

# Package ‘GWENA’

May 4, 2024

**Title** Pipeline for augmented co-expression analysis

**Version** 1.14.0

**Description** The development of high-throughput sequencing led to increased use of co-expression analysis to go beyond single feature (i.e. gene) focus. We propose GWENA (Gene Whole co-Expression Network Analysis), a tool designed to perform gene co-expression network analysis and explore the results in a single pipeline. It includes functional enrichment of modules of co-expressed genes, phenotypical association, topological analysis and comparison of networks configuration between conditions.

**License** GPL-3

**Encoding** UTF-8

**LazyData** true

**LazyDataCompression** xz

**RoxygenNote** 7.1.1

**Depends** R (>= 4.1)

**Suggests** testthat (>= 2.1.0), knitr (>= 1.25), rmarkdown (>= 1.16), prettydoc (>= 0.3.0), htr (>= 1.4.1), S4Vectors (>= 0.22.1), BiocStyle (>= 2.15.8)

**Imports** WGCNA (>= 1.67), dplyr (>= 0.8.3), dynamicTreeCut (>= 1.63-1), ggplot2 (>= 3.1.1), gprofiler2 (>= 0.1.6), magrittr (>= 1.5), tibble (>= 2.1.1), tidyr (>= 1.0.0), NetRep (>= 1.2.1), igraph (>= 1.2.4.1), RColorBrewer (>= 1.1-2), purrr (>= 0.3.3), rlist (>= 0.4.6.1), matrixStats (>= 0.55.0), SummarizedExperiment (>= 1.14.1), stringr (>= 1.4.0), cluster (>= 2.1.0), grDevices (>= 4.0.4), methods, graphics, stats, utils

**VignetteBuilder** knitr

**biocViews** Software, GeneExpression, Network, Clustering, GraphAndNetwork, GeneSetEnrichment, Pathways, Visualization, RNASeq, Transcriptomics, mRNAMicroarray, Microarray, NetworkEnrichment, Sequencing, GO

**BugReports** <https://github.com/Kumquatium/GWENA/issues>

**git\_url** <https://git.bioconductor.org/packages/GWENA>

**git\_branch** RELEASE\_3\_19

**git\_last\_commit** 165ba47

**git\_last\_commit\_date** 2024-04-30

**Repository** Bioconductor 3.19

**Date/Publication** 2024-05-03

**Author** Gwenaëlle Lemoine [aut, cre] (<<https://orcid.org/0000-0003-4747-1937>>),  
Marie-Pier Scott-Boyer [ths],  
Arnaud Droit [fnd]

**Maintainer** Gwenaëlle Lemoine <[lemoine.gwenaelle@gmail.com](mailto:lemoine.gwenaelle@gmail.com)>

## Contents

.check_data_expr . . . . .	3
.check_gost . . . . .	3
.check_module . . . . .	4
.check_network . . . . .	4
.contingencyTable . . . . .	5
.cor_func_match . . . . .	5
associate_phenotype . . . . .	6
bio_enrich . . . . .	7
build_graph_from_sq_mat . . . . .	8
build_net . . . . .	8
compare_conditions . . . . .	10
detect_modules . . . . .	12
filter_low_var . . . . .	13
filter_RNA_seq . . . . .	14
get_fit.cor . . . . .	15
get_fit.expr . . . . .	16
get_hub_degree . . . . .	17
get_hub_genes . . . . .	18
get_hub_high_co . . . . .	19
get_hub_kleinberg . . . . .	19
get_sub_clusters . . . . .	20
gg_palette . . . . .	21
gtex_expr . . . . .	21
gtex_traits . . . . .	22
is_data_expr . . . . .	22
is_gost . . . . .	23
is_module . . . . .	24
is_network . . . . .	24
join_gost . . . . .	25
kuehne_expr . . . . .	26
kuehne_traits . . . . .	26
matchsub . . . . .	27

`.check_data_expr` 3

<code>orderAsNumeric</code>	28
<code>plot_comparison_stats</code>	28
<code>plot_enrichment</code>	29
<code>plot_expression_profiles</code>	30
<code>plot_module</code>	31
<code>plot_modules_merge</code>	33
<code>plot_modules_phenotype</code>	35
<code>quiet</code>	36
<code>z_summary</code>	36

**Index** 38

---

`.check_data_expr`      *Run checks on an object to test if it's a data\_expr*

---

**Description**

Check an object to be a data.frame or a matrix compatible of genes and samples.

**Usage**

```
.check_data_expr(data_expr)
```

**Arguments**

`data_expr`      matrix or data.frame, expression data with genes as column and samples as row.

**Value**

Throw an error if doesn't correspond

---

`.check_gost`      *Run checks on an object to test if it's a gost result*

---

**Description**

Take a list that should be a gost result and check if format is good.

**Usage**

```
.check_gost(gost_result)
```

**Arguments**

`gost_result`      list, gprofiler2::gost result

**Value**

Throw an error if doesn't correspond

---

<code>.check_module</code>	<i>Run checks on an object to test if it's a module or a list of modules</i>
----------------------------	--

---

**Description**

Check content of a given object to determine if it's a module or a list of modules, meaning a single vector of characters which are gene names, or a named list of these vectors

**Usage**

```
.check_module(module, is_list = FALSE)
```

**Arguments**

<code>module</code>	vector or list, object to test to be a module or list of modules
<code>is_list</code>	boolean, indicate if module must be tested as a single module or a list of modules

**Value**

Throw an error if doesn't correspond

---

<code>.check_network</code>	<i>Run checks on an object to test if it's a network</i>
-----------------------------	--

---

**Description**

Check content of a given object to determine if it's a network, meaning a squared matrix of similarity score between genes.

**Usage**

```
.check_network(network)
```

**Arguments**

<code>network</code>	matrix or data.frame, object to test to be a network
----------------------	--

**Value**

Throw an error if doesn't correspond

---

.contingencyTable      *Calculate a contingency table of module overlap between datasets*

---

**Description**

Calculate a contingency table of module overlap between datasets

**Usage**

```
.contingencyTable(modAssignments, mods, tiNodelist)
```

**Arguments**

- modAssignments    a list where the first element is the 'moduleAssignments' vector in the discovery dataset, and the second element is the 'moduleAssignments' vector in the test dataset.
- mods                the 'modules' vector for the discovery dataset.
- tiNodelist         a vector of node IDs in the test dataset.

**Value**

A list containing a contingency table, a vector of the proportion of nodes present in the test dataset for each module, a vector containing the number of nodes present in the test dataset for each module, a vector of the node names present in both the discovery and test datasets, a vector of modules that are both requested and have nodes present in the test dataset, and the modAssignments vector containing only nodes present in the test dataset.

---

.cor\_func\_match      *Match a correlation function based on a name*

---

**Description**

Translate a function name into an R function.

**Usage**

```
.cor_func_match(cor_func = c("pearson", "spearman", "bicolor"))
```

**Arguments**

- cor\_func            string of the name of the correlation to be use

**Value**

A function corresponding to the correlation required

---

associate\_phenotype    *Modules phenotypic association*

---

## Description

Compute the correlation between all modules and the phenotypic variables

## Usage

```
associate_phenotype(
  eigengenes,
  phenotypes,
  cor_func = c("pearson", "spearman", "kendall", "other"),
  your_func = NULL,
  id_col = NULL,
  ...
)
```

## Arguments

eigengenes	matrix or data.frame, eigengenes of the modules. Provided by the output of modules_detection.
phenotypes	matrix or data.frame, phenotypes for each sample to associate.
cor_func	string, name of the correlation function to be used. Must be one of "pearson", "spearman", "kendall", "other". If "other", your_func must be provided
your_func	function returning a correlation matrix. Final values must be in [-1;1] range
id_col	string or vector of string, optional name of the columns containing the common id between eigengenes and phenotypes.
...	any arguments compatible with <code>cor</code> .

## Value

A list of two data.frames : associations modules/phenotype and p.values associated to this associations

## Examples

```
eigengene_mat <- data.frame(mod1 = rnorm(20, 0.1, 0.2),
  mod2 = rnorm(20, 0.2, 0.2))
phenotype_mat <- data.frame(phenA = sample(c("X", "Y", "Z"), 20,
  replace = TRUE),
  phenB = sample(c("U", "V"), 20, replace = TRUE),
  stringsAsFactors = FALSE)
association <- associate_phenotype(eigengene_mat, phenotype_mat)
```

---

bio_enrich	<i>Modules enrichment</i>
------------	---------------------------

---

## Description

Enrich genes list from modules.

## Usage

```
bio_enrich(module, custom_gmt = NULL, ...)
```

## Arguments

module	vector or list, vector of gene names representing a module or a named list of this modules.
custom_gmt	string or list, path to a gmt file or a list of these path.
...	any other parameter you can provide to gprofiler2::gost function.

## Value

A gprofiler2::gost output, meaning a named list containing a 'result' data.frame with enrichment information on the different databases and custom gmt files, and a 'meta' list containing informations on the input args, the version of gost, timestamp, etc. For more detail, see ?gprofiler2::gost.

## Examples

```
custom_path <- system.file("extdata", "h.all.v6.2.symbols.gmt",  
                           package = "GWENA", mustWork = TRUE)  
  
single_module <- c("BIRC3", "PMAIP1", "CASP8", "JUN", "BCL2L11", "MCL1",  
                  "IL1B", "SPTAN1", "DIABLO", "BAX", "BIK", "IL1A", "BID",  
                  "CDKN1A", "GADD45A")  
single_module_enriched <- bio_enrich(single_module, custom_path)  
  
multi_module <- list(mod1 = single_module,  
                    mod2 = c("TAF1C", "TARBP2", "POLH", "CETN2", "POLD1",  
                             "CANT1", "PDE4B", "DGCR8", "RAD51", "SURF1",  
                             "PNP", "ADA", "NME3", "GTF3C5", "NT5C"))  
multi_module_enriched <- bio_enrich(multi_module, custom_path)
```

build\_graph\_from\_sq\_mat

*Return graph from squared matrix network*

---

### Description

Takes a squared matrix containing the pairwise similarity scores for each gene and return a igraph object.

### Usage

```
build_graph_from_sq_mat(sq_mat)
```

### Arguments

sq\_mat                    matrix or data.frame, squared matrix representing

### Value

An igraph object

### Examples

```
mat <- matrix(runif(40*40), 40)
build_graph_from_sq_mat(mat)
```

---

build\_net

*Network building by co-expression score computation*

---

### Description

Compute the adjacency matrix, then the TOM to build the network. Than detect the modules by hierarchical clustering and thresholding

### Usage

```
build_net(
  data_expr,
  fit_cut_off = 0.9,
  cor_func = c("pearson", "spearman", "bicor", "other"),
  your_func = NULL,
  power_value = NULL,
  block_size = NULL,
  stop_if_fit_pb = FALSE,
  pct_power_ic = 0.7,
```



```

    network_type = c("unsigned", "signed", "signed hybrid"),
    tom_type = c("unsigned", "signed", "signed Nowick", "unsigned 2", "signed 2", "none"),
    keep_matrices = c("none", "cor", "adja", "both"),
    n_threads = NULL,
    ...
)

```

### Arguments

data_expr	matrix or data.frame or SummarizedExperiment, expression data with genes as column and samples as row.
fit_cut_off	float, cut off by which R <sup>2</sup> (coefficient of determination) will be thresholded. Must be in ]0;1[.
cor_func	string, name of the correlation function to be used. Must be one of "pearson", "spearman", "bicor", "other". If "other", your_func must be provided
your_func	function returning correlation values. Final values must be in [-1;1]
power_value	integer, power to be applied to the adjacency matrix. If NULL, will be estimated by trying different power law fitting.
block_size	integer, size of blocks by which operations can be proceed. Helping if working with low capacity computers. If null, will be estimated.
stop_if_fit_pb	boolean, does not finding a fit above fit_cut_off, or having a power too low or too high (based on WGCNA FAQ recommended powers) should stop process, or just print a warning and return the highest fitting power.
pct_power_ic	float, confidence interval by which the power fitted should be evaluated for too high or too low a power.
network_type	string, type of network to be used. Either "unsigned", "signed", "signed hybrid". See details.
tom_type	string, type of the topological overlap matrix to be computed. Either "none", "unsigned", "signed", "signed Nowick", "unsigned 2", "signed 2" and "signed Nowick 2". See detail at <a href="#">TOMsimilarityFromExpr</a> .
keep_matrices	string, matrices to keep in final object. Can be one of "none", "cor", "adja", "both". It is usefull to keep both if you plant to use <a href="#">compare_conditions</a> .
n_threads	integer, number of threads that can be used to paralellise the computing
...	any other parameter compatible with <a href="#">adjacency.fromSimilarity</a>

### Value

list containing network matrix, metadata of input parameters and power fitting information.

### Examples

```
net <- build_net(kuehne_expr[, seq_len(350)], n_threads = 1)
```

---

compare\_conditions      *Compare modules topology between conditions*

---

### Description

Take modules built from multiples conditions and search for preservation, non-preservation or one of them, against one or mutiple conditions of reference. Use 7 topological features to perform the differents test, and use permutation to validate results.

### Usage

```
compare_conditions(
  data_expr_list,
  adja_list,
  cor_list = NULL,
  modules_list,
  ref = names(data_expr_list)[1],
  test = NULL,
  cor_func = c("pearson", "spearman", "bicor", "other"),
  your_func = NULL,
  n_perm = 10000,
  test_alternative_hyp = c("greater", "less", "two.sided"),
  pvalue_th = 0.01,
  n_threads = NULL,
  ...
)
```

### Arguments

data_expr_list	list of matrix or data.frame or SummarizedExperiment, list of expression data by condition, with genes as column and samples as row.
adja_list	list of adjacency matrices, list of square tables by condition, representing connectivity between each genes as returned by build_net.
cor_list	list of matrices and/or data.frames, list of square tables by condition, representing correlation between each gene. Must be the same used to create networks in <a href="#">build_net</a> . If NULL, will be re-calculated according to cor_func.
modules_list	list of modules or nested list of modules, list of modules in one condition (will be considered as the one from reference) or a condition named list with list of modules built in each one.
ref	string or vector of strings, condition(s) name to be used as reference for permutation tests, or "cross comparison" if you want to compare each condition with the other as reference. Default will be the name of the first element in data_expr_list.
test	string or vector of strings, condition(s) name to be tested for permutation tests. If NULL, all conditions except these in ref will be taken. If ref is set to "cross comparison", any test specified will be ignored.



```
lapply(mod_by_cond, `[`, "modules"),
n_perm = 100)
```

---

detect_modules	<i>Modules detection in a network</i>
----------------	---------------------------------------

---

### Description

Detect the modules by hierarchical clustering .

### Usage

```
detect_modules(
  data_expr,
  network,
  min_module_size = min(20, ncol(data_expr)/2),
  clustering_th = NULL,
  merge_close_modules = TRUE,
  merge_threshold = 0.75,
  detailed_result = TRUE,
  pam_respects_dendro = FALSE,
  ...
)
```

### Arguments

data_expr	matrix or data.frame or SummarizedExperiment, expression data with genes as column and samples as row.
network	matrix or data.frame, strength of gene co-expression (edge values).
min_module_size	integer, lowest number of gene allowed in a module. If none provided, estimated.
clustering_th	float, threshold to be used by the clustering method. For now <a href="#">cutreeDynamic</a> .
merge_close_modules	boolean, does closest modules (based on eigengene) should be merged together.
merge_threshold	float, eigengenes correlation value over which close modules will be merged. Must be in ]0;1[. See <a href="#">mergeCloseModules</a>
detailed_result	boolean, does pre-merge modules (if applicable) and dendrogram included in output.

pam\_respects\_dendro  
 boolean, If TRUE, the Partitioning Around Medoids (PAM) stage will respect the dendrogram in the sense that objects and small clusters will only be assigned to clusters that belong to the same branch that the objects or small clusters being assigned belong to.

... any other parameter compatible with [mergeCloseModules](#)

### Value

list containing modules detected, modules\_eigengenes, and if asked for, modules pre-merge and dendrograms of genes and merged modules

### Examples

```
df <- kuehne_expr[1:24, 1:350]
net <- build_net(df, n_threads = 1)
detect_modules(df, net$network)
```

---

filter\_low\_var                    *Filtering genes with low variability*

---

### Description

Remove low varying genes based on the percentage given and the type of variation specified.

### Usage

```
filter_low_var(data_expr, pct = 0.8, type = c("mean", "median", "mad"))
```

### Arguments

data\_expr            matrix or data.frame or SummarizedExperiment, table of expression values (either microarray or RNA-seq), with genes as column and samples as row

pct                    float, percentage of gene to keep, value must be in ]0;1[

type                    string, function name used for filtration. Should be either "mean", "median", or "mad"

### Value

A data.frame of filtered genes

### Examples

```
df <- matrix(abs(rnorm(15*45)), 15)
colnames(df) <- paste0("gene_", seq_len(ncol(df)))
rownames(df) <- paste0("sample_", seq_len(nrow(df)))
df_filtered <- filter_low_var(df)
```

---

filter_RNA_seq	<i>Filtering of low counts</i>
----------------	--------------------------------

---

### Description

Keeping genes with at least one sample with count above min\_count in RNA-seq data.

### Usage

```
filter_RNA_seq(  
  data_expr,  
  min_count = 5,  
  method = c("at least one", "mean", "all")  
)
```

### Arguments

data_expr	matrix or data.frame or SummarizedExperiment, table of expression values (either microarray or RNA-seq), with genes as column and samples as row.
min_count	integer, minimal number of count to be considered in method.
method	string, name of the method for filtering. Must be one of "at least one", "mean", or "all"

### Details

Low counts in RNA-seq can bring noise to gene co-expression module building, so filtering them help to improve quality.

### Value

A data.frame of filtered genes

### Examples

```
df <- matrix(abs(rnorm(15*45)), 15) * 3  
colnames(df) <- paste0("gene_", seq_len(ncol(df)))  
rownames(df) <- paste0("sample_", seq_len(nrow(df)))  
df_filtered <- filter_RNA_seq(df)
```

---

get_fit.cor	<i>Calculating best fit of a power law on correlation matrix computed on expression data</i>
-------------	--

---

### Description

Adjust a correlation matrix depending of the type of network, then try to parameter a power law for best fit

### Usage

```
get_fit.cor(
  cor_mat,
  fit_cut_off = 0.9,
  network_type = c("unsigned", "signed", "signed hybrid"),
  block_size = NULL,
  ...
)
```

### Arguments

cor_mat	matrix or data.frame of genes correlation.
fit_cut_off	float, cut off by which R <sup>2</sup> (coefficient of determination) will be thresholded. Must be in ]0;1[.
network_type	string giving type of network to be used. Either "unsigned", "signed", "signed hybrid". See details.
block_size	integer giving size of blocks by which operations can be proceed. Helping if working with low capacity computers. If null, will be estimated.
...	any other parameter compatible with <a href="#">pickSoftThreshold.fromSimilarity</a>

### Details

network\_type indicate which transformation will be applied on the correlation matrix to return the similarity score.

**signed** will modify the range [-1;1] to [0.5;1.5] (because of log10 beeing used for scale free index computation)

**unsigned** will return absolute value (moving from [-1;1] to [0;1])

**signed hybrid** will replace all negative values by 0 (moving from [-1;1] to [0;1])

### Value

A list containing power of the law for best fit, fit table, and metadata about the arguments used.

### Examples

```
get_fit.cor(cor_mat = cor(kuehne_expr[, seq_len(100)]))
```

get\_fit.expr

*Calculating best fit of a power law on expression data***Description**

Computes correlation matrix of the gene expression data, adjust it depending of the type of network, then try to parameter a power law for best fit

**Usage**

```
get_fit.expr(
  data_expr,
  fit_cut_off = 0.9,
  cor_func = c("pearson", "spearman", "bicor", "other"),
  your_func = NULL,
  network_type = c("unsigned", "signed", "signed hybrid"),
  block_size = NULL,
  ...
)
```

**Arguments**

data_expr	matrix or data.frame or SummarizedExperiment, expression data with genes as column and samples as row.
fit_cut_off	float, cut off by which R <sup>2</sup> (coefficient of determination) will be thresholded. Must be in ]0;1[.
cor_func	string specifying correlation function to be used. Must be one of "pearson", "spearman", "bicor", "other". If "other", your_func must be provided
your_func	function returning correlation values. Final values must be in [-1;1]
network_type	string giving type of network to be used. Either "unsigned", "signed", "signed hybrid". See details.
block_size	integer giving size of blocks by which operations can be proceed. Helping if working with low capacity computers. If null, will be estimated.
...	any other parameter compatible with <a href="#">pickSoftThreshold.fromSimilarity</a>

**Details**

network\_type indicate which transformation will be applied on the correlation matrix to return the similarity score.

**signed** will modify the range [-1;1] to [0.5;1.5] (because of log10 beeing used for scale free index computation)

**unsigned** will return absolute value (moving from [-1;1] to [0;1])

**signed hybrid** will replace all negative values by 0 (moving from [-1;1] to [0;1])



**Value**

A list containing power of the law for best fit, fit table, and metadata about the arguments used.

**Examples**

```
get_fit.expr(kuehne_expr[, seq_len(100)])
```

---

get_hub_degree	<i>Determine hub genes based on degree</i>
----------------	--

---

**Description**

Remove edges from the graph which value is under `weight_th` then compute degree of each node (gene). Hub gene are genes whose degree value is above average degree value of the thresholded network.

**Usage**

```
get_hub_degree(network, modules = NULL, weight_th = 0.2)
```

**Arguments**

<code>network</code>	matrix or data.frame, square table representing connectivity between each genes as returned by <code>build_net</code> . Can be whole network or a single module.
<code>modules</code>	list, modules defined as list of gene vectors. If null, network is supposed to be the whole network or an already split module
<code>weight_th</code>	decimal, weight threshold under or equal to which edges will be removed

**Details**

GWENA natively build networks using WGCNA. These networks are complete in a graph theory sens, meaning all nodes are connected to each other. Therefore a threshold need to be applied so degree of all nodes isn't the same.

**Value**

A list of vectors, or single vector of gene names

**Examples**

```
mat <- matrix(runif(40*40), 40)
colnames(mat) <- paste0("gene_", seq_len(ncol(mat)))
rownames(mat) <- paste0("gene_", seq_len(nrow(mat)))
get_hub_degree(mat)
```

---

 get\_hub\_genes

*Determine hub genes inside each module*


---

### Description

Return genes considered as hub genes inside each module of a network following the selected method. Method will be launched with default parameters. If specific parameters desired, please use directly the function get\_hub\_... itself.

### Usage

```
get_hub_genes(
  network,
  modules = NULL,
  method = c("highest connectivity", "superior degree", "Kleinberg's score")
)
```

### Arguments

network	matrix or data.frame, square table representing connectivity between each genes as returned by build_net. Can be whole network or a single module.
modules	list, modules defined as list of gene vectors. If null, network is supposed to be the whole network or an already split module
method	string, name of the method to be used for hub gene detection. See details.

### Details

**highest connectivity** Select the top n (n depending on parameter given) highest connected genes. Similar to WGCNA::chooseTopHubInEachModule.

**superior degree** Select genes which degree is greater than average connection degree of the network. Definition from network theory.

**Kleinberg's score** Select genes which Kleinberg's score superior to provided threshold.

### Value

A list of vectors representing hub genes, by module

### Examples

```
mat <- matrix(runif(40*40), 40)
colnames(mat) <- paste0("gene_", seq_len(ncol(mat)))
rownames(mat) <- paste0("gene_", seq_len(nrow(mat)))
get_hub_genes(mat)
```

---

get_hub_high_co	<i>Determine hub genes based on connectivity</i>
-----------------	--

---

**Description**

Compute connectivity of each gene by module if provided or for whole network if not, and return the top\_n highest connected ones.

**Usage**

```
get_hub_high_co(network, modules = NULL, top_n = 5)
```

**Arguments**

network	matrix or data.frame, square table representing connectivity between each genes as returned by build_net. Can be whole network or a single module.
modules	list, modules defined as list of gene vectors. If null, network is supposed to be the whole network or an already split module
top_n	integer, number of genes to be considered as hub genes

**Value**

A list of vectors, or single vector of gene names

**Examples**

```
mat <- matrix(runif(40*40), 40)
colnames(mat) <- paste0("gene_", seq_len(ncol(mat)))
rownames(mat) <- paste0("gene_", seq_len(nrow(mat)))
get_hub_high_co(mat)
```

---

get_hub_kleinberg	<i>Determine hub genes based on Kleinberg's score</i>
-------------------	---

---

**Description**

Compute Kleinberg's score (defined as the principal eigenvector of  $A \cdot t(A)$ , where A is the similarity matrix of the graph) of each gene by module if provided or for whole network if not, and return the top\_n highest ones.

**Usage**

```
get_hub_kleinberg(network, modules = NULL, top_n = 5, k_th = NULL)
```

**Arguments**

network	matrix or data.frame, square table representing connectivity between each genes as returned by build_net. Can be whole network or a single module.
modules	list, modules defined as list of gene vectors. If null, network is supposed to be the whole network or an already split module
top_n	integer, number genes to be considered as hub genes
k_th	decimal, Kleinberg's score threshold above or equal to which genes are considered as hubs

**Details**

If you provide a top\_n value, you can't provide a k\_th value and vice versa. If none of them is provided, top\_n = 5. For more information on Kleinberg's score, look at [hub\\_score](#) from igrph.

**Value**

A list of vectors, or single vector of gene names

**Examples**

```
mat <- matrix(runif(40*40), 40)
colnames(mat) <- paste0("gene_", seq_len(ncol(mat)))
rownames(mat) <- paste0("gene_", seq_len(nrow(mat)))
get_hub_degree(mat)
get_hub_kleinberg(mat, top_n = NULL, k_th = 0.9)
```

---

get_sub_clusters	<i>Detect sub clusters</i>
------------------	----------------------------

---

**Description**

Use a partitioning around medoid (PAM, or k-medoid) clustering method to detect clusters into a provided module using the strength matrix of the network

**Usage**

```
get_sub_clusters(network, seq_k = seq_len(15), fit_plot = TRUE, ...)
```

**Arguments**

network	matrix or data.frame, strength of gene co-expression (edge values).
seq_k	vector, sequence of k number of cluster to test
fit_plot	boolean, does the plot with silhouette coefficient depending on the k tested should be plotted.
...	any other parameter compatible with the <a href="#">pam</a> function.

**Value**

data.frame, a two cols table with the gene id in the first one, and the cluster number assignation in the second one.

**Examples**

```
df <- kuehne_expr[1:24, 1:350]
net <- build_net(df, n_threads = 1)
mods <- detect_modules(df, net$network)
net_mod_1 <- net$network[mods$modules$`1`, mods$modules$`1`]
get_sub_clusters(net_mod_1)
```

---

gg_palette	<i>Mimicking</i>	<i>ggplot</i>	<i>palette</i>	<i>Source</i>	:
	<a href="https://stackoverflow.com/questions/8197559/emulate-ggplot2-default-color-palette">https://stackoverflow.com/questions/8197559/emulate-ggplot2-default-color-palette</a>				

---

**Description**

Mimicking ggplot palette Source : <https://stackoverflow.com/questions/8197559/emulate-ggplot2-default-color-palette>

**Usage**

```
gg_palette(n)
```

**Arguments**

n integer, number of colors wanted

**Value**

character vector, hexadecimal colors of length n

---

gtex_expr	<i>Transcriptomic muscle data from GTEx consortium RNA-seq data</i>
-----------	---

---

**Description**

A subset of GTEx RNA-seq dataset containing read counts collapsed to gene level.

**Usage**

```
gtex_expr
```

**Format**

A data frame with 50 rows (samples) and 15000 columns (genes)

**Source**

<https://gtexportal.org/home/datasets>

---

gtex\_traits

*Traits data linked to samples in transcriptomic data from GTEx*

---

**Description**

A dataset containing phenotypes of donors. From public data. Note: protected data contain more information but require dbGap accessh (see <https://gtexportal.org/home/protectedDataAccess>).

**Usage**

gtex\_traits

**Format**

A data frame with 50 rows (samples) and 4 columns :

**SUBJID** Subject ID, GTEx Public Donor ID

**SEX** Sex, donor's Identification of sex based upon self-report : 1=Male, 2=Female

**AGE** Age range, elapsed time since birth in years

**DTHHRDY** Hardy Scale : 0=Ventilator Case, 1=Violent and fast death, 2=Fast death of natural causes, 3=Intermediate death, 4=Slow death)

**Source**

<https://gtexportal.org/home/datasets>

---

is\_data\_expr

*Determine if an object is a data\_expr in sens of GWENA*

---

**Description**

Check an object to be a data.frame or a matrix compatible of genes and samples.

**Usage**

is\_data\_expr(data\_expr)

**Arguments**

data\_expr      matrix or data.frame, expression data with genes as column and samples as row.

**Value**

list, a boolean as first element and in second element NULL or the reason why boolean is set to FALSE

**Examples**

```
expr <- matrix(runif(15*40), 15)
colnames(expr) <- paste0("gene_", seq_len(ncol(expr)))
rownames(expr) <- paste0("gene_", seq_len(nrow(expr)))
is_data_expr(expr)
```

---

is_gost	<i>Determine if an object is a gost object</i>
---------	--

---

**Description**

Check content of a given object to determine if it's a gost object

**Usage**

```
is_gost(gost_result)
```

**Arguments**

gost\_result      list, gprofiler2::gost result

**Value**

list, a boolean as first element and in second element NULL or the reason why boolean is set to FALSE

**Examples**

```
single_module <- c("BIRC3", "PMAIP1", "CASP8", "JUN", "BCL2L11", "MCL1",
                  "IL1B", "SPTAN1", "DIABLO", "BAX", "BIK", "IL1A", "BID",
                  "CDKN1A", "GADD45A")
single_module_enriched <- bio_enrich(single_module)
is_gost(single_module_enriched)
```

---

is_module	<i>Determine if an object is a module or a list of modules</i>
-----------	--

---

**Description**

Check content of a given object to determine if it's a module or a list of modules, meaning a single vector of characters which are gene names, or a named list of these vectors

**Usage**

```
is_module(module, is_list = FALSE)
```

**Arguments**

module	vector or list, object to test to be a module or list of modules
is_list	boolean, indicate if module must be tested as a single module or a list of modules

**Value**

list, a boolean as first element and in second element NULL or the reason why boolean is set to FALSE

**Examples**

```
single_module <- c("BIRC3", "PMAIP1", "CASP8", "JUN", "BCL2L11", "MCL1",
                  "IL1B", "SPTAN1", "DIABLO", "BAX", "BIK", "IL1A", "BID",
                  "CDKN1A", "GADD45A")
is_module(single_module)

multi_module <- list(mod1 = single_module,
                    mod2 = c("TAF1C", "TARBP2", "POLH", "CETN2", "POLD1",
                            "CANT1", "PDE4B", "DGCR8", "RAD51", "SURF1", "PNP",
                            "ADA", "NME3", "GTF3C5", "NT5C"))
is_module(multi_module$modules, is_list = TRUE)
```

---

is_network	<i>Determine if an object is a network</i>
------------	--

---

**Description**

Check content of a given object to determine if it's a network, meaning a squared matrix of similarity score between genes.

**Usage**

```
is_network(network)
```



**Arguments**

network            matrix or data.frame, object to test to be a network

**Value**

list, a boolean as first element and in second element NULL or the reason why boolean is set to FALSE

**Examples**

```
net <- matrix(runif(40*40), 40)
colnames(net) <- paste0("gene_", seq_len(ncol(net)))
rownames(net) <- paste0("gene_", seq_len(nrow(net)))
is_network(net)
```

---

join_gost	<i>Join gprofiler2::gost results</i>
-----------	--------------------------------------

---

**Description**

Takes list of gprofiler2::gost results and join them. Usefull to join results of gprofiler2::gost with custom gmt to other gprofiler2::gost results.

**Usage**

```
join_gost(gost_result)
```

**Arguments**

gost\_result        list of gprofiler2::gost result

**Details**

First element of the list is taken as reference for checks on gost\_result elements compatibility. If warnings returned, value from reference will be used. Also, timestamp is set to timestamp of the join

**Value**

A gprofiler2::gost result

**Examples**

```
query <- c("ENSG00000184349", "ENSG00000158955", "ENSG00000091140",
           "ENSG00000163114", "ENSG00000163132", "ENSG00000019186")
g1 <- gprofiler2::gost(query, sources = "GO")
g2 <- gprofiler2::gost(query, sources = "REAC")
gj <- join_gost(list(g1,g2))
```

---

kuehne_expr	<i>Transcriptomic data from the Kuehne et al. publication</i>
-------------	---

---

**Description**

A dataset containing the expression levels collapsed to the gene level. Obtained from script provided in additional data n°10 runned on GSE85358 and reduced from probe to gene by WGCNA::collapseRows with median as fuction.

**Usage**

kuehne\_expr

**Format**

A data frame with 48 rows (samples) and 15801 columns (genes).

**Source**

<https://bmcgenomics.biomedcentral.com/articles/10.1186/s12864-017-3547-3>

---

kuehne_traits	<i>Traits data linked to samples in transcriptomic data from the Kuehne et al. publication</i>
---------------	--

---

**Description**

A dataset containing the phenotype of the donors and technical information about the experiment

**Usage**

kuehne\_traits

**Format**

A data frame with 48 rows (samples) and 5 columns :

**Slide** Reference number of the microarray's slide.

**Array** Array number, 8 by slide usually

**Exp** Experiment number

**Condition** Either old (between 55 and 66 years old) or young (between 20 to 25 years old)

**Age** Real age of the donor

**Source**

<https://bmcgenomics.biomedcentral.com/articles/10.1186/s12864-017-3547-3>

---

matchsub

*Functions from NetRep packages*

---

## Description

Sources: <https://github.com/sritchie73/NetRep> CRAN page: <https://cran.r-project.org/web/packages/NetRep>  
Package licence: GPL-2

## Usage

x %nin% table

x %sub\_in% table

x %sub\_nin% table

## Arguments

x                    vector or NULL: the values to be matched. [Long vectors](#) are supported.

table                vector or NULL: the values to be matched against. [Long vectors](#) are not supported.

## Details

Reason of the copy: the contingency function weren't exported from NetRep package and neither CRAN or Bioconductor allows to use un-exported function through the ':::' operator.

Note: functions description have been modified from the original work Value Matching and Subsetting

This set of functions provides shortcuts for value matching and subsetting, on top of the functionality provided by [%in%](#).

[%nin%](#) returns a logical vector indicating if elements of x are not in table, This is the opposite of [%in%](#).

[%sub\\_in%](#) returns the elements x that are [%in%](#) table rather than a logical vector.

[%sub\\_nin%](#) returns the elements x that are [%nin%](#) table rather than a logical vector.

## Value

A vector of the same length as x.

---

orderAsNumeric	<i>Order the module vector numerically</i>
----------------	--

---

**Description**

The module assignments may be numeric, but coded as characters.

**Usage**

```
orderAsNumeric(vec)
```

**Arguments**

vec	module vector to order
-----	------------------------

**Value**

the order of the vector

---

plot_comparison_stats	<i>Heatmap of comparison statistics</i>
-----------------------	---

---

**Description**

Plot heatmap of p values for the module comparison statistics evaluated through the permutation test.

**Usage**

```
plot_comparison_stats(
  comparison_pvalues,
  pvalue_th = 0.05,
  low_color = "#031643",
  pvalue_th_color = "#A0A3D3",
  insignificant_color = "#FFFFFF",
  text_angle = 90
)
```

**Arguments**

comparison_pvalues	matrix or data.frame, table containing the p values for the statistics on each module
pvalue_th	decimal, threshold of pvalue below which statistics are considered as significant
low_color, pvalue_th_color, insignificant_color	string, color to use as lower, middle, and higher end of the legend. Can either be the color name or hexadecimal code (e.g.: "red" or "#FF1234" )
text_angle	integer, angle in [0,360] of the x axis labels.

**Value**

A ggplot object representing a heatmap of the comparison statistics for each module

**Examples**

```
df <- data.frame(avg.weight = abs(rnorm(4, 0.1, 0.1)),
                 coherence = abs(rnorm(4, 0.1, 0.1)),
                 cor.cor = abs(rnorm(4, 0.1, 0.1)),
                 cor.degree = abs(rnorm(4, 0.1, 0.1)),
                 cor.contrib = abs(rnorm(4, 0.1, 0.1)),
                 avg.cor = abs(rnorm(4, 0.1, 0.1)),
                 avg.contrib = abs(rnorm(4, 0.1, 0.1)))
plot_comparison_stats(df)
```

---

plot_enrichment	<i>Plot module from bio_enrich</i>
-----------------	------------------------------------

---

**Description**

Wrapper of the `gprofiler2::gostplot` function. Adding support of colorblind palet and selection of subsets if initial multiple query, and/or sources to plot.

**Usage**

```
plot_enrichment(
  enrich_output,
  modules = "all",
  sources = "all",
  colorblind = TRUE,
  custom_palette = NULL,
  ...
)
```

**Arguments**

<code>enrich_output</code>	list, <code>bio_enrich</code> result which are in fact <code>gprofiler2::gost</code> output.
<code>modules</code>	string or vector of characters designing the modules to plot. "all" by default to plot every module.
<code>sources</code>	string or vector of characters designing the sources to plot. "all" by default to plot every source.
<code>colorblind</code>	boolean, indicates if a colorblind friendly palette should be used.
<code>custom_palette</code>	vector of character, colors to be used for plotting.
<code>...</code>	any other parameter you can provide to <code>gprofiler2::gostplot</code> .

**Details**

Note: The colorblind friendly palette is limited to maximum 8 colors, therefore 8 sources of enrichment.

**Value**

A plotly object representing enrichment for specified modules

**Examples**

```
custom_path <- system.file("extdata", "h.all.v6.2.symbols.gmt",
                           package = "GWENA", mustWork = TRUE)
multi_module <- list(mod1 = c("BIRC3", "PMAIP1", "CASP8", "JUN", "BCL2L11",
                              "MCL1", "IL1B", "SPTAN1", "DIABLO", "BAX",
                              "BIK", "IL1A", "BID", "CDKN1A", "GADD45A"),
                    mod2 = c("TAF1C", "TARBP2", "POLH", "CETN2", "POLD1",
                              "CANT1", "PDE4B", "DGCR8", "RAD51", "SURF1",
                              "PNP", "ADA", "NME3", "GTF3C5", "NT5C"))
multi_module_enriched <- bio_enrich(multi_module, custom_path)
plot_enrichment(multi_module_enriched)
```

---

plot\_expression\_profiles

*Modules expression profiles*

---

**Description**

Plot expression profiles for all modules with eigengene highlighted

**Usage**

```
plot_expression_profiles(
  data_expr,
  modules,
  eigengenes = NULL,
  alpha_expr = 0.3,
  ...
)
```

**Arguments**

data_expr	matrix or data.frame or SummarizedExperiment, expression data with genes as column and samples as row.
modules	vector, id (whole number or string) of modules associated to each gene.
eigengenes	matrix or data.frame, eigengenes of the provided modules. If null, new ones will be computed with a PCA.

alpha\_expr      numeric, transparency of the expression lines. Must be a value between 0 (transparent) and 1 (opaque)

...              additional parameters to pass to ggplot2::theme

### Details

The sign of the eigengenes from `detect_modules` may differ from the ones computed by the pca if no eigengenes is provided to `plot_expression_profiles` and therefore the plot itself. This is due to the sign indeterminacy property from the singular value decomposition.

### Value

A ggplot representing expression profile and eigengene by module

### Examples

```
df <- kuehne_expr[1:24, 1:350]
net <- build_net(df, n_threads = 1)
detection <- detect_modules(df, net$network, detailed_result = TRUE)
plot_expression_profiles(df, detection$modules, detection$modules_eigengenes)
```

---

plot\_module

*Plot co-expression network*

---

### Description

Display a graph representing the co-expression network and different informations like hubs, enrichments

### Usage

```
plot_module(
  graph_module,
  hubs = NULL,
  groups = NULL,
  lower_weight_th = NULL,
  upper_weight_th = NULL,
  title = "Module",
  degree_node_scaling = TRUE,
  node_scaling_min = 1,
  edge_scaling_min = 0.2,
  node_scaling_max = 6,
  edge_scaling_max = 1,
  nb_row_legend = 6,
  layout = "auto",
  zoom = 1,
  vertex.label.cex = 0.7,
```

```

vertex.label.color = "gray20",
vertex.label.family = "Helvetica",
edge.color = "gray70",
vertex.frame.color = "white",
vertex.color = "gray60",
vertex.label.dist = 1,
legend_cex = 0.8,
groups_palette = NULL,
window_x_min = -1,
window_x_max = 1,
window_y_min = -1,
window_y_max = 1,
legend = TRUE,
...
)

```

### Arguments

**graph\_module** igraph object, module to plot.

**hubs** character vector or numeric vector with names, optionnal, vector of gene names or vector of numeric values named with gene names.

**groups** matrix or data.frame, a two cols table with the gene id in the first one, and the group assignation in the second one.

**lower\_weight\_th, upper\_weight\_th**  
decimal, weight threshold above lower\_weight\_th or below upper\_weight\_th which edges will be removed.

**title** string, main title that will be displayed on the plot.

**degree\_node\_scaling**  
boolean, indicates if node size should represent the degree of this node.

**node\_scaling\_min, node\_scaling\_max**  
integer, if degree\_node\_scaling is TRUE, it is the min/max size of the node, else it is the exact size of all node.

**edge\_scaling\_min, edge\_scaling\_max**  
integer, min/max width of the edge

**nb\_row\_legend** integer, number of levels in the legend.

**layout** numeric matrix or function or string, numeric matrix for nodes coordinates, or function for layout, or name of a layout function available in igraph. Default "auto" will choose the best layout depending on the graph. For more information, see [igraph.plotting](#).

**zoom** integer, scaling factor by which it's possible to have compact graph (< 1) or larger graph (> 1) display.

**vertex.label.cex, legend\_cex**  
float, font size for vertex labels. It is interpreted as a multiplication factor of some device-dependent base font size. If 0, no labels displayed.



vertex.label.color, edge.color, vertex.frame.color, vertex.color	character and/or integer vector, color of the labels. It may either contain integer values, named colors or RGB specified colors with three or four bytes. All strings starting with '#' are assumed to be RGB color specifications. It is possible to mix named color and RGB colors.
vertex.label.family	character, font family to be used for vertex labels.
vertex.label.dist	integer, distance of the label from the center of the vertex. If it is 0 then the label is centered on the vertex. If it is 1 then the label is displayed beside the vertex.
groups_palette	character and/or integer vector, vertices group palette of colors for the groups specified. It may either contain integer values, named colors or RGB specified colors with three or four bytes. All strings starting with '#' are assumed to be RGB color specifications. It is possible to mix named color and RGB colors.
window_x_min	decimal, value for the bottom limit of the window.
window_x_max	decimal, value for the top limit of the window.
window_y_min	decimal, value for the left limit of the window.
window_y_max	decimal, value for the right limit of the window.
legend	boolean, indicates if the legend should be plotted.
...	any other parameter compatible with the <code>plot.igraph</code> function.

### Details

Take care if you intend to compare modules' graphs, the same size of node will not correspond to the same values because of the scaling.

### Value

matrix, layout of the graph as a two column matrix (x, y)

### Examples

```
mat <- matrix(runif(40*40), 40)
g <- build_graph_from_sq_mat(mat)
plot_module(g, lower_weight_th = -0.5, upper_weight_th = 0.5)
```

---

plot\_modules\_merge      *Modules merge plot*

---

### Description

Plot a bipartite graph to see in which modules all modules have been merged

**Usage**

```

plot_modules_merge(
  modules_premerge,
  modules_merged,
  zoom = 1,
  vertex_size = 6,
  vertex_label_color = "gray20",
  vertex_label_family = "Helvetica",
  vertex_label_cex = 0.8,
  vertex_color = "lightskyblue",
  vertex_frame_color = "white",
  window_x_min = -1,
  window_x_max = 1,
  window_y_min = -1,
  window_y_max = 1,
  ...
)

```

**Arguments**

modules_premerge	vector, id (whole number or string) of module before merge associated to each gene.
modules_merged	vector, id (whole number or string) of module after merge associated to each gene.
zoom	decimal, value to which the display will be increased/decreased.
vertex_size	integer, size of the vertices.
vertex_label_color, vertex_color, vertex_frame_color	string, name of the color or hexadecimal code.
vertex_label_family	string, font family name.
vertex_label_cex	decimal, value for font size.
window_x_min	decimal, value for the bottom limit of the window.
window_x_max	decimal, value for the top limit of the window.
window_y_min	decimal, value for the left limit of the window.
window_y_max	decimal, value for the right limit of the window.
...	additional arguments to be passed to <code>igraph::plot.igraph()</code> .

**Details**

Both vectors must be in the same gene order before passing them to the function. No check is applied on this.

**Value**

The layout of the plot



```

                                stringsAsFactors = FALSE)
association <- associate_phenotype(eigengene_mat, phenotype_mat)
plot_modules_phenotype(association)

```

---

`quiet`*Muting a function*

---

**Description**

Prevent a function to output multiple message. Source: <https://r.789695.n4.nabble.com/Suppressing-output-e-g-from-cat-td859876.html>

**Usage**

```
quiet(func)
```

**Arguments**

`func`            Function who need to be muted.

**Value**

Nothing, just mute the called function

---

`z_summary`*Calculating Z summary*

---

**Description**

Use the topological metrics and permutations from output of `modulePreservation` to compute a Z summary (a composite preservation statistic) as defined by <https://doi.org/10.1371/journal.pcbi.1001057>

**Usage**

```
z_summary(observed_stat, permutations_array)
```

**Arguments**

`observed_stat`    matrix, bidimensional matrix containing the topological matrix computed for each module by `modulePreservation` (the element observed). Modules are in row, metrics are in column.

`permutations_array`    matrix, tridimensional matrix containing the topological matrix computed for each module by `modulePreservation` (the element observed). Modules are in dim 1, metrics are in dim 2, permutations are in dim 3.

## Details

The original Zsummary composite preservation statistic was defined by Langfelder et al. (2011). However this method use the metric from [modulePreservation](#) since they it handle better large and multiple testing correction.

## Value

A named vector of the z summary statistic with the module id as name.

## Examples

```

expr_by_cond <- list(cond1 = kuehne_expr[1:24, 1:350],
                    cond2 = kuehne_expr[25:48, 1:350])
net_by_cond <- lapply(expr_by_cond, build_net, cor_func = "spearman",
                    n_threads = 1, keep_matrices = "both")

mods_labels <- setNames(
  sample(1:6, 350, replace = TRUE,
        prob = c(0.05, 0.4, 0.25, 0.15, 0.1, 0.05)),
  colnames(expr_by_cond$cond1))

netrep_res <- NetRep::modulePreservation(
  network = lapply(net_by_cond, `[`, "adja_mat"),
  data = lapply(expr_by_cond, as.matrix),
  correlation = lapply(net_by_cond, `[`, "cor_mat"),
  moduleAssignments = mods_labels, nPerm = 100)

z_summary(netrep_res$observed, netrep_res$nulls)

mod_by_cond <- mapply(detect_modules, expr_by_cond,
  lapply(net_by_cond, `[`, "network"),
  MoreArgs = list(detailed_result = TRUE),
  SIMPLIFY = FALSE)

comparison <- compare_conditions(expr_by_cond,
  lapply(net_by_cond, `[`, "adja_mat"),
  lapply(net_by_cond, `[`, "cor_mat"),
  lapply(mod_by_cond, `[`, "modules"),
  n_perm = 100)

z_summary(comparison$result$cond1$cond2$observed,
  comparison$result$cond1$cond2$nulls)

```

# Index

- \* **datasets**
  - gtex\_expr, 21
  - gtex\_traits, 22
  - kuehne\_expr, 26
  - kuehne\_traits, 26
- \* **internal**
  - matchsub, 27
  - orderAsNumeric, 28
  - .check\_data\_expr, 3
  - .check\_gost, 3
  - .check\_module, 4
  - .check\_network, 4
  - .contingencyTable, 5
  - .cor\_func\_match, 5
  - %nin% (matchsub), 27
  - %sub\_in% (matchsub), 27
  - %sub\_nin% (matchsub), 27
  - %in%, 27
- adjacency.fromSimilarity, 9
- associate\_phenotype, 6
- bio\_enrich, 7
- build\_graph\_from\_sq\_mat, 8
- build\_net, 8, 10, 11
- compare\_conditions, 9, 10
- cor, 6
- cutreeDynamic, 12
- detect\_modules, 12, 31
- filter\_low\_var, 13
- filter\_RNA\_seq, 14
- get\_fit.cor, 15
- get\_fit.expr, 16
- get\_hub\_degree, 17
- get\_hub\_genes, 18
- get\_hub\_high\_co, 19
- get\_hub\_kleinberg, 19
- get\_sub\_clusters, 20
- gg\_palette, 21
- gtex\_expr, 21
- gtex\_traits, 22
- hub\_score, 20
- igraph.plotting, 32
- is\_data\_expr, 22
- is\_gost, 23
- is\_module, 24
- is\_network, 24
- join\_gost, 25
- kuehne\_expr, 26
- kuehne\_traits, 26
- Long vectors, 27
- matchsub, 27
- mergeCloseModules, 12, 13
- modulePreservation, 11, 36, 37
- orderAsNumeric, 28
- pam, 20
- pickSoftThreshold.fromSimilarity, 15, 16
- plot.igraph, 33
- plot\_comparison\_stats, 28
- plot\_enrichment, 29
- plot\_expression\_profiles, 30, 31
- plot\_module, 31
- plot\_modules\_merge, 33
- plot\_modules\_phenotype, 35
- quiet, 36
- TOMsimilarityFromExpr, 9
- z\_summary, 36