Package: CeTF (via r-universe)

May 30, 2026

Type Package

Title Coexpression for Transcription Factors using Regulatory Impact Factors and Partial Correlation and Information Theory analysis

Version 1.24.0

Description This package provides the necessary functions for performing the Partial Correlation coefficient with Information Theory (PCIT) (Reverter and Chan 2008) and Regulatory Impact Factors (RIF) (Reverter et al. 2010) algorithm. The PCIT algorithm identifies meaningful correlations to define edges in a weighted network and can be applied to any correlation-based network including but not limited to gene co-expression networks, while the RIF algorithm identify critical Transcription Factors (TF) from gene expression data. These two algorithms when combined provide a very relevant layer of information for gene expression studies (Microarray, RNA-seq and single-cell RNA-seq data).

Imports circlize, ComplexHeatmap, clusterProfiler, DESeq2, dplyr, GenomicTools.fileHandler, GGally, ggnetwork, ggplot2, ggpubr, ggrepel, graphics, grid, igraph, Matrix, network, Rcpp, RCy3, stats, SummarizedExperiment, S4Vectors, utils, methods

Suggests airway, kableExtra, knitr, org.Hs.eg.db, rmarkdown, testthat

SystemRequirements libcurl4-openssl-dev, libxml2-dev, libssl-dev, gfortran, build-essential, libz-dev, zlib1g-dev

Depends R (>= 4.0)

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 7.2.3

biocViews Sequencing, RNASeq, Microarray, GeneExpression, Transcription, Normalization, DifferentialExpression, SingleCell, Network, Regression, ChIPSeq, ImmunoOncology, Coverage

VignetteBuilder knitr

LinkingTo Rcpp, RcppArmadillo

Config/pak/sysreqs libcairo2-dev cmake libfontconfig1-dev libfreetype6-dev libglpk-dev make libicu-dev libpng-dev libuv1-dev libxml2-dev libssl-dev perl python3 libzmq3-dev zlib1g-dev

Repository https://bioc-release.r-universe.dev

Date/Publication 2026-04-28 12:51:33 UTC

RemoteUrl https://github.com/bioc/CeTF

RemoteRef RELEASE_3_23

RemoteSha a297f0726643fdf2b37b796c21a1ae289927d600

Contents

`bivar.awk` *Summary statistics from two variables*

Description

Read two columns of data values (say X and Y) and computes summary statistics including N, Mean, SD, Min and Max for X and Y, as well as the correlation between X and Y and the regression of Y on X.

Usage

```
bivar.awk(x)
```

Arguments

`x`	A dataframe with two columns (variables).

Value

Returns an summary statistics for two variables.

Examples

```
# creating a random dataframe with two columns (variables)
tab <- data.frame(a = sample(1:1000, 100, replace=TRUE),
                  b = sample(1:1000, 100, replace=TRUE))

# running bivar.awk function
bivar.awk(tab)
```

`CeTF-class` *The CeTF Class*

Description

The CeTF class is data storage class that stores all the results from `runAnalysis` function.

Value

Returns an CeTF object.

Slots

`Data` Includes the raw, tpm and norm (see `normExp`) data.

`DE` Includes the uniquely differentially expressed genes/TFs and the statistics for all genes (see `expDiff`).

`Input` Includes input matrices for RIF (see `RIF`) and PCIT (see `PCIT`) for both conditions in `runAnalysis` function analysis.

`Output` Includes the matrix output from RIF analysis (see `RIF`) and a matrix with PCIT output, and other two matrix with raw and significant adjacency (see `PCIT`) for both conditions inside of `runAnalysis` function analysis.

`Network` Network with Gene-Gene and Gene-TF interactions for both conditions (see `PCIT`), main TFs resulted from the complete analysis, all the TFs identified in the input data and matrix annotating all genes and TFs.

`CeTFdemo` *CeTFdemo class object example*

Description

A CeTFdemo class object to run the examples in functions. This object was generated after running the runAnalysis function. This object is the same generated in vignette for complete analysis. Note that this example data is reduced and don't have the Data slot.

Usage

```
data(CeTFdemo)
```

Format

An CeTF class object

Examples

```
data(CeTFdemo)
```

`CircosTargets` *Circos plot for the Transcription Factors/genes targets.*

Description

Generate an plot for Transcription Targets (TFs) or any gene targets. This plot consists of sorting all the chromosomes of any specie based in GTF annotation file and showing how the selected TF(s)/gene(s) targets are distributed. If a target is connected to the same chromosome as the selected one so the connection is defined as *cis*, otherwise it is a *trans* connection.

Usage

```
CircosTargets(object, file, nomenclature, selection, cond)
```

Arguments

`object`	CeTF class object resulted from `runAnalysis` function.
`file`	GTF file or path.
`nomenclature`	Gene nomenclature: *SYMBOL* or *ENSEMBL*.
`selection`	Specify a single or up to 4 TF/gene to be visualized for.
`cond`	The options are *condition1* or *condition2* based on the conditions previously defined in `runAnalysis` function.

Details

The black links are between different chromosomes while the red links are between the same chromosome.

Value

Returns an plot with a specific(s) TF/gene and its targets in order to visualize the chromosome location of each one.

Examples

```
## Not run:
CircosTargets(object = out,
file = '/path/to/gtf/specie.gtf',
nomenclature = 'SYMBOL',
selection = 'TCF4',
cond = 'condition1')

## End(Not run)
```

clustCoef *Calculate the clustering coefficient*

Description

Calculate the clustering coefficient for an adjacency matrix.

Usage

```
clustCoef(mat)
```

Arguments

mat	An adjacency matrix. Calculating the clustering coefficient only makes sense if some connections are zero i.e. no connection.

Value

Returns the clustering coefficient(s) for the adjacency matrix.

References

Nathan S. Watson-Haigh, Haja N. Kadarmideen, and Antonio Reverter (2010). PCIT: an R package for weighted gene co-expression networks based on partial correlation and information theory approaches. Bioinformatics. 26(3) 411-413. https://academic.oup.com/bioinformatics/article/26/3/411/215002

Examples

```
# loading a simulated counts data
data('simNorm')

# running PCIT analysis
results <- PCIT(simNorm)

# getting the clustering coefficient
clustCoef(results$adj_sig)
```

`clustCoefPercentage` *Calculate the clustering coefficient as a percentage*

Description

Given an adjacency matrix, calculate the clustering coefficient as a percentage of non-zero adjacencies.

Usage

```
clustCoefPercentage(mat)
```

Arguments

`mat`	An adjacency matrix. Calculating the clustering coefficient percentage only makes sense if some connections are zero i.e. no connection.

Value

Returns the clustering coefficient as a porcentage.

References

Nathan S. Watson-Haigh, Haja N. Kadarmideen, and Antonio Reverter (2010). PCIT: an R package for weighted gene co-expression networks based on partial correlation and information theory approaches. Bioinformatics. 26(3) 411-413. https://academic.oup.com/bioinformatics/article/26/3/411/215002

Examples

```
# loading a simulated counts data
data('simNorm')

# running PCIT analysis
results <- PCIT(simNorm)

# getting the clustering coefficient as percentage
clustCoefPercentage(results$adj_sig)
```

densityPlot *Density distribution of correlation coefficients and significant PCIT values*

Description

Generate the density plot for adjacency matrices. This function uses the raw adjacency matrix and significant adjacency matrix resulted from PCIT function.

Usage

```
densityPlot(mat1, mat2, threshold = 0.5)
```

Arguments

mat1	Raw adjacency matrix.
mat2	Significant adjacency matrix.
threshold	Threshold of correlation module to plot (default: 0.5).

Value

Returns an density plot of raw correlation with significant PCIT values.

Examples

```
# loading a simulated normalized data
data('simNorm')

# getting the PCIT results
results <- PCIT(simNorm[1:20, ])

# using the PCIT results to get density distribution of correlation coefficients
densityPlot(mat1 = results$adj_raw,
            mat2 = results$adj_sig,
            threshold = 0.5)
```

diffusion *Network diffusion analysis*

Description

Expand node selection using network propagation algorithms generating the expanded network for a core of genes and the network plot of this subnetwork.

Usage

```
diffusion(object, cond, genes, cyPath, name = "top_diffusion", label = TRUE)
```

Arguments

object	CeTF object resulted from runAnalysis function.
cond	Which conditions to be used to perform the diffusion analysis. The options are: network1 (1th condition) and network2 (2th condition).
genes	A single gene or a vector of characters indicating which genes will be used to perform diffusion analysis.
cyPath	System path of Cytoscape software (see *details* for further informations).
name	Network output name (default: top_diffusion)
label	If label is TRUE, shows the names of nodes (default: TRUE).

Details

To perform the diffusion analysis is necessary to install the latest Cytoscape software version (https://cytoscape.org/).

The *cyPath* argument varies depending on the operating system used, for example:

1. **For Windows users:** C:/Program Files/Cytoscape_v3.8.0/Cytoscape.exe
2. **For Linux users:** /home/user/Cytoscape_v3.8.0/Cytoscape
3. **For macOS users:** /Applications/Cytoscape_v3.8.0/cytoscape.sh

Value

Returns a list with the plot of the network and a table with the diffusion network.

Examples

```
## Not run:
data(CeTFdemo)

result <- diffusion(object = CeTFdemo,
                    cond = 'network1',
                    genes = c('ENSG00000185591', 'ENSG00000179094'),
                    cyPath = 'C:/Program Files/Cytoscape_v3.7.2/Cytoscape.exe',
                    name = 'top_diffusion',
                    label = TRUE)

## End(Not run)
```

enrichdemo *Enrichment data*

Description

Enrichemnt result from CeTFdemo using the genes of condition 1 network.

Usage

```
data(enrichdemo)
```

Format

An list

Examples

```
data(enrichdemo)
```

enrichPlot *Plots to visualize the enrichment analysis results*

Description

Generate three types of plots to visualize the enrichment analysis results from getEnrich function. The plots are an circular barplot, barplot and dotplot.

Usage

```
enrichPlot(res, showCategory = 10, type = "circle")
```

Arguments

res	A dataframe with getEnrich results.
showCategory	Number of enriched terms to display (default: 10).
type	Type of plot: circle, bar or dot (default: circle).

Value

Returns a circle, bar or dot plot of enrichment analysis results.

Examples

```
# loading enrichdemo
data(enrichdemo)

# circle barplot
enrichPlot(res = enrichdemo$results,
           showCategory = 10,
           type = 'circle')

# barplot
enrichPlot(res = enrichdemo$results,
           showCategory = 10,
           type = 'bar')

# dotplot
enrichPlot(res = enrichdemo$results,
           showCategory = 10,
           type = 'dot')
```

`expDiff` *Differential expression analysis*

Description

This function returns the differentially expressed genes when comparing two conditions.

Usage

```
expDiff(
  exp,
  anno = NULL,
  conditions = NULL,
  lfc = 1.5,
  padj = 0.05,
  diffMethod = "Reverter"
)
```

Arguments

`exp`	Count data where the rows are genes and coluns the samples.
`anno`	A single column dataframe. The column name must be 'cond', and the rownames must be the names of samples.
`conditions`	A character vector containing the name of the two conditions. The first name will be selected as reference.
`lfc`	log2 fold change module threshold to define a gene as differentially expressed (default: 1.5).

padj	Significance value to define a gene as differentially expressed (default: 0.05).
diffMethod	Choose between Reverter or DESeq2 method (default: 'Reverter'). The DESeq2 method is only for counts data (see details).

Details

The **Reverter** option to diffMethod parameter works as follows:

1. Calculation of mean between samples of each condition for all genes;
2. Subtraction between mean of control condition relative to other condition;
3. Calculation of variance of subtraction previously obtained;
4. The last step calculates the differential expression using the following formula, where x is the result of substraction (item 2) and var is the variance calculated in item 3:

$$diff = \frac{x - (sum(x)/length(x))}{\sqrt{var}}$$

The **DESeq2** option to diffMethod parameter is recommended only for count data. This method apply the differential expression analysis based on the negative binomial distribution (see DESeq).

Value

Returns an list with all calculations of differentially expressed genes and the subsetted differentially expressed genes by lfc and/or padj.

References

REVERTER, Antonio et al. Simultaneous identification of differential gene expression and connectivity in inflammation, adipogenesis and cancer. Bioinformatics, v. 22, n. 19, p. 2396-2404, 2006. https://academic.oup.com/bioinformatics/article/22/19/2396/240742

Examples

```
# loading a simulated counts data
data('simCounts')

# creating the dataframe with annotation for each sample
anno <- data.frame(cond = c(rep('cond1', 10), rep('cond2', 10)))

# renaming colums of simulated counts data
colnames(simCounts) <- paste(colnames(simCounts), anno$cond, sep = '_')

# renaming anno rows
rownames(anno) <- colnames(simCounts)

# performing differential expression analysis using Reverter method
out <- expDiff(exp = simCounts,
               anno = anno,
               conditions = c('cond1', 'cond2'),
               lfc = 2,
```

```
                padj = 0.05,
                diffMethod = 'Reverter')
```

---

getData *Data accessor for a CeTF class object.*

Description

The Data accessor access the raw, tpm and normalized data from runAnalysis function analysis.

Usage

```
getData(x, type = "raw")

## S4 method for signature 'CeTF'
getData(x, type = "raw")
```

Arguments

x	CeTF-class object
type	Type of data: raw, tpm or norm (default: raw)

Value

Returns the raw, tpm or normalized data.

See Also

runAnalysis.

Examples

```
# load the CeTF class object resulted from runAnalysis function
data(CeTFdemo)

getData(CeTFdemo)
```

getDE *Differential Expression accessor for a CeTF class object.*

Description

The DE accessor access the differential expression resulted from runAnalysis function analysis.

Usage

```
getDE(x, type = "unique")

## S4 method for signature 'CeTF'
getDE(x, type = "unique")
```

Arguments

x	CeTF-class object
type	Type of DE matrix: unique and all (default: unique)

Value

Returns the DE genes with the statistics.

See Also

runAnalysis.

Examples

```
# load the CeTF class object resulted from runAnalysis function
data(CeTFdemo)

getDE(CeTFdemo)
```

getEnrich *Enrichment analysis for genes of network*

Description

Enrichment analysis of a set of genes derived from the network of any condition using clusterProfiler. Given a vector of genes, this function will return the enrichment related to the selected database.

Usage

```
getEnrich(
  genes,
  organismDB,
  keyType,
  ont,
  fdrMethod = "BH",
  fdrThr = 0.05,
  minGSSize = 5,
  maxGSSize = 500
)
```

Arguments

genes	Should be an R vector object containing the interesting gene list.
organismDB	clusterProfiler supports a lot of different organisms. Users can check the following link (https://www.bioconductor.org/packages/release/data/annotation/) and search for annotations starting with *org.*.
keyType	The ID type of the input genes (i.e. SYMBOL, ENTREZID, ENSEMBL, etc.).
ont	The functional categories for the enrichment analysis. The available ontologies are Biological Process (BP), Molecular Function (MF) and Cellular Component (CC).
fdrMethod	Has five FDR methods: holm, hochberg, hommel, bonferroni, BH, BY, fdr and none(default: BH).
fdrThr	The significant threshold for selected pathways (default: 0.05).
minGSSize	Will be exclude the categories with the number of annotated genes less than minGSSize for enrichment analysis (default: 5).
maxGSSize	Will be exclude the categories with the number of annotated genes larger than maxGSSize for enrichment analysis (default: 500).

Value

Returns an list with the results of the enrichment analysis of the genes and a network with the database ID (column 1) and the corresponding genes (column 2).

Examples

```
## Not run:
# load the CeTF class object resulted from runAnalysis function
library(org.Hs.eg.db)
data(CeTFdemo)

# getting the genes in network of condition 1
genes <- unique(c(as.character(NetworkData(CeTFdemo, 'network1')[, 'gene1']),
                  as.character(NetworkData(CeTFdemo, 'network1')[, 'gene2'])))

# performing getEnrich analysis
```

```
cond1 <- getEnrich(genes = genes, organismDB = org.Hs.eg.db, keyType = 'ENSEMBL',
                   ont = 'BP', fdrMethod = "BH", fdrThr = 0.05, minGSSize = 5,
                   maxGSSize = 500)

## End(Not run)
```

---

getGroupGO *Functional Profile of a gene set at specific GO level*

Description

Functional Profile of a gene set at specific GO level. Given a vector of genes, this function will return the GO profile at a specific level.

Usage

```
getGroupGO(genes, ont = "BP", keyType, annoPkg, level = 3)
```

Arguments

genes	Character vector with the genes to perform the functional profile.
ont	One of 'MF', 'BP', and 'CC' subontologies (default: 'BP').
keyType	Key type of inputted genes (i.e. 'ENSEMBL', 'SYMBOL', 'ENTREZID').
annoPkg	Package of annotation of specific organism (i.e. org.Hs.eg.db, org.Bt.eg.db, org.Rn.eg.db, etc).
level	Specific GO Level (default: 3).

Value

Returns an list with the results of the functional profile of the genes and a network with the ontologies (column 1) and the corresponding genes (column 2).

Examples

```
## Not run:
# load the annotation package
library(org.Hs.eg.db)

# load the CeTF class object resulted from runAnalysis function
data(CeTFdemo)

# getting the genes in network of condition 1
genes <- unique(c(as.character(NetworkData(CeTFdemo, 'network1')[, 'gene1']),
                  as.character(NetworkData(CeTFdemo, 'network1')[, 'gene2'])))

# performing getGroupGO analysis
cond1 <- getGroupGO(genes = genes,
```

```
                    ont = 'BP',
                    keyType = 'ENSEMBL',
                    annoPkg = org.Hs.eg.db,
                    level = 3)

## End(Not run)
```

---

gtfToBed *Converts GTF to BED*

Description

Converts a GTF to BED format.

Usage

```
gtfToBed(gtf)
```

Arguments

gtf	A GTF as data.frame.

Value

Returns a data.frame in BED format.

---

heatPlot *Heatmap-like functional classification*

Description

Heatmap-like functional classification to visualize the enrichment analysis results from getEnrich function. The plot contains the heatmap with the associated pathways genes, the significance of the enrichment and a barplot with the enrichment ratio.

Usage

```
heatPlot(res, diff, showCategory = 10, font_size = 6)
```

Arguments

res	A dataframe with getEnrich results.
diff	A dataframe with all differentialy expressed genes obtained from runAnalysis function. For better understanding, simply use the getDE accessor with 'all' option.
showCategory	Number of enriched terms to display (default: 10).
font_size	Size of gene row names (default: 6).

Value

Returns a Heatmap-like functional classification

Examples

```
# loading enrichdemo and CeTFdemo object
data(enrichdemo)
data(CeTFdemo)

heatPlot(res = enrichdemo$results,
         diff = getDE(CeTFdemo, 'all'),
         showCategory = 10)
```

`histPlot` *Histogram of connectivity distribution*

Description

Generate the histogram for adjacency matrix to show the clustering coefficient distribution.

Usage

```
histPlot(mat)
```

Arguments

`mat`	Adjacency matrix resulting from PCIT analysis in which has some zero values.

Value

Returns the histogram of connectivity distribution.

Examples

```
# loading a simulated normalized data
data(simNorm)

# getting the PCIT results for first 30 genes
results <- PCIT(simNorm[1:30, ])

# plotting the histogram for PCIT significance results
histPlot(results$adj_sig)
```

InputData *Input data accessor for a CeTF class object.*

Description

The input accessor access the input matrices used for RIF and PCIT analysis to both conditions resulted from `runAnalysis` function analysis.

Usage

```
InputData(x, analysis = "rif")

## S4 method for signature 'CeTF'
InputData(x, analysis = "rif")
```

Arguments

x	`CeTF-class` object
analysis	Type of analysis: rif, pcit1, pcit2. The numbers 1 and 2 correspond to the respective condition (default: rif).

Value

Returns the Inputs used for RIF and PCIT.

See Also

`runAnalysis`.

Examples

```
# load the CeTF class object resulted from runAnalysis function
data(CeTFdemo)

InputData(CeTFdemo)
```

netConditionsPlot *Network plot of gene-gene/gene-TFs interactions*

Description

Generate the network plot of gene-gene/gene-TFs interactions for both conditions.

Usage

```
netConditionsPlot(x)
```

Arguments

x	CeTF object resulted from runAnalysis function.

Value

Returns the network plot for both conditions.

Examples

```
# loading a simulated counts data
data('simCounts')

# performing runAnalysis function
out <- runAnalysis(mat = simCounts,
                   conditions=c('cond1', 'cond2'),
                   lfc = 3,
                   padj = 0.05,
                   TFs = paste0('TF_', 1:1000),
                   nSamples1 = 10,
                   nSamples2= 10,
                   tolType = 'mean',
                   diffMethod = 'Reverter',
                   data.type = 'counts')

# plotting networks conditions
netConditionsPlot(out)
```

---

netGOTFPlot *Plot a network for Ontologies, genes and TFs*

Description

Generate the plot of groupGO network result of getGroupGO function, and the integrated network of genes, GOs and TFs.

Usage

```
netGOTFPlot(
  netCond,
  resultsGO,
  netGO,
  anno,
  groupBy = "pathways",
  TFs = NULL,
  genes = NULL,
```

```
  keyTFs = NULL,
  size = 0.5,
  type = NULL
)
```

Arguments

netCond	Network of a specific condition. Can be found in result of runAnalysis (see NetworkData and NetworkData).
resultsGO	Dataframe with the results of getGroupGO (first element of list). This result can be filtered by applying filters for pathways selection.
netGO	Dataframe with the results of getGroupGO (second element of list).
anno	Annotation of gene or TFs. Can be found in result of runAnalysis function (see NetworkData).
groupBy	Which variables do you want to group in GO type? The options are: 'pathways', 'TFs' and 'genes' (default: 'pathways').
TFs	A character with selected TFs.
genes	A character with selected genes.
keyTFs	TFs identified as importants by runAnalysis (see NetworkData). This argument is used only if the type argument equals Integrated.
size	Size of nodes labels (default: 0.5).
type	Type of plot selected (GO or Integrated). If GO will plot the associated GO grouped by some variable, and if Integrated will plot a integrated network with genes, GO and TFs.

Value

Returns a list with the plot of the network for GO or integrated output under a condition and the table used to plot the network.

Examples

```
## Not run:
# load the annotation package
library(org.Hs.eg.db)

# load the CeTF class object resulted from runAnalysis function
data(CeTFdemo)

# getting the genes in network of condition 1
genes <- unique(c(as.character(NetworkData(CeTFdemo, 'network1')[, 'gene1']),
                  as.character(NetworkData(CeTFdemo, 'network1')[, 'gene2'])))

# performing getGroupGO analysis
cond1 <- getGroupGO(genes = genes,
                    ont = 'BP',
                    keyType = 'ENSEMBL',
                    annoPkg = org.Hs.eg.db,
```

```
                    level = 3)

# selecting only first 12 pathways
t1 <- head(cond1$results, 12)

# subsetting the network to have only the first 12 pathways
t2 <- subset(cond1$netGO, cond1$netGO$gene1 %in% as.character(t1[, 'ID']))

# generate the GO plot grouping by pathways
pt <- netGOTFPlot(netCond = NetworkData(CeTFdemo, 'network1'),
              resultsGO = t1,
              netGO = t2,
              anno = NetworkData(CeTFdemo, 'annotation'),
              groupBy = 'pathways',
              keyTFs = NetworkData(CeTFdemo, 'keytfs'),
              type = 'GO')
pt$plot
head(pt$tab$`GO:0006807`)

# generate the Integrated plot
pt <- netGOTFPlot(netCond = NetworkData(CeTFdemo, 'network1'),
              resultsGO = t1,
              netGO = t2,
              anno = NetworkData(CeTFdemo, 'annotation'),
              groupBy = 'pathways',
              keyTFs = NetworkData(CeTFdemo, 'keytfs'),
              type = 'Integrated')
pt$plot
head(pt$tab)

## End(Not run)
```

`NetworkData` *Networks data accessor for a CeTF class object.*

Description

The networks accessor access the networks, key TFs and annotations for each gene and TF resulted from PCIT analysis and `runAnalysis` function analysis.

Usage

```
NetworkData(x, type = "network1")

## S4 method for signature 'CeTF'
NetworkData(x, type = "network1")
```

Arguments

x	CeTF-class object
type	Type of data: network1, network2, keytfs, tfs or annotation.The numbers 1 and 2 correspond to the respective condition (default: network1).

Value

Returns the Outputs used for RIF and PCIT.

See Also

runAnalysis.

Examples

```
# load the CeTF class object resulted from runAnalysis function
data(CeTFdemo)

NetworkData(CeTFdemo)
```

---

normExp *Normalized expression transformation*

Description

Normalize the expression data of any type of experiment by columns, applying log(x + 1)/log(2).

Usage

```
normExp(tab)
```

Arguments

tab	A matrix or dataframe of expression data (i.e. TPM, counts, FPKM).

Value

Returns a table with normalized values.

Examples

```
# loading a simulated counts data
data('simCounts')

# getting the TPM matrix from counts
tpm <- apply(simCounts, 2, function(x) {
            (1e+06 * x)/sum(x)
            })

# normalizing TPM data
norm <- normExp(tpm)
```

`OutputData` *Output data accessor for a CeTF class object.*

Description

The output accessor access the output matrices and lists used for RIF and PCIT analysis to both conditions resulted from `runAnalysis` function analysis.

Usage

```
OutputData(x, analysis = "rif", type = "tab")

## S4 method for signature 'CeTF'
OutputData(x, analysis = "rif", type = "tab")
```

Arguments

`x`	`CeTF-class` object
`analysis`	Type of analysis: rif, pcit1, pcit2. The numbers 1 and 2 correspond to the respective condition (default: rif).
`type`	Type of matrix for PCIT output: tab, adj_raw or adj_sig (default: tab).

Value

Returns the Outputs used for RIF and PCIT.

See Also

`runAnalysis`.

Examples

```
# load the CeTF class object resulted from runAnalysis function
data(CeTFdemo)

OutputData(CeTFdemo)
```

PCIT *Partial Correlation and Information Theory (PCIT) analysis*

Description

The PCIT algorithm is used for reconstruction of gene co-expression networks (GCN) that combines the concept partial correlation coefficient with information theory to identify significant gene to gene associations defining edges in the reconstruction of GCN.

Usage

```
PCIT(input, tolType = "mean")
```

Arguments

input	A correlation matrix.
tolType	Type of tolerance (default: 'mean') given the 3 pairwise correlations (see tolerance).

Value

Returns an list with the significant correlations, raw adjacency matrix and significant adjacency matrix.

References

REVERTER, Antonio; CHAN, Eva KF. Combining partial correlation and an information theory approach to the reversed engineering of gene co-expression networks. Bioinformatics, v. 24, n. 21, p. 2491-2497, 2008. https://academic.oup.com/bioinformatics/article/24/21/2491/192682

Examples

```
# loading a simulated normalized data
data('simNorm')

# getting the PCIT results for first 30 genes
results <- PCIT(simNorm[1:30, ])

# printing PCIT output first 15 rows
head(results$tab, 15)
```

pcitC *A helper to calculate PCIT implemented in C/C++*

Description

Calculates the correlation matrix using PCIT algorithm

Usage

```
pcitC(cor, tolType)
```

Arguments

cor	A correlation matrix.
tolType	Type of tolerance (default: 'mean') given the 3 pairwise correlations (see tolerance.

Value

Correlation matrix resulted from PCIT algorithm.

See Also

(see PCIT)

Examples

```
library(Matrix)

# loading a simulated normalized data
data('simNorm')

# calculating the correlation matrix
suppressWarnings(gene_corr <- cor(t(simNorm[1:30, ])))
gene_corr[is.na(gene_corr)] <- 0

# getting the PCIT correlation results for first 30 genes
results <- pcitC(cor = Matrix(gene_corr, sparse = TRUE),
                 tolType = 1)
```

refGenes — *List of reference genes for 5 different organisms to perform enrichment*

Description

List with protein codings for 5 organisms that must be used as reference genes for functional enrichment. This list was generated using Ensembl GTF. The organisms are: Human (*Homo sapiens*), Mouse (*Mus musculus*), Zebrafish (*Danio rerio*), Cow (*Bos taurus*) and Rat (*Rattus norvegicus*).

Usage

```
data(refGenes)
```

Format

An list.

References

https://www.ensembl.org/info/data/ftp/index.html

Examples

```
data(refGenes)
```

RIF — *Regulatory Impact Factors (RIF) analysis*

Description

The RIF algorithm identify critical transcript factors (TF) from gene expression data.

Usage

```
RIF(input, nta = NULL, ntf = NULL, nSamples1 = NULL, nSamples2 = NULL)
```

Arguments

input	A matrix of expression with differentially expressed genes and transcript factors in rows, and the samples in columns.
nta	Number of Differentially Expressed (DE) genes.
ntf	Number of Transcription Factors (TFs).
nSamples1	Number of samples of condition 1.
nSamples2	Number of samples of condition 2.

Details

The input matrix must have the rows and columns ordered by the following request:

1. **rows**: DE genes followed by TFs;
2. **columns**: samples of condition1 followed by samples of condition2.

Value

Returns an dataframe with the regulatory impact factors metric for each transcript factor.

References

REVERTER, Antonio et al. Regulatory impact factors: unraveling the transcriptional regulation of complex traits from expression data. Bioinformatics, v. 26, n. 7, p. 896-904, 2010. https://academic.oup.com/bioinformatics/article/26/7/896/212064

Examples

```
# load RIF input example
data('RIF_input')

# performing RIF analysis
RIF_out <- RIF(input = RIF_input,
               nta = 104,
               ntf = 50,
               nSamples1 = 10,
               nSamples2 = 10)
```

RIF_input *Regulatory Impact Factors (RIF) input*

Description

Data used to the examples of RIF analysis. This data was generated based on simulated counts and normalized data.

Usage

```
data(RIF_input)
```

Format

An dataframe.

Examples

```
data(RIF_input)
```

RIFPlot *Relationship plots between RIF1, RIF2 and DE genes*

Description

Generate plots for the relationship between the RIF output analysis (RIF1 and RIF2) and for differentially expressed genes (DE).

Usage

```
RIFPlot(object, color = "darkblue", type = "RIF")
```

Arguments

object	CeTF object resulted from runAnalysis function.
color	Color of points (default: darkblue)
type	Type of plot. The available options are: RIF or DE (default: RIF)

Details

This function can only be used after using the runAnalysis function as it uses the CeTF class object as input.

Value

Returns a relationship plot between RIF1 and RIF2 or a plot with the relationship between RIF1 or RIF2 with DE genes.

Examples

```
# load the CeTF class object resulted from runAnalysis function
data(CeTFdemo)

# performing RIFPlot for RIF
RIFPlot(object = CeTFdemo,
        color  = 'darkblue',
        type   = 'RIF')


# performing RIFPlot for DE
RIFPlot(object = CeTFdemo,
        color  = 'darkblue',
        type   = 'DE')
```

`runAnalysis` *Whole analysis of Regulatory Impact Factors (RIF) and Partial Correlation and Information Theory analysis (PCIT)*

Description

This function uses RIF and PCIT algorithms to run the whole pipeline analysis. The pipeline is composed by 4 steps:

1. **Step 1:** Data adjustment;
2. **Step 2:** Differential expression analysis;
3. **Step 3:** Regulatory Impact Factors analysis;
4. **Step 4:** Partial Correlation and Information Theory analysis.

Usage

```
runAnalysis(
  mat,
  conditions = NULL,
  lfc = 2.57,
  padj = 0.05,
  TFs = NULL,
  nSamples1 = NULL,
  nSamples2 = NULL,
  tolType = "mean",
  diffMethod = "Reverter",
  data.type = NULL
)
```

Arguments

`mat`	Count data where the rows are genes and coluns the samples (conditions).
`conditions`	A vector of characters identifying the names of conditions (i.e. c('normal', 'tumor')).
`lfc`	logFoldChange module threshold to define a gene as differentially expressed (default: 2.57).
`padj`	Significance value to define a gene as differentially expressed (default: 0.05).
`TFs`	A vector of character with all transcripts factors of specific organism.
`nSamples1`	Number of samples that correspond to first condition.
`nSamples2`	Number of samples that correspond to second condition.
`tolType`	Tolerance calculation type (see `tolerance`) (default: 'mean').
`diffMethod`	Method to calculate Differentially Expressed (DE) genes (see `expDiff`) (default: 'Reverter')
`data.type`	Type of input data. If is *expression* (FPKM, TPM, etc) or *counts*.

Value

Returns an CeTF class object with output variables of each step of analysis.

See Also

CeTF-class

Examples

```
data('simCounts')
out <- runAnalysis(mat = simCounts,
                   conditions=c('cond1', 'cond2'),
                   lfc = 3,
                   padj = 0.05,
                   TFs = paste0('TF_', 1:1000),
                   nSamples1 = 10,
                   nSamples2= 10,
                   tolType = 'mean',
                   diffMethod = 'Reverter',
                   data.type = 'counts')
```

---

simCounts *Simulated counts data*

Description

Simulated counts data created using PROPER package. This data contains 21,000 genes, 1,000 transcript factors and 20 samples (divided in two conditions).

Usage

```
data(simCounts)
```

Format

An dataframe.

Examples

```
data(simCounts)
```

simNorm *Simulated normalized data*

Description

Simulated normalized data created using PROPER package. This data contains 69 genes, and 10 samples (correspondent to only one condition).

Usage

```
data(simNorm)
```

Format

An dataframe.

Examples

```
data(simNorm)
```

SmearPlot *Smear plot for Differentially Expressed genes and TFs*

Description

Generate an plot for Differentially Expressed (DE) genes and for specific TF that shows the relationship between log(baseMean) and Difference of Expression or log2FoldChange. This plot enables to visualize the distribution of DE genes and TF in both conditions.

Usage

```
SmearPlot(
  object,
  diffMethod,
  lfc = 1.5,
  conditions,
  TF = NULL,
  padjust = 0.05,
  label = FALSE,
  type = NULL
)
```

Arguments

object	CeTF class object resulted from runAnalysis function.
diffMethod	Method used to calculate Differentially Expressed (DE) genes (see expDiff).
lfc	logFoldChange module threshold to define a gene as differentially expressed (default: 1.5).
conditions	A vector of characters identifying the names of conditions (i.e. c('normal', 'tumoral')).
TF	Specify a single TF to be visualized for (used only if type argument equals TF).
padjust	Significance value to define a gene as differentially expressed in DESeq2 diffMethod option (default: 0.05).
label	If label is TRUE, shows the names of single TF and its respectives (default: FALSE).
type	Type of plot (DE or TF). If DE, will plot the smear plot for all differentally expressed genes and TFs for both conditions, and if TF, will plot the smear plot for a specific TF and their targets. targets for both conditions (default: FALSE).

Value

Returns an plot of log2(baseMean) by log2FoldChange or difference of expression for genes and TFs differentially expressed or for a single TF and its targets for both conditions.

Examples

```
# load the CeTF class object resulted from runAnalysis function
data(CeTFdemo)

#performing SmearPlot for DE genes and TFs
SmearPlot(object = CeTFdemo,
          diffMethod = 'Reverter',
          lfc = 1.5,
          conditions = c('untrt', 'trt'),
          type = 'DE')

#performing SmearPlot for DE genes and TFs
SmearPlot(object = CeTFdemo,
          diffMethod = 'Reverter',
          lfc = 1.5,
          conditions = c('untrt', 'trt'),
          TF = 'ENSG00000205189',
          label = FALSE,
          type = 'TF')
```

TFs *Transcripition Factors data*

Description

Transcripition Factors data from Kai Wang and Hiroki Nishida, 2015 for *Homo sapiens*.

Usage

```
data(TFs)
```

Format

An character vector with TFs for *Homo sapiens*.

References

See https://doi.org/10.1186/s12859-015-0552-x

Examples

```
data(TFs)
```

tolerance *Tolerance level between 3 pairwise correlations implemented in C/C++*

Description

Calculates the local tolerance for every trio of genes.

Usage

```
tolerance(a, b, c, tolType)
```

Arguments

a	Correlation value between the genes A and B.
b	Correlation value between the genes B and C.
c	Correlation value between the genes A and C.
tolType	Calculation type for tolerance (1 for mean, 2 for min and 3 for max).

Value

Returns the value of tolerance.

See Also

See vignette for more details about the pairwise correlations.

Examples

```
tolerance(0.5, -0.65, 0.23, tolType = 1)
tolerance(0.5, -0.65, 0.23, tolType = 2)
tolerance(0.5, -0.65, 0.23, tolType = 3)
```

Index