1 and 1 if `pd_strategy="min_eigenvalue"`. If `continuous=FALSE`, `v_within` is the set of possible precision values. If `continuous=TRUE`, `v_within` is the range of possible precision values.
`v_sign`	vector of possible signs for precision matrix entries. Possible inputs are: `-1` for positive partial correlations, `1` for negative partial correlations, or `c(-1, 1)` for both positive and negative partial correlations.
`continuous`	logical indicating whether to sample precision values from a uniform distribution between the minimum and maximum values in `v_within` (diagonal blocks) or `v_between` (off-diagonal blocks) (`continuous=TRUE`) or from proposed values in `v_within` (diagonal blocks) or `v_between` (off-diagonal blocks) (`continuous=FALSE`).
`pd_strategy`	method to ensure that the generated precision matrix is positive definite (and hence can be a covariance matrix). If `pd_strategy="diagonally_dominant"`, the precision matrix is made diagonally dominant by setting the diagonal entries to the sum of absolute values on the corresponding row and a constant u. If `pd_strategy="min_eigenvalue"`, diagonal entries are set to the sum of the absolute value of the smallest eigenvalue of the precision matrix with zeros on the diagonal and a constant u.
`ev_xx`	expected proportion of explained variance by the first Principal Component (PC1) of a Principal Component Analysis. This is the largest eigenvalue of the correlation (if `scale_ev=TRUE`) or covariance (if `scale_ev=FALSE`) matrix divided by the sum of eigenvalues. If `ev_xx=NULL` (the default), the constant u is chosen by maximising the contrast of the correlation matrix.
`scale_ev`	logical indicating if the proportion of explained variance by PC1 should be computed from the correlation (`scale_ev=TRUE`) or covariance (`scale_ev=FALSE`) matrix. If `scale_ev=TRUE`, the correlation matrix is used as parameter of the multivariate normal distribution.
`u_list`	vector with two numeric values defining the range of values to explore for constant u.
`tol`	accuracy for the search of parameter u as defined in `optimise`.

Details

For a univariate outcome (`length(pk)=1`), the simulation is done in four steps where (i) predictors contributing to outcome definition are randomly sampled (with probability `nu_xz` for a given predictor to be picked), (ii) the conditional independence structure between the predictors is simulated (with probability `nu_within` for a given pair of predictors to be correlated, conditionally on all other variables), (iii) generation of a precision matrix (inverse covariance matrix) for all variables, where nonzero entries correspond to the predictors contributing to outcome definition or conditional correlation between the predictors, and (iv) data for both predictors and outcome is simulated from a single multivariate Normal distribution using the inverse precision matrix as covariance matrix.

To ensure that the generated precision matrix Ω is positive definite, the diagonal entries are defined as described in `MakePositiveDefinite`. The conditional variance of the outcome Ω_{YY} is chosen so that the proportion of variance in the outcome that is explained by the predictors is `ev_xz`.

For a multivariate outcome (`length(pk)>1`), we introduce independent groups of predictors and orthogonal latent variables (groups are defined in `pk`). Each latent variable is defined as a function of variables belonging to one group of predictors. The precision matrix is defined as described above for univariate outcomes. Subject to the re-ordering of its rows, this precision matrix is block-diagonal, encoding the independence between sets of variables made of (i) the groups of predictors, and (ii) their corresponding latent variable. The outcome variables are then constructed from a linear combination of the latent variables, allowing for contributing predictors belonging to different groups.

The use of latent variables in the multivariate case ensures that we can control the proportion of variance in the latent variable explained by the predictors (`ev_xz`).

Value

A list with:

`xdata`	simulated predictor data.
`ydata`	simulated outcome data.
`proba`	simulated probability of belonging to each outcome class. Only used for `family="binomial"` or `family="multinomial"`.
`logit_proba`	logit of the simulated probability of belonging to each outcome class. Only used for `family="binomial"` or `family="multinomial"`.
`zdata`	simulated data for orthogonal latent outcomes.
`beta`	matrix of true beta coefficients used to generate outcomes in `ydata` from predictors in `xdata`.
`theta`	binary matrix indicating the predictors from `xdata` contributing to the definition of each of the outcome variables in `ydata`.
`eta`	matrix of coefficients used in the linear combination of latent variables from `zdata` to define observed outcomes in `ydata`.
`theta_zy`	binary matrix indicating the latent variables from `zdata` used in the definition of observed outcomes in `ydata`.
`xi`	matrix of true beta coefficients used to generate orthogonal latent outcomes in `zdata` from predictors in `xdata`.
`theta_xz`	binary matrix indicating the predictors from `xdata` contributing to the definition of each of the latent outcome variables in `zdata`.
`omega_xz`	precision matrix for variables in `xdata` and `zdata`.
`adjacency`	binary matrix encoding the conditional independence structure between variables from `xdata` (`var`), `zdata` (`latent`) and `ydata` (`outcome`).

References

Bodinier B, Filippi S, Nost TH, Chiquet J, Chadeau-Hyam M (2021). “Automated calibration for stability selection in penalised regression and graphical models: a multi-OMICs network application exploring the molecular response to tobacco smoking.” `https://arxiv.org/abs/2106.02521`.

See Also

Other simulation functions: `SimulateAdjacency()`, `SimulateComponents()`, `SimulateGraphical()`

Examples

```
oldpar <- par(no.readonly = TRUE)
par(mar = c(5, 5, 5, 5))

## Continuous outcomes

# Univariate outcome
set.seed(1)
simul <- SimulateRegression(pk = 15)
print(simul)
plot(simul)

# Multivariate outcome
set.seed(1)
simul <- SimulateRegression(pk = c(5, 7, 3))
print(simul)
plot(simul)

# Independent predictors
set.seed(1)
simul <- SimulateRegression(pk = c(5, 3), nu_within = 0)
print(simul)
plot(simul)

# Blocks of strongly inter-connected predictors
set.seed(1)
simul <- SimulateRegression(
  pk = c(5, 5), nu_within = 0.5,
  v_within = c(0.5, 1), v_sign = -1, continuous = TRUE, pd_strategy = "min_eigenvalue"
)
print(simul)
Heatmap(
  mat = cor(simul$xdata),
  col = c("navy", "white", "red"),
  legend_range = c(-1, 1)
)
plot(simul)


## Categorical outcomes

# Binary outcome
set.seed(1)
simul <- SimulateRegression(pk = 20, family = "binomial")
print(simul)
table(simul$ydata[, 1])

# Categorical outcome
```

```
set.seed(1)
simul <- SimulateRegression(pk = 20, family = "multinomial")
print(simul)
apply(simul$ydata, 2, sum)

par(oldpar)
```

Index