Package: fake (via r-universe)

September 4, 2024

Title Flexible Data Simulation Using the Multivariate Normal Distribution

Version 1.4.0

Date 2023-04-13

Author Barbara Bodinier [aut, cre]

Maintainer Barbara Bodinier <barbara.bodinier@gmail.com>

Description This R package can be used to generate artificial data conditionally on pre-specified (simulated or user-defined) relationships between the variables and/or observations. Each observation is drawn from a multivariate Normal distribution where the mean vector and covariance matrix reflect the desired relationships. Outputs can be used to evaluate the performances of variable selection, graphical modelling, or clustering approaches by comparing the true and estimated structures (B Bodinier et al (2021) <arXiv:2106.02521>).

License GPL (>= 3)

Language en-GB

Encoding UTF-8

RoxygenNote 7.2.3

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

Suggests testthat (>= 3.0.0),

Config/testthat/edition 3

RdMacros Rdpack

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

RemoteUrl https://github.com/barbarabodinier/fake

RemoteRef HEAD

RemoteSha 37f655cd02eabe0e2535c2f8315f4fc66725bc82

Contents

`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))
```

Concordance *Concordance statistic*

Description

Computes the concordance statistic given observed binary outcomes and predicted probabilities of event. In logistic regression, the concordance statistic is equal to the area under the Receiver Operating Characteristic (ROC) curve and estimates the probability that an individual who experienced the event ($Y_i = 1$) had a higher probability of event than an individual who did not experience the event ($Y_i = 0$).

Usage

```
Concordance(observed, predicted)
```

Arguments

observed	vector of binary outcomes.
predicted	vector of predicted probabilities.

Value

The concordance statistic.

See Also

Other goodness of fit functions: ROC()

Examples

```
# Data simulation
set.seed(1)
proba <- runif(n = 200)
ydata <- rbinom(n = length(proba), size = 1, prob = proba)

# Observed concordance in simulated data
Concordance(observed = ydata, predicted = proba)
```

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)
```

ExpectedCommunities *Expected community structure*

Description

Computes expected metrics related to the community structure of a graph simulated with given parameters.

Usage

```
ExpectedCommunities(pk, nu_within = 0.1, nu_between = 0, nu_mat = NULL)
```

Arguments

pk	vector of the number of variables per group in the simulated dataset. 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 `theta` is not provided.
nu_within	probability of having an edge between two nodes belonging to the same group, as defined in `pk`. 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`). Only used if `nu_mat` is not provided.
nu_between	probability of having an edge between two nodes belonging to different groups, as defined in `pk`. By default, the same density is used for within and between blocks (`nu_within=nu_between`). Only used if `length(pk)>1`. Only used if `nu_mat` is not provided.
nu_mat	matrix of probabilities of having an edge between nodes belonging to a given pair of node groups defined in `pk`.

Details

Given a group of nodes, the within degree d_i^w of node i is defined as the number of nodes from the same group node i is connected to. The between degree d_i^b is the number of nodes from other groups node i is connected to. A weak community in the network is defined as a group of nodes for which the total within degree (sum of the d_i^w for all nodes in the community) is stricly greater than the total between degree (sum of d_i^b for all nodes in the community). For more details, see Network Science by Albert-Laszlo Barabasi.

The expected total within and between degrees for the groups defined in `pk` in a network simulated using `SimulateAdjacency` can be computed given the group sizes (stored in `pk`) and probabilities of having an edge between nodes from a given group pair (defined by `nu_within` and `nu_between` or by `nu_mat`). The expected presence of weak communities can be inferred from these quantities.

The expected modularity, measuring the difference between observed and expected number of within-community edges, is also returned. For more details on this metric, see `modularity`.

Value

A list with:

`total_within_degree_c`
: total within degree by node group, i.e. sum of expected within degree over all nodes in a given group.

`total_between_degree`
: total between degree by node group, i.e. sum of expected between degree over all nodes in a given group.

`weak_community`
: binary indicator for a given node group to be an expected weak community.

`total_number_edges_c`
: matrix of expected number of edges between nodes from a given node pair.

`modularity`
: expected modularity (see `modularity`).

See Also

`SimulateGraphical`, `SimulateAdjacency`, `MinWithinProba`

Examples

```
# Simulation parameters
pk <- rep(20, 4)
nu_within <- 0.8
nu_between <- 0.1

# Expected metrics
expected <- ExpectedCommunities(
  pk = pk,
  nu_within = nu_within,
  nu_between = nu_between
)

# Example of simulated graph
set.seed(1)
```

```
theta <- SimulateAdjacency(
  pk = pk,
  nu_within = nu_within,
  nu_between = nu_between
)

# Comparing observed and expected numbers of edges
bigblocks <- BlockMatrix(pk)
BlockStructure(pk)
sum(theta[which(bigblocks == 2)]) / 2
expected$total_number_edges_c[1, 2]

# Comparing observed and expected modularity
igraph::modularity(igraph::graph_from_adjacency_matrix(theta, mode = "undirected"),
  membership = rep.int(1:length(pk), times = pk)
)
expected$modularity
```

`ExpectedConcordance` *Expected concordance statistic*

Description

Computes the expected concordance statistic given true probabilities of event. In logistic regression, the concordance statistic is equal to the area under the Receiver Operating Characteristic (ROC) curve and estimates the probability that an individual who experienced the event ($Y_i = 1$) had a higher probability of event than an individual who did not experience the event ($Y_i = 0$).

Usage

```
ExpectedConcordance(probabilities)
```

Arguments

`probabilities`	vector of probabilities of event.

Value

The expected concordance statistic.

See Also

`Concordance`

Examples

```
# Simulation of probabilities
set.seed(1)
proba <- runif(n = 1000)

# Expected concordance
ExpectedConcordance(proba)

# Simulation of binary outcome
ydata <- rbinom(n = length(proba), size = 1, prob = proba)

# Observed concordance in simulated data
Concordance(observed = ydata, predicted = proba)
```

`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,
  cex.legend = 1,
  ...
)
```

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.
cex.legend	font size for legend.
...	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(100), ncol = 10)
rownames(mat) <- paste0("r", 1:nrow(mat))
colnames(mat) <- paste0("c", 1:ncol(mat))

# Generating heatmaps
Heatmap(mat = mat)
Heatmap(mat = mat, text = TRUE, format = "f", digits = 2)
Heatmap(
  mat = mat,
  col = c("lightgrey", "blue", "black"),
  legend = FALSE
)

par(oldpar)
```

LayeredDAG *Layered Directed Acyclic Graph*

Description

Returns the adjacency matrix of a layered Directed Acyclic Graph. In this graph, arrows go from all members of a layer to all members of the following layers. There are no arrows between members of the same layer.

Usage

```
LayeredDAG(layers, n_manifest = NULL)
```

Arguments

layers	list of vectors. Each vector in the list corresponds to a layer. There are as many layers as items in the list. Alternatively, this argument can be a vector of the number of variables per layer.
n_manifest	vector of the number of manifest (observed) variables measuring each of the latent variables. If n_manifest is provided, the variables defined in argument layers are considered latent. All entries of n_manifest must be strictly positive.

Value

The adjacency matrix of the layered Directed Acyclic Graph.

Examples

```
# Example with 3 layers specified in a list
layers <- list(
  c("x1", "x2", "x3"),
  c("x4", "x5"),
  c("x6", "x7", "x8")
)
dag <- LayeredDAG(layers)
plot(dag)

# Example with 3 layers specified in a vector
dag <- LayeredDAG(layers = c(3, 2, 3))
plot(dag)
```

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
)
```

MinWithinProba *Within-group probabilities for communities*

Description

Computes the smallest within-group probabilities that can be used to simulate a graph where communities can be expected for given probabilities of between-group probabilities and group sizes.

Usage

```
MinWithinProba(pk, nu_between = 0, nu_mat = NULL)
```

Arguments

pk	vector of the number of variables per group in the simulated dataset. 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 `theta` is not provided.
nu_between	probability of having an edge between two nodes belonging to different groups, as defined in `pk`. By default, the same density is used for within and between blocks (`nu_within=nu_between`). Only used if `length(pk)>1`. Only used if `nu_mat` is not provided.
nu_mat	matrix of probabilities of having an edge between nodes belonging to a given pair of node groups defined in `pk`. Only off-diagonal entries are used.

Details

The vector of within-group probabilities is the smallest one that can be used to generate an expected total within degree D_k^w strictly higher than the expected total between degree D_k^b for all communities k (see `ExpectedCommunities`). Namely, using the suggested within-group probabilities would give expected $D_k^w = D_k^b + 1$.

Value

A vector of within-group probabilities.

See Also

`ExpectedCommunities`, `SimulateAdjacency`, `SimulateGraphical`

Examples

```
# Simulation parameters
pk <- rep(20, 4)
nu_between <- 0.1

# Estimating smallest nu_within
nu_within <- MinWithinProba(pk = pk, nu_between = nu_between)

# Expected metrics
ExpectedCommunities(
  pk = pk,
  nu_within = max(nu_within),
  nu_between = nu_between
)
```

plot.roc_curve *Receiver Operating Characteristic (ROC) curve*

Description

Plots the True Positive Rate (TPR) as a function of the False Positive Rate (FPR) for different thresholds in predicted probabilities.

Usage

```
## S3 method for class 'roc_curve'
plot(x, add = FALSE, ...)
```

Arguments

x	output of ROC.
add	logical indicating if the curve should be added to the current plot.
...	additional plotting arguments (see par).

Value

A base plot.

See Also

ROC, Concordance

Examples

```
# Data simulation
set.seed(1)
simul <- SimulateRegression(
  n = 500, pk = 20,
  family = "binomial", ev_xy = 0.8
)

# Logistic regression
fitted <- glm(simul$ydata ~ simul$xdata, family = "binomial")$fitted.values

# Constructing the ROC curve
roc <- ROC(predicted = fitted, observed = simul$ydata)
plot(roc)
```

ROC *Receiver Operating Characteristic (ROC)*

Description

Computes the True and False Positive Rates (TPR and FPR, respectively) and Area Under the Curve (AUC) by comparing the true (observed) and predicted status using a range of thresholds on the predicted score.

Usage

```
ROC(observed, predicted, n_thr = NULL)
```

Arguments

observed	vector of binary outcomes.
predicted	vector of predicted scores.
n_thr	number of thresholds to use to construct the ROC curve. For faster computations on large data, values below `length(predicted)-1` can be used.

Value

A list with:

TPR	True Positive Rate.
FPR	False Positive Rate.
AUC	Area Under the Curve.

See Also

Other goodness of fit functions: `Concordance()`

Examples

```
# Data simulation
set.seed(1)
simul <- SimulateRegression(
  n = 500, pk = 20,
  family = "binomial", ev_xy = 0.8
)

# Logistic regression
fitted <- glm(simul$ydata ~ simul$xdata, family = "binomial")$fitted.values

# Constructing the ROC curve
roc <- ROC(predicted = fitted, observed = simul$ydata)
plot(roc)
```

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,
  nu_mat = NULL,
  ...
)
```

Arguments

`pk`	vector of the number of variables per group in the simulated dataset. 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 `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`	probability of having an edge between two nodes belonging to the same group, as defined in `pk`. 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`). Only used if `nu_mat` is not provided.
`nu_between`	probability of having an edge between two nodes belonging to different groups, as defined in `pk`. By default, the same density is used for within and between blocks (`nu_within=nu_between`). Only used if `length(pk)>1`. Only used if `nu_mat` is not provided.

nu_mat	matrix of probabilities of having an edge between nodes belonging to a given pair of node groups defined in pk.
...	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: SimulateClustering(), SimulateComponents(), SimulateCorrelation(), SimulateGraphical(), SimulateRegression(), SimulateStructural()

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
```

`SimulateClustering` *Simulation of data with underlying clusters*

Description

Simulates mixture multivariate Normal data with clusters of items (rows) sharing similar profiles along (a subset of) attributes (columns).

Usage

```
SimulateClustering(
  n = c(10, 10),
  pk = 10,
  sigma = NULL,
  theta_xc = NULL,
  nu_xc = 1,
  ev_xc = 0.5,
  output_matrices = FALSE
)
```

Arguments

`n`	vector of the number of items per cluster in the simulated data. The total number of items is `sum(n)`.
`pk`	vector of the number of attributes in the simulated data.
`sigma`	optional within-cluster correlation matrix.
`theta_xc`	optional binary matrix encoding which attributes (columns) contribute to the clustering structure between which clusters (rows). If `theta_xc=NULL`, variables contributing to the clustering are sampled with probability `nu_xc`.
`nu_xc`	expected proportion of variables contributing to the clustering over the total number of variables. Only used if `theta_xc` is not provided.
`ev_xc`	vector of expected proportion of variance in each of the contributing attributes that can be explained by the clustering.
`output_matrices`	logical indicating if the cluster and attribute specific means and cluster specific covariance matrix should be included in the output.

Details

The data is simulated from a Gaussian mixture where for all $i \in 1, \ldots, n$:

$Z_i i.i.d.\ M(1, \kappa)$

$X_i | Z_i indep.\ N_p(\mu_{Z_i}, \Sigma)$

where $M(1, \kappa)$ is the multinomial distribution with parameters 1 and κ, the vector of length G (the number of clusters) with probabilities of belonging to each of the clusters, and $N_p(\mu_{Z_i}, \Sigma)$ is the multivariate Normal distribution with a mean vector μ_{Z_i} that depends on the cluster membership encoded in Z_i and the same covariance matrix Σ within all G clusters.

The mean vectors $\mu_g, g \in 1, \ldots G$ are simulated so that the desired proportion of variance in each of attributes explained by the clustering (argument `ev_xc`) is reached.

The covariance matrix Σ is obtained by re-scaling a correlation matrix (argument `sigma`) to ensure that the desired proportions of explained variances by the clustering (argument `ev_xc`) are reached.

Value

A list with:

`data`	simulated data with `sum(n)` observation and `sum(pk)` variables
`theta`	simulated (true) cluster membership.
`theta_xc`	binary vector encoding variables contributing to the clustering structure.
`ev`	vector of marginal expected proportions of explained variance for each variable.
`mu_mixture`	simulated (true) cluster-specific means. Only returned if `output_matrices=TRUE`.
`sigma`	simulated (true) covariance matrix. Only returned if `output_matrices=TRUE`.

See Also

`MakePositiveDefinite`

Other simulation functions: `SimulateAdjacency()`, `SimulateComponents()`, `SimulateCorrelation()`, `SimulateGraphical()`, `SimulateRegression()`, `SimulateStructural()`

Examples

```
oldpar <- par(no.readonly = TRUE)
par(mar = rep(7, 4))

## Example with 3 clusters

# Data simulation
set.seed(1)
simul <- SimulateClustering(
  n = c(10, 30, 15),
  nu_xc = 1,
  ev_xc = 0.5
)
print(simul)
plot(simul)
```

```
# Checking the proportion of explained variance
x <- simul$data[, 1]
z <- as.factor(simul$theta)
summary(lm(x ~ z)) # R-squared


## Example with 2 variables contributing to clustering

# Data simulation
set.seed(1)
simul <- SimulateClustering(
  n = c(20, 10, 15), pk = 10,
  theta_xc = c(1, 1, rep(0, 8)),
  ev_xc = 0.8
)
print(simul)
plot(simul)

# Visualisation of the data
Heatmap(
  mat = simul$data,
  col = c("navy", "white", "red")
)
simul$ev # marginal proportions of explained variance

# Visualisation along contributing variables
plot(simul$data[, 1:2], col = simul$theta, pch = 19)


## Example with different levels of separation

# Data simulation
set.seed(1)
simul <- SimulateClustering(
  n = c(20, 10, 15), pk = 10,
  theta_xc = c(1, 1, rep(0, 8)),
  ev_xc = c(0.99, 0.5, rep(0, 8))
)

# Visualisation along contributing variables
plot(simul$data[, 1:2], col = simul$theta, pch = 19)


## Example with correlated contributors

# Data simulation
pk <- 10
adjacency <- matrix(0, pk, pk)
adjacency[1, 2] <- adjacency[2, 1] <- 1
set.seed(1)
sigma <- SimulateCorrelation(
  pk = pk,
  theta = adjacency,
```

```
  pd_strategy = "min_eigenvalue",
  v_within = 0.6, v_sign = -1
)$sigma
simul <- SimulateClustering(
  n = c(200, 100, 150), pk = pk, sigma = sigma,
  theta_xc = c(1, 1, rep(0, 8)),
  ev_xc = c(0.9, 0.8, rep(0, 8))
)

# Visualisation along contributing variables
plot(simul$data[, 1:2], col = simul$theta, pch = 19)

# Checking marginal proportions of explained variance
mymodel <- lm(simul$data[, 1] ~ as.factor(simul$theta))
summary(mymodel)$r.squared
mymodel <- lm(simul$data[, 2] ~ as.factor(simul$theta))
summary(mymodel)$r.squared

par(oldpar)
```

`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 dataset.
pk	vector of the number of variables per group in the simulated dataset. 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 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	probability of having an edge between two nodes belonging to the same group, as defined in pk. 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). Only used if nu_mat 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) (if continuous=TRUE) or from proposed values in v_within (diagonal blocks) or v_between (off-diagonal blocks) (if 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 true mean is zero and true variance is one for all simulated variables. The observed mean and variance may be slightly off by chance.
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(), SimulateClustering(), SimulateCorrelation(), SimulateGraphical(), SimulateRegression(), SimulateStructural()

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)
```

SimulateCorrelation *Simulation of a correlation matrix*

Description

Simulates a correlation matrix. This is done in three steps with (i) the simulation of an undirected graph encoding conditional independence, (ii) the simulation of a (positive definite) precision matrix given the graph, and (iii) the re-scaling of the inverse of the precision matrix.

Usage

```
SimulateCorrelation(
  pk = 10,
  theta = NULL,
  implementation = HugeAdjacency,
  topology = "random",
  nu_within = 0.1,
  nu_between = NULL,
  nu_mat = 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,
  output_matrices = FALSE,
  ...
)
```

Arguments

pk	vector of the number of variables per group in the simulated dataset. 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 `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	probability of having an edge between two nodes belonging to the same group, as defined in `pk`. 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`). Only used if `nu_mat` is not provided.
nu_between	probability of having an edge between two nodes belonging to different groups, as defined in `pk`. By default, the same density is used for within and between blocks (`nu_within=nu_between`). Only used if `length(pk)>1`. Only used if `nu_mat` is not provided.
nu_mat	matrix of probabilities of having an edge between nodes belonging to a given pair of node groups defined in `pk`.
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) (if continuous=TRUE) or from proposed values in v_within (diagonal blocks) or v_between (off-diagonal blocks) (if 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.
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

In Step 1, the conditional independence structure between the variables is simulated. This is done using SimulateAdjacency.

In Step 2, the precision 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).

In Step 3, the covariance is calculated as the inverse of the precision matrix. The correlation matrix is then obtained by re-scaling the covariance matrix (see cov2cor).

Value

A list with:

sigma	simulated correlation matrix.
omega	simulated precision matrix. Only returned if output_matrices=TRUE.
theta	adjacency matrix of the simulated graph. Only returned if output_matrices=TRUE.

See Also

SimulatePrecision, MakePositiveDefinite

Other simulation functions: SimulateAdjacency(), SimulateClustering(), SimulateComponents(), SimulateGraphical(), SimulateRegression(), SimulateStructural()

Examples

```
oldpar <- par(no.readonly = TRUE)
par(mar = rep(7, 4))

# Random correlation matrix
set.seed(1)
simul <- SimulateCorrelation(pk = 10)
Heatmap(simul$sigma,
  col = c("navy", "white", "darkred"),
  text = TRUE, format = "f", digits = 2,
  legend_range = c(-1, 1)
)

# Correlation matrix with homogeneous block structure
set.seed(1)
simul <- SimulateCorrelation(
  pk = c(5, 5),
  nu_within = 1,
  nu_between = 0,
  v_sign = -1,
  v_within = 1
)
Heatmap(simul$sigma,
  col = c("navy", "white", "darkred"),
  text = TRUE, format = "f", digits = 2,
  legend_range = c(-1, 1)
)

# Correlation matrix with heterogeneous block structure
set.seed(1)
simul <- SimulateCorrelation(
  pk = c(5, 5),
  nu_within = 0.5,
  nu_between = 0,
  v_sign = -1
)
Heatmap(simul$sigma,
  col = c("navy", "white", "darkred"),
  text = TRUE, format = "f", digits = 2,
  legend_range = c(-1, 1)
)

par(oldpar)
```

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,
  nu_mat = 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 dataset.
pk	vector of the number of variables per group in the simulated dataset. 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 `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 — probability of having an edge between two nodes belonging to the same group, as defined in pk. 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). Only used if nu_mat is not provided.

nu_between — probability of having an edge between two nodes belonging to different groups, as defined in pk. By default, the same density is used for within and between blocks (nu_within=nu_between). Only used if length(pk)>1. Only used if nu_mat is not provided.

nu_mat — matrix of probabilities of having an edge between nodes belonging to a given pair of node groups defined in pk.

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) (if continuous=TRUE) or from proposed values in v_within (diagonal blocks) or v_between (off-diagonal blocks) (if 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 true mean is zero and true variance is one for all simulated variables. The observed mean and variance may be slightly off by chance.
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

Other simulation functions: SimulateAdjacency(), SimulateClustering(), SimulateComponents(), SimulateCorrelation(), SimulateRegression(), SimulateStructural()

Examples

```
oldpar <- par(no.readonly = TRUE)
par(mar = rep(7, 4))

# 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
)

par(oldpar)
```

`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 dataset. 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 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) (if continuous=TRUE) or from proposed values in v_within (diagonal blocks) or v_between (off-diagonal blocks) (if 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,
  xdata = NULL,
  family = "gaussian",
  q = 1,
  theta = NULL,
  nu_xy = 0.2,
  beta_abs = c(0.1, 1),
  beta_sign = c(-1, 1),
  continuous = TRUE,
  ev_xy = 0.7
)
```

Arguments

`n`	number of observations in the simulated dataset. Not used if `xdata` is provided.
`pk`	number of predictor variables. A subset of these variables contribute to the outcome definition (see argument `nu_xy`). Not used if `xdata` is provided.
`xdata`	optional data matrix for the predictors with variables as columns and observations as rows. A subset of these variables contribute to the outcome definition (see argument `nu_xy`).
`family`	type of regression model. Possible values include `"gaussian"` for continuous outcome(s) or `"binomial"` for binary outcome(s).
`q`	number of outcome variables.

theta — binary matrix with as many rows as predictors and as many columns as outcomes. A nonzero entry on row i and column j indicates that predictor i contributes to the definition of outcome j.

nu_xy — vector of length q with expected proportion of predictors contributing to the definition of each of the q outcomes.

beta_abs — vector defining the range of nonzero regression coefficients in absolute values. If continuous=FALSE, beta_abs is the set of possible precision values. If continuous=TRUE, beta_abs is the range of possible precision values. Note that regression coefficients are re-scaled if family="binomial" to ensure that the desired concordance statistic can be achieved (see argument ev_xy) so they may not be in this range.

beta_sign — vector of possible signs for regression coefficients. Possible inputs are: 1 for positive coefficients, -1 for negative coefficients, or c(-1, 1) for both positive and negative coefficients.

continuous — logical indicating whether to sample regression coefficients from a uniform distribution between the minimum and maximum values in beta_abs (if continuous=TRUE) or from proposed values in beta_abs (if continuous=FALSE).

ev_xy — vector of length q with expected goodness of fit measures for each of the q outcomes. If family="gaussian", the vector contains expected proportions of variance in each of the q outcomes that can be explained by the predictors. If family="binomial", the vector contains expected concordance statistics (i.e. area under the ROC curve) given the true probabilities.

Value

A list with:

xdata — input or simulated predictor data.

ydata — simulated outcome data.

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.

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(), SimulateClustering(), SimulateComponents(), SimulateCorrelation(), SimulateGraphical(), SimulateStructural()

Examples

```
## Independent predictors

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

# Univariate binary outcome
set.seed(1)
simul <- SimulateRegression(pk = 15, family = "binomial")
table(simul$ydata)

# Multiple continuous outcomes
set.seed(1)
simul <- SimulateRegression(pk = 15, q = 3)
summary(simul)


## Blocks of correlated predictors

# Simulation of predictor data
set.seed(1)
xsimul <- SimulateGraphical(pk = rep(5, 3), nu_within = 0.8, nu_between = 0, v_sign = -1)
Heatmap(cor(xsimul$data),
  legend_range = c(-1, 1),
  col = c("navy", "white", "darkred")
)

# Simulation of outcome data
simul <- SimulateRegression(xdata = xsimul$data)
print(simul)
summary(simul)


## Choosing expected proportion of explained variance

# Data simulation
set.seed(1)
simul <- SimulateRegression(n = 1000, pk = 15, q = 3, ev_xy = c(0.9, 0.5, 0.2))
summary(simul)

# Comparing with estimated proportion of explained variance
summary(lm(simul$ydata[, 1] ~ simul$xdata))
summary(lm(simul$ydata[, 2] ~ simul$xdata))
summary(lm(simul$ydata[, 3] ~ simul$xdata))


## Choosing expected concordance (AUC)

# Data simulation
set.seed(1)
```

```
simul <- SimulateRegression(
  n = 500, pk = 10,
  family = "binomial", ev_xy = 0.9
)

# Comparing with estimated concordance
fitted <- glm(simul$ydata ~ simul$xdata,
  family = "binomial"
)$fitted.values
Concordance(observed = simul$ydata, predicted = fitted)
```

`SimulateStructural` *Data simulation for Structural Causal Modelling*

Description

Simulates data from a multivariate Normal distribution where relationships between the variables correspond to a Structural Causal Model (SCM). To ensure that the generated SCM is identifiable, the nodes are organised by layers, with no causal effects within layers.

Usage

```
SimulateStructural(
  n = 100,
  pk = c(5, 5, 5),
  theta = NULL,
  n_manifest = NULL,
  nu_between = 0.5,
  v_between = c(0.5, 1),
  v_sign = c(-1, 1),
  continuous = TRUE,
  ev = 0.5,
  ev_manifest = 0.8,
  output_matrices = FALSE
)
```

Arguments

`n`	number of observations in the simulated dataset.
`pk`	vector of the number of (latent) variables per layer.
`theta`	optional binary adjacency matrix of the Directed Acyclic Graph (DAG) of causal relationships. This DAG must have a structure with layers so that a variable can only be a parent of variable in one of the following layers (see `LayeredDAG` for examples). The layers must be provided in `pk`.

n_manifest	vector of the number of manifest (observed) variables measuring each of the latent variables. If n_manifest=NULL, there are sum(pk) manifest variables and no latent variables. Otherwise, there are sum(pk) latent variables and sum(n_manifest) manifest variables. All entries of n_manifest must be strictly positive.
nu_between	probability of having an edge between two nodes belonging to different layers, as defined in pk. If length(pk)=1, this is the expected density of the graph.
v_between	vector defining the (range of) nonzero path coefficients. If continuous=FALSE, v_between is the set of possible values. If continuous=TRUE, v_between is the range of possible values.
v_sign	vector of possible signs for path coefficients. Possible inputs are: 1 for positive coefficients, -1 for negative coefficients, or c(-1, 1) for both positive and negative coefficients.
continuous	logical indicating whether to sample path coefficients from a uniform distribution between the minimum and maximum values in v_between (if continuous=FALSE) or from proposed values in v_between (if continuous=FALSE).
ev	vector of proportions of variance in each of the (latent) variables that can be explained by its parents. If there are no latent variables (if n_manifest=NULL), these are the proportions of explained variances in the manifest variables. Otherwise (if n_manifest is provided), these are the proportions of explained variances in the latent variables.
ev_manifest	vector of proportions of variance in each of the manifest variable that can be explained by its latent parent. Only used if n_manifest is provided.
output_matrices	logical indicating if the true path coefficients, residual variances, and precision and (partial) correlation matrices should be included in the output.

Value

A list with:

data	simulated data with n observations for manifest variables.
theta	adjacency matrix of the simulated Directed Acyclic Graph encoding causal relationships.
Amat	simulated (true) asymmetric matrix A in RAM notation. Only returned if output_matrices=TRUE.
Smat	simulated (true) symmetric matrix S in RAM notation. Only returned if output_matrices=TRUE.
Fmat	simulated (true) filter matrix F in RAM notation. Only returned if output_matrices=TRUE.
sigma	simulated (true) covariance matrix. Only returned if output_matrices=TRUE.

References

Jacobucci R, Grimm KJ, McArdle JJ (2016). "Regularized structural equation modeling." *Structural equation modeling: a multidisciplinary journal*, **23**(4), 555–566. doi:10.1080/10705511.2016.1154793.

See Also

SimulatePrecision, MakePositiveDefinite, Contrast

Other simulation functions: SimulateAdjacency(), SimulateClustering(), SimulateComponents(), SimulateCorrelation(), SimulateGraphical(), SimulateRegression()

Examples

```
# Simulation of a layered SCM
set.seed(1)
pk <- c(3, 5, 4)
simul <- SimulateStructural(n = 100, pk = pk)
print(simul)
summary(simul)
plot(simul)

# Choosing the proportions of explained variances for endogenous variables
set.seed(1)
simul <- SimulateStructural(
  n = 1000,
  pk = c(2, 3),
  nu_between = 1,
  ev = c(NA, NA, 0.5, 0.7, 0.9),
  output_matrices = TRUE
)

# Checking expected proportions of explained variances
1 - simul$Smat["x3", "x3"] / simul$sigma["x3", "x3"]
1 - simul$Smat["x4", "x4"] / simul$sigma["x4", "x4"]
1 - simul$Smat["x5", "x5"] / simul$sigma["x5", "x5"]

# Checking observed proportions of explained variances (R-squared)
summary(lm(simul$data[, 3] ~ simul$data[, which(simul$theta[, 3] != 0)]))
summary(lm(simul$data[, 4] ~ simul$data[, which(simul$theta[, 4] != 0)]))
summary(lm(simul$data[, 5] ~ simul$data[, which(simul$theta[, 5] != 0)]))

# Simulation including latent and manifest variables
set.seed(1)
simul <- SimulateStructural(
  n = 100,
  pk = c(2, 3),
  n_manifest = c(2, 3, 2, 1, 2)
)
plot(simul)

# Showing manifest variables in red
if (requireNamespace("igraph", quietly = TRUE)) {
  mygraph <- plot(simul)
  ids <- which(igraph::V(mygraph)$name %in% colnames(simul$data))
  igraph::V(mygraph)$color[ids] <- "red"
  igraph::V(mygraph)$frame.color[ids] <- "red"
  plot(mygraph)
}

# Choosing proportions of explained variances for latent and manifest variables
set.seed(1)
simul <- SimulateStructural(
  n = 100,
  pk = c(3, 2),
```

```
  n_manifest = c(2, 3, 2, 1, 2),
  ev = c(NA, NA, NA, 0.7, 0.9),
  ev_manifest = 0.8,
  output_matrices = TRUE
)
plot(simul)

# Checking expected proportions of explained variances
(simul$sigma_full["f4", "f4"] - simul$Smat["f4", "f4"]) / simul$sigma_full["f4", "f4"]
(simul$sigma_full["f5", "f5"] - simul$Smat["f5", "f5"]) / simul$sigma_full["f5", "f5"]
(simul$sigma_full["x1", "x1"] - simul$Smat["x1", "x1"]) / simul$sigma_full["x1", "x1"]
```

Index