Package 'netZooR'

January 10, 2025

Type Package

Title Unified methods for the inference and analysis of gene regulatory networks

Version 1.10.0

Date 2023-03-27

Maintainer Marouen Ben Guebila <marouen.b.guebila@gmail.com>

Description netZooR unifies the implementations of several Network Zoo methods (netzoo, netzoo.github.io) into a single package by creating interfaces between network inference and network analysis methods. Currently, the package has 3 methods for network inference including PANDA and its optimized implementation OTTER (network reconstruction using mutliple lines of biological evidence), LIONESS (single-sample network inference), and EGRET (genotype-specific networks). Network analysis methods include CONDOR (community detection), ALPACA (differential community detection), CRANE (significance estimation of differential modules), MONSTER (estimation of network transition states). In addition, YARN allows to process gene expresssion data for tissue-specific analyses and SAMBAR infers missing mutation data based on pathway information.

Depends R (>= 4.2.0), igraph, reticulate, pandaR, yarn, matrixcalc

biocViews NetworkInference, Network, GeneRegulation, GeneExpression, Transcription, Microarray, GraphAndNetwork

Imports RCy3, viridisLite, STRINGdb, Biobase, GOstats, AnnotationDbi, matrixStats, GO.db, org.Hs.eg.db, Matrix, gplots, nnet, data.table, vegan, stats, utils, reshape, reshape2, penalized, parallel, doParallel, foreach, ggplot2, ggdendro, grid, MASS, assertthat, tidyr, methods, dplyr, graphics

License GPL-3

Encoding UTF-8

LazyData false

Suggests testthat (>= 2.1.0), knitr, rmarkdown, pkgdown

VignetteEngine knitr

VignetteBuilder knitr

RoxygenNote 7.2.3

BugReports https://github.com/netZoo/netZooR/issues

URL https://github.com/netZoo/netZooR, https://netzoo.github.io/

git_url https://git.bioconductor.org/packages/netZooR

git_branch RELEASE_3_20

git_last_commit 4cb3b30

git_last_commit_date 2024-10-29

Repository Bioconductor 3.20

Date/Publication 2025-01-09

Author Marouen Ben Guebila [aut, cre]
(<https://orcid.org/0000-0001-5934-966X>),
Tian Wang [aut] (<https://orcid.org/0000-0002-2767-3243>),
John Platig [aut],
Marieke Kuijjer [aut] (<https://orcid.org/0000-0001-6280-3130>),
Megha Padi [aut] (<https://orcid.org/0000-0002-3446-4562>),
Rebekka Burkholz [aut],
Des Weighill [aut] (<https://orcid.org/0000-0003-4979-5871>),
Kate Shutta [aut] (<https://orcid.org/0000-0003-0402-3771>)

Contents

adjMatToElist *converts adjacency matrix to edge list*

Description

converts adjacency matrix to edge list

Usage

```
adjMatToElist(adj_mat)
```

Arguments

adj_mat	adjacency matrix

Value

edge list

alpaca *Main ALPACA function*

Description

This function compares two networks and finds the sets of nodes that best characterize the change in modular structure

Usage

```
alpaca(net.table, file.stem, verbose = FALSE)
```

Arguments

net.table	A table of edges, with the first column representing the TFs ("from" nodes) and the second column representing the targets ("to" nodes). The third column contains the edge weights corresponding to the control or healthy network, and the fourth column contains the edge weights for the disease network or network of interest.
file.stem	The folder location and title under which all results will be stored.
verbose	Indicates whether the full differential modularity matrix should also be written to a file. Defaults to FALSE. modularity

Value

List where first element is the membership vector and second element is the contribution score of each node to its module's total differential modularity

Examples

```
example_path <- system.file("extdata", "Example_2comm.txt",
package = "netZooR", mustWork = TRUE)
simp.mat <- read.table(example_path,header=TRUE)
simp.alp <- alpaca(simp.mat,NULL,verbose=FALSE)
```

---

alpacaCommunityStructureRotation

Comparing node community membership between two networks

Description

This function uses the pseudo-inverse to find the optimal linear transformation mapping the community structures of two networks, then ranks nodes in the network by how much they deviate from the linear mapping.

Usage

```
alpacaCommunityStructureRotation(net1.memb, net2.memb)
```

Arguments

net1.memb	The community membership for Network 1.
net2.memb	The community membership for Network 2.

Value

A ranked list of nodes.

Examples

```
a <- 1 #place holder
```

---

alpacaComputeDifferentialScoreFromDWBM

Compute Differential modularity score from differential modularity matrix

---

Description

This functions takes the precomputed differential modularity matrix and the genLouvain membership to compute the differential modularity score.

Usage

```
alpacaComputeDifferentialScoreFromDWBM(dwbm, louv.memb)
```

Arguments

dwbm	differential modularity matrix
louv.memb	louvain community membership

Value

Vector of differntial modularity score

`alpacaComputeDWBMmatmScale` *Differential modularity matrix*

Description

This function computes the differential modularity matrix for weighted bipartite networks. The community structure of the healthy network is rescaled by the ratio of m (the total edge weight) of each network.

Usage

```
alpacaComputeDWBMmatmScale(edge.mat, ctrl.memb)
```

Arguments

`edge.mat`	A table of edges, with the first column representing the TFs ("from" nodes) and the second column representing the targets ("to" nodes). The third column contains the edge weights corresponding to the control or healthy network, and the fourth column contains the edge weights for the disease network or network of interest.
`ctrl.memb`	The community membership for the control (healthy) network.

Value

The differential modularity matrix, with rows representing "from" nodes and columns representing "to" nodes.

Examples

```
a <- 1 # place holder
```

`alpacaComputeWBMmat` *Compute modularity matrix for weighted bipartite network*

Description

This function computes the modularity matrix for a weighted bipartite network.

Usage

```
alpacaComputeWBMmat(edge.mat)
```

Arguments

`edge.mat`	A table of edges, with the first column representing the TFs ("from" nodes) and the second column representing the targets ("to" nodes). The third column contains the edge weights corresponding to the network of interest.

Value

Modularity matrix with rows representing TFs ("from" nodes) and columns repesenting targets ("to" nodes)

Examples

```
a <- 1 # example place holder
```

`alpacaCrane` *Find the robust nodes in ALPACA community using CRANE*

Description

Find the robust nodes in ALPACA community using CRANE

Usage

```
alpacaCrane(input, alp, alpha = 0.1, beta = 0, iteration = 30, isParallel = F)
```

Arguments

`input`	same input for alpaca: first column TF, second column Genes, third column edge weights from baseline condition, fourth column edge weights from disease condition.
`alp`	alpca object in list format (output from alpaca package)
`alpha`	alpha paramter perturbs each edge weights
`beta`	beta parameter perturbs the strength of each node. Set this to 0 if you want nodes to have node strength identical to the orignal network.
`iteration`	Number of CRANE distributions to create. Higher value leads to better ranking but longer runtime.
`isParallel`	TRUE = use Multithread / FALSE = do not use Multithread

Value

list of data frames

Examples

```
## Not run:

input=cbind(nonAng,ang[,3])
alp=alpaca(input,NULL,verbose = F)
alpListObject=alpacaCrane(input, alp, isParallel = T)


## End(Not run)
```

alpacaDeltaZAnalysis *Edge subtraction method (CONDOR optimizaton)*

Description

Takes two networks, subtracts edges individually, and then clusters the subtracted network using CONDOR.

Usage

```
alpacaDeltaZAnalysis(net.table, file.stem)
```

Arguments

net.table	A table of edges, with the first column representing the TFs ("from" nodes) and the second column representing the targets ("to" nodes). The third column contains the edge weights corresponding to the control or healthy network, and the fourth column contains the edge weights for the disease network or network of interest.
file.stem	The folder location and title under which all results will be stored.

Value

List where first element is the membership vector and second element is the contribution score of each node to its community's modularity in the final edge-subtracted network

Examples

```
a <- 1 # example place holder
```

alpacaDeltaZAnalysisLouvain *Edge subtraction method (Louvain optimizaton)*

Description

Takes two networks, subtracts edges individually, and then clusters the subtracted network using Louvain method.

Usage

```
alpacaDeltaZAnalysisLouvain(net.table, file.stem)
```

Arguments

net.table	A table of edges, with the first column representing the TFs ("from" nodes) and the second column representing the targets ("to" nodes). The third column contains the edge weights corresponding to the control or healthy network, and the fourth column contains the edge weights for the disease network or network of interest.
file.stem	The folder location and title under which all results will be stored.

Value

List where first element is the membership vector and second element is the contribution score of each node to its community's modularity in the final edge-subtracted network

Examples

```
a <- 1 # example place holder
```

`alpacaExtractTopGenes` *Extract core target genes in differential modules*

Description

This function outputs the top target genes in each module, ranked by their contribution to the differential modularity of the particular module in which they belong.

Usage

```
alpacaExtractTopGenes(module.result, set.lengths)
```

Arguments

`module.result`	A table of edges, with the first column representing the TFs ("from" nodes) and the second column representing the targets ("to" nodes). The third column contains the edge weights corresponding to the control or healthy network, and the fourth column contains the edge weights for the disease network or network of interest.
`set.lengths`	The desired lengths of the top gene lists.

Value

List with two elements. First element is a list of the top target genes in each cluster. Second element is a vector with the names of the gene sets. The names are in the format "number_length", where number is the module number label and length is the length of the gene set.

Examples

```
example_path <- system.file("extdata", "Example_2comm.txt",
package = "netZooR", mustWork = TRUE)
simp.mat <- read.table(example_path,header=TRUE)
simp.alp <- alpaca(simp.mat,NULL,verbose=FALSE)
alpacaExtractTopGenes(simp.alp, set.lengths=c(2,2))
```

alpacaGenLouvain *Generalized Louvain optimization*

Description

This function implements the Louvain optimization scheme on a general symmetric matrix. First, nodes are all placed in separate communities, and merged iteratively according to which merge moves result in the greatest increase in the modularity sum. Note that nodes are iterated in the order of the input matrix (not randomly) so that all results are reproducible. Second, the final community membership is used to form a alpacaMetaNetwork whose nodes represent communities from the prevous step, and which are connected by effective edge weights. The merging process is then repeated on the alpacaMetaNetwork. These two steps are repeated until the modularity sum does not increase more than a very small tolerance factor. New

Usage

```
alpacaGenLouvain(B)
```

Arguments

B	Symmetric modularity matrix

Value

The community membership vector

Examples

```
a <- 1 # example place holder
```

alpacaGetMember *get the member vector from alpaca object*

Description

get the member vector from alpaca object

Usage

```
alpacaGetMember(alp, target = "all")
```

Arguments

alp	alpaca object
target	tf, gene, or all

Value

member vector

alpacaGOtabtogenes *The top GO term associated genes in each module*

Description

Get all the genes in the top-scoring lists which are annotated with the enriched GO terms. Only GO terms with at least 3 genes in the overlap are included.

Usage

```
alpacaGOtabtogenes(go.result, dm.top)
```

Arguments

go.result	The result of the GO term analysis (alpacaListToGo)
dm.top	The result of extracting the top genes of the differential modules (dm.top)

Value

A vector with strings representing gene lists, each element of the vector has the genes in that GO term and community pasted together with spaces in between.

Examples

```
a <- 1 # example place holder
```

alpacaGoToGenes *Map GO terms to gene symbols*

Description

This function extracts all the gene symbols associated with a GO term and its descendants. (v1)

Usage

```
alpacaGoToGenes(go.term)
```

Arguments

go.term	The GO Biological Process ID (string).

Value

A vector of all gene symbols associated with the GO term.

Examples

```
a <- 1 # example place holder
```

alpacaListToGo *GO term enrichment for a list of gene sets*

Description

GO term enrichment is run using the GOstats package, and corrected for multiple testing using the Benjamini-Hochberg method.

Usage

```
alpacaListToGo(gene.list, univ.vec, comm.nums)
```

Arguments

gene.list	A list consisting of vectors of genes; genes must be identified by their official gene symbols.
univ.vec	A vector of all gene symbols that were present in the original network. This set is used as the universe for running the hypergeometric test in GOstats.
comm.nums	A vector of names for the gene sets in the input parameter "gene.list". These are used to create the table of final results.

Value

A table whose rows represent enriched GO terms (p_adj<0.05) and columns describe useful properties, like the name of the GO term, the label of the gene set which is enriched in that GO term, the adjusted p-value and Odds Ratio.

Examples

```
a <- 1 # example place holder
```

alpacaMetaNetwork *Create alpacaMetaNetwork for Louvain optimization*

Description

Computes the "effective" adjacency matrix of a alpacaMetaNetwork whose nodes represent communities in the larger input matrix.

Usage

```
alpacaMetaNetwork(J, S)
```

Arguments

J	The modularity matrix
S	The community membership vector from the previous round of agglomeration.

Value

The differential modularity matrix, with rows representing "from" nodes and columns representing "to" nodes.

Examples

```
a <- 1 # example place holder
```

---

alpacaNodeToGene *Remove tags from gene names*

Description

In gene regulatory networks, transcription factors can act as both "from" nodes (regulators) and "to" nodes (target genes), so the network analysis functions automatically tag the two columns to differentiate them. This function removes those tags from the gene identifiers.

Usage

```
alpacaNodeToGene(x)
```

Arguments

x	Tagged node identifier

Value

Untagged node name

Examples

```
a <- 1 # example place holder
```

---

alpacaObjectToDfList *Converts alpaca output into list of data frames*

Description

Converts alpaca output into list of data frames

Usage

```
alpacaObjectToDfList(alp)
```

Arguments

alp	alpaca object

Value

list of data frames

alpacaRotationAnalysis

Community comparison method (CONDOR optimizaton)

Description

Takes two networks, finds community structure of each one individually using CONDOR, and then ranks the nodes that show the biggest difference in their community membership.

Usage

```
alpacaRotationAnalysis(net.table)
```

Arguments

net.table	A table of edges, with the first column representing the TFs ("from" nodes) and the second column representing the targets ("to" nodes). The third column contains the edge weights corresponding to the control or healthy network, and the fourth column contains the edge weights for the disease network or network of interest.

Value

Vector of nodes ordered by how much they change their community membership between the two networks.

Examples

```
a <- 1 # example place holder
```

alpacaRotationAnalysisLouvain

Community comparison method (CONDOR optimizaton)

Description

Takes two networks, finds community structure of each one individually using a generalization of the Louvain method, and then ranks the nodes that show the biggest difference in their community membership.

Usage

```
alpacaRotationAnalysisLouvain(net.table)
```

Arguments

net.table	A table of edges, with the first column representing the TFs ("from" nodes) and the second column representing the targets ("to" nodes). The third column contains the edge weights corresponding to the control or healthy network, and the fourth column contains the edge weights for the disease network or network of interest.

Value

Vector of nodes ordered by how much they change their community membership between the two networks.

Examples

```
a <- 1 # example place holder
```

---

alpacaSimulateNetwork *Simulated networks*

Description

This function creates a pair of networks given user-defined parameters for the modular structure of the first (healthy) network and the type of added module in the second (disease) network.

Usage

```
alpacaSimulateNetwork(
  comm.sizes,
  edge.mat,
  num.module,
  size.module,
  dens.module
)
```

Arguments

comm.sizes	A two-column matrix indicating the number of "from" nodes (left column) and number of "to" nodes (right column) in each community (row).
edge.mat	A matrix indicating the number of edges from the TFs in community i (rows) to target genes in community j (columns).
num.module	The number of modules that will be added to simulate the disease network.
size.module	A two-column matrix indicating the number of "from" and "to" nodes in each new module (row) that will be added to simulate the disease network.
dens.module	A vector of length num.module, indicating the edge density of each added module.

Value

A list with two elements. The first element is a four-column edge table of the same form that is input into the differential modularity function. The second element is a list of all the new nodes in the modules that were added to create the disease network.

Examples

```
a <- 1 # example place holder
```

alpacaTestNodeRank *Enrichment in ranked list*

Description

This function computes the enrichment of selected nodes in a ranked list, using Wilcoxon, Kolmogorov-Smirnov, and Fisher exact tests.

Usage

```
alpacaTestNodeRank(node.ordered, true.pos)
```

Arguments

node.ordered	An ordered list of nodes (high-scoring to low-scoring).
true.pos	The selected set of nodes being tested for enrichment among the ranked list.

Value

A vector of 4 values. 1) Wilcoxon p-value, 2) KS p-value, 3) Fisher p-value, 4) Fisher odds ratio.

Examples

```
a <- 1 # example place holder
```

alpacaTidyConfig *Renumbering community membership vector*

Description

This is a helper function alpacaGenLouvain. It re-numbers the communities so that they run from 1 to N increasing through the vector.

Usage

```
alpacaTidyConfig(S)
```

Arguments

S	The community membership vector derived from the previous round of agglomeration.

Value

The renumbered membership vector.

Examples

```
a <- 1 # example place holder
```

alpacaTopEnsembltoTopSym *Translating gene identifiers to gene symbols*

Description

Takes a list of gene sets named using gene identifiers and converts them to a list of symbols given a user-defined annotation table.

Usage

```
alpacaTopEnsembltoTopSym(mod.top, annot.vec)
```

Arguments

mod.top	A list of gene sets (gene identifiers)
annot.vec	A vector of gene symbols with gene identifiers as the names of the vector, that defines the translation between annotations.

Value

A list of sets of gene symbols.

Examples

```
a <- 1 # example place holder
```

alpacaWBMlouvain *Generalized Louvain method for bipartite networks*

Description

This function implements a generalized form of the Louvain method for weighted bipartite networks.

Usage

```
alpacaWBMlouvain(net.frame)
```

Arguments

net.frame	A table of edges, with the first column representing the TFs ("from" nodes) and the second column representing the targets ("to" nodes). The third column contains the edge weights corresponding to the network of interest.

Value

List where first element is the community membership vector and second element is the contribution score of each node to its community's portion of the bipartite modularity.

Examples

```
a <- 1 # example place holder
```

`condorCluster` *Main clustering function for condor.*

Description

This function performs community structure clustering using the bipartite modularity described in `condorModularityMax`. This function uses a standard (non-bipartite) community structure clustering of the uni-partite, weighted projection of the original bipartite graph as an initial guess for the bipartite modularity.

Usage

```
condorCluster(
  condor.object,
  cs.method = "LCS",
  project = TRUE,
  low.memory = FALSE,
  deltaQmin = "default"
)
```

Arguments

`condor.object`	Output of make.condor.object. This function uses `condor.object$edges`
`cs.method`	is a string to specify which unipartite community structure algorithm should be used for the seed clustering. Options are `LCS` (`multilevel.community`), `LEC` (`leading.eigenvector.community`), `FG` (`fastgreedy.community`).
`project`	Provides options for initial seeding of the bipartite modularity maximization. If TRUE, the nodes in the first column of `condor.object$edges` are projected and clustered using `cs.method`. If FALSE, the complete bipartite network is clustered using the unipartite clustering methods listed in `cs.method`.
`low.memory`	If TRUE, uses `condorModularityMax` instead of `condorMatrixModularity`. This is a slower implementation of the modularity maximization, which does not store any matrices in memory. Useful on a machine with low RAM. However, runtimes are (much) longer.
`deltaQmin`	convergence parameter determining the minimum required increase in the modularity for each iteration. Default is min(10^-4,1/(number of edges)), with number of edges determined by `nrow(condor.object$edges)`. User can set this parameter by passing a numeric value to deltaQmin.

Value

`condor.object` with `condorModularityMax` output included.

Examples

```
r = c(1,1,1,2,2,2,3,3,3,4,4);
b = c(1,2,3,1,2,4,2,3,4,3,4);
reds <- c("Alice","Sue","Janine","Mary")
blues <- c("Bob","John","Ed","Hank")
elist <- data.frame(red=reds[r],blue=blues[b])
condor.object <- createCondorObject(elist)
condor.object <- condorCluster(condor.object)
```

`condorCoreEnrich` *Compare qscore distribution of a subset of nodes to all other nodes.*

Description

Compute one-sided KS and wilcox tests to determine if a subset of nodes has a stochastically larger qscore distribution.

Usage

```
condorCoreEnrich(test_nodes, q, perm = FALSE, plot.hist = FALSE, nsamp = 1000)
```

Arguments

`test_nodes`	is a list containing the subset of nodes (of one node class –blue or red–only) to be tested
`q`	is a two column data frame containing the node names in the first column and the q-scores in the second column.
`perm`	if TRUE, run permutation tests. Else, run `ks.test` and `wilcox.test` only.
`plot.hist`	if TRUE, produces two histograms of test statistics from permutation tests, one for KS and one for wilcoxon and a red dot for true labeling. Only works if perm=TRUE.
`nsamp`	Number of permutation tests to run

Value

if `perm=FALSE`, the analytical p-values from `ks.test` and `wilcox.test`

if `perm=TRUE`, the permutation p-values are provided in addition to the analytical values.

Note

`ks.test` and `wilcox.test` will throw warnings due to the presence of ties, so the p-values will be approximate. See those functions for further details.

Examples

```
r = c(1,1,1,2,2,2,3,3,3,4,4);
b = c(1,2,3,1,2,4,2,3,4,3,4);
reds <- c("Alice","Sue","Janine","Mary")
blues <- c("Bob","John","Ed","Hank")
elist <- data.frame(red=reds[r],blue=blues[b])
condor.object <- createCondorObject(elist)
condor.object <- condorCluster(condor.object)
condor.object <- condorQscore(condor.object)
q_in <- condor.object$qscores$red.qscore
out <- condorCoreEnrich(c("Alice","Mary"),q=q_in,perm=TRUE,plot.hist=TRUE)
```

`condorCreateObject` *creates condor object*

Description

creates condor object

Usage

```
condorCreateObject(elist)
```

Arguments

`elist`	edge list

Value

condor object

`condorMatrixModularity`
Iteratively maximize bipartite modularity.

Description

This function is based on the bipartite modularity as defined in "Modularity and community detection in bipartite networks" by Michael J. Barber, Phys. Rev. E 76, 066102 (2007) This function uses a slightly different implementation from the paper. It does not use the "adaptive BRIM" method for identifying the number of modules. Rather, it simply continues to iterate until the difference in modularity between iterations is less that 10^-4. Starting from a random initial condition, this could take some time. Use `condorCluster` for quicker runtimes and likely better clustering, it initializes the blue node memberships by projecting the blue nodes into a unipartite "blue" network and then identify communities in that network using a standard unipartite community detection algorithm run on the projected network. See `condorCluster` for more details on that. This function loads the entire adjacency matrix in memory, so if your network has more than ~50,000 nodes, you may want to use `condorModularityMax`, which is slower, but does not store the matrices in memory. Or, of course, you could move to a larger machine.

Usage

```
condorMatrixModularity(
  condor.object,
  T0 = cbind(seq_len(q), rep(1, q)),
  weights = 1,
  deltaQmin = "default"
)
```

Arguments

condor.object	is a list created by createCondorObject. condor.object$edges must contain the edges in the giant connected component of a bipartite network
T0	is a two column data.frame with the initial community assignment for each "blue" node, assuming there are more reds than blues, though this is not strictly necessary. The first column contains the node name, the second column the community assignment.
weights	edgeweights for each edge in edgelist.
deltaQmin	convergence parameter determining the minimum required increase in the modularity for each iteration. Default is min(10^-4,1/(number of edges)), with number of edges determined by nrow(condor.object$edges). User can set this parameter by passing a numeric value to deltaQmin.

Value

Qcoms data.frame with modularity of each community.

modularity modularity value after each iteration.

red.memb community membership of the red nodes

blue.memb community membership of the blue.nodes

Examples

```
r = c(1,1,1,2,2,2,3,3,3,4,4);
b = c(1,2,3,1,2,4,2,3,4,3,4);
reds <- c("Alice","Sue","Janine","Mary")
blues <- c("Bob","John","Ed","Hank")
elist <- data.frame(red=reds[r],blue=blues[b])
condor.object <- createCondorObject(elist)
#randomly assign blues to their own community
T0 <- data.frame(nodes=blues,coms=seq_len(4))
condor.object <- condorMatrixModularity(condor.object,T0=T0)
```

---

condorModularityMax *Iteratively maximize bipartite modularity.*

Description

This function is based on the bipartite modularity as defined in "Modularity and community detection in bipartite networks" by Michael J. Barber, Phys. Rev. E 76, 066102 (2007) This function uses a slightly different implementation from the paper. It does not use the "adaptive BRIM" method for identifying the number of modules. Rather, it simply continues to iterate until the difference in modularity between iterations is less that 10^-4. Starting from a random initial condition, this could take some time. Use `condorCluster` for quicker runtimes and likely better clustering, it initializes the blue node memberships by projecting the blue nodes into a unipartite "blue" network and then identify communities in that network using a standard unipartite community detection algorithm run on the projected network. See `condorCluster` for more details that.

Usage

```
condorModularityMax(
  condor.object,
  T0 = cbind(seq_len(q), rep(1, q)),
  weights = 1,
  deltaQmin = "default"
)
```

Arguments

`condor.object`	is a list created by `createCondorObject`. `condor.object$edges` must contain the edges in the giant connected component of a bipartite network
`T0`	is a two column data.frame with the initial community assignment for each "blue" node, assuming there are more reds than blues, though this is not strictly necessary. The first column contains the node name, the second column the community assignment.
`weights`	edgeweights for each edge in `edgelist`.
`deltaQmin`	convergence parameter determining the minimum required increase in the modularity for each iteration. Default is min(10^-4,1/(number of edges)), with number of edges determined by `nrow(condor.object$edges)`. User can set this parameter by passing a numeric value to deltaQmin.

Value

Qcoms data.frame with modularity of each community.

modularity modularity value after each iteration.

red.memb community membership of the red nodes

blue.memb community membership of the blue.nodes

Examples

```
r = c(1,1,1,2,2,2,3,3,3,4,4);
b = c(1,2,3,1,2,4,2,3,4,3,4);
reds <- c("Alice","Sue","Janine","Mary")
blues <- c("Bob","John","Ed","Hank")
elist <- data.frame(red=reds[r],blue=blues[b])
condor.object <- createCondorObject(elist)
#randomly assign blues to their own community
T0 <- data.frame(nodes=blues,coms=1)
condor.object <- condorModularityMax(condor.object,T0=T0)
```

condorPlotCommunities *Plot adjacency matrix with links grouped and colored by community*

Description

This function will generate the network link 'heatmap' with colored dots representing within-community links and black dots between-community links

Usage

```
condorPlotCommunities(
  condor.object,
  color_list,
  point.size = 0.01,
  xlab = "SNP",
  ylab = "Gene"
)
```

Arguments

condor.object	output of either condorCluster or condorModularityMax
color_list	vector of colors accepted by col inside the plot function. There must be as many colors as communities.
point.size	passed to cex in the plot
xlab	x axis label
ylab	y axis label

Value

produces a plot output.

Note

For the condor paper http://arxiv.org/abs/1509.02816, I used 35 colors from the "Tarnish" palette with "hard" clustering

References

http://tools.medialab.sciences-po.fr/iwanthue/ for a nice color generator at

Examples

```
r = c(1,1,1,2,2,2,3,3,3,4,4);
b = c(1,2,3,1,2,4,2,3,4,3,4);
reds <- c("Alice","Sue","Janine","Mary")
blues <- c("Bob","John","Ed","Hank")
elist <- data.frame(red=reds[r],blue=blues[b])
condor.object <- createCondorObject(elist)
condor.object <- condorCluster(condor.object)
condorPlotCommunities(condor.object,
color_list=c("darkgreen","darkorange"),point.size=2,
xlab="Women",ylab="Men")
```

condorPlotHeatmap *Plot weighted adjacency matrix with links grouped by community*

Description

This function will generate the network link 'heatmap' for a weighted network

Usage

```
condorPlotHeatmap(condor.object, main = "", xlab = "blues", ylab = "reds")
```

Arguments

condor.object	output of either condorCluster or condorModularityMax
main	plot title
xlab	x axis label
ylab	y axis label

Value

produces a plot output.

Examples

```
data(small1976)
condor.object <- createCondorObject(small1976)
condor.object <- condorCluster(condor.object, project=FALSE)
condorPlotHeatmap(condor.object)
```

condorQscore *Calculate Qscore for all nodes*

Description

Qscore is designed to calculate the fraction of the modularity contributed by each node to its community's modularity

Usage

```
condorQscore(condor.object)
```

Arguments

condor.object	output of condorCluster or condorModularityMax

Value

condor.object list has condor.object$qscores added to it. this includes two data.frames, blue.qscore and red.qscore which have the qscore for each red and blue node.

Examples

```
r = c(1,1,1,2,2,2,3,3,3,4,4);
b = c(1,2,3,1,2,4,2,3,4,3,4);
reds <- c("Alice","Sue","Janine","Mary")
blues <- c("Bob","John","Ed","Hank")
elist <- data.frame(red=reds[r],blue=blues[b])
condor.object <- createCondorObject(elist)
condor.object <- condorCluster(condor.object)
condor.object <- condorQscore(condor.object)
```

---

condorRun *Run CONDOR clustering*

Description

Run CONDOR clustering

Usage

```
condorRun(elist, qscore = F)
```

Arguments

elist	edge list
qscore	TRUE = output qscore / FALSE = do not output qscore

Value

condor object

---

craneBipartite *Pertrubs the bipartite network with fixed node strength*

Description

Pertrubs the bipartite network with fixed node strength

Usage

```
craneBipartite(df, alpha = 0.1, beta = 0, getAdj = F, randomStart = F)
```

Arguments

df	Adjacency Matrix or Edge list
alpha	alpha paramter perturbs each edge weights
beta	beta parameter perturbs the strength of each node. Set this to 0 if you want nodes to have node strength identical to the orignal network.
getAdj	TRUE = this will return adjacency matrix instead of edge list
randomStart	FALSE = initialize the first row with completely random edge weights.

Value

edge list

Examples

```
## Not run:

# Using Edge list as input
elist=craneBipartite(nonAng)
elist=craneBipartite(nonAng,alpha=0.3)

# Using Edge list as input and Adjcency Matrix as output
adjMatrix=craneBipartite(nonAng,alpha=0.1,getAdj=T)

# Using Edge list as input and Adjcency Matrix as output
A=elistToAdjMat(nonAng)
elist=craneBipartite(A)


## End(Not run)
```

`craneUnipartite` *Pertrubs the unipartite network with fixed node strength from adjacency matrix*

Description

Pertrubs the unipartite network with fixed node strength from adjacency matrix

Usage

```
craneUnipartite(A, alpha = 0.1, isSelfLoop = F)
```

Arguments

`A`	Adjacency Matrix
`alpha`	alpha paramter perturbs each edge weights
`isSelfLoop`	TRUE/FALSE if self loop exists. co-expression matrix will have a self-loop of 1. Thus TRUE

Value

adjacency matrix

createCondorObject *Create list amenable to analysis using* condor *package.*

Description

Converts an edge list into a list which is then an input for other functions in the condor package.

Usage

```
createCondorObject(edgelist, return.gcc = TRUE)
```

Arguments

edgelist	a data.frame with 'red' nodes in the first column and 'blue' nodes in the second column, representing links from the node in the first column to the node in the second column. There must be more unique 'red' nodes than 'blue' nodes. Optionally, a third column may be provided to create a weighted network.
return.gcc	if TRUE, returns the giant connected component

Value

G is an igraph graph object with a 'color' attribute based on the colnames of edgelist. This can be accessed via V(g)$color, which returns a vector indicating red/blue. Use V(g)$name with V(g)$color to identify red/blue node names

edges corresponding to graph G. If return.gcc=TRUE, includes only those edges in the giant connected component.

Qcoms output from condorCluster or condorModularityMax

modularity NULL output from condorCluster or condorModularityMax

red.memb NULL output from condorCluster or condorModularityMax

blue.memb NULL output from condorCluster or condorModularityMax

qscores NULL output from condorQscore

Examples

```
r = c(1,1,1,2,2,2,3,3,3,4,4);
b = c(1,2,3,1,2,4,2,3,4,3,4);
reds <- c("Alice","Sue","Janine","Mary")
blues <- c("Bob","John","Ed","Hank")
elist <- data.frame(red=reds[r],blue=blues[b])
condor.object <- createCondorObject(elist)
```

createPandaStyle *Create a Cytoscape visual style for PANDA network*

Description

This function is able to create a Cytoscape visual style for any PANDA network output.

Usage

```
createPandaStyle(style_name = "PandaStyle")
```

Arguments

style_name	Character string indicating the style name. Defaults to "PandaStyle"

Value

A visual style in Cytoscape Control Panel under "Style" button.

Examples

```
# Here we will load a customized visual style for our network, in which TF
# nodes are orange circles, target gene nodes are blue squares, and edges
# shade and width are the edge weight (likelyhood of regulatory interaction
# between the TF and gene). You can further customize the network style
# directly from Cytoscape.

createPandaStyle(style_name="PandaStyle")
```

degreeAdjust *Function to adjust the degree so that the hub nodes are not penalized in z-score transformation*

Description

Function to adjust the degree so that the hub nodes are not penalized in z-score transformation

Usage

```
degreeAdjust(A)
```

Arguments

A	Input adjacency matrix

`dragon` *Run DRAGON in R.*

Description

Description: Estimates a multi-omic Gaussian graphical model for two input layers of paired omic data.

Usage

```
dragon(
  layer1,
  layer2,
  pval = FALSE,
  gradient = "finite_difference",
  verbose = FALSE
)
```

Arguments

`layer1`	: first layer of omics data; rows: samples (order must match layer2), columns: variables
`layer2`	: second layer of omics data; rows: samples (order must match layer1), columns: variables.
`pval`	: calculate p-values for network edges. Not yet implemented in R; available in netZooPy.
`gradient`	: method for estimating parameters of p-value distribution, applies only if p-val == TRUE. default = "finite_difference"; other option = "exact"
`verbose`	: verbosity level (TRUE/FALSE)

Value

A list of model results. cov : the shrunken covariance matrix

- `cov` the shrunken covariance matrix
- `prec` the shrunken precision matrix
- `ggm` the shrunken Gaussian graphical model; matrix of partial correlations. Self-edges (diagonal elements) are set to zero.
- `lambdas` Vector of omics-specific tuning parameters (lambda1, lambda2) for `layer1` and `layer2`
- `gammas` Reparameterized tuning parameters; gamma = 1 - lambda^2
- `risk_grid` Risk grid, for assessing optimization. Grid boundaries are in terms of gamma.

elistAddTags *Adds "_A" to first column and "_B" to second column*

Description

Adds "_A" to first column and "_B" to second column

Usage

```
elistAddTags(elist)
```

Arguments

elist	edge list

Value

edge list

elistIsEdgeOrderEqual *check if first two columns are identical*

Description

check if first two columns are identical

Usage

```
elistIsEdgeOrderEqual(elist1, elist2)
```

Arguments

elist1	edge list
elist2	edge list

Value

boolean

elistRemoveTags *undo elistAddTags*

Description

undo elistAddTags

Usage

```
elistRemoveTags(elist)
```

Arguments

elist	edge list

Value

edge list

elistSort *Sorts the edge list based on first two columns in alphabetical order*

Description

Sorts the edge list based on first two columns in alphabetical order

Usage

```
elistSort(elist)
```

Arguments

elist	edge list

Value

edge list

elistToAdjMat *Converts edge list to adjacency matrix*

Description

Converts edge list to adjacency matrix

Usage

```
elistToAdjMat(elist, isBipartite = F)
```

Arguments

elist	edge list
isBipartite	TRUE = for bipartite / FALSE = for unipartite

Value

Adjcency Matrix

exon.size *Gene length*

Description

A vector of gene lengths. This will be used to normalize the gene mutation scores by the gene's length. This example is based on hg19 gene symbols. The gene length is based on the number of non-overlapping exons. Data were downloaded and pre-processed as described in Kuijjer et al.

This data is a toy example data for SAMBAR, it contains length of Exons.

This data is a toy example data for SAMBAR, it contains gene annotations

Usage

```
data(exon.size)

data(exon.size)

data(genes)
```

Format

A integer vector of size 23459, with gene symbols as names

A list containing Exon sizes for 23459 genes

A vector containing names of 23459 genes

Value

A list of length 1

A vector of length 23459

References

Kuijjer, Marieke Lydia, et al. "Cancer subtype identification using somatic mutation data." British journal of cancer 118.11 (2018): 1492-1501.

Kuijjer, Marieke Lydia, et al. "Cancer subtype identification using somatic mutation data." British journal of cancer 118.11 (2018): 1492-1501.

`genes` *Example of a gene list*

Description

List of cancer-associated genes to subset the mutation data to, as described in Kuijjer et al.

Usage

```
data(genes)
```

Format

A character vector of length 2352

`isElist` *Check if data frame is an edge list*

Description

Check if data frame is an edge list

Usage

```
isElist(df)
```

Arguments

`df`	some data frame

Value

Boolean

`jutterDegree` *CRANE Beta perturbation function. This function will add noice to the node strength sequence.*

Description

CRANE Beta perturbation function. This function will add noice to the node strength sequence.

Usage

```
jutterDegree(nodeD, beta, beta_slope = T)
```

Arguments

`nodeD`	Vector of node strength
`beta`	beta
`beta_slope`	TRUE=use predetermined slope to add noise / FALSE = use constant value for noise

Value

vector with new strength distribution

`lioness` *Compute LIONESS (Linear Interpolation to Obtain Network Estimates for Single Samples)*

Description

Compute LIONESS (Linear Interpolation to Obtain Network Estimates for Single Samples)

Usage

```
lioness(
  expr,
  motif = NULL,
  ppi = NULL,
  network.inference.method = "panda",
  ncores = 1,
  ...
)
```

Arguments

`expr`	A mandatory expression dataset, as a genes (rows) by samples (columns) data.frame
`motif`	A motif dataset, a data.frame, matrix or exprSet containing 3 columns. Each row describes an motif associated with a transcription factor (column 1) a gene (column 2) and a score (column 3) for the motif.
`ppi`	A Protein-Protein interaction dataset, a data.frame containing 3 columns. Each row describes a protein-protein interaction between transcription factor 1(column 1), transcription factor 2 (column 2) and a score (column 3) for the interaction.
`network.inference.method`	String specifying choice of network inference method. Default is "panda". Options include "pearson".
`ncores`	int specifying the number of cores to be used. Default is 1. (Note: constructing panda networks can be memory-intensive, and the number of cores should take into consideration available memory.)
`...`	additional arguments for panda analysis

Value

A list of length N, containing objects of class "panda" corresponding to each of the N samples in the expression data set.
"regNet" is the regulatory network
"coregNet" is the coregulatory network
"coopNet" is the cooperative network

References

Kuijjer, M.L., Tung, M., Yuan, G., Quackenbush, J. and Glass, K., 2015. Estimating sample-specific regulatory networks. arXiv preprint arXiv:1505.06440. Kuijjer, M.L., Hsieh, PH., Quackenbush, J. et al. lionessR: single sample network inference in R. BMC Cancer 19, 1003 (2019). https://doi.org/10.1186/s12885-019-6235-7

Examples

```
data(pandaToyData)
lionessRes <- lioness(expr = pandaToyData$expression[,1:3], motif = pandaToyData$motif, ppi = pandaToyData$ppi
```

`lionessPy` *Run python implementation of LIONESS*

Description

LIONESS(Linear Interpolation to Obtain Network Estimates for Single Samples) is a method to estimate sample-specific regulatory networks. [(LIONESS publication)]).

Usage

```
lionessPy(
  expr_file,
  motif_file = NULL,
  ppi_file = NULL,
  computing = "cpu",
  precision = "double",
  save_tmp = TRUE,
  modeProcess = "union",
  remove_missing = FALSE,
  start_sample = 1,
  end_sample = "None",
  save_single_network = FALSE,
  save_dir = "lioness_output",
  save_fmt = "npy"
)
```

Arguments

expr_file	Character string indicating the file path of expression values file, with each gene(in rows) across samples(in columns).
motif_file	An optional character string indicating the file path of a prior transcription factor binding motifs dataset. When this argument is not provided, analysis will continue with Pearson correlation matrix.
ppi_file	An optional character string indicating the file path of protein-protein interaction edge dataset. Also, this can be generated with a list of proteins of interest by sourcePPI.
computing	'cpu' uses Central Processing Unit (CPU) to run PANDA; 'gpu' use the Graphical Processing Unit (GPU) to run PANDA. The default value is "cpu".
precision	'double' computes the regulatory network in double precision (15 decimal digits); 'single' computes the regulatory network in single precision (7 decimal digits) which is fastaer, requires half the memory but less accurate. The default value is 'double'.
save_tmp	'TRUE' saves middle data like expression matrix and normalized networks; 'FALSE' deletes the middle data. The default value is 'TURE'.
modeProcess	'legacy' refers to the processing mode in netZooPy<=0.5, 'union': takes the union of all TFs and genes across priors and fills the missing genes in the priors with zeros; 'intersection': intersects the input genes and TFs across priors and removes the missing TFs/genes. Default values is 'union'.
remove_missing	Only when modeProcess='legacy': remove_missing='TRUE' removes all unmatched TF and genes; remove_missing='FALSE' keeps all tf and genes. The default value is FALSE.
start_sample	Numeric indicating the start sample number, The default value is 1.
end_sample	Numeric indicating the end sample number. The default value is 'None' meaning no end sample, i.e. print out all samples.
save_single_network	Boolean vector, "TRUE" wirtes out the single network in npy/txt/mat formats, directory and format are specifics by params "save_dir" and "save_fmt". The default value is 'FALSE'

save_dir — Character string indicating the folder name of output lioness networks for each sample by defined. The default is a folder named "lioness_output" under current working directory. This paramter is valid only when save_single_network = TRUE.

save_fmt — Character string indicating the format of lioness network of each sample. The dafault is "npy". The option is txt, npy, or mat. This paramter is valid only when save_single_network = TRUE.

Value

A data frame with columns representing each sample, rows representing the regulator-target pair in PANDA network generated by pandaPy. Each cell filled with the related score, representing the estimated contribution of a sample to the aggregate network.

Examples

```
# refer to the input datasets files of control in inst/extdat as example
control_expression_file_path <- system.file("extdata", "expr10_reduced.txt", package = "netZooR",
    mustWork = TRUE)
motif_file_path <- system.file("extdata", "chip_reduced.txt", package = "netZooR", mustWork = TRUE)
   ppi_file_path <- system.file("extdata", "ppi_reduced.txt", package = "netZooR", mustWork = TRUE)

# Run LIONESS algorithm

control_lioness_result <- lionessPy(expr_file = control_expression_file_path,
    motif_file = motif_file_path, ppi_file = ppi_file_path,
    modeProcess="union",start_sample=1, end_sample=1, precision="single")
```

---

monster — *MOdeling Network State Transitions from Expression and Regulatory data (MONSTER)*

Description

This function runs the MONSTER algorithm. Biological states are characterized by distinct patterns of gene expression that reflect each phenotype's active cellular processes. Driving these phenotypes are gene regulatory networks in which transcriptions factors control when and to what degree individual genes are expressed. Phenotypic transitions, such as those that occur when disease arises from healthy tissue, are associated with changes in these networks. MONSTER is an approach to understanding these transitions. MONSTER models phenotypic-specific regulatory networks and then estimates a "transition matrix" that converts one state to another. By examining the properties of the transition matrix, we can gain insight into regulatory changes associated with phenotypic state transition. Important note: the direct regulatory network observed from gene expression is currently implemented as a regular correlation as opposed to the partial correlation described in the paper. Citation: Schlauch, Daniel, et al. "Estimating drivers of cell state transitions using gene regulatory network models." BMC systems biology 11.1 (2017): 139. https://doi.org/10.1186/s12918-017-0517-y

Usage

```
monster(
  expr,
  design,
  motif,
  nullPerms = 100,
  ni_method = "BERE",
  ni.coefficient.cutoff = NA,
  numMaxCores = 1,
  outputDir = NA,
  alphaw = 0.5,
  mode = "buildNet"
)
```

Arguments

expr	Gene Expression dataset, can be matrix or data.frame of expression values or ExpressionSet.
design	Binary vector indicating case control partition. 1 for case and 0 for control.
motif	Regulatory data.frame consisting of three columns. For each row, a transcription factor (column 1) regulates a gene (column 2) with a defined strength (column 3), usually taken to be 0 or 1
nullPerms	number of random permutations to run (default 100). Set to 0 to only calculate observed transition matrix. When mode is is 'buildNet' it randomly permutes the case and control expression samples, if mode is 'regNet' it will randomly permute the case and control networks.
ni_method	String to indicate algorithm method. Must be one of "bere","pearson","cd","lda", or "wcd". Default is "bere"
ni.coefficient.cutoff	numeric to specify a p-value cutoff at the network inference step. Default is NA, indicating inclusion of all coefficients.
numMaxCores	requires doParallel, foreach. Runs MONSTER in parallel computing environment. Set to 1 to avoid parallelization, NA will take the default parallel pool in the computer.
outputDir	character vector specifying a directory or path in which which to save MONSTER results, default is NA and results are not saved.
alphaw	A weight parameter between 0 and 1 specifying proportion of weight to give to indirect compared to direct evidence. The default is 0.5 to give an equal weight to direct and indirect evidence.
mode	A parameter telling whether to build the regulatory networks ('buildNet') or to use provided regulatory networks ('regNet'). If set to 'regNet', then the parameters motif, ni_method, ni.coefficient.cutoff, and alphaw will be set to NA.

Value

An object of class "monsterAnalysis" containing results

Examples

```
# Example with the network reconstruction step
data(yeast)
design <- c(rep(0,20),rep(NA,10),rep(1,20))
yeast$exp.cc[is.na(yeast$exp.cc)] <- mean(as.matrix(yeast$exp.cc),na.rm=TRUE)
#monsterRes <- monster(yeast$exp.cc[1:500,], design, yeast$motif, nullPerms=10, numMaxCores=1)
# Example with provided networks

pandaResult <- panda(pandaToyData$motif, pandaToyData$expression, pandaToyData$ppi)
case=getRegNet(pandaResult)
nelemReg=dim(getRegNet(pandaResult))[1]*dim(getRegNet(pandaResult))[2]
nGenes=length(colnames(getRegNet(pandaResult)))
control=matrix(rexp(nelemReg, rate=.1), ncol=nGenes)
colnames(control) = colnames(case)
rownames(control) = rownames(case)
expr = as.data.frame(cbind(control,case))
design=c(rep(0,nGenes),rep(1, nGenes))
monsterRes <- monster(expr, design, motif=NA, nullPerms=10, numMaxCores=1, mode='regNet')
```

---

monsterBereFull *Bipartite Edge Reconstruction from Expression data (composite method with direct/indirect)*

Description

This function generates a complete bipartite network from gene expression data and sequence motif data. This NI method serves as a default method for inferring bipartite networks in MONSTER. Running monsterBereFull can generate these networks independently from the larger MONSTER method.

Usage

```
monsterBereFull(
  motif.data,
  expr.data,
  alpha = 0.5,
  lambda = 10,
  score = "motifincluded"
)
```

Arguments

motif.data	A motif dataset, a data.frame, matrix or exprSet containing 3 columns. Each row describes an motif associated with a transcription factor (column 1) a gene (column 2) and a score (column 3) for the motif.
expr.data	An expression dataset, as a genes (rows) by samples (columns) data.frame
alpha	A weight parameter specifying proportion of weight to give to indirect compared to direct evidence. See documentation.
lambda	if using penalized, the lambda parameter in the penalized logistic regression
score	String to indicate whether motif information will be readded upon completion of the algorithm

Value

An matrix or data.frame

Examples

```
data(yeast)
monsterRes <- monsterBereFull(yeast$motif, yeast$exp.cc, alpha=.5)
```

`monsterCalculateTmPValues`
Calculate p-values for a tranformation matrix

Description

This function calculates the significance of an observed transition matrix given a set of null transition matrices

Usage

```
monsterCalculateTmPValues(monsterObj, method = "z-score")
```

Arguments

`monsterObj`	monsterAnalysis Object
`method`	one of 'z-score' or 'non-parametric'

Value

vector of p-values for each transcription factor

Examples

```
# data(yeast)
# design <- c(rep(0,20),rep(NA,10),rep(1,20))
# yeast$exp.cc[is.na(yeast$exp.cc)] <- mean(as.matrix(yeast$exp.cc),na.rm=TRUE)
# monsterRes <- monster(yeast$exp.cc, design, yeast$motif, nullPerms=100, numMaxCores=4)
data(monsterRes)
monsterCalculateTmPValues(monsterRes)
```

monsterCheckDataType *Checks that data is something MONSTER can handle*

Description

Checks that data is something MONSTER can handle

Usage

```
monsterCheckDataType(expr)
```

Arguments

expr	Gene Expression dataset

Value

expr Gene Expression dataset in the proper form (may be the same as input)

Examples

```
expr.matrix <- matrix(rnorm(2000),ncol=20)
monsterCheckDataType(expr.matrix)
#TRUE
data(yeast)
class(yeast$exp.cc)
monsterCheckDataType(yeast$exp.cc)
#TRUE
```

monsterdTFIPlot *This function plots the Off diagonal mass of an observed Transition Matrix compared to a set of null TMs*

Description

This function plots the Off diagonal mass of an observed Transition Matrix compared to a set of null TMs

Usage

```
monsterdTFIPlot(
  monsterObj,
  rescale = "none",
  plot.title = NA,
  highlight.tfs = NA,
  nTFs = -1
)
```

Arguments

monsterObj	monsterAnalysis Object
rescale	string indicating whether to reorder transcription factors according to their statistical significance and to rescale the values observed to be standardized by the null distribution ('significance'), to reorder transcription factors according to the largest dTFIs ('magnitude') with the TF x axis labels proportional to their significance , or finally without ordering them ('none'). When rescale is set to 'significance', the results can be different between two MONSTER runs if the number of permutations is not large enough to sample the null, that is why it is the seed should be set prior to calling MONSTER to get reproducible results. If rescale is set to another value such as 'magnitude' or 'none', it will produce deterministic results between two identical MONSTER runs.
plot.title	String specifying the plot title
highlight.tfs	vector specifying a set of transcription factors to highlight in the plot
nTFs	number of TFs to plot in x axis. -1 takes all TFs.

Value

ggplot2 object for transition matrix comparing observed distribution to that estimated under the null

Examples

```
# data(yeast)
# yeast$exp.cc[is.na(yeast$exp.cc)] <- mean(as.matrix(yeast$exp.cc),na.rm=TRUE)
# design <- c(rep(0,20),rep(NA,10),rep(1,20))
# monsterRes <- monster(yeast$exp.cc, design, yeast$motif, nullPerms=100, numMaxCores=4)#'
data(monsterRes)
monsterdTFIPlot(monsterRes)
```

---

monsterGetTm *monsterGetTm*

Description

acessor for the transition matrix in MONSTER object

Usage

```
monsterGetTm(x)
```

Arguments

x	an object of class "monsterAnalysis"

Value

Transition matrix

Examples

```
data(monsterRes)
tm <- monsterGetTm(monsterRes)
```

monsterHclHeatmapPlot *Transformation matrix plot*

Description

This function plots a hierachically clustered heatmap and corresponding dendrogram of a transaction matrix

Usage

```
monsterHclHeatmapPlot(monsterObj, method = "pearson")
```

Arguments

monsterObj	monsterAnalysis Object
method	distance metric for hierarchical clustering. Default is "Pearson correlation"

Value

ggplot2 object for transition matrix heatmap

Examples

```
# data(yeast)
# design <- c(rep(0,20),rep(NA,10),rep(1,20))
# yeast$exp.cc[is.na(yeast$exp.cc)] <- mean(as.matrix(yeast$exp.cc),na.rm=TRUE)
# monsterRes <- monster(yeast$exp.cc, design, yeast$motif, nullPerms=10, numMaxCores=1)
data(monsterRes)
monsterHclHeatmapPlot(monsterRes)
```

monsterMonsterNI *Bipartite Edge Reconstruction from Expression data*

Description

This function generates a complete bipartite network from gene expression data and sequence motif data

Usage

```
monsterMonsterNI(
  motif.data,
  expr.data,
  verbose = FALSE,
  randomize = "none",
  method = "bere",
  ni.coefficient.cutoff = NA,
  alphaw = 1,
  regularization = "none",
  score = "motifincluded",
  cpp = FALSE
)
```

Arguments

motif.data	A motif dataset, a data.frame, matrix or exprSet containing 3 columns. Each row describes an motif associated with a transcription factor (column 1) a gene (column 2) and a score (column 3) for the motif.
expr.data	An expression dataset, as a genes (rows) by samples (columns)
verbose	logical to indicate printing of output for algorithm progress.
randomize	logical indicating randomization by genes, within genes or none
method	String to indicate algorithm method. Must be one of "bere","pearson","cd","lda", or "wcd". Default is "bere". Important note: the direct regulatory network observed from gene expression is currently implemented as a regular correlation as opposed to the partial correlation described in the paper (please see Schlauch et al., 2017, https://doi.org/10.1186/s12918-017-0517-y)
ni.coefficient.cutoff	numeric to specify a p-value cutoff at the network inference step. Default is NA, indicating inclusion of all coefficients.
alphaw	A weight parameter between 0 and 1 specifying proportion of weight to give to indirect compared to direct evidence. The default is 0.5 to give an equal weight to direct and indirect evidence.
regularization	String parameter indicating one of "none", "L1", "L2"
score	String to indicate whether motif information will be readded upon completion of the algorithm to give to indirect compared to direct evidence. See documentation.
cpp	logical use C++ for maximum speed, set to false if unable to run.

Value

matrix for inferred network between TFs and genes

Examples

```
data(yeast)
cc.net <- monsterMonsterNI(yeast$motif,yeast$exp.cc)
```

---

monsterPlotMonsterAnalysis

monsterPlotMonsterAnalysis

Description

plots the sum of squares of off diagonal mass (differential TF Involvement)

Usage

```
monsterPlotMonsterAnalysis(x, ...)
```

Arguments

x	an object of class "monsterAnalysis"
...	further arguments passed to or from other methods.

Value

Plot of the dTFI for each TF against null distribution

Examples

```
data(yeast)
yeast$exp.cc[is.na(yeast$exp.cc)] <- mean(as.matrix(yeast$exp.cc),na.rm=TRUE)
design <- c(rep(1,25),rep(0,10),rep(NA,15))
#monsterRes <- monster(yeast$exp.cc, design,
#yeast$motif, nullPerms=10, numMaxCores=1)
#monsterPlotMonsterAnalysis(monsterRes)
```

`monsterPrintMonsterAnalysis` *monsterPrintMonsterAnalysis*

Description

summarizes the results of a MONSTER analysis

Usage

```
monsterPrintMonsterAnalysis(x, ...)
```

Arguments

x	an object of class "monster"
...	further arguments passed to or from other methods.

Value

Description of transition matrices in object

Examples

```
data(yeast)
yeast$exp.cc[is.na(yeast$exp.cc)] <- mean(as.matrix(yeast$exp.cc),na.rm=TRUE)
design <- c(rep(1,25),rep(0,10),rep(NA,15))
#monster(yeast$exp.cc,design,yeast$motif, nullPerms=10, numMaxCores=1)
```

monsterRes *MONSTER results from example cell-cycle yeast transition*

Description

This data contains the MONSTER result from analysis of Yeast Cell cycle, included in data(yeast). This result arbitrarily takes the first 20 gene expression samples in yeast$cc to be the baseline condition, and the final 20 samples to be the final condition.

Usage

```
data(monsterRes)
```

Format

MONSTER obj #' @references Schlauch, Daniel, et al. "Estimating drivers of cell state transitions using gene regulatory network models." BMC systems biology 11.1 (2017): 1-10.

monsterTransformationMatrix *Bi-partite network analysis tools*

Description

This function analyzes a bi-partite network.

Usage

```
monsterTransformationMatrix(
  network.1,
  network.2,
  by.tfs = TRUE,
  standardize = FALSE,
  remove.diagonal = TRUE,
  method = "ols"
)
```

Arguments

network.1	starting network, a genes by transcription factors data.frame with scores for the existence of edges between
network.2	final network, a genes by transcription factors data.frame with scores for the existence of edges between
by.tfs	logical indicating a transcription factor based transformation. If false, gives gene by gene transformation matrix
standardize	logical indicating whether to standardize the rows and columns
remove.diagonal	logical for returning a result containing 0s across the diagonal
method	character specifying which algorithm to use, default='ols'

Value

matrix object corresponding to transition matrix

Examples

```
data(yeast)
cc.net.1 <- monsterMonsterNI(yeast$motif,yeast$exp.cc[1:1000,1:20])
cc.net.2 <- monsterMonsterNI(yeast$motif,yeast$exp.cc[1:1000,31:50])
monsterTransformationMatrix(cc.net.1, cc.net.2)
```

`monsterTransitionNetworkPlot`

This function uses igraph to plot the transition matrix (directed graph) as a network. The edges in the network should be read as A 'positively/negatively contributes to' the targeting of B in the target state.

Description

This function uses igraph to plot the transition matrix (directed graph) as a network. The edges in the network should be read as A 'positively/negatively contributes to' the targeting of B in the target state.

Usage

```
monsterTransitionNetworkPlot(
  monsterObj,
  numEdges = 100,
  numTopTFs = 10,
  rescale = "significance"
)
```

Arguments

`monsterObj`	monsterAnalysis Object
`numEdges`	The number of edges to display
`numTopTFs`	The number of TFs to display, only when rescale='significance'
`rescale`	string to specify the order of edges. If set to 'significance', the TFs with the largest dTFI significance (smallest dTFI p-values) will be filtered first before plotting the edges with the largest magnitude in the transition matrix. Otherwise the filtering step will be skipped and the edges with the largest transitions will be plotted. The plotted graph represents the top numEdges edges between the numTopTFs if rescale=='significance' and top numEdges edges otherwise. The edge weight represents the observed transition edges standardized by the null and the node size in the graph is proportional to the p-values of the dTFIs of each TF. When rescale is set to 'significance', the results can be different between two MONSTER runs if the number of permutations is not large enough to sample the null, that is why it is the seed should be set prior to calling MONSTER to get reproducible results. If rescale is set to another value such as 'none', it will produce deterministic results between two identical MONSTER runs.

Value

plot the transition matrix (directed graph) as a network.

Examples

```
# data(yeast)
# yeast$exp.cc[is.na(yeast$exp.cc)] <- mean(as.matrix(yeast$exp.cc),na.rm=TRUE)
# design <- c(rep(0,20),rep(NA,10),rep(1,20))
# monsterRes <- monster(yeast$exp.cc, design, yeast$motif, nullPerms=100, numMaxCores=4)#'
data(monsterRes)
monsterTransitionNetworkPlot(monsterRes, rescale='significance')
monsterTransitionNetworkPlot(monsterRes, rescale='none')
```

---

monsterTransitionPCAPlot

Principal Components plot of transformation matrix

Description

This function plots the first two principal components for a transaction matrix

Usage

```
monsterTransitionPCAPlot(
  monsterObj,
  title = "PCA Plot of Transition",
  clusters = 1,
  alpha = 1
)
```

Arguments

monsterObj	a monsterAnalysis object resulting from a monster analysis
title	The title of the plot
clusters	A vector indicating the number of clusters to compute
alpha	A vector indicating the level of transparency to be plotted

Value

ggplot2 object for transition matrix PCA

Examples

```
# data(yeast)
# design <- c(rep(0,20),rep(NA,10),rep(1,20))
# yeast$exp.cc[is.na(yeast$exp.cc)] <- mean(as.matrix(yeast$exp.cc),na.rm=TRUE)
# monsterRes <- monster(yeast$exp.cc, design, yeast$motif, nullPerms=100, numMaxCores=4)#'
data(monsterRes)
# Color the nodes according to cluster membership
clusters <- kmeans(monsterGetTm(monsterRes),3)$cluster
monsterTransitionPCAPlot(monsterRes,
title="PCA Plot of Transition - Cell Cycle vs Stress Response",
clusters=clusters)
```

mut.ucec *Example of mutation data*

Description

Somatic mutations of Uterine Corpus Endometrial Carcinoma from The Cancer Genome Atlas. Data were downloaded and pre-processed as described in Kuijjer et al.

This data is a toy example data for SAMBAR, it contains gene annotations

Usage

```
data(mut.ucec)

data(mut.ucec)
```

Format

A table with 248 rows and 19754 columns

A binary dataframe where 1 indicates a mutation and 0 otherwise.

Value

A table of 19754 genes by 248 samples

References

Kuijjer, Marieke Lydia, et al. "Cancer subtype identification using somatic mutation data." British journal of cancer 118.11 (2018): 1492-1501.

otter *Run OTTER in R*

Description

Description: OTTER infers gene regulatory networks using TF DNA binding motif (W), TF PPI (P), and gene coexpression (C) through minimzing the following objective: min f(W) with f(W) = (1-lambda)*||WW' - P||^2 + lambda*||W'W - C||^2 + (gamma/2)*||W||^2

Usage

```
otter(W, P, C, lambda = 0.035, gamma = 0.335, Iter = 60, eta = 1e-05, bexp = 1)
```

Arguments

W	: TF-gene regulatory network based on TF motifs as a matrix of size (t,g), g=number of genes, t=number of TFs
P	: TF-TF protein interaction network as a matrix of size (t,t)
C	: gene coexpression as a matrix of size (g,g)
lambda	: tuning parameter in [0,1] (higher gives more weight to C)
gamma	: regularization parameter
Iter	: number of iterations of the algorithm
eta	: learning rate
bexp	: exponent influencing learning rate (higher means smaller)
	Outputs:

Details

Inputs:

Value

W : Predicted TF-gene complete regulatory network as an adjacency matrix of size (t,g).

Examples

```
W=matrix(rexp(100, rate=.1), ncol=10)
C=matrix(rexp(100, rate=.1), ncol=10)
P=matrix(rexp(100, rate=.1), ncol=10)

# Run OTTER algorithm
W <- otter(W, P, C)
```

---

pandaDiffEdges *Identify differential edges in two PANDA networks*

Description

To determine the probability that an edge is "different" between the networks, we first subtracted the z-score weight values estimated by PANDA for the two networks and then determined the value of the inverse cumulative distribution for this difference. The product of these two probabilities represents the probability than an edge is both "supported" and "different." We select edges for which this combined probability is greater than a threshold probability (default value is 0.8).

Usage

```
pandaDiffEdges(
  panda.net1,
  panda.net2,
  threshold = 0.8,
  condition_name = "cond.1"
)
```

Arguments

panda.net1	vector indicating the PANDA networks of one condition or phenotype.
panda.net2	vector indicating the PANDA networks of another compared condition orphenotype.
threshold	numerical vector indicating a threshold probability to select select edges.
condition_name	string vector indicating the condition name of net1

Value

a data.frame with five columns: tf, gene, motif, Score and defined condition name(the row with "T" in this column means this egde belongs to first condition or phenotype, "F" means this edge belongs to the second condition or phenotype)

Examples

```
# refer to four input datasets files in inst/extdat
treated_expression_file_path <- system.file("extdata", "expr4_matched.txt",
package = "netZooR", mustWork = TRUE)
control_expression_file_path <- system.file("extdata", "expr10_matched.txt",
package = "netZooR", mustWork = TRUE)
motif_file_path <- system.file("extdata", "chip_matched.txt", package = "netZooR", mustWork = TRUE)
ppi_file_path <- system.file("extdata", "ppi_matched.txt", package = "netZooR", mustWork = TRUE)


# Run PANDA for treated and control network
#treated_all_panda_result <- pandaPy(expr_file = treated_expression_file_path,
#motif_file= motif_file_path, ppi_file = ppi_file_path, modeProcess="legacy", remove_missing = TRUE )
#control_all_panda_result <- pandaPy(expr_file = control_expression_file_path,
#motif_file= motif_file_path, ppi_file= ppi_file_path, modeProcess="legacy", remove_missing = TRUE )

# access PANDA regulatory network
#treated_net <- treated_all_panda_result$panda
#control_net <- control_all_panda_result$panda

#merged.panda <- pandaDiffEdges(treated_net, control_net, condition_name="treated")
```

---

pandaPy *Run Python implementation PANDA in R*

Description

PANDA(Passing Attributes between Networks for Data Assimilation) is a message-passing model to reconstruct gene regulatory network, which integrates multiple sources of biological data-including protein-protein interaction data, gene expression data, and transcription factor binding motifs data to reconstruct genome-wide, condition-specific regulatory networks. [(Glass et al. 2013)]) This function is designed to run the a derived PANDA implementation in Python Library "netZooPy" netZooPy.

Usage

```
pandaPy(
  expr_file,
  motif_file = NULL,
  ppi_file = NULL,
  computing = "cpu",
  precision = "double",
  save_memory = FALSE,
  save_tmp = TRUE,
  keep_expression_matrix = FALSE,
  modeProcess = "union",
  remove_missing = FALSE,
  with_header = FALSE
)
```

Arguments

expr_file	Character string indicating the file path of expression values file, with each gene(in rows) across samples(in columns).
motif_file	An optional character string indicating the file path of a prior transcription factor binding motifs dataset. When this argument is not provided, analysis will continue with Pearson correlation matrix.
ppi_file	An optional character string indicating the file path of protein-protein interaction edge dataset. Also, this can be generated with a list of proteins of interest by sourcePPI.
computing	'cpu' uses Central Processing Unit (CPU) to run PANDA; 'gpu' use the Graphical Processing Unit (GPU) to run PANDA. The default value is "cpu".
precision	'double' computes the regulatory network in double precision (15 decimal digits); 'single' computes the regulatory network in single precision (7 decimal digits) which is fastaer, requires half the memory but less accurate. The default value is 'double'.
save_memory	'TRUE' removes temporary results from memory. The result network is weighted adjacency matrix of size (nTFs, nGenes); 'FALSE' keeps the temporary files in memory. The result network has 4 columns in the form gene - TF - weight in motif prior - PANDA edge. PANDA indegree/outdegree of panda network, only if save_memory = FALSE. The default value is 'FALSE'.
save_tmp	'TRUE' saves middle data like expression matrix and normalized networks; 'FALSE' deletes the middle data. The default value is 'TURE'.
keep_expression_matrix	'TRUE' keeps the input expression matrix as an attribute in the result Panda object.'FALSE' deletes the expression matrix attribute in the Panda object. The default value is 'FALSE'.
modeProcess	'legacy' refers to the processing mode in netZooPy<=0.5, 'union': takes the union of all TFs and genes across priors and fills the missing genes in the priors with zeros; 'intersection': intersects the input genes and TFs across priors and removes the missing TFs/genes. Default values is 'union'.
remove_missing	Only when modeProcess='legacy': remove_missing='TRUE' removes all unmatched TF and genes; remove_missing='FALSE' keeps all tf and genes. The default value is 'FALSE'.
with_header	if TRUE reads header of expression matrix

Value

When save_memory=FALSE(default), this function will return a list of three items: Use $panda to access the standard output of PANDA as data frame, which consists of four columns: "TF", "Gene", "Motif" using 0 or 1 to indicate if this edge belongs to prior motif dataset, and "Score".

Use $indegree to access the indegree of PANDA network as data frame, which consists of two columns: "Gene", "Score".

Use $outdegree to access the outdegree of PANDA network as data frame, which consists of two columns: "TF", "Score".

When save_memory=TRUE, this function will return a weigheted adjacency matirx of size (nTFs, nGenes), use $WAMpanda to access.

Examples

```
# take the treated TB dataset as example here.
# refer to the datasets files path in inst/extdat

treated_expression_file_path <- system.file("extdata", "expr4_matched.txt",
package = "netZooR", mustWork = TRUE)
treated_expression_file_path <- system.file("extdata", "expr4_matched.txt",
 package = "netZooR", mustWork = TRUE)
motif_file_path <- system.file("extdata", "chip_matched.txt", package = "netZooR", mustWork = TRUE)
ppi_file_path <- system.file("extdata", "ppi_matched.txt", package = "netZooR", mustWork = TRUE)


# Run PANDA for treated and control network

treated_all_panda_result <- pandaPy(expr_file = treated_expression_file_path,
motif_file = motif_file_path, ppi_file = ppi_file_path,
modeProcess="legacy", remove_missing = TRUE )

# access PANDA regulatory network
treated_net <- treated_all_panda_result$panda

# access PANDA regulatory indegree network.
indegree_net <- treated_all_panda_result$indegree

# access PANDA regulatory outdegree networks
outdegree_net <- treated_all_panda_result$outdegree
```

---

pandaToAlpaca *Use two PANDA network to generate an ALPACA result*

Description

ALPACA(ALtered Partitions Across Community Architectures) is a method for comparing two genome-scale networks derived from different phenotypic states to identify condition-specific modules. [(Padi and Quackenbush 2018)]) This function compares two networks generate by pandaPy in this package and finds the sets of nodes that best characterize the change in modular structure.

Usage

```
pandaToAlpaca(panda.net1, panda.net2, file.stem = "./alpaca", verbose = FALSE)
```

Arguments

panda.net1	data.frame indicating an complete network of one condition generated by pandaPy
panda.net2	data.frame indicating an complete network of another condition generated by pandaPy
file.stem	String indicating the folder path and prefix of result files, where all results will be stored.
verbose	Boolean vector indicating whether the full differential modularity matrix should also be written to a file. The default values is 'FALSE'.

Value

A string message showing the location of output file if file.stem is given, and a List where the first element is the membership vector and second element is the contribution score of each node to its module's total differential modularity

Examples

```
# refer to four input datasets files in inst/extdat
treated_expression_file_path <- system.file("extdata", "expr4_matched.txt",
package = "netZooR", mustWork = TRUE)
control_expression_file_path <- system.file("extdata", "expr10_matched.txt",
package = "netZooR", mustWork = TRUE)
motif_file_path <- system.file("extdata", "chip_matched.txt", package = "netZooR", mustWork = TRUE)
ppi_file_path <- system.file("extdata", "ppi_matched.txt", package = "netZooR", mustWork = TRUE)


# Run PANDA for treated and control network

treated_panda_net <- pandaPy(expr_file = treated_expression_file_path,
motif_file = motif_file_path, ppi_file = ppi_file_path,
modeProcess="legacy", remove_missing = TRUE )$panda
control_panda_net <- pandaPy(expr_file = control_expression_file_path,
motif_file = motif_file_path, ppi_file = ppi_file_path,
modeProcess="legacy", remove_missing = TRUE )$panda

# Run ALPACA
alpaca<- pandaToAlpaca(treated_panda_net, control_panda_net, "./TB", verbose=TRUE)
```

---

pandaToCondorObject *Turn PANDA network into a CONDOR object*

Description

CONDOR (COmplex Network Description Of Regulators) implements methods for clustering bi-apartite networks and estimatiing the contribution of each node to its community's modularity, [(Platig et al. 2016)]) This function uses the result of PANDA algorithm as the input dataset to run CONDOR algorithm. More about condor package and usage.

Usage

```
pandaToCondorObject(panda.net, threshold)
```

Arguments

panda.net	Data Frame indicating the result of PANDA regulatory network, created by pandaPy
threshold	Numeric vector of the customered threshold to select edges. Default value is the the midpoint between the median edge-weight of prior (3rd column "Motif" is 1.0) edges and the median edge-weight of non-prior edges (3rd column "Motif" is 0.0) in PANDA network. and the median edge-weight of non-prior edges (3rd column "Motif" is 0.0) in PANDA network.

Value

a CONDOR object, see createCondorObject.

Examples

```
# refer to three input datasets files in inst/extdat
treated_expression_file_path <- system.file("extdata", "expr4_matched.txt",
package = "netZooR", mustWork = TRUE)
motif_file_path <- system.file("extdata", "chip_matched.txt", package = "netZooR", mustWork = TRUE)
ppi_file_path <- system.file("extdata", "ppi_matched.txt", package = "netZooR", mustWork = TRUE)


# Run PANDA to construct the treated network

treated_all_panda_result <- pandaPy(expr_file = treated_expression_file_path,
motif_file= motif_file_path, ppi_file = ppi_file_path,
modeProcess="legacy", remove_missing = TRUE )

# access PANDA regulatory network
treated_net <- treated_all_panda_result$panda

# Obtain the condor.object from PANDA network
treated_condor_object <- pandaToCondorObject(treated_net, threshold = 0)

# cluster condor.object
treated_condor_object <- condorCluster(treated_condor_object, project = FALSE)

# package igraph and package viridisLite are already loaded with this package.
library(viridisLite)
treated_color_num <- max(treated_condor_object$red.memb$com)
treated_color <- viridis(treated_color_num, alpha = 1, begin = 0, end = 1,
direction = 1, option = "D")
condorPlotCommunities(treated_condor_object, color_list=treated_color,
point.size=0.04, xlab="Target", ylab="Regulator")
```

`runEgret` *Run EGRET in R*

Description

Description: NOTE: Beta version. EGRET infers individual-specific gene regulatory networks using inidividual level data - a genotype vcf file (v) and QBiC binding predictions (q) - as well as population/reference level data - eQTLs (b), a motif-gene prior (m), PPI network (p), and gene expression (e). An annotation file g is also used to map TF names to their corresponding ensemble ids.

Usage

```
runEgret(b, v, q, m, e, p, g, t)
```

Arguments

`b`	: Data frame of eQTL data, each row containing an eQTL which exist within motif regions adjacent to the eGene, with columns TF, gene, variant position,variant chromosome, eQTL beta value.
`v`	: Data frame of VCF file containing SNPs of the individual in question
`q`	: Data frame of QBiC predictions of the effect of eQTL variants on TF binding. Each row represents an eQTL variant with a predicted negative (disruptive) effect on the binding of the TF corresponding to the motif in which the eQTL variant resides. Colums are: eQTL variant as chr[chrNum]_position, TF, adjacent eGene, QBiC binding effect size and QBiC binding effect (should be negative)
`m`	: Motif prior data frame. Each row represents an edge in the bipartite motif prior, with columns TF, gene and edge weight. The edge weight should be 1 or 0 based on the presence/absence of the TF motif in the promoter region of the gene.
`e`	: Gene expression data frame in which each row represents a gene and each column represents the expression of that gene in a sample. The first column should contain gene IDs.
`p`	: PPI network data frame. Each row represents an edgem with columns TF, TF and interaction weight.
`g`	: Data frame mapping gene names to gene ids, with columns containing the gene ID the corresponding gene name.
`t`	: A string containing a name for the EGRET run. Output files will be labelled with this tag.
	Outputs:

Details

Inputs:

Value

EGRET : Predicted genotye-specific gene regulatory network saved as tag_egret.RData

BASELINE : A Baseline (PANDA) genotype-agnostic gene regulatory network saved as tag_panda.RData

Examples

```
# Run EGRET algorithm
toy_qbic_path <- system.file("extdata", "toy_qbic.txt", package = "netZooR",
mustWork = TRUE)
toy_genotype_path <- system.file("extdata", "toy_genotype.vcf",
package = "netZooR", mustWork = TRUE)
toy_motif_path <- system.file("extdata", "toy_motif_prior.txt",
package = "netZooR", mustWork = TRUE)
toy_expr_path <- system.file("extdata", "toy_expr.txt",
package = "netZooR", mustWork = TRUE)
toy_ppi_path <- system.file("extdata", "toy_ppi_prior.txt",
package = "netZooR", mustWork = TRUE)
toy_eqtl_path <- system.file("extdata", "toy_eQTL.txt",
package = "netZooR", mustWork = TRUE)
toy_map_path <- system.file("extdata", "toy_map.txt",
package = "netZooR", mustWork = TRUE)
qbic <- read.table(file = toy_qbic_path, header = FALSE)
vcf <- read.table(toy_genotype_path, header = FALSE, sep = "\t",
stringsAsFactors = FALSE,
colClasses = c("character", "numeric", "character", "character", "character",
"character", "character", "character", "character", "character"))
motif <- read.table(toy_motif_path, sep = "\t", header = FALSE)
expr <- read.table(toy_expr_path, header = FALSE, sep = "\t", row.names = 1)
ppi <- read.table(toy_ppi_path, header = FALSE, sep = "\t")
qtl <- read.table(toy_eqtl_path, header = FALSE)
nameGeneMap <- read.table(toy_map_path, header = FALSE)
tag <- "my_toy_egret_run"

runEgret(qtl,vcf,qbic,motif,expr,ppi,nameGeneMap,tag)
```

---

sambar *Main SAMBAR function.*

Description

Main SAMBAR function.

Usage

```
sambar(
  mutdata = mut.ucec,
  esize = exon.size,
 signatureset = system.file("extdata", "h.all.v6.1.symbols.gmt", package = "netZooR",
    mustWork = TRUE),
  cangenes = genes,
  kmin = 2,
  kmax = 4
)
```

Arguments

mutdata	Mutation data in matrix format. The number of mutations should be listed for samples (rows) and genes (columns).
esize	A integer vector of gene lengths, with gene symbols as names.
signatureset	A file containing gene sets (signatures) in .gmt format. These gene sets will be used to de-sparsify the gene-level mutation scores.
cangenes	A vector of genes, for example of cancer-associated genes. This will be used to subset the gene-level mutation data to.
kmin	The minimum number of subtypes the user wants to assess. Defaults to 2.
kmax	The maximum number of subtypes the user wants to assess. Defaults to 4.

Value

A list of samples and the subtypes to which these samples are assigned, for each k.

Examples

```
data("exon.size")
data("mut.ucec")
data("genes")
sambar(mutdata=mut.ucec, esize=exon.size, signatureset=system.file("extdata",
"h.all.v6.1.symbols.gmt", package="netZooR", mustWork=TRUE),
       cangenes=genes, kmin=2, kmax=4)
```

---

sambarConvertgmt *Convert .gmt files into a binary matrix.*

Description

Convert .gmt files into a binary matrix.

Usage

```
sambarConvertgmt(signature, cagenes)
```

Arguments

signature	A file containing gene sets (signatures) in .gmt format. These gene sets will be used to de-sparsify the gene-level mutation scores.
cagenes	A vector of genes, for example of cancer-associated genes. This will be used to subset the gene-level mutation data to.

Value

A matrix containing gene set mutation scores.

sambarCorgenelength *Normalize gene mutation scores by gene length.*

Description

Normalize gene mutation scores by gene length.

Usage

```
sambarCorgenelength(x, cagenes, exonsize)
```

Arguments

x	Mutation data, in the format of a matrix, including the number of mutations for samples (rows) and genes (columns).
cagenes	A vector of genes, for example of cancer-associated genes. This will be used to subset the gene-level mutation data to.
exonsize	A vector of gene lengths. This will be used to normalize the gene mutation scores.

Value

Mutation rate-adjusted gene mutation scores.

sambarDesparsify *De-sparsify gene-level mutation scores into gene set-level mutation scores.*

Description

De-sparsify gene-level mutation scores into gene set-level mutation scores.

Usage

```
sambarDesparsify(edgx, mutratecorx)
```

Arguments

edgx	A binary matrix containing information on which genes belong to which gene sets. Output from the sambarConvertgmt function.
mutratecorx	Gene-level mutation scores corrected for the number of gene sets each gene belongs to (from sambar function).

Value

De-sparsified mutation data.

small1976 *Pollinator-plant interactions*

Description

A dataset containing the number of interactions 34 plants and 13 pollinators. The variables are as follows:

Usage

```
data(small1976)
```

Format

A data frame with 442 rows and 3 variables

Details

- pollinator. Species name of insect pollinator
- plant. Species name of plant
- interactions. Number of visitors caught on each plant species

References

https://www.nceas.ucsb.edu/interactionweb/html/small_1976.html

sourcePPI *Source the Protein-Protein interaction in STRING database*

Description

This function uses a list of Transcription Factors (TF) of interest to source the Protein-Protein interactions (PPI) from STRING database using all types of interactions not only the physical subnetwork Important: this function produces a simple unweighted network for tutorial purposes, and does not support weighted PPI edges for the moment. For more complex PPI network modeling, consider pulling the PPI network directly from STRINGdb directly or through their R package.

Usage

```
sourcePPI(TF, STRING.version = "10", species.index, ...)
```

Arguments

TF	a data frame with one column indicating the TF of interest
STRING.version	a numeric vector indicating the STRING version. Default valuve is 10
species.index	a numeric vector indicating NCBI taxonomy identifiers
...	any ddtional arguments passed to

Value

A PPI data.frame which contains three columns: "from" and "to" indicating the direction of protein-protein interaction, and "score" indicating the interaction score between two proteins.

Examples

```
# the example motif file
motif_file_path <- system.file("extdata", "chip_matched.txt", package = "netZooR", mustWork = TRUE)
motif <- read.table(motif_file_path, sep="\t")
# create a TF data frame with one column
TF <-data.frame(motif[,1])
# create PPI data frame by searching in STRING version 10
# and specifying specie to "Mycobacterium tuberculosis H37Rv".
# STRING verison 11 is only accessible to R 4.0.

if(R.Version()$major=="3"){PPI <- sourcePPI(TF, STRING.version="10",
species.index=83332, score_threshold=0)}
if(R.Version()$major=="4"){PPI <- sourcePPI(TF, STRING.version="11",
species.index=83332, score_threshold=0)}

# write out locally then can be used in \code{\link{pandaPy}}.
```

---

spider *Seeding PANDA Interactions to Derive Epigenetic Regulation*

Description

This function runs the SPIDER algorithm

Usage

```
spider(
  motif,
  expr = NULL,
  epifilter = NULL,
  ppi = NULL,
  alpha = 0.1,
  hamming = 0.001,
  iter = NA,
  output = c("regulatory", "coexpression", "cooperative"),
  zScale = TRUE,
  progress = FALSE,
  randomize = c("None", "within.gene", "by.gene"),
  cor.method = "pearson",
  scale.by.present = FALSE,
  edgelist = FALSE,
  remove.missing.ppi = FALSE,
  remove.missing.motif = FALSE,
  remove.missing.genes = FALSE,
  mode = "union"
)
```

Arguments

motif	A motif dataset, a data.frame, matrix or exprSet containing 3 columns. Each row describes an motif associated with a transcription factor (column 1) a gene (column 2) and a score (column 3) for the motif.
expr	An expression dataset, as a genes (rows) by samples (columns) data.frame
epifilter	A binary matrix that is of the same size as motif that will be used as a mask to filter motif for open chromatin region. Motif interactions that fall in open chromatin region will be kept and the others are removed.
ppi	A Protein-Protein interaction dataset, a data.frame containing 3 columns. Each row describes a protein-protein interaction between transcription factor 1(column 1), transcription factor 2 (column 2) and a score (column 3) for the interaction.
alpha	value to be used for update variable, alpha (default=0.1)
hamming	value at which to terminate the process based on hamming distance (default 10^-3)
iter	sets the maximum number of iterations SPIDER can run before exiting.
output	a vector containing which networks to return. Options include "regulatory", "coregulatory", "cooperative".
zScale	Boolean to indicate use of z-scores in output. False will use [0,1] scale.
progress	Boolean to indicate printing of output for algorithm progress.
randomize	method by which to randomize gene expression matrix. Default "None". Must be one of "None", "within.gene", "by.genes". "within.gene" randomization scrambles each row of the gene expression matrix, "by.gene" scrambles gene labels.
cor.method	Correlation method, default is "pearson".
scale.by.present	Boolean to indicate scaling of correlations by percentage of positive samples.
edgelist	Boolean to indicate if edge lists instead of matrices should be returned.
remove.missing.ppi	Boolean to indicate whether TFs in the PPI but not in the motif data should be removed. Only when mode==’legacy’.
remove.missing.motif	Boolean to indicate whether genes targeted in the motif data but not the expression data should be removed. Only when mode==’legacy’.
remove.missing.genes	Boolean to indicate whether genes in the expression data but lacking information from the motif prior should be removed. Only when mode==’legacy’.
mode	The data alignment mode. The mode ’union’ takes the union of the genes in the expression matrix and the motif and the union of TFs in the ppi and motif and fills the matrics with zeros for nonintersecting TFs and gens, ’intersection’ takes the intersection of genes and TFs and removes nonintersecting sets, ’legacy’ is the old behavior with PANDAR version 1.19.3. #’ Parameters remove.missing.ppi, remove.missingmotif, remove.missing.genes work only with mode==’legacy’.

Value

An object of class "panda" containing matrices describing networks achieved by convergence with SPIDER algorithm.
"regNet" is the regulatory network
"coregNet" is the coregulatory network
"coopNet" is the cooperative network

References

Sonawane, Abhijeet Rajendra, et al. "Constructing gene regulatory networks using epigenetic data." npj Systems Biology and Applications 7.1 (2021): 1-13.

Examples

```
data(pandaToyData)
pandaToyData$epifilter = pandaToyData$motif
nind=floor(runif(5000, min=1, max=dim(pandaToyData$epifilter)[1]))
pandaToyData$epifilter[nind,3] = 0
#spiderRes <- spider(pandaToyData$motif,pandaToyData$expression,
#                    pandaToyData$epifilter,pandaToyData$ppi,hamming=.1,progress=TRUE)
```

---

visPandaInCytoscape *Plot PANDA network in Cytoscape*

Description

This function is able to modify PANDA network and plot in Cytoscape. Please make sure that Cytoscape is installed and open it before calling this function.

Usage

```
visPandaInCytoscape(panda.net, network_name = "PANDA")
```

Arguments

panda.net	Character string indicating the input PANDA network in data frame structure type.
network_name	Character string indicating the name of Cytoscape network.

Value

PANDA network in Cytoscape

Examples

```
# refer to the input datasets files of control TB dataset in inst/extdat as example
control_expression_file_path <- system.file("extdata", "expr10_matched.txt",
package = "netZooR", mustWork = TRUE)
motif_file_path <- system.file("extdata", "chip_matched.txt", package = "netZooR", mustWork = TRUE)
ppi_file_path <- system.file("extdata", "ppi_matched.txt", package = "netZooR", mustWork = TRUE)

# Run PANDA algorithm
```

```
control_all_panda_result <- panda.py(expr = control_expression_file_path, motif = motif_file_path,
ppi = ppi_file_path, mode_process="legacy", rm_missing = TRUE )

# access PANDA regulatory network
control_net <- control_all_panda_result$panda

# select top 1000 edges in PANDA network by edge weight.
panda.net <- head(control_net[order(control_net$force,decreasing = TRUE),], 1000)

# run this function to create a network in Cytoscape.
visPandaInCytoscape(panda.net, network.name="PANDA")
```

`yeast` *Toy data derived from three gene expression datasets and a mapping from transcription factors to genes.*

Description

This data is a list containing gene expression data from three separate yeast studies along with data mapping yeast transcription factors with genes based on the presence of a sequence binding motif for each transcription factor in the vicinity of each gene. The motif data.frame, yeast$motif, describes a set of pairwise connections where a specific known sequence motif of a transcription factor was found upstream of the corresponding gene. The expression data, yeast$exp.ko, yeast$exp.cc, and yeast$exp.sr, are three gene expression datasets measured in conditions of gene knockout, cell cycle, and stress response, respectively.

Usage

```
data(yeast)
```

Format

A list containing 4 data.frames

Value

A list of length 4

References

Glass K, Huttenhower C, Quackenbush J, Yuan GC. Passing Messages Between Biological Networks to Refine Predicted Interactions. PLoS One. 2013 May 31;8(5):e64832.

Index