Package 'SeqNet'

December 18, 2019

Title Generate RNA-Seq Data from Gene-Gene Association Networks

Version 1.1.0

Author Tyler Grimes [aut, cre],
Somnath Datta [aut]

Maintainer Tyler Grimes <tyler.grimes@ufl.edu>

Description Methods to generate random gene-gene association networks and simulate RNA-seq data from them. Includes functions to generate random networks of any size and perturb them to obtain differential networks. Network objects are built from individual, overlapping modules that represent pathways. The resulting network has various topological properties that are characteristic of gene regulatory networks. RNA-seq data can be generated such that the association among gene expression profiles reflect the underlying network. A reference RNA-seq dataset can be provided to model realistic marginal distributions. Plotting functions are available to visualize a network, compare two networks, and compare the expression of two genes across multiple networks.

Depends R (>= 3.6.0)

Imports fitdistrplus, ggplot2, grDevices, graphics, igraph, mvtnorm, purrr, RColorBrewer, tibble, Rcpp, rlang, stats, utils, methods

Suggests knitr, rmarkdown, testthat

License GPL-2 | GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 6.1.1

LinkingTo Rcpp

NeedsCompilation yes

Repository CRAN

Date/Publication 2019-12-18 18:40:02 UTC

R topics documented:

`add_modules_to_network` *Internal function for adding a set of modules to the network*

Description

Internal function for adding a set of modules to the network

Usage

```
add_modules_to_network(network, module_list)
```

Arguments

`network`	The network to modify.
`module_list`	A list of 'network_module' objects to add to the network.

Value

The modified network.

add_random_module_to_network *Adds a random module of a given size to the network*

Description

Adds a random module of a given size to the network

Usage

```
add_random_module_to_network(network, module_size, ...)
```

Arguments

network	The 'network' object to modify.
module_size	The size of the module to generate.
...	Additional arguments passed into random_module().

Value

The modified 'network' object.

Examples

```
# This function provides an alternative way to iteratively add random
# modules to the network. It uses a weighted sampling of nodes, where
# nodes that haven't been selected for a module have a higher probability
# of being sampled for the new module.
nw <- create_empty_network(100)
plot(nw) # An empty network of 100 nodes.
# Add random modules of size 10 to the network, 1 at a time.
# By plotting the network each time, we can watch it grow.
set.seed(12345)
plot(nw <<- add_random_module_to_network(nw, 10))
plot(nw <<- add_random_module_to_network(nw, 10))
plot(nw <<- add_random_module_to_network(nw, 10))
plot(nw <<- add_random_module_to_network(nw, 10))
plot(nw <<- add_random_module_to_network(nw, 10))
plot(nw <<- add_random_module_to_network(nw, 10))
# Etc.
```

`all_networks_contain_same_modules`
Internal function to check if a list of networks all contain the same modules.

Description

Internal function to check if a list of networks all contain the same modules.

Usage

```
all_networks_contain_same_modules(network_list)
```

Arguments

`network_list`	A list of 'network' objects.

Value

A logical value; `TRUE` indicates the networks contain the same modules, and `FALSE` indicates otherwise. Note, this only checks that the modules contain the same nodes - the structure of the modules are allowed to differ.

`all_networks_contain_same_nodes`
Internal function to check if a list of networks all contain the same nodes.

Description

Internal function to check if a list of networks all contain the same nodes.

Usage

```
all_networks_contain_same_nodes(network_list)
```

Arguments

`network_list`	A list of 'network' objects.

Value

A logical value; `TRUE` indicates the networks contain the same nodes, and `FALSE` indicates otherwise.

as_single_module *Collapses all modules in network into a single module*

Description

This modification can be used if it is desired to simulate from a single GGM rather than averaging over the GGMs for each module.

Usage

```
as_single_module(network)
```

Arguments

network	The 'network' object to modify

Value

The modified 'network' object.

Examples

```
# This function can be used prior to generating weights for the network
# connections. With multiple modules in the network, the weighted network may
# gain conditional dependencies between nodes across modules. If the network
# is reduced to a single module prior to generating weights, then the
# weighted and unweighted networks will maintain the same structure.
nw <- random_network(20, n_modules = 3)
g <- plot(nw)
nw <- gen_partial_correlations(nw)
plot(nw, g) # Additional edges appear from conditional dependencies across modules.
nw <- remove_weights(nw) # Remove weights to avoid warning message in next call.
nw <- as_single_module(nw)
nw <- gen_partial_correlations(nw)
plot(nw, g) # With only one module, the weighted network has the same structure.
```

check_adjacency_cpp *C++ implementation to check if a matrix is an adjacency matrix*

Description

C++ implementation to check if a matrix is an adjacency matrix

Usage

```
check_adjacency_cpp(m)
```

Arguments

m	A matrix to check.

Value

Returns 0 if the matrix is an adjacency matrix. If the matrix is not square, returns 1; if the diagonal entries are not all zero, returns 2; if the matrix is not symmetric, returns 3; if the matrix contains values other than 0 or 1, returns 4.

---

components_in_adjacency

C++ implementation to obtain connected components in a graph.

Description

C++ implementation to obtain connected components in a graph.

Usage

```
components_in_adjacency(adj)
```

Arguments

adj	An adjacency matrix.

Value

Returns a matrix with 2 columns containing the indicies in the lower-triangle of the matrix that are nonzero.

---

connect_module_structure

Connect disconnected components in an adjacency matrix

Description

Connect disconnected components in an adjacency matrix

Usage

```
connect_module_structure(adj, weights = NULL, alpha = 100, beta = 1,
  epsilon = 10^-5)
```

Arguments

adj	An adjacency matrix to modify.
weights	(Optional) weights used for sampling nodes.
alpha	A positive value used to parameterize the Beta distribution.
beta	A positive value used to parameterize the Beta distribution.
epsilon	A small constant added to the sampling probability of each node.

Value

A modified adjacency matrix

Note

This function is used in random_module_structure to reconnect any disconnected components after edge removal and rewiring. When connecting two components, a node is sampled from each component with probability that is dependent on node degree; those two nodes are then connected, which connects the components.

Examples

```
# This function is used in `random_module_structure()` to reconnect any
# disconnected components. To demonstrate, we'll create a random structure,
# remove connections to one of the nodes (that node will then be a disconnected
# component), and use `connect_module_structure()` to reconnect it back to
# the main component.
adj <- random_module_structure(10)
adj <- remove_connections_to_node(adj, 1, prob_remove = 1)
# Note that there are now two components in the network:
components_in_adjacency(adj)
g <- plot_network(adj)
# After connecting, the network contains one component.
adj <- connect_module_structure(adj)
components_in_adjacency(adj)
plot_network(adj, g)
```

---

create_cytoscape_file *Create an edge table file for Cytoscape*

Description

The returned data frame can be saved as a .csv file. Then, in Cytopscape use File -> Import -> Network -> File. Select the .csv file containing the data frame generated by this function. There will be a popup window. The source, interaction, and target columns should automatically be identified. Click OK.

Usage

```
create_cytoscape_file(g)
```

Arguments

g	A 'network_plot' object. See plot_network.

Examples

```
nw <- random_network(10)
g <- plot(nw)
nw_plot_cytoscape <- create_cytoscape_file(g)

# Save the edge table in a .csv file to be used in cytoscape.
write.table(nw_plot_cytoscape, file.path(tempdir(), "file_name.csv"),
            sep = ",", row.names = FALSE, col.names = TRUE, quote = FALSE)
```

---

create_empty_module *Create a module*

Description

Create a module

Usage

```
create_empty_module(nodes)
```

Arguments

nodes	A numeric vector indicating which nodes in the network are contained in this module.

Value

A 'network_module' object.

Examples

```
module <- create_empty_module(1:10)
plot(module) # A module with no edges.
```

create_empty_network *Create a network object.*

Description

Creates a 'network' object containing no modules.

Usage

```
create_empty_network(p)
```

Arguments

p	The number of nodes in the network

Value

A network object.

Examples

```
nw <- create_empty_network(10)
plot(nw) # A network with no edges.
```

create_modules_for_network *Randomly sample subsets of genes for each module*

Description

Creates a collection of modules containing randomly samples genes.

Usage

```
create_modules_for_network(n_modules, p, avg_module_size = 50,
  sd_module_size = 50, min_module_size = 10, max_module_size = 200,
  sample_link_nodes_fn = sample_link_nodes,
  sample_module_nodes_fn = sample_module_nodes, ...)
```

Arguments

n_modules	The number of modules to include in the network.
p	The number of nodes in the network.
avg_module_size	The average number of nodes in a module.
sd_module_size	The standard deviation of module size.
min_module_size	The minimum number of nodes in a module.
max_module_size	A positive value. Any generated module sizes above this value will be reduced to 'max_module_size'. Set to 'Inf' to avoid this truncation.
sample_link_nodes_fn	A function used for sampling link nodes for a new module.
sample_module_nodes_fn	A function used for sampling nodes for a new module.
...	Additional arguments passed to random_module.

Value

A list containing the indicies for genes contained in each module.

Examples

```
# Create a two modules (having random structures and sizes) from a pool
# of 100 nodes.
create_modules_for_network(n_modules = 2, p = 100)
# Set n_modules = NULL to continue making modules until all nodes have
# been selected at least once.
create_modules_for_network(n_modules = NULL, p = 100)
```

---

create_module_from_adjacency_matrix

Create a module from an adjacency matrix

Description

The edges in the module will be set to the edges in the adjacency matrix. The edges are undirected, and only the lower triangle of the matrix is considered. See set_module_edges for more details.

Usage

```
create_module_from_adjacency_matrix(adjacency_matrix, nodes = NULL,
  module_name = NULL, run_checks = TRUE)
```

Arguments

adjacency_matrix	The adjacency matrix used to create the module.
nodes	A numeric vector indicating which nodes in the network are contained in this module.
module_name	(optional) Character string specifying the name of the module. If NULL, the module will be unnamed.
run_checks	If TRUE, then the adjacency_matrix argument is checked.

Value

A 'network_module' object.

Examples

```
nw <- random_network(10)
nw <- gen_partial_correlations(nw)
adj_mat <- get_adjacency_matrix(nw)
create_module_from_adjacency_matrix(adj_mat)
```

---

create_module_from_association_matrix

Create a module from an association matrix

Description

The edge weights in the module will be set to the corresponding values in the association matrix. The edges are undirected, and only the lower triangle of the matrix is considered. See set_module_weights for more details.

Usage

```
create_module_from_association_matrix(association_matrix, nodes = NULL,
  module_name = NULL)
```

Arguments

association_matrix	The association matrix used to create the module.
nodes	A numeric vector indicating which nodes in the network are contained in this module.
module_name	(optional) Character string specifying the name of the module. If NULL, the module will be unnamed.

Value

A 'network_module' object.

Examples

```
nw <- random_network(10)
nw <- gen_partial_correlations(nw)
assoc_mat <- get_association_matrix(nw)
create_module_from_association_matrix(assoc_mat)
```

`create_network_from_adjacency_matrix` *Create a network object from an adjacency matrix*

Description

Create a network object from an adjacency matrix

Usage

```
create_network_from_adjacency_matrix(adjacency_matrix, ...)
```

Arguments

`adjacency_matrix`	The adjacency matrix for the network. Since the adjacency matrix only provides information on the global connections, the resulting 'network' object will consist of a single module containing these connections.
`...`	Additional arguments passed to `create_module_from_adjacency_matrix`.

Value

A network object.

Examples

```
adj_mat <- random_module_structure(10)
nw <- create_network_from_adjacency_matrix(adj_mat)
all(adj_mat == get_adjacency_matrix(nw))
```

create_network_from_association_matrix

Create a network object from an association matrix

Description

Create a network object from an association matrix

Usage

```
create_network_from_association_matrix(association_matrix, ...)
```

Arguments

association_matrix	The association matrix for the network. Since the association matrix only provides information on the global connections, the resulting 'network' object will consist of a single weighted module containing these connections. The edge weights, i.e. the partial correlations, will correspond to the nonzero values in the matrix.
...	Additional arguments passed to create_module_from_association_matrix.

Value

A network object.

Examples

```
# Create a random weighted network and extract the association matrix from it.
nw <- random_network(10)
nw <- gen_partial_correlations(nw)
assoc_mat <- get_association_matrix(nw)
# Any association matrix can be used to directly create a network object.
# However, the created network will only contain one module.
nw_from_assoc <- create_network_from_association_matrix(assoc_mat)
all(get_adjacency_matrix(nw) == get_adjacency_matrix(nw_from_assoc))
```

create_network_from_modules

Create a network object.

Description

Generates a 'network' object from a list of 'network_modules', The modules are assumed to have their local network structure already generated. Individual modules can be generated using the random_module function.

Usage

```
create_network_from_modules(p, module_list,
  node_names = as.character(1:p), ...)
```

Arguments

p	The number of nodes in the graph
module_list	A named list of 'network_module' objects.
node_names	(optional) Vector of strings providing names for each node in the graph. Default names are "1", "2", ..., "p".
...	Arguments to be passed to other methods. Possible arguments include:

prob_rewire	The probability of removing a connection from the local network structure; this is applied to each edge created
prob_remove	The probability of rewiring a connection from the local network structure; this is applied every connection of e
neig_size	The initial degree of each node when constructing the ring lattice. See random_module_structure.
alpha	A positive value used to parameterize the Beta distribution used to sample nodes based on their degree. Larger
beta	A positive value used to parameterize the Beta distribution used to sample nodes based on their degree. Set to
epsilon	A small constant added to the sampling probability of each node. See random_module_structure.

Value

A network object.

Examples

```
# Networks can be crafted manually by first constructing the individual
# modules, then putting them together to create a network.
module_1 <- random_module(1:10) # A module containing nodes 1-10
module_2 <- random_module(5:15) # A module containing nodes 5-15
# Create a network containing 20 nodes and the two modules.
nw <- create_network_from_modules(20, list(module_1, module_2))
nw
# Note: nodes 16-20 are not in a module, so they have no connections.
plot(nw)
```

---

dzinb *The Zero-Inflated Negative Binomial Distribution*

Description

The Zero-Inflated Negative Binomial Distribution

Usage

```
dzinb(x, size, mu, rho = 0, log = FALSE)
```

Arguments

x	A vector of quantities.
size	The dispersion paramater used in dnbinom.
mu	The mean parameter used in dnbinom.
rho	The zero-inflation parameter.
log	Logical; if TRUE, then log(d) is returned.

Value

The value(s) of the density function evaluated at x.

Examples

```
x <- rzinb(10, 1, 10, 0.1)
p <- pzinb(x, 1, 10, 0.1)
y <- qzinb(p, 1, 10, 0.1)
all(x == y)
# Compute P(0 < X < 5) for X ~ ZINB(1, 10, 0.1)
sum(dzinb(0:5, 1, 10, 0.1))
```

---

ecdf_cpp *C++ implementation of empirical CDF*

Description

Constructs the empirical CDF, F, for a set of observations, x, and returns F(x).

Usage

```
ecdf_cpp(x)
```

Arguments

x	The observation to construct the empirical CDF from.

Value

Returns the values for F(x).

edges_from_adjacency_cpp *C++ implementation for obtaining an edge list from adjacency matrix*

Description

C++ implementation for obtaining an edge list from adjacency matrix

Usage

```
edges_from_adjacency_cpp(adj)
```

Arguments

adj	An adjacency matrix.

Value

Returns a matrix with 2 columns containing the indicies in the lower-triangle of the matrix that are nonzero.

---

est_params_from_reference *Estimate ZINB parameters from reference data*

Description

The observations in the reference dataset should be as homogeneous as possible. For example, we should not expect differential expression or differential connectivity of genes within the sample. If the data are heterogeneous, the estimation of the parameters may be unreliable.

Usage

```
est_params_from_reference(reference, verbose = TRUE)
```

Arguments

reference	Either a vector or data.frame of counts from a reference gene expression profile. If a data.frame is provided, each column should correspond to a gene.
verbose	Boolean indicator for message output.

Value

Returns a list containing a matrix of parameter estimates 'size', 'mu', and 'rho' for each gene in the reference, and the reference dataset used. The parameter matrix can be used in gen_zinb.

Examples

```
# The internal reference dataset already contains ZINB parameter estimates,
# so running est_params_from_reference() is not necessary. To simulate
# ZINB data from a different RNA-seq reference dataset, the data can
# be passed into gen_zinb() directly using the 'reference' argument, and
# est_params_from_reference() will be used automatically (i.e. the user
# does not need to call this function directly).

# An example using the reference dataset
data(reference)
# The RNA-seq dataset should have samples as rows and genes as columns:
rnaseq <- reference$rnaseq
# Estimate ZINB params for first ten genes.
params <- est_params_from_reference(rnaseq[, 1:10])$params
# However, the previous call is not needed for simulated ZINB data.
# The RNA-seq dataset can be passed directly to `gen_zinb()`.
nw <- random_network(10)
x <- gen_zinb(20, nw, reference = rnaseq[, 1:10])$x # Pass in 'rnaseq' directly.
```

---

gen_gaussian *Generate observations from a Gaussian graphical model.*

Description

Generates data based on the multivariate normal distribution parameterized by a zero mean vector and a covariance matrix. Observations are generated for each module in the network individually, and the covariance matrix is set to the inverse of the standardized association matrix for the module. Observations are combined for gene i by taking the sum across the m_i modules containing it and dividing by sqrt(m_i).

Usage

```
gen_gaussian(n, ...)
```

Arguments

n	The number of samples to generate. If multiple networks are provided, n samples are generated per network.
...	The 'network' object(s) to generate data from. Can be a single network, many networks, or a single list of networks.

Value

A list containing the n by p matrix of samples and the 'network' object used to generate them.

Examples

```
nw <- random_network(10) # Create a random network with 10 nodes.
nw <- gen_partial_correlations(nw) # Add weights to connections in the network.
x <- gen_gaussian(20, nw)$x # Simulate 20 Gaussian observations from network.
```

---

gen_partial_correlations

Generate partial correlations for a list of networks.

Description

Random partial correlations are generated to weigh the network connections. If multiple networks are provided, the networks must contain the same nodes and the same modules (the connections within modules may differ). Any connection that is common across different networks will also have the same partial correlation weight across networks.

Usage

```
gen_partial_correlations(..., k = 2.5, rweights = function(n)
  (-1)^rbinom(n, 1, 0.5) * runif(n, 0.5, 1))
```

Arguments

...	The 'network' objects to modify.
k	An positive number used to ensure that the matrix inverse is numerically stable. k = 2.5 is the default value; higher values will allow for larger values of partial correlations (and will result in a wider distribution of Pearson correlations).
rweights	A generator for initial weights in the network. By default, values are generated uniformly from (-1, -0.5) U (0.5, 1). The weights will be adjusted so that the sign of a generated weight and the sign of the corresponding partial correlation agree.

Value

An updated network object containing random weights. If multiple networks were provided, then a list of network objects is returned.

Examples

```
nw <- random_network(10) # Create a random network with 10 nodes.
nw <- gen_partial_correlations(nw) # Add weights to connections in the network.
```

gen_rnaseq *Generate RNA-seq data from an underlying network*

Description

The expression data are generated based on the gene-gene associations of an underlying network. An association structure is imposed by first generating data from a multivariate Gaussian distribution. Those data are then used to sample from the empirical distribution of gene expression profiles in the reference dataset using the inverse transform method.

Usage

```
gen_rnaseq(n, network, reference = NULL, verbose = TRUE)
```

Arguments

n	The number of samples to generate.
network	A 'network' object or list of 'network' objects.
reference	A data.frame containing reference gene expression data. Rows should correspond to samples and columns to genes. If NULL, then the kidney dataset is used.
verbose	Boolean indicator for message output.

Value

A list containing the simulated expression data and the reference dataset. If a list of networks were provided, then the results for each network are returned as a list.

Examples

```
nw <- random_network(10) # Create a random network with 10 nodes.
nw <- gen_partial_correlations(nw) # Add weights to connections in the network.
# If no reference is provided, the internal RNA-seq reference dataset is used.
x <- gen_rnaseq(20, nw)$x # Simulate 20 observations from the network.
```

gen_zinb *Generate ZINB counts from an underlying network*

Description

The count data are generated based on the gene-gene associations of an udnerlying network. An association structure is imposed by first generating data from a multivariate Gaussian distribution, and counts are then obtained through the inverse tranformation method. To generate realistic counts, either a reference dataset or parameters for the ZINB model (size, mu, rho) can be provided. Parameters can be estimated from a reference using the est_params_from_reference function.

Usage

```
gen_zinb(n, network, reference = NULL, params = NULL,
  library_sizes = NULL, adjust_library_size = NULL, verbose = TRUE)
```

Arguments

n	The number of samples to generate.
network	A 'network' object or list of 'network' objects.
reference	Either a vector or data.frame of counts from a reference gene expression profile. If a data.frame is provided, each column should correspond to a gene. If both reference and params are NULL, then parameters are estimated from the kidney dataset.
params	A matrix of ZINB parameter values; each column should contain the size, mu, and rho parameters for a gene.
library_sizes	A vector of library sizes. Used only if reference is NULL.
adjust_library_size	A boolean value. If TRUE, the library size of generated counts are adjusted based on the reference library sizes. If both reference and library_sizes is NULL, then no adjustment is made. By default, this adjustment is made if the necessary information is provided.
verbose	Boolean indicator for message output.

Value

A list containing the generated counts and the ZINB parameters used to create them. If a list of networks were provided, then the results for each network are returned as a list.

Examples

```
nw <- random_network(10) # Create a random network with 10 nodes.
nw <- gen_partial_correlations(nw) # Add weights to connections in the network.
# If no reference is provided, ZINB data are generated using an internal reference.
x <- gen_zinb(20, nw)$x # Simulate 20 ZINB observations from the network.
```

---

get_adjacency_matrix *Get adjacency matrix*

Description

The adjacency matrix is constructed from all modules in a network.

Usage

```
get_adjacency_matrix(x, ...)
```

Arguments

x	Either a 'network', 'network_module', or 'matrix' object.
...	Additional arguments.

Value

An adjacency matrix with entry ij = 1 if node i and j are connected, and 0 otherwise. The diagonal entries are all zero.

Note

The connections in an adjacency matrix and association matrix may differ if the network contains multiple modules. The adjacency matrix only considers direct connections in the network, whereas the association matrix takes into account the fact that overlapping modules can create conditional dependencies between two genes in seperate modules (i.e. genes that don't have a direct connection in the graph).

Examples

```
# Create a random network with 10 nodes and add random edge weights.
nw <- random_network(10)
nw <- gen_partial_correlations(nw)
# Get adjacency matrix for the network or individual modules in the network.
get_adjacency_matrix(nw)
module <- nw$modules[[1]]
get_adjacency_matrix(module)
```

---

get_adjacency_matrix.default
Get adjacency matrix

Description

The adjacency matrix is constructed from all modules in a network.

Usage

```
## Default S3 method:
get_adjacency_matrix(x, ...)
```

Arguments

x	Either a 'network', 'network_module', or 'matrix' object.
...	Additional arguments.

Value

An adjacency matrix with entry ij = 1 if node i and j are connected, and 0 otherwise. The diagonal entries are all zero.

Examples

```
# Create a random network with 10 nodes and add random edge weights.
nw <- random_network(10)
nw <- gen_partial_correlations(nw)
# Get adjacency matrix for the network or individual modules in the network.
get_adjacency_matrix(nw)
module <- nw$modules[[1]]
get_adjacency_matrix(module)
```

---

get_adjacency_matrix.matrix

Get adjacency matrix

Description

The adjacency matrix is constructed from all modules in a network.

Usage

```
## S3 method for class 'matrix'
get_adjacency_matrix(x, ...)
```

Arguments

x	Either a 'network', 'network_module', or 'matrix' object.
...	Additional arguments.

Value

An adjacency matrix with entry ij = 1 if node i and j are connected, and 0 otherwise. The diagonal entries are all zero.

Examples

```
# Create a random network with 10 nodes and add random edge weights.
nw <- random_network(10)
nw <- gen_partial_correlations(nw)
# Get adjacency matrix for the network or individual modules in the network.
get_adjacency_matrix(nw)
module <- nw$modules[[1]]
get_adjacency_matrix(module)
```

get_adjacency_matrix.network *Get adjacency matrix*

Description

The adjacency matrix is constructed from all modules in a network.

Usage

```
## S3 method for class 'network'
get_adjacency_matrix(x, ...)
```

Arguments

x	Either a 'network', 'network_module', or 'matrix' object.
...	Additional arguments.

Value

An adjacency matrix with entry ij = 1 if node i and j are connected, and 0 otherwise. The diagonal entries are all zero.

Examples

```
# Create a random network with 10 nodes and add random edge weights.
nw <- random_network(10)
nw <- gen_partial_correlations(nw)
# Get adjacency matrix for the network or individual modules in the network.
get_adjacency_matrix(nw)
module <- nw$modules[[1]]
get_adjacency_matrix(module)
```

get_adjacency_matrix.network_module *Get adjacency matrix*

Description

The adjacency matrix is constructed from all modules in a network.

Usage

```
## S3 method for class 'network_module'
get_adjacency_matrix(x, ...)
```

Arguments

x	Either a 'network', 'network_module', or 'matrix' object.
...	Additional arguments.

Value

An adjacency matrix with entry ij = 1 if node i and j are connected, and 0 otherwise. The diagonal entries are all zero.

Examples

```
# Create a random network with 10 nodes and add random edge weights.
nw <- random_network(10)
nw <- gen_partial_correlations(nw)
# Get adjacency matrix for the network or individual modules in the network.
get_adjacency_matrix(nw)
module <- nw$modules[[1]]
get_adjacency_matrix(module)
```

---

get_association_matrix

Get association matrix

Description

Get association matrix

Usage

```
get_association_matrix(x, tol = 10^-13, ...)
```

Arguments

x	Either a 'network', 'network_module', or 'matrix' object.
tol	A small tolerance threshold; any entry that is within 'tol' from zero is set to zero.
...	Additional arguments.

Value

An association matrix with entry ij != 0 if node i and j are connected, and 0 otherwise. The diagonal entries are all zero.

Note

The connections in an adjacency matrix and association matrix may differ if the network contains multiple modules. The adjacency matrix only considers direct connections in the network, whereas the association matrix takes into account the fact that overlapping modules can create conditional dependencies between two genes in seperate modules (i.e. genes that don't have a direct connection in the graph).

Examples

```
# Create a random network with 10 nodes and add random edge weights.
nw <- random_network(10)
nw <- gen_partial_correlations(nw)
# Get adjacency matrix for the network or individual modules in the network.
get_association_matrix(nw)
module <- nw$modules[[1]]
get_association_matrix(module)
```

---

get_association_matrix.default
Get association matrix

Description

Get association matrix

Usage

```
## Default S3 method:
get_association_matrix(x, tol = 10^-13, ...)
```

Arguments

x	Either a 'network', 'network_module', or 'matrix' object.
tol	A small tolerance threshold; any entry that is within 'tol' from zero is set to zero.
...	Additional arguments.

Value

An association matrix with entry ij != 0 if node i and j are connected, and 0 otherwise. The diagonal entries are all zero.

Examples

```
# Create a random network with 10 nodes and add random edge weights.
nw <- random_network(10)
nw <- gen_partial_correlations(nw)
# Get adjacency matrix for the network or individual modules in the network.
get_association_matrix(nw)
module <- nw$modules[[1]]
get_association_matrix(module)
```

get_association_matrix.matrix *Get association matrix*

Description

Get association matrix

Usage

```
## S3 method for class 'matrix'
get_association_matrix(x, tol = 10^-13, ...)
```

Arguments

x	Either a 'network', 'network_module', or 'matrix' object.
tol	A small tolerance threshold; any entry that is within 'tol' from zero is set to zero.
...	Additional arguments.

Value

An association matrix with entry ij != 0 if node i and j are connected, and 0 otherwise. The diagonal entries are all zero.

Examples

```
# Create a random network with 10 nodes and add random edge weights.
nw <- random_network(10)
nw <- gen_partial_correlations(nw)
# Get adjacency matrix for the network or individual modules in the network.
get_association_matrix(nw)
module <- nw$modules[[1]]
get_association_matrix(module)
```

get_association_matrix.network *Get association matrix*

Description

Get association matrix

Usage

```
## S3 method for class 'network'
get_association_matrix(x, tol = 10^-13, ...)
```

Arguments

x	Either a 'network', 'network_module', or 'matrix' object.
tol	A small tolerance threshold; any entry that is within 'tol' from zero is set to zero.
...	Additional arguments.

Value

An association matrix with entry ij != 0 if node i and j are connected, and 0 otherwise. The diagonal entries are all zero.

Examples

```
# Create a random network with 10 nodes and add random edge weights.
nw <- random_network(10)
nw <- gen_partial_correlations(nw)
# Get adjacency matrix for the network or individual modules in the network.
get_association_matrix(nw)
module <- nw$modules[[1]]
get_association_matrix(module)
```

---

get_association_matrix.network_module
Get association matrix

Description

Get association matrix

Usage

```
## S3 method for class 'network_module'
get_association_matrix(x, tol = 10^-13, ...)
```

Arguments

x	Either a 'network', 'network_module', or 'matrix' object.
tol	A small tolerance threshold; any entry that is within 'tol' from zero is set to zero.
...	Additional arguments.

Value

An association matrix with entry ij != 0 if node i and j are connected, and 0 otherwise. The diagonal entries are all zero.

Examples

```
# Create a random network with 10 nodes and add random edge weights.
nw <- random_network(10)
nw <- gen_partial_correlations(nw)
# Get adjacency matrix for the network or individual modules in the network.
get_association_matrix(nw)
module <- nw$modules[[1]]
get_association_matrix(module)
```

`get_degree_distribution` *Get the degree distribution for a network.*

Description

Counts the connections to each node within each structure. Note, this is not the same as the degree distribution from the adjacency matrix obtained from the network, which collapses the individual structures into one graph.

Usage

```
get_degree_distribution(network)
```

Arguments

`network`	A network object.

Value

A vector of length p, containing the degree for each node in the network.

Examples

```
set.seed(13245)
nw <- random_network(10)
deg <- get_degree_distribution(nw) # Degree of each node.
table(deg) # Frequency table of degrees.
# Five nodes have degree 2, three nodes have degree 3, etc.
```

get_edge_weights_from_module *Get edge weights.*

Description

Get edge weights.

Usage

```
get_edge_weights_from_module(module)
```

Arguments

module	The 'network_module' object to get edge weights for.

Value

A vector containing the weights of each edge. If the edges are unweighted, then a vector of 1's is returned. If there are no edges, in the module, then NULL is returned.

Examples

```
nw <- random_network(10)
nw <- gen_partial_correlations(nw)
module <- nw$modules[[1]]
get_edge_weights_from_module(module)
```

get_layout_for_modules *Internal function used to create coordinates based on a set of modules*

Description

Internal function used to create coordinates based on a set of modules

Usage

```
get_layout_for_modules(g, modules)
```

Arguments

g	An 'igraph' object
modules	A list containing sets of indicies indicating the nodes in g that belong to each module

Value

A matrix of coordinates for plotting

---

get_network_arguments *Internal function used to extract 'network' objects from argument list.*

Description

Internal function used to extract 'network' objects from argument list.

Usage

```
get_network_arguments(...)
```

Arguments

...	The 'network' object(s) or list of networks.

Value

A list of 'network' objects.

---

get_network_characteristics *Characteristics of the network topology*

Description

The average degree, clustering coefficient, and average path length are calculated.

Usage

```
get_network_characteristics(network, global_only = FALSE)
```

Arguments

network	A 'network', 'network_module', or 'matrix' object.
global_only	If TRUE, only the global characteristics are calculated.

Value

A list containing characteristics of the network.

Examples

```
nw <- random_network(10)
get_network_characteristics(nw)
```

get_network_modules *Get a list of modules from the network*

Description

Get a list of modules from the network

Usage

```
get_network_modules(network)
```

Arguments

network	A 'network' object.

Value

A list whose length is the number of modules in the network; each element is a vector containing the indicies of the nodes that belong to that module.

Examples

```
set.seed(12345)
# Create a random network of 50 nodes and modules of sizes between 5-20.
nw <- random_network(50, n_modules = 5, min_module_size = 5,
                     max_module_size = 20, avg_module_size = 10,
                     sd_module_size = 5)
get_network_modules(nw) # Indicies of nodes contained in each module.
```

get_node_names *Get node names*

Description

Get node names

Usage

```
get_node_names(x, ...)
```

Arguments

x	Either a 'network', 'network_module', or 'matrix' object.
...	Additional arguments.

Value

A vector containing the node names or node indicies.

Note

Modules do not retain the names of each node, so the node indicies are returned instead. These can be used to index into the vector of node names obtained from the network.

Examples

```
# Create a random network with 10 nodes.
nw <- random_network(10)
get_node_names(nw) # Default names are 1, 2, ..., 10.
nw <- set_node_names(nw, paste("node", 1:10, sep = "_"))
get_node_names(nw) # Print out updated node names.
# Modules only contain the indicies to nodes, not the node names
module <- nw$modules[[1]]
get_node_names(module)
# When converting the network to a matrix, node names appear as column names.
adj_matrix <- get_adjacency_matrix(nw)
colnames(adj_matrix)
```

---

get_node_names.default

Get node names

Description

Get node names

Usage

```
## Default S3 method:
get_node_names(x, ...)
```

Arguments

x	Either a 'network', 'network_module', or 'matrix' object.
...	Additional arguments.

Value

A vector containing the node names or node indicies.

Examples

```
# Create a random network with 10 nodes.
nw <- random_network(10)
get_node_names(nw) # Default names are 1, 2, ..., 10.
nw <- set_node_names(nw, paste("node", 1:10, sep = "_"))
get_node_names(nw) # Print out updated node names.
# Modules only contain the indicies to nodes, not the node names
module <- nw$modules[[1]]
get_node_names(module)
# When converting the network to a matrix, node names appear as column names.
adj_matrix <- get_adjacency_matrix(nw)
colnames(adj_matrix)
```

---

get_node_names.matrix *Get node names*

Description

Get node names

Usage

```
## S3 method for class 'matrix'
get_node_names(x, ...)
```

Arguments

x	Either a 'network', 'network_module', or 'matrix' object.
...	Additional arguments.

Value

A vector containing the node names or node indicies.

Examples

```
# Create a random network with 10 nodes.
nw <- random_network(10)
get_node_names(nw) # Default names are 1, 2, ..., 10.
nw <- set_node_names(nw, paste("node", 1:10, sep = "_"))
get_node_names(nw) # Print out updated node names.
# Modules only contain the indicies to nodes, not the node names
module <- nw$modules[[1]]
get_node_names(module)
# When converting the network to a matrix, node names appear as column names.
adj_matrix <- get_adjacency_matrix(nw)
colnames(adj_matrix)
```

get_node_names.network

Get node names

Description

Get node names

Usage

```
## S3 method for class 'network'
get_node_names(x, ...)
```

Arguments

x	Either a 'network', 'network_module', or 'matrix' object.
...	Additional arguments.

Value

A vector containing the node names or node indicies.

Examples

```
# Create a random network with 10 nodes.
nw <- random_network(10)
get_node_names(nw) # Default names are 1, 2, ..., 10.
nw <- set_node_names(nw, paste("node", 1:10, sep = "_"))
get_node_names(nw) # Print out updated node names.
# Modules only contain the indicies to nodes, not the node names
module <- nw$modules[[1]]
get_node_names(module)
# When converting the network to a matrix, node names appear as column names.
adj_matrix <- get_adjacency_matrix(nw)
colnames(adj_matrix)
```

get_node_names.network_module

Get node names

Description

Get node names

Usage

```
## S3 method for class 'network_module'
get_node_names(x, ...)
```

Arguments

x	Either a 'network', 'network_module', or 'matrix' object.
...	Additional arguments.

Value

A vector containing the node names or node indicies.

Examples

```
# Create a random network with 10 nodes.
nw <- random_network(10)
get_node_names(nw) # Default names are 1, 2, ..., 10.
nw <- set_node_names(nw, paste("node", 1:10, sep = "_"))
get_node_names(nw) # Print out updated node names.
# Modules only contain the indicies to nodes, not the node names
module <- nw$modules[[1]]
get_node_names(module)
# When converting the network to a matrix, node names appear as column names.
adj_matrix <- get_adjacency_matrix(nw)
colnames(adj_matrix)
```

---

get_sigma *Get the covariance matrix*

Description

The associations in each module are taken as partial correlations, and the covariance matrix is calculated from these assuming that expression for gene i is the weighted average over each module using 1/sqrt(m_i) as the weight, where m_i is the number of modules containing gene i.

Usage

```
get_sigma(x, ...)
```

Arguments

x	Either a 'network', 'network_module', or 'matrix' object.
...	Additional arguments.

Value

A covariance matrix.

Note

If a matrix is provided, it is assumed to be a partial correlation matrix; a warning is given in this case.
To avoid the warning message, convert the matrix into a network object using create_network_from_association_matrix

Examples

```
# Create a random network with 10 nodes and add random edge weights.
nw <- random_network(10)
nw <- gen_partial_correlations(nw)
# Get covariance matrix for the network or individual modules in the network.
get_sigma(nw)
module <- nw$modules[[1]]
get_sigma(module)
```

`get_sigma.default` *Get the covariance matrix*

Description

The associations in each module are taken as partial correlations, and the covariance matrix is calculated from these assuming that expression for gene i is the weighted average over each module using 1/sqrt(m_i) as the weight, where m_i is the number of modules containing gene i.

Usage

```
## Default S3 method:
get_sigma(x, ...)
```

Arguments

`x`	Either a 'network', 'network_module', or 'matrix' object.
`...`	Additional arguments.

Value

A covariance matrix.

Examples

```
# Create a random network with 10 nodes and add random edge weights.
nw <- random_network(10)
nw <- gen_partial_correlations(nw)
# Get covariance matrix for the network or individual modules in the network.
get_sigma(nw)
module <- nw$modules[[1]]
get_sigma(module)
```

get_sigma.matrix *Get the covariance matrix*

Description

The associations in each module are taken as partial correlations, and the covariance matrix is calculated from these assuming that expression for gene i is the weighted average over each module using 1/sqrt(m_i) as the weight, where m_i is the number of modules containing gene i.

Usage

```
## S3 method for class 'matrix'
get_sigma(x, ...)
```

Arguments

x	Either a 'network', 'network_module', or 'matrix' object.
...	Additional arguments.

Value

A covariance matrix.

Examples

```
# Create a random network with 10 nodes and add random edge weights.
nw <- random_network(10)
nw <- gen_partial_correlations(nw)
# Get covariance matrix for the network or individual modules in the network.
get_sigma(nw)
module <- nw$modules[[1]]
get_sigma(module)
```

get_sigma.network *Get the covariance matrix*

Description

The associations in each module are taken as partial correlations, and the covariance matrix is calculated from these assuming that expression for gene i is the weighted average over each module using 1/sqrt(m_i) as the weight, where m_i is the number of modules containing gene i.

Usage

```
## S3 method for class 'network'
get_sigma(x, ...)
```

Arguments

x	Either a 'network', 'network_module', or 'matrix' object.
...	Additional arguments.

Value

A covariance matrix.

Examples

```
# Create a random network with 10 nodes and add random edge weights.
nw <- random_network(10)
nw <- gen_partial_correlations(nw)
# Get covariance matrix for the network or individual modules in the network.
get_sigma(nw)
module <- nw$modules[[1]]
get_sigma(module)
```

---

get_sigma.network_module

Get the covariance matrix

Description

The associations in each module are taken as partial correlations, and the covariance matrix is calculated from these assuming that expression for gene i is the weighted average over each module using 1/sqrt(m_i) as the weight, where m_i is the number of modules containing gene i.

Usage

```
## S3 method for class 'network_module'
get_sigma(x, ...)
```

Arguments

x	Either a 'network', 'network_module', or 'matrix' object.
...	Additional arguments.

Value

A covariance matrix.

Examples

```
# Create a random network with 10 nodes and add random edge weights.
nw <- random_network(10)
nw <- gen_partial_correlations(nw)
# Get covariance matrix for the network or individual modules in the network.
get_sigma(nw)
module <- nw$modules[[1]]
get_sigma(module)
```

---

get_summary_for_node *Get summary for a node in the network.*

Description

Get summary for a node in the network.

Usage

```
get_summary_for_node(node, network)
```

Arguments

node	The node to summarize. Can be a character string corresponding to a name of a node in the network, or an integer value from 1 to p corresponding to the index of a node.
network	A network object.

Value

A list containing summary information for the node; this includes a vector of indicies to other nodes in the network it is connected to, and a vector of incidices to modules that contain the node.

Examples

```
set.seed(12345)
nw <- random_network(100)
get_summary_for_node(1, nw)
# Node 1 is contained in modules 1 and 2, and it is connected to nodes
# 2, 4, 11, 13, 23, and 29.
```

heatmap_network *Plot heatmap representation of a network*

Description

This function plots the given network as a heatmap to visualize its connections. If the network is weighted, then the heatmap will use greyscale colors to represent connection strengths; black squares correspond to the strongest connections, while lighter color squares are weaker connections.

Usage

```
heatmap_network(network, main = NULL,
  col = colorRampPalette(RColorBrewer::brewer.pal(8, "Greys"))(50), ...)
```

Arguments

network	Either a network object or association matrix of the network.
main	A string containing the title for the graph.
col	Color palatte used for heatmap. See link[stats]{heatmap} for details.
...	Additional arguments passed to link[stats]{heatmap}.

Value

The matrix used to create the heatmap.

Examples

```
set.seed(12345)
nw <- random_network(10)
nw <- set_node_names(nw, paste("node", 1:10, sep = "_"))
heatmap_network(nw, "Unweighted Network")
nw <- gen_partial_correlations(nw)
heatmap_network(nw, "Weighted Network")
```

is_PD *Internal function to check if a matrix is positive definite*

Description

Internal function to check if a matrix is positive definite

Usage

```
is_PD(x)
```

Arguments

x	A matrix to check.

Value

Returns TRUE if the matrix is positive definite and FALSE otherwise.

---

is_symmetric_cpp *C++ implementation to check if a matrix is symmetric*

Description

C++ implementation to check if a matrix is symmetric

Usage

```
is_symmetric_cpp(m, tol = 1e-12)
```

Arguments

m	A matrix to check.
tol	A Numeric scalar >= 0. Differences smaller than tol are ignored.

Value

Returns TRUE if the matrix is symmetric and FALSE otherwise.

---

is_weighted *Check if an object is weighted*

Description

Check if an object is weighted

Usage

```
is_weighted(x, ...)
```

Arguments

x	Either a 'network', 'network_module', or 'matrix' object.
...	Additional arguments. object are weighted by 0s and 1s, and returns TRUE otherwise. If there are no connections in the module, then this function returns TRUE.

Examples

```
# Create a random network with 10 nodes.
nw <- random_network(10)
# The network, and hence all of its modules, are unweighted.
is_weighted(nw)
sapply(nw$modules, is_weighted)
# Add random weights to the connections.
nw <- gen_partial_correlations(nw)
# The network, and hence all of its modules, are now weighted.
is_weighted(nw)
sapply(nw$modules, is_weighted)
```

---

is_weighted.default *Check if an object is weighted*

Description

Check if an object is weighted

Usage

```
## Default S3 method:
is_weighted(x, ...)
```

Arguments

x	Either a 'network', 'network_module', or 'matrix' object.
...	Additional arguments. object are weighted by 0s and 1s, and returns TRUE otherwise. If there are no connections in the module, then this function returns TRUE.

Examples

```
# Create a random network with 10 nodes.
nw <- random_network(10)
# The network, and hence all of its modules, are unweighted.
is_weighted(nw)
sapply(nw$modules, is_weighted)
# Add random weights to the connections.
nw <- gen_partial_correlations(nw)
# The network, and hence all of its modules, are now weighted.
is_weighted(nw)
sapply(nw$modules, is_weighted)
```

is_weighted.matrix *Check if an object is weighted*

Description

Check if an object is weighted

Usage

```
## S3 method for class 'matrix'
is_weighted(x, ...)
```

Arguments

x	Either a 'network', 'network_module', or 'matrix' object.
...	Additional arguments. object are weighted by 0s and 1s, and returns TRUE otherwise. If there are no connections in the module, then this function returns TRUE.

Examples

```
# Create a random network with 10 nodes.
nw <- random_network(10)
# The network, and hence all of its modules, are unweighted.
is_weighted(nw)
sapply(nw$modules, is_weighted)
# Add random weights to the connections.
nw <- gen_partial_correlations(nw)
# The network, and hence all of its modules, are now weighted.
is_weighted(nw)
sapply(nw$modules, is_weighted)
```

is_weighted.network *Check if an object is weighted*

Description

Check if an object is weighted

Usage

```
## S3 method for class 'network'
is_weighted(x, ...)
```

Arguments

x	Either a 'network', 'network_module', or 'matrix' object.
...	Additional arguments. object are weighted by 0s and 1s, and returns TRUE otherwise. If there are no connections in the module, then this function returns TRUE.

Examples

```
# Create a random network with 10 nodes.
nw <- random_network(10)
# The network, and hence all of its modules, are unweighted.
is_weighted(nw)
sapply(nw$modules, is_weighted)
# Add random weights to the connections.
nw <- gen_partial_correlations(nw)
# The network, and hence all of its modules, are now weighted.
is_weighted(nw)
sapply(nw$modules, is_weighted)
```

---

is_weighted.network_module

Check if an object is weighted

Description

Check if an object is weighted

Usage

```
## S3 method for class 'network_module'
is_weighted(x, ...)
```

Arguments

x	Either a 'network', 'network_module', or 'matrix' object.
...	Additional arguments. object are weighted by 0s and 1s, and returns TRUE otherwise. If there are no connections in the module, then this function returns TRUE.

Examples

```
# Create a random network with 10 nodes.
nw <- random_network(10)
# The network, and hence all of its modules, are unweighted.
is_weighted(nw)
sapply(nw$modules, is_weighted)
# Add random weights to the connections.
nw <- gen_partial_correlations(nw)
```

```
# The network, and hence all of its modules, are now weighted.
is_weighted(nw)
sapply(nw$modules, is_weighted)
```

`perturb_network` *Perturbs the connections in a network*

Description

The network is perturbed by removing connections from hubs and/or rewiring other nodes in the network. By default, one hub is turned off (i.e. its connections are removed each with probability `rewire_hub_prob = 0.5`), and no other nodes are changed. Hub nodes are defined as those having degree above three standard deviations from the average degree, and nodes are sampled from these to be turned off; if there are no hub nodes, then those with the largest degree are turned off.

Usage

```
perturb_network(network, n_hubs = 1, n_nodes = 0,
  rewire_hub_prob = 0.5, rewire_other_prob = 0.5, ...)
```

Arguments

`network`	The network to modify.
`n_hubs`	The number of hub nodes to turn off.
`n_nodes`	The number of non-hub nodes to rewire. When rewiring, the degree of the node is unchanged.
`rewire_hub_prob`	The probability that a connection is removed from a hub that is selected to be turned off. If `rewire_hub_prob = 1`, then all of the connections to the hub are removed.
`rewire_other_prob`	The probability that a connection is rewired from a non-hub that is selected for rewiring. If `rewire_other_prob = 1`, then all of the connections to the hub are rewired; however, this does not mean that all connections will be changed, as some connections may be removed but later rewired back.
`...`	Additional arguments passed to `rewire_connections_to_node` and `remove_connections_to_node`.

Value

The modified network.

Examples

```
# Create a random network, perturb the network, then plot the differential network.
set.seed(12345)
nw <- random_network(100)
# Rewire 2 random hub genes and 10 other random genes:
nw_diff <- perturb_network(nw, n_hubs = 2, n_nodes = 10)
plot_network_diff(nw, nw_diff)
```

plot.network *Plot function for 'network' object*

Description

This function plots the given network. If the result of another plot is provided, this plot will be modified for easier comparison.

Usage

```
## S3 method for class 'network'
plot(x, compare_graph = NULL, show_modules = FALSE,
  as_subgraph = FALSE, ...)
```

Arguments

x	A 'network' object.
compare_graph	The plot of another network to use for comparison.
show_modules	If TRUE, the modules will highlighted in the graph. Defaults to FALSE if there is exactly one module in the network and to TRUE otherwise.
as_subgraph	If TRUE, only nodes of positive degree will be shown. Defaults to FALSE if there are 100 or fewer nodes in the network and to TRUE otherwise.
...	Additional arguments passed to plot_modules or plot_network.

Value

Creates a plot of the module and returns a graph object. See plot_modules and plot_network for details.

A 'network_plot' object for the network. This object can be passed back into a future call of plot.network through the compare_graph argument, which will setup the plot for easier comparison between the old graph and the new graph of network.

Examples

```
nw <- random_network(10)
plot(nw)
```

plot.network_module *Plot function for 'network_module' object.*

Description

Plot function for 'network_module' object.

Usage

```
## S3 method for class 'network_module'
plot(x, ...)
```

Arguments

x	A 'network_module' object.
...	Additional arguments passed to plot_network.

Value

Creates a plot of the module and returns a graph object. See plot_network for details.

Examples

```
module <- random_module(1:10)
plot(module)
```

plot.network_plot *Plot function for 'network_plot' class*

Description

Plot function for 'network_plot' class

Usage

```
## S3 method for class 'network_plot'
plot(x, ...)
```

Arguments

x	A 'network_plot' object obtained from plot.network or plot_network.
...	Additional arguments passed to plot.igraph.

Examples

```
nw <- random_network(10)
g <- plot(nw)
# Can change the plot by modifying the instance `g`.
# For example, make vertex size and edge width twice as big.
g$edge.width <- 2 * g$edge.width
g$vertex.size <- 2 * g$vertex.size
# Change color of verticies, edges, and vertex labels.
g$edge.color <- "orange"
g$vertex.color <- "navy"
g$vertex.label.color <- "white"
plot(g)
```

`plot_gene_pair` *Scatter plot of two gene expressions*

Description

Plots the expression of two genes for visual assessment of association.

Usage

```
plot_gene_pair(x_list, geneA, geneB, method = "loess", se_alpha = 0.1,
  do_facet_wrap = FALSE, scales = "fixed")
```

Arguments

`x_list`	A named list containing one or more n by p gene expression profiles, one for each group or subpopulation under consideration.
`geneA`	The name of the first gene to plot. Must be either a character string matching a column name in each matrix of `x_list` or an integer to index the columns.
`geneB`	The name of the second gene to plot. Must be either a character string matching a column name in each matrix of `x_list` or an integer to index the columns.
`method`	Charater string either "lm" or "loess" used for plotting. For no line, set `method = NULL`.
`se_alpha`	Sets transparancy of confidence interval around association trend line. Set to 0 to remove the confidence interval.
`do_facet_wrap`	If `TRUE`, the groups are plotted in seperate graphs.
`scales`	Only used if `do_facet_wrap` is `TRUE`. See `facet_wrap` for details.

Value

Returns the generated plot.

Examples

```
data(reference)
rnaseq <- reference$rnaseq
genes <- colnames(rnaseq)
plot_gene_pair(rnaseq, genes[1], genes[2])
# Suppose we had multiple data frames.
control <- rnaseq[1:100, 1:10]
treatment1 <- rnaseq[101:200, 1:10]
treatment2 <- rnaseq[201:250, 1:10]
plot_gene_pair(list(ctrl = control, trt1 = treatment1, trt2 = treatment2),
               genes[1], genes[2], method = NA)
plot_gene_pair(list(ctrl = control, trt = treatment1),
               genes[1], genes[2], do_facet_wrap = TRUE, method = "lm")
```

---

plot_modules *Visualize a network and its modules*

Description

This function plots a network and highlights the individual modules. An attempt is made to layout the nodes so that any visual overlaps among modules correspond to true overlaps in the network, however it is possible that a node may appear to be in multiple modules in the visualization when it does not actually belong to multiple modules. If the result of another plot is provided using the `compare_graph` argument, then the layout of this network will be based on that plot and convex hulls are drawn to trace out the modules; in this case it is likely that the displayed modules will contain extraneous nodes.

Usage

```
plot_modules(network, compare_graph = NULL, as_subgraph = TRUE,
  modules = NULL, node_scale = 4, edge_scale = 1,
  node_color = adjustcolor("orange", 0.5),
  group_color = RColorBrewer::brewer.pal(9, "Set1"),
  generate_layout = igraph::nicely, include_vertex_labels = TRUE,
  show_legend = FALSE, legend_position = "topright",
  legend_horizontal = FALSE, display_plot = TRUE, ...)
```

Arguments

`network`	A 'network' object to plot. Alternatively, an adjacency or association matrix can be provided, in which case the 'modules' argument should be specified.
`compare_graph`	The plot of another network to use for comparison.
`as_subgraph`	If `TRUE`, only nodes of positive degree will be shown.
`modules`	A list of modules for the network; this is used to provide a member list of each module when the `network` argument is not a 'network' object. To get this list from a network, use `get_network_modules`.

node_scale	Used for scaling of nodes.
edge_scale	Used for scaling of edges.
node_color	The color used for the nodes.
group_color	A vector of colors used for the modules.
generate_layout	A function to generate the layout of a graph; used if coords is NULL. See layout_ from **igraph** for details. Other options include as_star, in_circle, and with_fr, among many others.
include_vertex_labels	If TRUE, the verticies will be labeled.
show_legend	If TRUE, a legend for the modules is shown. Default is FALSE
legend_position	The location of the legend. Can be any one of "bottomright", "bottom", "bottomleft", "left", "topleft", "top", "topright", "right" or "center".
legend_horizontal	If TRUE, the legend will be displayed horizontally.
display_plot	If TRUE (default), the plot will be generated and displayed.
...	Additional arguments passed to plot.igraph.

Value

A 'network_plot' object for the network. This object can be passed back into a future call of plot.network through the compare_graph argument, which will setup the plot for easier comparison between the old graph and the new graph of network.

Examples

```
set.seed(1)
# Networks can be plotted with modules highlighted.
nw <- random_network(100)
g <- plot_network(nw)
plot_modules(nw, g) # Overlay convex hulls around modules in previous layout.
```

---

plot_network *Visualize a network*

Description

This function is used to plot a network. The network argument can be a network object, network module, an adjacency matrix, or an association matrix. If the result of another plot is provided using the compare_graph argument, then the layout of this network will be based on that plot.

Usage

```
plot_network(network, compare_graph = NULL, as_subgraph = FALSE,
  node_scale = 4, edge_scale = 1, node_color = adjustcolor("orange",
  0.5), generate_layout = igraph::nicely, include_vertex_labels = TRUE,
  display_plot = TRUE, ...)
```

Arguments

network	A 'network', 'network_module', or 'matrix' object.
compare_graph	The plot of another network to use for comparison.
as_subgraph	If TRUE, only nodes of positive degree will be shown.
node_scale	Used for scaling of nodes.
edge_scale	Used for scaling of edges.
node_color	The color used for the nodes.
generate_layout	A function to generate the layout of a graph; used if coords is NULL. See layout_ from **igraph** for details. Other options include as_star, in_circle, and with_fr, among many others.
include_vertex_labels	If TRUE, the verticies will be labeled.
display_plot	If TRUE (default), the plot will be generated and displayed.
...	Additional arguments passed to plot.igraph.

Value

Creates a plot of the network and returns a graph object. The graph object can be passed back into a future call of plot.network through the compare_edge argument, which will setup the plot for easier comparison between the old graph and the graph of network.

Examples

```
set.seed(0)
# Basic plotting for networks, modules, and matricies
nw <- random_network(10)
plot(nw)
module <- random_module(1:10)
plot(module)
adj_mat <- get_adjacency_matrix(nw)
plot_network(adj_mat)
# To compare multiple networks, the layout from the first plot can be used
# in subsequent plots using the second argument, `compare_graph`.
nw1 <- random_network(10)
nw2 <- remove_connections_to_node(nw1, 6, prob_remove = 1)
g <- plot(nw1)
plot(nw2, g)
# If the network contains many nodes of degree 0, plotting as subgraph
# may be preferred.
```

```
nw <- random_network(100, n_modules = 1)
plot(nw)
plot(nw, as_subgraph = TRUE)
# Networks can be plotted with modules highlighted.
nw <- random_network(100)
g <- plot_network(nw)
plot_modules(nw, g)
# For large networks, the vertex labels can clutter the graph; these can
# be removed using the `include_vertex_labels` argument.
nw <- random_network(250)
g <- plot(nw)
plot(nw, g, include = FALSE)
```

`plot_network_diff` *Plot the difference between two networks*

Description

This function plots the difference in connectivity between two networks. For two identical networks, the graph will be empty. For non-identical networks, black edges are shared by both networks but differ in magnitude or direction (if the networks are weighted), tan edges are in `network_1` but not `network_2`, and red edges are in `network_2` but not `network_1`. All edges are scaled according to the strongest association in either network.

Usage

```
plot_network_diff(network_1, network_2, compare_graph = NULL,
  as_subgraph = FALSE, node_scale = 4, edge_scale = 1,
  node_color = adjustcolor("orange", 0.5), edge_colors = c("black",
  "wheat", "red"), generate_layout = igraph::nicely,
  include_vertex_labels = TRUE, ...)
```

Arguments

`network_1`	A 'network' or 'matrix' object.
`network_2`	A 'network' or 'matrix' object.
`compare_graph`	The plot of another network to use for comparison.
`as_subgraph`	If TRUE, only nodes of positive degree will be shown.
`node_scale`	Used for scaling of nodes.
`edge_scale`	Used for scaling of edges.
`node_color`	The color used for the nodes.
`edge_colors`	A vector of three colors used for edges; the first colors edges common to both network, the second colors edges in `network_1` but not `network_2`, and the third colors edges that are in `network_2` but not `network_2`. Default is `c("black","wheat","red")`.

generate_layout
: A function to generate the layout of a graph; used if coords is NULL. See layout_ from **igraph** for details. Other options include as_star, in_circle, and with_fr, among many others.

include_vertex_labels
: If TRUE, the verticies will be labeled.

...
: Additional arguments passed to plot.igraph.

Value

Creates a plot of the network and returns a graph object. The graph object can be passed back into a future call of plot_network, plot_network_diff, or plot_network_sim through the compare_edge argument, which will setup the plot for easier comparison between the old graph and the graph of network.

Examples

```
# Create two networks, the second being a perturbation of the first.
nw1 <- random_network(20)
nw2 <- perturb_network(nw1, n_nodes = 5)
# Can compare networks by plotting each using the same layout.
g <- plot(nw1)
plot(nw2, g)
# Or, the differential network can be plotted.
plot_network_diff(nw1, nw2, g)
```

---

plot_network_sim *Plot the similarity between two networks*

Description

This function plots the similarity of connections between two networks. Both networks must be weighted. The width of each edge corresponds to the strength of similarity and is calculated by sqrt(abs((s1 + s2)s1s2)), where s1 and s2 are the weights for a particular connection in network_1 and network_2, respectively

Usage

```
plot_network_sim(network_1, network_2, compare_graph = NULL, ...)
```

Arguments

network_1
: A weighted 'network' or 'matrix' object.

network_2
: A weighted 'network' or 'matrix' object.

compare_graph
: The plot of another network to use for comparison.

...
: Additional arguments passed to plot_network.

Value

Creates a plot of the network and returns a graph object. The graph object can be passed back into a future call of plot_network, plot_network_diff or plot_network_sim through the compare_edge argument, which will setup the plot for easier comparison between the old graph and the graph of network.

Examples

```
# Create two networks, the second being a perturbation of the first.
nw1 <- random_network(20)
nw2 <- perturb_network(nw1, n_nodes = 5)
nw1 <- gen_partial_correlations(nw1)
nw2 <- gen_partial_correlations(nw2)
# Can compare networks by plotting each using the same layout.
g <- plot(nw1)
plot(nw2, g)
# Or, plot the differential network or similarity network
plot_network_diff(nw1, nw2, g)
plot_network_sim(nw1, nw2, g)
# Note the behavior when both networks are the same.
plot_network_diff(nw1, nw1, g) # No differences produces an empty network
plot_network_sim(nw1, nw1, g) # Edge widths are still scaled by connection strength.
```

---

print.network *Print function for 'network' object.*

Description

Print function for 'network' object.

Usage

```
## S3 method for class 'network'
print(x, ...)
```

Arguments

x	A 'network' object.
...	Additional arguments are ignored.

Value

Prints a summary of the module.

Examples

```
nw <- random_network(10)
nw
print(nw)
```

print.network_module *Print function for 'network_module' object.*

Description

Print function for 'network_module' object.

Usage

```
## S3 method for class 'network_module'
print(x, ...)
```

Arguments

x	A 'network_module' object.
...	Additional arguments are ignored.

Value

Prints a summary of the module.

Examples

```
module <- random_module(1:10)
module
print(module)
```

print.network_plot *Print function for 'network_plot' class*

Description

Displays the network plot.

Usage

```
## S3 method for class 'network_plot'
print(x, ...)
```

Arguments

x	A 'network_plot' object obtained from plot.network or plot_network.
...	Additional arguments passed to plot.

Examples

```
nw <- random_network(10)
g <- plot(nw, display_plot = FALSE) # Doesn't display the plot.
g # Displays the plot.
```

---

pzinb *The Zero-Inflated Negative Binomial Distribution*

Description

The Zero-Inflated Negative Binomial Distribution

Usage

```
pzinb(q, size, mu, rho, lower.tail = TRUE, log.p = FALSE)
```

Arguments

q	A vector of quantities.
size	The dispersion paramater used in dnbinom
mu	The mean parameter used in dnbinom.
rho	The zero-inflation parameter.
lower.tail	Logical; if TRUE, then probabilities are P(X <= x). Otherwise, P(X > x).
log.p	Logical; if TRUE, then log(p) is returned.

Examples

```
x <- rzinb(10, 1, 10, 0.1)
p <- pzinb(x, 1, 10, 0.1)
y <- qzinb(p, 1, 10, 0.1)
all(x == y)
# Compute P(0 < X < 5) for X ~ ZINB(1, 10, 0.1)
sum(dzinb(0:5, 1, 10, 0.1))
```

qzinb *The Zero-Inflated Negative Binomial Distribution*

Description

The Zero-Inflated Negative Binomial Distribution

Usage

```
qzinb(p, size, mu, rho, lower.tail = TRUE, log.p = FALSE)
```

Arguments

p	A vector of probabilities
size	The dispersion paramater used in dnbinom
mu	The mean parameter used in dnbinom.
rho	The zero-inflation parameter.
lower.tail	Logical; if TRUE, then probabilities are P(X <= x). Otherwise, P(X > x).
log.p	Logical; if TRUE, then exp(p) is used.

Examples

```
x <- rzinb(10, 1, 10, 0.1)
p <- pzinb(x, 1, 10, 0.1)
y <- qzinb(p, 1, 10, 0.1)
all(x == y)
# Compute P(0 < X < 5) for X ~ ZINB(1, 10, 0.1)
sum(dzinb(0:5, 1, 10, 0.1))
```

random_module *Create a random module*

Description

Create a random module

Usage

```
random_module(nodes, module_name = NULL, ...)
```

Arguments

nodes	A numeric vector indicating which nodes in the network are contained in this module.
module_name	(optional) Character string specifying the name of the module. If NULL, the module will be unnamed.
...	Additional arguments passed to random_module_structure.

Value

A 'network_module' object.

Examples

```
module <- random_module(1:10)
```

`random_module_structure` *Create a random network structure for a module*

Description

A single, connected graph is created. The graph is initialized as a ring lattice, and edges are randomly rewired and/or removed. The procedure is similar to the Watts-Strogatz method, but the sampling of edges to modify can be based on the degree of each node.

Usage

```
random_module_structure(size, prob_rewire = 1, prob_remove = 0.5,
  weights = NULL, neig_size = 3, alpha = 100, beta = 1,
  epsilon = 10^-5, ...)
```

Arguments

`size`	The number of nodes to include in the graph.
`prob_rewire`	The probability of rewiring an edge.
`prob_remove`	The probability of removing an edge.
`weights`	(Optional) Weights used for sampling nodes. See `rewire_connections_to_node` and `remove_connections_to_node` for details.
`neig_size`	The neighborhood size within which the nodes of the ring lattice are connected. The initial degree of each node is `2 * neig_size`, so long as `size >= (1 + 2 * neig_size)`.
`alpha`	A positive value used to parameterize the Beta distribution.
`beta`	A positive value used to parameterize the Beta distribution.
`epsilon`	A small constant added to the sampling probability of each node.
`...`	Additional arguments are ignored.

Value

An adjacency matrix representing the network structure.

Examples

```
# Create a random module structure (an adjacency matrix) for 10 nodes.
adj_mat <- random_module_structure(10)
# A network object can be created using this structure.
module <- create_module_from_adjacency_matrix(adj_mat)
nw <- create_network_from_modules(10, module)
```

`random_network` *Create a network object.*

Description

Creates an unweighted 'network' object containing randomly generated modules.

Usage

```
random_network(p, n_modules = NULL, ...)
```

Arguments

p	The number of nodes in the network. If p is much larger than 10^4, computation may begin to slow depending on the average module size and the amount of overlap among modules.
n_modules	The number of modules to include in the network. If NULL, then modules are created until all nodes in the network have positive degree.
...	Arguments to be passed to other methods. Possible arguments include:
nu	A value in [0, 1] used to control the amount of overlap among modules. Smaller values result in less over
prob_rewire	The probability of removing a connection from the local network structure; this is applied to each edge c
prob_remove	The probability of rewiring a connection from the local network structure; this is applied every connectio
neig_size	The initial degree of each node when constructing the ring lattice. See random_module_structure.
alpha	A positive value used to parameterize the Beta distribution used to sample nodes based on their degree. L
beta	A positive value used to parameterize the Beta distribution used to sample nodes based on their degree. S
epsilon	A small constant added to the sampling probability of each node. See random_module_structure.
avg_module_size	See create_modules_for_network.
sd_module_size	See create_modules_for_network.
min_module_size	See create_modules_for_network.
Max_module_size	See create_modules_for_network.

Value

An unweighted network object.

Examples

```
# Create a random network of 10 nodes
```

```
nw <- random_network(10)
nw
# Add a random weight to each connection.
nw <- gen_partial_correlations(nw)
# Plot the network
plot(nw)
```

`reference` *RNA-seq reference dataset*

Description

The reference is a breast invasive carcinoma dataset containing gene expression profiles generated by The Cancer Genome Atlas (TCGA) and downloaded using the LinkedOmics portal. The dataset contains 1093 samples and 15944 genes. The reference is a list containing a data frame of the expression data and a data frame of estimated ZINB parameters for each expression profile.

Usage

```
reference
```

Format

A list containing two data frames:

$rnaseq A 1093 by 15944 data frame containing the raw RNA-seq expression counts

$params A 3 by 15944 data frame containing the estimated ZINB parameters for each expression profile

Source

http://www.linkedomics.org/data_download/TCGA-BRCA/

`remove_connections` *Remove connections in a network*

Description

Remove connections in a network

Usage

```
remove_connections(x, prob_remove, run_checks = TRUE, ...)
```

Arguments

x	The 'network', 'network_module', or 'matrix' object to modify.
prob_remove	A value between 0 and 1. Each edge will be removed with probability equal to prob_remove.
run_checks	If TRUE and 'x' is a matrix, then it is checked that 'x' is an adjacency matrix. This catches the case where 'x' is a weighted matrix, in which case the weights are removed and a warning is given.
...	Additional arguments.

Value

The modified adjacency matrix.

Examples

```
# Create a random network with 10 nodes.
nw <- random_network(20)
# Remove connections in the network each with probability 1/2.
nw_rewired <- remove_connections(nw, 0.5)
# Plot the two networks for comparison
g <- plot(nw)
plot(nw_rewired, g) # Pass in g to mirror the layout.
# Or plot the differential network.
plot_network_diff(nw, nw_rewired)
```

---

remove_connections.default

Remove connections in a network

Description

Remove connections in a network

Usage

```
## Default S3 method:
remove_connections(x, prob_remove, run_checks = TRUE,
  ...)
```

Arguments

x	The 'network', 'network_module', or 'matrix' object to modify.
prob_remove	A value between 0 and 1. Each edge will be removed with probability equal to prob_remove.
run_checks	If TRUE and 'x' is a matrix, then it is checked that 'x' is an adjacency matrix. This catches the case where 'x' is a weighted matrix, in which case the weights are removed and a warning is given.
...	Additional arguments.

Value

The modified adjacency matrix.

Examples

```
# Create a random network with 10 nodes.
nw <- random_network(20)
# Remove connections in the network each with probability 1/2.
nw_rewired <- remove_connections(nw, 0.5)
# Plot the two networks for comparison
g <- plot(nw)
plot(nw_rewired, g) # Pass in g to mirror the layout.
# Or plot the differential network.
plot_network_diff(nw, nw_rewired)
```

---

remove_connections.matrix

Remove connections in a network

Description

Remove connections in a network

Usage

```
## S3 method for class 'matrix'
remove_connections(x, prob_remove, run_checks = TRUE,
  ...)
```

Arguments

x	The 'network', 'network_module', or 'matrix' object to modify.
prob_remove	A value between 0 and 1. Each edge will be removed with probability equal to prob_remove.
run_checks	If TRUE and 'x' is a matrix, then it is checked that 'x' is an adjacency matrix. This catches the case where 'x' is a weighted matrix, in which case the weights are removed and a warning is given.
...	Additional arguments.

Value

The modified adjacency matrix.

Examples

```
# Create a random network with 10 nodes.
nw <- random_network(20)
# Remove connections in the network each with probability 1/2.
nw_rewired <- remove_connections(nw, 0.5)
# Plot the two networks for comparison
g <- plot(nw)
plot(nw_rewired, g) # Pass in g to mirror the layout.
# Or plot the differential network.
plot_network_diff(nw, nw_rewired)
```

`remove_connections.network`

Remove connections in a network

Description

Remove connections in a network

Usage

```
## S3 method for class 'network'
remove_connections(x, prob_remove, run_checks = TRUE,
  ...)
```

Arguments

`x`	The 'network', 'network_module', or 'matrix' object to modify.
`prob_remove`	A value between 0 and 1. Each edge will be removed with probability equal to `prob_remove`.
`run_checks`	If `TRUE` and 'x' is a matrix, then it is checked that 'x' is an adjacency matrix. This catches the case where 'x' is a weighted matrix, in which case the weights are removed and a warning is given.
`...`	Additional arguments.

Value

The modified adjacency matrix.

Examples

```
# Create a random network with 10 nodes.
nw <- random_network(20)
# Remove connections in the network each with probability 1/2.
nw_rewired <- remove_connections(nw, 0.5)
# Plot the two networks for comparison
g <- plot(nw)
```

```
plot(nw_rewired, g) # Pass in g to mirror the layout.
# Or plot the differential network.
plot_network_diff(nw, nw_rewired)
```

`remove_connections.network_module`
Remove connections in a network

Description

Remove connections in a network

Usage

```
## S3 method for class 'network_module'
remove_connections(x, prob_remove,
  run_checks = TRUE, ...)
```

Arguments

`x`	The 'network', 'network_module', or 'matrix' object to modify.
`prob_remove`	A value between 0 and 1. Each edge will be removed with probability equal to `prob_remove`.
`run_checks`	If `TRUE` and 'x' is a matrix, then it is checked that 'x' is an adjacency matrix. This catches the case where 'x' is a weighted matrix, in which case the weights are removed and a warning is given.
`...`	Additional arguments.

Value

The modified adjacency matrix.

Examples

```
# Create a random network with 10 nodes.
nw <- random_network(20)
# Remove connections in the network each with probability 1/2.
nw_rewired <- remove_connections(nw, 0.5)
# Plot the two networks for comparison
g <- plot(nw)
plot(nw_rewired, g) # Pass in g to mirror the layout.
# Or plot the differential network.
plot_network_diff(nw, nw_rewired)
```

remove_connections_to_node *Remove connections to a node*

Description

Remove connections to a node

Usage

```
remove_connections_to_node(x, node, prob_remove, run_checks = TRUE, ...)
```

Arguments

x	The 'network', 'network_module', or 'matrix' object to modify.
node	The node to unwire.
prob_remove	A value between 0 and 1. Each connection to node will be removed with probability equal to prob_remove.
run_checks	If TRUE and 'x' is a matrix, then it is checked that 'x' is an adjacency matrix. This catches the case where 'x' is a weighted matrix, in which case the weights are removed and a warning is given.
...	Additional arguments.

Value

The modified adjacency matrix.

Examples

```
# Create a random network with 10 nodes.
nw <- random_network(10)
# Remove all connections to node 1.
nw_rewired <- remove_connections_to_node(nw, 1, 1)
# Plot the two networks for comparison
g <- plot(nw)
plot(nw_rewired, g) # Pass in g to mirror the layout.
# Or plot the differential network.
plot_network_diff(nw, nw_rewired)
```

`remove_connections_to_node.default` *Remove connections to a node*

Description

Remove connections to a node

Usage

```
## Default S3 method:
remove_connections_to_node(x, node, prob_remove,
  run_checks = TRUE, ...)
```

Arguments

`x`	The 'network', 'network_module', or 'matrix' object to modify.
`node`	The node to unwire.
`prob_remove`	A value between 0 and 1. Each connection to `node` will be removed with probability equal to `prob_remove`.
`run_checks`	If `TRUE` and 'x' is a matrix, then it is checked that 'x' is an adjacency matrix. This catches the case where 'x' is a weighted matrix, in which case the weights are removed and a warning is given.
`...`	Additional arguments.

Value

The modified adjacency matrix.

Examples

```
# Create a random network with 10 nodes.
nw <- random_network(10)
# Remove all connections to node 1.
nw_rewired <- remove_connections_to_node(nw, 1, 1)
# Plot the two networks for comparison
g <- plot(nw)
plot(nw_rewired, g) # Pass in g to mirror the layout.
# Or plot the differential network.
plot_network_diff(nw, nw_rewired)
```

`remove_connections_to_node.matrix` *Remove connections to a node*

Description

Remove connections to a node

Usage

```
## S3 method for class 'matrix'
remove_connections_to_node(x, node, prob_remove,
  run_checks = TRUE, ...)
```

Arguments

`x`	The 'network', 'network_module', or 'matrix' object to modify.
`node`	The node to unwire.
`prob_remove`	A value between 0 and 1. Each connection to `node` will be removed with probability equal to `prob_remove`.
`run_checks`	If `TRUE` and 'x' is a matrix, then it is checked that 'x' is an adjacency matrix. This catches the case where 'x' is a weighted matrix, in which case the weights are removed and a warning is given.
`...`	Additional arguments.

Value

The modified adjacency matrix.

Examples

```
# Create a random network with 10 nodes.
nw <- random_network(10)
# Remove all connections to node 1.
nw_rewired <- remove_connections_to_node(nw, 1, 1)
# Plot the two networks for comparison
g <- plot(nw)
plot(nw_rewired, g) # Pass in g to mirror the layout.
# Or plot the differential network.
plot_network_diff(nw, nw_rewired)
```

`remove_connections_to_node.network` *Remove connections to a node*

Description

Remove connections to a node

Usage

```
## S3 method for class 'network'
remove_connections_to_node(x, node, prob_remove,
  run_checks = TRUE, ...)
```

Arguments

`x`	The 'network', 'network_module', or 'matrix' object to modify.
`node`	The node to unwire.
`prob_remove`	A value between 0 and 1. Each connection to `node` will be removed with probability equal to `prob_remove`.
`run_checks`	If `TRUE` and 'x' is a matrix, then it is checked that 'x' is an adjacency matrix. This catches the case where 'x' is a weighted matrix, in which case the weights are removed and a warning is given.
`...`	Additional arguments.

Value

The modified adjacency matrix.

Examples

```
# Create a random network with 10 nodes.
nw <- random_network(10)
# Remove all connections to node 1.
nw_rewired <- remove_connections_to_node(nw, 1, 1)
# Plot the two networks for comparison
g <- plot(nw)
plot(nw_rewired, g) # Pass in g to mirror the layout.
# Or plot the differential network.
plot_network_diff(nw, nw_rewired)
```

`remove_connections_to_node.network_module` *Remove connections to a node*

Description

Remove connections to a node

Usage

```
## S3 method for class 'network_module'
remove_connections_to_node(x, node, prob_remove,
  run_checks = TRUE, ...)
```

Arguments

`x`	The 'network', 'network_module', or 'matrix' object to modify.
`node`	The node to unwire.
`prob_remove`	A value between 0 and 1. Each connection to `node` will be removed with probability equal to `prob_remove`.
`run_checks`	If `TRUE` and 'x' is a matrix, then it is checked that 'x' is an adjacency matrix. This catches the case where 'x' is a weighted matrix, in which case the weights are removed and a warning is given.
`...`	Additional arguments.

Value

The modified adjacency matrix.

Examples

```
# Create a random network with 10 nodes.
nw <- random_network(10)
# Remove all connections to node 1.
nw_rewired <- remove_connections_to_node(nw, 1, 1)
# Plot the two networks for comparison
g <- plot(nw)
plot(nw_rewired, g) # Pass in g to mirror the layout.
# Or plot the differential network.
plot_network_diff(nw, nw_rewired)
```

remove_weights *Removes the weights of all connections*

Description

Removes the weights of all connections

Usage

```
remove_weights(x, ...)
```

Arguments

x	Either a 'network', 'network_module', or 'matrix' object.
...	Additional arguments.

Value

The modified object.

Examples

```
# Create a random network with 10 nodes and add random edge weights.
nw <- random_network(10)
nw <- gen_partial_correlations(nw)
is_weighted(nw)
# Remove the edge weights from the network.
nw <- remove_weights(nw)
is_weighted(nw)
```

remove_weights.default *Removes the weights of all connections*

Description

Removes the weights of all connections

Usage

```
## Default S3 method:
remove_weights(x, ...)
```

Arguments

x	Either a 'network', 'network_module', or 'matrix' object.
...	Additional arguments.

Value

The modified object.

Examples

```
# Create a random network with 10 nodes and add random edge weights.
nw <- random_network(10)
nw <- gen_partial_correlations(nw)
is_weighted(nw)
# Remove the edge weights from the network.
nw <- remove_weights(nw)
is_weighted(nw)
```

---

remove_weights.matrix *Removes the weights of all connections*

Description

Removes the weights of all connections

Usage

```
## S3 method for class 'matrix'
remove_weights(x, ...)
```

Arguments

x	Either a 'network', 'network_module', or 'matrix' object.
...	Additional arguments.

Value

The modified object.

Examples

```
# Create a random network with 10 nodes and add random edge weights.
nw <- random_network(10)
nw <- gen_partial_correlations(nw)
is_weighted(nw)
# Remove the edge weights from the network.
nw <- remove_weights(nw)
is_weighted(nw)
```

remove_weights.network

Removes the weights of all connections

Description

Removes the weights of all connections

Usage

```
## S3 method for class 'network'
remove_weights(x, ...)
```

Arguments

x	Either a 'network', 'network_module', or 'matrix' object.
...	Additional arguments.

Value

The modified object.

Examples

```
# Create a random network with 10 nodes and add random edge weights.
nw <- random_network(10)
nw <- gen_partial_correlations(nw)
is_weighted(nw)
# Remove the edge weights from the network.
nw <- remove_weights(nw)
is_weighted(nw)
```

remove_weights.network_module

Removes the weights of all connections

Description

Removes the weights of all connections

Usage

```
## S3 method for class 'network_module'
remove_weights(x, ...)
```

Arguments

x	Either a 'network', 'network_module', or 'matrix' object.
...	Additional arguments.

Value

The modified object.

Examples

```
# Create a random network with 10 nodes and add random edge weights.
nw <- random_network(10)
nw <- gen_partial_correlations(nw)
is_weighted(nw)
# Remove the edge weights from the network.
nw <- remove_weights(nw)
is_weighted(nw)
```

---

replace_module_in_network

Internal function for replacing a module in the network

Description

Internal function for replacing a module in the network

Usage

```
replace_module_in_network(module_index, module, network)
```

Arguments

module_index	The index of the module to replace.
module	The new module to replace with.
network	The network to modify.

Value

The modified network.

`rewire_connections` *Rewire connections*

Description

Rewire connections

Usage

```
rewire_connections(x, prob_rewire = 1, weights = NULL, alpha = 100,
  beta = 1, epsilon = 10^-5, run_checks = TRUE, ...)
```

Arguments

`x`	The 'network', 'network_module', or 'matrix' object to modify.
`prob_rewire`	A value between 0 and 1. The connections to each node will be rewired with probability equal to `prob_rewire`.
`weights`	(Optional) A vector of weights for each node. These are used in addition to the degree of each node when sampling a node to rewire to.
`alpha`	A positive value used to parameterize the Beta distribution.
`beta`	A positive value used to parameterize the Beta distribution.
`epsilon`	A small constant added to the sampling probability of each node.
`run_checks`	If `TRUE` and 'x' is a matrix, then it is checked that 'x' is an adjacency matrix. This catches the case where 'x' is a weighted matrix, in which case the weights are removed and a warning is given.
`...`	Additional arguments.

Value

The modified module.

Note

When applied to a network object, all modules in the network are rewired. If

Examples

```
# Create a random network with 10 nodes.
nw <- random_network(10)
# Rewire nodes in the network each with probability 1/2
nw_rewired <- rewire_connections(nw, 0.5)
# Plot the two networks for comparison
g <- plot(nw)
plot(nw_rewired, g) # Pass in g to mirror the layout.
# Or plot the differential network.
plot_network_diff(nw, nw_rewired, g)
```

rewire_connections.default
Rewire connections

Description

Rewire connections

Usage

```
## Default S3 method:
rewire_connections(x, prob_rewire = 1,
  weights = NULL, alpha = 100, beta = 1, epsilon = 10^-5,
  run_checks = TRUE, ...)
```

Arguments

x	The 'network', 'network_module', or 'matrix' object to modify.
prob_rewire	A value between 0 and 1. The connections to each node will be rewired with probability equal to prob_rewire.
weights	(Optional) A vector of weights for each node. These are used in addition to the degree of each node when sampling a node to rewire to.
alpha	A positive value used to parameterize the Beta distribution.
beta	A positive value used to parameterize the Beta distribution.
epsilon	A small constant added to the sampling probability of each node.
run_checks	If TRUE and 'x' is a matrix, then it is checked that 'x' is an adjacency matrix. This catches the case where 'x' is a weighted matrix, in which case the weights are removed and a warning is given.
...	Additional arguments.

Value

The modified module.

Examples

```
# Create a random network with 10 nodes.
nw <- random_network(10)
# Rewire nodes in the network each with probability 1/2
nw_rewired <- rewire_connections(nw, 0.5)
# Plot the two networks for comparison
g <- plot(nw)
plot(nw_rewired, g) # Pass in g to mirror the layout.
# Or plot the differential network.
plot_network_diff(nw, nw_rewired, g)
```

`rewire_connections.matrix` *Rewire connections*

Description

Rewire connections

Usage

```
## S3 method for class 'matrix'
rewire_connections(x, prob_rewire = 1, weights = NULL,
  alpha = 100, beta = 1, epsilon = 10^-5, run_checks = TRUE, ...)
```

Arguments

`x`	The 'network', 'network_module', or 'matrix' object to modify.
`prob_rewire`	A value between 0 and 1. The connections to each node will be rewired with probability equal to `prob_rewire`.
`weights`	(Optional) A vector of weights for each node. These are used in addition to the degree of each node when sampling a node to rewire to.
`alpha`	A positive value used to parameterize the Beta distribution.
`beta`	A positive value used to parameterize the Beta distribution.
`epsilon`	A small constant added to the sampling probability of each node.
`run_checks`	If `TRUE` and 'x' is a matrix, then it is checked that 'x' is an adjacency matrix. This catches the case where 'x' is a weighted matrix, in which case the weights are removed and a warning is given.
`...`	Additional arguments.

Value

The modified module.

Examples

```
# Create a random network with 10 nodes.
nw <- random_network(10)
# Rewire nodes in the network each with probability 1/2
nw_rewired <- rewire_connections(nw, 0.5)
# Plot the two networks for comparison
g <- plot(nw)
plot(nw_rewired, g) # Pass in g to mirror the layout.
# Or plot the differential network.
plot_network_diff(nw, nw_rewired, g)
```

rewire_connections.network
Rewire connections

Description

Rewire connections

Usage

```
## S3 method for class 'network'
rewire_connections(x, prob_rewire = 1,
  weights = NULL, alpha = 100, beta = 1, epsilon = 10^-5,
  run_checks = TRUE, ...)
```

Arguments

x	The 'network', 'network_module', or 'matrix' object to modify.
prob_rewire	A value between 0 and 1. The connections to each node will be rewired with probability equal to prob_rewire.
weights	(Optional) A vector of weights for each node. These are used in addition to the degree of each node when sampling a node to rewire to.
alpha	A positive value used to parameterize the Beta distribution.
beta	A positive value used to parameterize the Beta distribution.
epsilon	A small constant added to the sampling probability of each node.
run_checks	If TRUE and 'x' is a matrix, then it is checked that 'x' is an adjacency matrix. This catches the case where 'x' is a weighted matrix, in which case the weights are removed and a warning is given.
...	Additional arguments.

Value

The modified module.

Examples

```
# Create a random network with 10 nodes.
nw <- random_network(10)
# Rewire nodes in the network each with probability 1/2
nw_rewired <- rewire_connections(nw, 0.5)
# Plot the two networks for comparison
g <- plot(nw)
plot(nw_rewired, g) # Pass in g to mirror the layout.
# Or plot the differential network.
plot_network_diff(nw, nw_rewired, g)
```

`rewire_connections.network_module` *Rewire connections*

Description

Rewire connections

Usage

```
## S3 method for class 'network_module'
rewire_connections(x, prob_rewire = 1,
  weights = NULL, alpha = 100, beta = 1, epsilon = 10^-5,
  run_checks = TRUE, ...)
```

Arguments

`x`	The 'network', 'network_module', or 'matrix' object to modify.
`prob_rewire`	A value between 0 and 1. The connections to each node will be rewired with probability equal to `prob_rewire`.
`weights`	(Optional) A vector of weights for each node. These are used in addition to the degree of each node when sampling a node to rewire to.
`alpha`	A positive value used to parameterize the Beta distribution.
`beta`	A positive value used to parameterize the Beta distribution.
`epsilon`	A small constant added to the sampling probability of each node.
`run_checks`	If `TRUE` and 'x' is a matrix, then it is checked that 'x' is an adjacency matrix. This catches the case where 'x' is a weighted matrix, in which case the weights are removed and a warning is given.
`...`	Additional arguments.

Value

The modified module.

Examples

```
# Create a random network with 10 nodes.
nw <- random_network(10)
# Rewire nodes in the network each with probability 1/2
nw_rewired <- rewire_connections(nw, 0.5)
# Plot the two networks for comparison
g <- plot(nw)
plot(nw_rewired, g) # Pass in g to mirror the layout.
# Or plot the differential network.
plot_network_diff(nw, nw_rewired, g)
```

rewire_connections_to_node *Rewire connections to a node*

Description

Rewire connections to a node

Usage

```
rewire_connections_to_node(x, node, prob_rewire = 1, weights = NULL,
  alpha = 100, beta = 1, epsilon = 10^-5, run_checks = TRUE, ...)
```

Arguments

x	The 'network', 'network_module', or 'matrix' object to modify.
node	The node to rewire.
prob_rewire	A value between 0 and 1, inclusive. Each connection to node will be rewired with probability equal to prob_rewire. Note, the degree of node is unchanged after this operation.
weights	(Optional) A vector of weights for each node. These are used in addition to the degree of each node when sampling nodes to rewire.
alpha	A positive value used to parameterize the Beta distribution.
beta	A positive value used to parameterize the Beta distribution.
epsilon	A small constant added to the sampling probability of each node.
run_checks	If TRUE and 'x' is a matrix, then it is checked that 'x' is an adjacency matrix. This catches the case where 'x' is a weighted matrix, in which case the weights are removed and a warning is given.
...	Additional arguments.

Value

The modified object.

Examples

```
# Create a random network with 10 nodes.
nw <- random_network(10)
# Rewire connections to the first node.
nw_rewired <- rewire_connections_to_node(nw, 1)
# Plot the two networks for comparison
g <- plot(nw)
plot(nw_rewired, g) # Pass in g to mirror the layout.
# Or plot the differential network.
plot_network_diff(nw, nw_rewired, g)
```

`rewire_connections_to_node.default` *Rewire connections to a node*

Description

Rewire connections to a node

Usage

```
## Default S3 method:
rewire_connections_to_node(x, node, prob_rewire = 1,
  weights = NULL, alpha = 100, beta = 1, epsilon = 10^-5,
  run_checks = TRUE, ...)
```

Arguments

`x`	The 'network', 'network_module', or 'matrix' object to modify.
`node`	The node to rewire.
`prob_rewire`	A value between 0 and 1, inclusive. Each connection to `node` will be rewired with probability equal to `prob_rewire`. Note, the degree of `node` is unchanged after this operation.
`weights`	(Optional) A vector of weights for each node. These are used in addition to the degree of each node when sampling nodes to rewire.
`alpha`	A positive value used to parameterize the Beta distribution.
`beta`	A positive value used to parameterize the Beta distribution.
`epsilon`	A small constant added to the sampling probability of each node.
`run_checks`	If `TRUE` and 'x' is a matrix, then it is checked that 'x' is an adjacency matrix. This catches the case where 'x' is a weighted matrix, in which case the weights are removed and a warning is given.
`...`	Additional arguments.

Value

The modified object.

Examples

```
# Create a random network with 10 nodes.
nw <- random_network(10)
# Rewire connections to the first node.
nw_rewired <- rewire_connections_to_node(nw, 1)
# Plot the two networks for comparison
g <- plot(nw)
plot(nw_rewired, g) # Pass in g to mirror the layout.
# Or plot the differential network.
plot_network_diff(nw, nw_rewired, g)
```

`rewire_connections_to_node.matrix` *Rewire connections to a node*

Description

Rewire connections to a node

Usage

```
## S3 method for class 'matrix'
rewire_connections_to_node(x, node, prob_rewire = 1,
  weights = NULL, alpha = 100, beta = 1, epsilon = 10^-5,
  run_checks = TRUE, ...)
```

Arguments

`x`	The 'network', 'network_module', or 'matrix' object to modify.
`node`	The node to rewire.
`prob_rewire`	A value between 0 and 1, inclusive. Each connection to `node` will be rewired with probability equal to `prob_rewire`. Note, the degree of `node` is unchanged after this operation.
`weights`	(Optional) A vector of weights for each node. These are used in addition to the degree of each node when sampling nodes to rewire.
`alpha`	A positive value used to parameterize the Beta distribution.
`beta`	A positive value used to parameterize the Beta distribution.
`epsilon`	A small constant added to the sampling probability of each node.
`run_checks`	If `TRUE` and 'x' is a matrix, then it is checked that 'x' is an adjacency matrix. This catches the case where 'x' is a weighted matrix, in which case the weights are removed and a warning is given.
`...`	Additional arguments.

Value

The modified object.

Examples

```
# Create a random network with 10 nodes.
nw <- random_network(10)
# Rewire connections to the first node.
nw_rewired <- rewire_connections_to_node(nw, 1)
# Plot the two networks for comparison
g <- plot(nw)
plot(nw_rewired, g) # Pass in g to mirror the layout.
# Or plot the differential network.
plot_network_diff(nw, nw_rewired, g)
```

`rewire_connections_to_node.network` *Rewire connections to a node*

Description

Rewire connections to a node

Usage

```
## S3 method for class 'network'
rewire_connections_to_node(x, node, prob_rewire = 1,
  weights = NULL, alpha = 100, beta = 1, epsilon = 10^-5,
  run_checks = TRUE, ...)
```

Arguments

`x`	The 'network', 'network_module', or 'matrix' object to modify.
`node`	The node to rewire.
`prob_rewire`	A value between 0 and 1, inclusive. Each connection to `node` will be rewired with probability equal to `prob_rewire`. Note, the degree of `node` is unchanged after this operation.
`weights`	(Optional) A vector of weights for each node. These are used in addition to the degree of each node when sampling nodes to rewire.
`alpha`	A positive value used to parameterize the Beta distribution.
`beta`	A positive value used to parameterize the Beta distribution.
`epsilon`	A small constant added to the sampling probability of each node.
`run_checks`	If `TRUE` and 'x' is a matrix, then it is checked that 'x' is an adjacency matrix. This catches the case where 'x' is a weighted matrix, in which case the weights are removed and a warning is given.
`...`	Additional arguments.

Value

The modified object.

Examples

```
# Create a random network with 10 nodes.
nw <- random_network(10)
# Rewire connections to the first node.
nw_rewired <- rewire_connections_to_node(nw, 1)
# Plot the two networks for comparison
g <- plot(nw)
plot(nw_rewired, g) # Pass in g to mirror the layout.
# Or plot the differential network.
plot_network_diff(nw, nw_rewired, g)
```

`rewire_connections_to_node.network_module` *Rewire connections to a node*

Description

Rewire connections to a node

Usage

```
## S3 method for class 'network_module'
rewire_connections_to_node(x, node,
  prob_rewire = 1, weights = NULL, alpha = 100, beta = 1,
  epsilon = 10^-5, run_checks = TRUE, ...)
```

Arguments

`x`	The 'network', 'network_module', or 'matrix' object to modify.
`node`	The node to rewire.
`prob_rewire`	A value between 0 and 1, inclusive. Each connection to `node` will be rewired with probability equal to `prob_rewire`. Note, the degree of `node` is unchanged after this operation.
`weights`	(Optional) A vector of weights for each node. These are used in addition to the degree of each node when sampling nodes to rewire.
`alpha`	A positive value used to parameterize the Beta distribution.
`beta`	A positive value used to parameterize the Beta distribution.
`epsilon`	A small constant added to the sampling probability of each node.
`run_checks`	If `TRUE` and 'x' is a matrix, then it is checked that 'x' is an adjacency matrix. This catches the case where 'x' is a weighted matrix, in which case the weights are removed and a warning is given.
`...`	Additional arguments.

Value

The modified object.

Examples

```
# Create a random network with 10 nodes.
nw <- random_network(10)
# Rewire connections to the first node.
nw_rewired <- rewire_connections_to_node(nw, 1)
# Plot the two networks for comparison
g <- plot(nw)
plot(nw_rewired, g) # Pass in g to mirror the layout.
# Or plot the differential network.
plot_network_diff(nw, nw_rewired, g)
```

ring_lattice_cpp *C++ implementation for creating a ring lattice*

Description

C++ implementation for creating a ring lattice

Usage

```
ring_lattice_cpp(p, neig_size)
```

Arguments

p	The number of nodes in the lattice.
neig_size	The neighborhood side within which nodes are connected.

Value

Returns the adjacency matrix for the ring lattice.

rzinb *The Zero-Inflated Negative Binomial Distribution*

Description

The Zero-Inflated Negative Binomial Distribution

Usage

```
rzinb(n, size, mu, rho)
```

Arguments

n	The number of random values to return.
size	The dispersion paramater used in dnbinom.
mu	The mean parameter used in dnbinom.
rho	The zero-inflation parameter.

Examples

```
x <- rzinb(10, 1, 10, 0.1)
p <- pzinb(x, 1, 10, 0.1)
y <- qzinb(p, 1, 10, 0.1)
all(x == y)
# Compute P(0 < X < 5) for X ~ ZINB(1, 10, 0.1)
sum(dzinb(0:5, 1, 10, 0.1))
```

sample_link_nodes *Sample link nodes for new module*

Description

Sample link nodes for new module

Usage

```
sample_link_nodes(n, nodes, degree, alpha = 100, beta = 1,
  epsilon = 10^-5, ...)
```

Arguments

n	The number of link nodes to sample.
nodes	The nodes to sample from.
degree	The degree of each node.
alpha	A positive value used to parameterize the Beta distribution.
beta	A positive value used to parameterize the Beta distribution.
epsilon	Used when sampling link nodes.
...	Additional arguments are ignored.

Value

A vector of selected nodes (possibly of length 1).

Note

This function is used by create_modules_for_network and is not meant to be used externally.

sample_module_nodes *Sample nodes for new module*

Description

Sample nodes for new module

Usage

```
sample_module_nodes(n, nodes, degree, nu = 0.01, ...)
```

Arguments

n	The number of nodes to sample.
nodes	The nodes available to sample from.
degree	The degree of each node.
nu	Multiplier for nodes that are already in one or more modules.
...	Additional arguments are ignored.

Value

A vector of selected nodes of length m.

Note

This function is used by `create_modules_for_network` and is not meant to be used externally.

---

sample_reference_data *Sample genes from reference dataset*

Description

Sample genes from reference dataset

Usage

```
sample_reference_data(reference_data, p, percent_ZI = NULL,
  threshold_ZI = 0.2)
```

Arguments

reference_data	The reference data.frame to use.
p	The number of genes (columns) to sample
percent_ZI	The percentage of genes to be zero inflated. If NULL, the genes are sampled at random; in this case, the empirical distribution of gene expression profiles will determine the probablility that a sampled gene is zero inflated.
threshold_ZI	The minimum proportion of zero counts for a gene to be considered as zero inflated.

Examples

```
data(reference)
rnaseq <- reference$rnaseq
rnaseq_subset <- sample_reference_data(rnaseq, 10)
```

set_module_edges *Internal function used to set the edges in a module*

Description

Internal function used to set the edges in a module

Usage

```
set_module_edges(module, edges)
```

Arguments

module	The 'network_module' object to modify.
edges	A matrix used to indicate the edges in the module. If the matrix is square and contains the same number of rows and columns as nodes in the module, then it is assumed to be an adjacency matrix and the nonzero lower-triangle values of the matrix are used to indicate edges in the module. If the matrix is not square, the first two columns are assumed to be an edge list.

Value

The modified 'network_module' object.

set_module_name *Set the name for a module*

Description

Set the name for a module

Usage

```
set_module_name(module, module_name)
```

Arguments

module	The 'network_module' object to modify.
module_name	A character string.

Value

The modified 'network_module' object.

Examples

```
nw <- random_network(10)
nw <- gen_partial_correlations(nw)
module <- nw$modules[[1]]
named_module <- set_module_name(module, "new name")
```

`set_module_weights` *Internal function to set the connection weights for a module*

Description

Internal function to set the connection weights for a module

Usage

```
set_module_weights(module, weights)
```

Arguments

`module`	The 'network_module' object to modify.
`weights`	A vector or matrix of weights for each connetions. If a vector, its length must equal the number of connections in the module. If a matrix, it should be square with the number of columns equal to the number of nodes in the module; only the entries in the lower triangle that correspond to connections in the module will be used.

Value

The modified 'network_module' object.

`set_node_names` *Set the node names in a network*

Description

Set the node names in a network

Usage

```
set_node_names(network, node_names)
```

Arguments

`network`	The network to modify.
`node_names`	A vector of strings containing the names for each node in the network. If a numeric vector is provided, the values will be coerced into strings. If 'node_names' is `NULL`, then the names will default to "1", "2", ..., "p".

Value

The modified network.

Examples

```
# Create a random network with 10 nodes.
nw <- random_network(10)
get_node_names(nw) # Default names are 1, 2, ..., 10.
nw <- set_node_names(nw, paste("node", 1:10, sep = "_"))
get_node_names(nw) # Print out updated node names.
# Modules only contain the indicies to nodes, not the node names
module <- nw$modules[[1]]
get_node_names(module)
# When converting the network to a matrix, node names appear as column names.
adj_matrix <- get_adjacency_matrix(nw)
colnames(adj_matrix)
```

---

update_module_with_random_weights

Generate small-world network structure for module

Description

The small-world network is generated using the Watts-Strogatz method. See watts.strogatz.game for details.

Usage

```
update_module_with_random_weights(module, rdist = function(n) {
  runif(n, 0.5, 1) * (-1)^rbinom(n, 1, 0.5) }, ...)
```

Arguments

module	The network_module object to modify.
rdist	A distribution function that generates random numbers. The first argument should specify the number of weights to generate. By default, weights are generated uniformly from the set (-1, -0.5)U(0.5, 1).
...	Additional parameters are ignored.

Value

An updated 'network_module' object.

Examples

```
# Create a random module.
module <- random_module(1:10)
is_weighted(module)
# Add a random weight to each connection.
module <- update_module_with_random_weights(module)
is_weighted(module)
```

Index