Package 'terapadog'

November 12, 2026

Type Package

Title Translational Efficiency Regulation Analysis using the PADOG Method

Version 1.4.0

Description This package performs a Gene Set Analysis with the approach adopted by PADOG on the genes that are reported as translationally regulated (ie. exhibit a significant change in TE) by the DeltaTE package. It can be used on its own to see the impact of translation regulation on gene sets, but it is also integrated as an additional analysis method within ReactomeGSA, where results are further contextualised in terms of pathways and directionality of the change.

License GPL-2

Encoding UTF-8

LazyData FALSE

Imports DESeq2, KEGGREST, stats, utils, dplyr, plotly, htmlwidgets, biomaRt, methods

Suggests apeglm, BiocStyle, knitr, rmarkdown, testthat

Collate preprocessing_helpers.R id_converter.R prepareTerapadogData.R terapadogBricks.R terapadog.R get_FCs.R assign_Regmode.R plotDTA.R

biocViews RiboSeq, Transcriptomics, GeneSetEnrichment, GeneRegulation, Reactome, Software

RoxygenNote 7.3.2

VignetteBuilder knitr

URL https://github.com/Gionmattia/terapadog

BugReports https://github.com/Gionmattia/terapadog/issues

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

git_branch RELEASE_3_23

git_last_commit 17c838f

git_last_commit_date 2026-04-28

Repository Bioconductor 3.23

Date/Publication 2026-11-11

Author Gionmattia Carancini [cre, aut] (ORCID:
<https://orcid.org/0000-0001-7936-4883>)

Maintainer Gionmattia Carancini <gionmattia@gmail.com>

Contents

assign_Regmode 2
check_input_df 3
check_integer_values 4
check_matching_colnames 4
check_value_range 5
detect_separator 6
generate_permutation_matrix 7
get_FCs 8
gsScoreFun 9
id_converter 10
plotDTA 11
prepareGeneSets 12
prepareTerapadogData 13
terapadog 15

Index **17**

`assign_Regmode` *This function reads the results of the function getFCs and assigns a regulatory mode to each gene based on the Fold Change (FC) for RNA-Seq Counts, Ribo-Seq Counts, or TE. It is intended for internal use and its output is the output of get_FCs.R*

Description

This function reads the results of the function getFCs and assigns a regulatory mode to each gene based on the Fold Change (FC) for RNA-Seq Counts, Ribo-Seq Counts, or TE. It is intended for internal use and its output is the output of get_FCs.R

Usage

```
assign_Regmode(res_df)
```

Arguments

`res_df`	A dataframe, output of the function getFCs

Value

A dataframe, with two extra columns with info on the Regulatory Mode

Examples

```
# Internal function, code cannot be run from here.
## Not run:
mockdata <- data.frame(
  Identifier = c("ENSG00000248713", "ENSG00000125780"),
  log2FoldChange = c(-0.69, 2),
  padj = c(0.16, 0.001),
  RIBO_FC = c(0.27, 3),
  RIBO_padj = c(0.45, 0.002),
  RNA_FC = c(1, 0.56),
  RNA_padj = c(0.001, 0.65)
)
result <- assign_Regmode(mockdata)
# Only the head of the result file will be returned
print(head(result))

## End(Not run)
```

---

check_input_df *Check if a data frame is not empty (no rows, no columns, or NULL)*

Description

Check if a data frame is not empty (no rows, no columns, or NULL)

Usage

```
check_input_df(df1)
```

Arguments

df1	A dataframe.

Value

NULL. Throws an error if df is empty or NULL.

Examples

```
#' # Internal function, code cannot be run from here.
## Not run:
rna_df <- data.frame(Sample1 = c(1,2,3), Sample2 = c(1,2,3), Sample3 = c(1,2,3))
check_input_df(rna_df)

## End(Not run)
```

check_integer_values *Check and Convert Dataframe Columns to Integers*

Description

This function checks whether each numeric column in the provided dataframe contains only integer values. If a column contains floating-point numbers, it rounds the values and converts them to integers.

Usage

```
check_integer_values(df)
```

Arguments

df	A dataframe to be checked and potentially modified.

Value

A dataframe with numeric columns as integers.

Examples

```
# Internal function, code cannot be run from here.
## Not run:
rna_file <- system.file("extdata", "rna_counts.tsv",
package = "terapadog")
rna_df <- read.table(rna_file, header = TRUE, sep = "\t")
check_integer_values(rna_df)

## End(Not run)
```

check_matching_colnames *Check if two data frames have the same column names*

Description

This function verifies that two data frames (RIBO and RNA) contain the same set of column names, regardless of their order. This is important for how terapadog handles these matching samples during shuffles.

Usage

```
check_matching_colnames(df1, df2)
```

Arguments

df1	A dataframe.
df2	A dataframe.

Value

NULL. Throws an error if column names do not match.

Examples

```
#' # Internal function, code cannot be run from here.
## Not run:
rna_file <- system.file("extdata", "rna_counts.tsv",
package = "terapadog")
ribo_file <- system.file("extdata", "ribo_counts.tsv",
package = "terapadog")
rna_df <- read.table(rna_file, header = TRUE, sep = "\t")
ribo_df <- read.table(ribo_file, header = TRUE, sep = "\t")
check_matching_colnames(rna_df, ribo_df)

## End(Not run)
```

---

check_value_range *Check the Range of Values in a Dataframe*

Description

This function checks if the range (max - min) of numeric values in the dataframe is below 1. If so, this is an indication the data was scaled or anyway processed in a way that makes it not suitable for DeltaTE's analysis within terapadog. The input counts must be raw counts.

Usage

```
check_value_range(df)
```

Arguments

df	A dataframe containing numeric values.

Value

NULL. Throws an error if the range does not exceed the threshold.

Examples

```
#' # Internal function, code cannot be run from here.
## Not run:
rna_file <- system.file("extdata", "rna_counts.tsv",
package = "terapadog")
rna_df <- read.table(rna_file, header = TRUE, sep = "\t")
check_value_range(rna_df)

## End(Not run)
```

`detect_separator` *Detect the separator used in a file*

Description

This function determines whether the file uses a comma or tab as a separator based on the file extension (.csv or .tsv).

Usage

```
detect_separator(path)
```

Arguments

`path`	A string representing the file path.

Value

A string representing the separator (comma or tab).

Examples

```
# Internal function, code cannot be run from here.
## Not run:
result <- detect_separator("this/function/is/quite/easy.csv")
print(result)

## End(Not run)
```

`generate_permutation_matrix` *Generate a Permutation Matrix for Group Assignments*

Description

This function generates a permutation matrix based on given sample groups, handling paired and unpaired designs. It also returns the group in the experiment with the least and most samples. Finally, it recomputes the necessary iterations, based on the permutations done.

Usage

```
generate_permutation_matrix(G, NI, paired, block, verbose)
```

Arguments

`G`	A factor vector indicating the group assignment for each sample.
`NI`	Integer. The maximum number of permutations allowed.
`paired`	Logical. If TRUE, the function accounts for paired samples.
`block`	A factor vector specifying the blocking variable for paired samples.
`verbose`	Logical. If TRUE, the function prints the number of permutations used.

Value

A list containing:

`combidx`	A matrix where each column is a permutation of sample indices.
`NI`	The number of permutations used.
`bigG`	A vector of group labels excluding the smallest group.
`minG`	The group label with the smallest number of samples.

Examples

```
G <- factor(rep(c("Wt", "Mut"), each = 3))
block <- factor(rep(c("Pat1", "Pat2", "Pat3", "Pat1", "Pat2", "Pat3")))
result <- terapadog:::generate_permutation_matrix(G, NI = 1000, paired = TRUE, block = block, verbose = FALSE)
```

get_FCs *This function execute the Differential Translation Analysis on its own using DeltaTE. The output is a dataframe with the FC in mRNA counts, RIBO counts or TE between the conditions in exam.*

Description

This function execute the Differential Translation Analysis on its own using DeltaTE. The output is a dataframe with the FC in mRNA counts, RIBO counts or TE between the conditions in exam.

Usage

```
get_FCs(expression.data, exp_de, paired = FALSE)
```

Arguments

expression.data	A matrix containing the counts from RNA and RIBO samples.
exp_de	A dataframe containing information regarding the samples. It has number of rows equal to the columns of esetm.
paired	Logical. Default is false. Set to TRUE if the experiment has paired samples in its design.

Value

A dataframe with the results of a Differential Translation Analysis. Each gene's change in RNA counts, RFP(/RIBO) counts and TE are reported, along with the relative adjusted p-values. The RegModes are also reported.

Examples

```
# The execution of a DTA can take some time and computational resources.
# Henceforth, the following code is not supposed to be run from the man page.
# Load the data
rna_file <- system.file("extdata", "rna_counts.tsv",
package = "terapadog")
ribo_file <- system.file("extdata", "ribo_counts.tsv",
package = "terapadog")
sample_file <- system.file("extdata", "sample_info.tsv",
package = "terapadog")
 # Use the paths to load the files.
prepared_data <- prepareTerapadogData(rna_file, ribo_file,
sample_file, "1", "2")
# Unpacks the expression.data and exp_de from the output
expression.data <- prepared_data$expression.data
exp_de <- prepared_data$exp_de
result <- get_FCs(expression.data, exp_de)
```

gsScoreFun *Compute Gene Set Scores after Translational Efficiency Analysis*

Description

This internal function performs gene set scoring by applying a modified version of the PADOG algorithm to genes undergoing changes in translational efficiency. As reported by the results of the DeltaTE package

Usage

```
gsScoreFun(
  G,
  block,
  ite,
  exp_de,
  esetm,
  paired,
  grouped_indexes,
  minG,
  bigG,
  gf,
  combidx,
  deINgs,
  gslistINesetm
)
```

Arguments

G	A factor vector representing the group for each sample.
block	A factor indicating paired samples.
ite	Integer, indicating the current iteration number.
exp_de	A dataframe containing metadata for each sample, including grouping information.
esetm	A matrix containing RNA and RIBO count data, where rows correspond to genes and columns to samples.
paired	Logical, indicating whether the study design is paired.
grouped_indexes	A dataframe mapping RNA and RIBO samples to their corresponding indices.
minG	A character value representing the smallest group.
bigG	A character vector representing the larger group.
gf	A vector containing gene weighting factors.
combidx	A matrix storing all possible permutations for group shuffling.
deINgs	A vector with genes that are both in the gene sets, as well in esetm.
gslistINesetm	A list of indices mapping genes existing both in gslist and esetm.

Details

The function performs differential translational analysis using DESeq2 and calculates gene set scores based on adjusted p-values from the differential analysis.

Value

A named matrix containing two rows:

MeanAdjP Mean adjusted p-value for each gene set.

WeightedAdjP Weighted adjusted p-value incorporating gene weighting factors.

`id_converter` *Convert the human gene identifier (hgnc_symbol or ensembl_gene_id) to entrezgene_id format for the analysis.*

Description

Convert the human gene identifier (hgnc_symbol or ensembl_gene_id) to entrezgene_id format for the analysis.

Usage

```
id_converter(esetm, id_type, save_report = FALSE, outdir = tempdir())
```

Arguments

`esetm`	A matrix with the gene count values and whose rownames are the gene Ids (gene symbol or ensembl gene ID).
`id_type`	A string representing the type of ID given as input. Must be either hgnc_symbol or ensembl_gene_id.
`save_report`	A boolean. By default, the duplicates report is not saved locally.
`outdir`	Path to a directory where to save the report. If none is given, a temporary directory will be chosen. No report will be creatted if save_report is set to FALSE.

Value

A matrix with gene IDs in the entrezgene_id format. Also provides a report on the duplicated mappings (conversion_report.txt) in the working dir.

Examples

```
# To showcase thisl internal function, a small example is made.
gene_ids <- c('ENSG00000103197', 'ENSG00000008710', 'ENSG00000167964'
, 'ENSG00000167964')
esetm <- matrix(c(
2.5, 3.1, 5.2, 0.1,
4.1, 2.9, 6.3, 0.5,
1.5, 3.7, 4.8, 0.1), nrow = 4, byrow = FALSE)
rownames(esetm) <- gene_ids
colnames(esetm) <- c("Sample 1", "Sample 2", "Sample 3")
# Call the function
esetm <- id_converter(esetm, "ensembl_gene_id")
print(head(esetm))
```

`plotDTA` *This function will plot an interactive html plot of the results of get_FCs.R That is to say, a plot of the genes undergoing translational regulation, coloured by RegMode. Genes whose RegMode was Undeterminable or Undetermined are omitted.*

Description

This function will plot an interactive html plot of the results of get_FCs.R That is to say, a plot of the genes undergoing translational regulation, coloured by RegMode. Genes whose RegMode was Undeterminable or Undetermined are omitted.

Usage

```
plotDTA(
  FC_results,
  save_plot = FALSE,
  path = file.path(tempdir(), "plot.html")
)
```

Arguments

`FC_results`	A dataframe containing the counts from RNA and RIBO samples.
`save_plot`	Boolean. Default is FALSE. If TRUE, will save plot to a specified directory or a temporary one if none are given.
`path`	A string, pointing to where to save the html plot. If none is given, the plot will be saved to a temporary directory. This parameter will be ignored if save_plot is set to FALSE.

Value

An interactive html plot.

Examples

```
# Creates a mock dataframe for this demonstration
df <- data.frame(
  Identifier = c("Gene A", "Gene B", "Gene C", "Gene D"),
  RegMode = c("Buffered", "Exclusive", "Undeterminable", "No Change"),
  RNA_FC = c(-0.40, -0.5, NA, 0.01),
  RIBO_FC = c(0.19, -0.3, 0.8, -0.02)
)
result <- plotDTA(df)
```

`prepareGeneSets` *This function retrieves KEGG-based gene sets, if 'gslist' is set to "KEGGRESTpathway". oOtherwise, it uses the user-supplied list of gene sets, after verifying that enough genes overlap with the provided expression data. It also computes a weighting factor 'gf' to down-weight genes that appear in many sets.*

Description

This function retrieves KEGG-based gene sets, if 'gslist' is set to "KEGGRESTpathway". oOtherwise, it uses the user-supplied list of gene sets, after verifying that enough genes overlap with the provided expression data. It also computes a weighting factor 'gf' to down-weight genes that appear in many sets.

Usage

```
prepareGeneSets(
  esetm,
  gslist = "KEGGRESTpathway",
  organism = "hsa",
  gs.names = NULL,
  Nmin = 3,
  verbose = FALSE
)
```

Arguments

`esetm`	A matrix of expression data, whose rownames are gene IDs. Required for overlap checks with the sets.
`gslist`	Either a user-supplied list of pathways, or the default string "KEGGRESTpathway", indicating that KEGG gene sets should be retrieved via the KEGGREST package.
`organism`	A three-letter string giving the organism code for KEGG. e.g. "hsa" for human. Defaults is "hsa".
`gs.names`	Character vector with the names of the gene sets. If specified, must have the same length as gslist.
`Nmin`	The minimum size of gene sets to be included in the analysis.
`verbose`	Logical. If true, shows number of gene sets being worked upon.

Value

A list with three elements: gslist, gs.names, gf. gslist is the list of pathways/sets. gs.names is the names of each gene pathway/set. gf is the vector of gene weights, computed from how frequents genes are across sets.

Examples

```
# Suppose 'esetm' is a matrix of counts, rownames are gene IDs.
esetm <- matrix(
data = c(10, 3, 25, 12, 8, 14, 7, 1, 5, 7, 2, 10, 11, 2, 27, 11, 8, 10, 6, 2, 4, 2),
nrow =11,
ncol = 2,
dimnames = list (
c("GeneA","GeneB","GeneC", "GeneD","GeneE", "GeneF", "GeneV", "GeneX","GeneY","GeneZ", "GeneW"),
c("Sample1", "Sample2"))
)
# If we have our own sets in a list:
mySets <- list(
  Path1 = c("GeneA","GeneB","GeneC"),
  Path2 = c("GeneX","GeneY","GeneZ", "GeneW"),
  Path3 = c("GeneA", "GeneZ", "GeneD", "GeneV"),
  Path4 = c("GeneA","GeneB","GeneY", "GeneE", "GeneF"),
  Path5 = c("GeneC", "GeneV")
)
gs.names <- c(Path1 = "Pathway_1",
Path2 = "Pathway_2",
Path3 = "Pathway_3",
Path4 = "Pathway_4",
Path5 = "Anne_PHathway")

geneSets <- terapadog:::prepareGeneSets(esetm, gslist = mySets, gs.names = gs.names)
```

`prepareTerapadogData` *Prepare Data by Loading and Validating RNA, RIBO Counts, and Metadata. This function reads RNA and RIBO count files, checks input data validity and merges them into a single numerical matrix (expression.data). It also prepares the metatadata needed by padog (exp_de).*

Description

Prepare Data by Loading and Validating RNA, RIBO Counts, and Metadata. This function reads RNA and RIBO count files, checks input data validity and merges them into a single numerical matrix (expression.data). It also prepares the metatadata needed by padog (exp_de).

Usage

```
prepareTerapadogData(
  path_to_RNA_counts,
```

```
  path_to_RIBO_counts,
  path_to_metadata,
  analysis.group.1,
  analysis.group.2
)
```

Arguments

`path_to_RNA_counts`
: A string representing the file path to the RNA counts data file (.csv or .tsv).

`path_to_RIBO_counts`
: A string representing the file path to the RIBO counts data file (.csv or .tsv).

`path_to_metadata`
: The file path to the metadata file (.csv or .tsv).

`analysis.group.1`
: A string specifying the baseline group in the experiment (e.g., WT, control, etc.).

`analysis.group.2`
: A string specifying the target group to compare against the baseline (e.g., mutant, disease, treatment, etc.).

Value

A list containing two data frames: expression.data and exp_de.

Examples

```
# Data is also available in the "extdata" folder of this package.
# The path will be automatically generated for the purpose of this example
rna_file <- system.file("extdata", "rna_counts.tsv",
package = "terapadog")
ribo_file <- system.file("extdata", "ribo_counts.tsv",
package = "terapadog")
sample_file <- system.file("extdata", "sample_info.tsv",
 package = "terapadog")
 # Use the paths to load the files.
prepared_data <- prepareTerapadogData(rna_file, ribo_file,
sample_file, "1", "2")
# Unpacks the expression.data and exp_de from the output
expression.data <- prepared_data$expression.data
exp_de <- prepared_data$exp_de
# For sake of brevity, only the data frame's head will be printed out
print(head(expression.data))
print(head(exp_de))
```

terapadog *Performs the main Gene Set Enrichement Analysis, by applying a modified version of the PADOG algorithm to genes undergoing changes in TE.*

Description

Performs the main Gene Set Enrichement Analysis, by applying a modified version of the PADOG algorithm to genes undergoing changes in TE.

Usage

```
terapadog(
  esetm = NULL,
  exp_de = NULL,
  paired = FALSE,
  gslist = "KEGGRESTpathway",
  organism = "hsa",
  gs.names = NULL,
  NI = 1000,
  Nmin = 3,
  verbose = TRUE
)
```

Arguments

esetm	A matrix containing the counts from RNA and RIBO samples. Rownames must be ensembl GENEIDs, while column names must be sample names. Refer to prepareTerapadogData.R to prepare input data.
exp_de	A dataframe containing information regarding the samples. It has number of rows equal to the columns of esetm. It has a formatted vocabulary, but can be obtained by running prepareTerapadogData.R.
paired	Logical. Specify is the study has a paired design or not. If it does, be sure that the pairs are specified in the "Block" column of the exp_de dataframe.
gslist	A list of named character vectors. Each vector is named after a KEGG pathway ID and each element within the vector is an ENSEMBL gene ID for a gene part of said pathway.
organism	A three letter string giving the name of the organism supported by the "KEGGREST" package.
gs.names	Character vector with the names of the gene sets. If specified, must have the same length as gslist.
NI	Number of iterations allowed to determine the gene set score significance p-values.
Nmin	The minimum size of gene sets to be included in the analysis.
verbose	Logical. If true, shows number of iterations done.

Value

A dataframe with the PADOG score for each pathway in exam.

Index

∗ **internal**
assign_Regmode, 2
check_input_df, 3
check_integer_values, 4
check_matching_colnames, 4
check_value_range, 5
detect_separator, 6
generate_permutation_matrix, 7
gsScoreFun, 9
prepareGeneSets, 12

assign_Regmode, 2

check_input_df, 3
check_integer_values, 4
check_matching_colnames, 4
check_value_range, 5

detect_separator, 6

generate_permutation_matrix, 7
get_FCs, 8
gsScoreFun, 9

id_converter, 10

plotDTA, 11
prepareGeneSets, 12
prepareTerapadogData, 13

terapadog, 15