Package: mclink (via r-universe)

June 10, 2026

Title Metabolic Pathway Completeness and Abundance Calculation

Version 1.1.1

Description Provides tools for analyzing metabolic pathway completeness, abundance, and transcripts using KEGG Orthology (KO) data from (meta)genomic and (meta)transcriptomic studies. Supports both completeness (presence/absence) and abundance-weighted analyses. Includes built-in KEGG reference datasets. For more details see Li et al. (2023) <doi:10.1038/s41467-023-42193-7>.

URL `https://github.com/LiuyangLee/mclink`

BugReports `https://github.com/LiuyangLee/mclink/issues`

License GPL-3

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Imports data.table (>= 1.17.0), dplyr (>= 1.1.4), stringr (>= 1.5.1), tibble (>= 3.2.1), utils

Depends R (>= 3.5)

LazyData false

Config/check/use_internal_tz TRUE

NeedsCompilation no

Author Liuyang Li [aut, cre] (<`https://orcid.org/0000-0001-6004-9437`>)

Maintainer Liuyang Li <`cyanobacteria@yeah.net`>

Config/pak/sysreqs libicu-dev

Repository https://liuyanglee.r-universe.dev

Date/Publication 2025-10-25 14:56:06 UTC

RemoteUrl https://github.com/liuyanglee/mclink

RemoteRef HEAD

RemoteSha 4398570c48895ad0070b8f06078f538a0f695ca1

Contents

add_rows_if_not_exists 2
ata_cal 3
convert_abundance_to_presence 4
create_sub_module_sample 4
escape_special_chars 5
extract_inner_brackets 6
group_ko_by_module 6
KO_pathway_ref 7
KO_Sample_wide 7
mclink 8
merge_module_name 10
process_all_modules 10
process_all_pathways 11
process_module_brackets 12
process_module_definition 13
process_module_loop_comma 14
process_module_loop_plu_comma 15
process_module_loop_plus 16
process_module_step 17
process_module_structure 18
process_step_comma 18
process_step_direct 19
process_step_plus 20
process_step_space 21
read_and_process_KO_table 21
read_and_process_pathway_infor 22
remove_outer_brackets 23

Index **24**

`add_rows_if_not_exists` *Add Missing Rows to a Data Frame*

Description

This function checks if specified rows exist in a data frame, and if not, adds them with all values set to 0. Useful for ensuring consistent KO representation across samples.

Usage

```
add_rows_if_not_exists(
  module_abundance,
  add_rows = c("K14126", "K14128", "K14127")
)
```

Arguments

`module_abundance`	A data frame with row names representing KO identifiers (e.g., K numbers) and numeric abundance values
`add_rows`	A character vector of row names (KO identifiers) that should be present in the output. Defaults to c("K14126","K14128","K14127")

Value

The original data frame with additional rows (if any were missing) where all values are set to 0. Row and column names are preserved.

`ata_cal` *Calculate Log2 Ratio of Sample Values to Row Means*

Description

This function calculates the log2 ratio of each value in a data frame to its corresponding row mean. Zero values are preserved as zeros in the output.

Usage

```
ata_cal(data = KO_Sample_table)
```

Arguments

`data`	A data frame or matrix containing numerical values to be processed. Rows represent features (e.g., KO terms) and columns represent samples. Default is KO_Sample_table.

Value

A data frame of the same dimensions as input, where each value is: - log2(sample_value/row_mean) when both sample_value and row_mean are non-zero - 0 when either sample_value or row_mean is zero The Mean_RA column used for calculations is removed from the output.

`convert_abundance_to_presence` *Convert Abundance Values to Presence/Absence Indicators*

Description

Transforms a numeric abundance matrix into a binary presence/absence matrix, where 1 indicates presence (abundance > 0) and 0 indicates absence. Preserves row names as Orthology_Entry column in the output.

Usage

```
convert_abundance_to_presence(module_abundance)
```

Arguments

`module_abundance`	A data frame containing KO abundance data, must include: - Rows named by Orthology_Entry (KO identifiers) - Numeric columns representing sample abundances - An Orthology_Entry column

Value

A data frame with: - Binary values (1 = present, 0 = absent) for each sample - Original row names preserved in Orthology_Entry column - Same dimensions as input (excluding the Orthology_Entry column)

`create_sub_module_sample` *Create and Export Pathway-Specific Module Sample Files*

Description

Processes pathway information and module sample data to create and export individual pathway-specific files containing scaled module data.

Usage

```
create_sub_module_sample(
  pathway_infor,
  Module_Sample_scale,
  out_DIR_Module_Sample_by_pathway,
  plus_scale_method,
  comma_scale_method
)
```

Arguments

`pathway_infor`	Data frame containing pathway information, see examples.
`Module_Sample_scale`	Data frame containing scaled module sample data with module names as row names
`out_DIR_Module_Sample_by_pathway`	Character string specifying output directory
`plus_scale_method`	Scaling method for plus-separated KOs ("mean", "min", or "max")
`comma_scale_method`	Scaling method for comma-separated KOs ("sum" or "max")

Value

None (writes files to disk)

`escape_special_chars` *Escape Special Characters in a String*

Description

Escapes all specified special characters in a string by adding a backslash before them. This is particularly useful for preparing strings for use in regular expressions or other contexts where special characters need to be treated as literals.

Usage

```
escape_special_chars(s)
```

Arguments

`s`	A character string to be processed

Value

A new string with all specified special characters escaped with backslashes

---

`extract_inner_brackets` *Extract Innermost Parentheses Content*

Description

Extracts all text segments enclosed in the innermost level of parentheses from a string. This is useful for parsing hierarchical or nested parenthetical expressions.

Usage

```
extract_inner_brackets(s)
```

Arguments

`s`	A character string to process (can contain multiple parenthetical groups)

Value

A character vector containing all innermost parenthesized segments Returns empty character vector if no matches found

---

`group_ko_by_module` *Group KO Abundance Data by Module*

Description

Processes KO abundance data to group by metabolic modules, converting presence/absence data into module-level KO lists. Handles missing KOs and maintains sample-specific KO profiles.

Usage

```
group_ko_by_module(pathway_infor, Sample_KO_abundance)
```

Arguments

`pathway_infor`	Data frame containing pathway information, see examples.
`Sample_KO_abundance`	Data frame of KO abundances with: - Rows as KO identifiers - Columns as samples - Orthology_Entry column

Value

A data frame where: - Rows are module names - Columns are samples - Cell values are space-separated lists of present KOs - Empty strings for modules with no detected KOs

---

`KO_pathway_ref` *KEGG Orthology (KO) Pathway Information Dataset*

Description

A comprehensive dataset mapping KEGG Orthology (KO) entries to metabolic pathways, including module hierarchy, definitions, and enzyme annotations.

Usage

```
KO_pathway_ref
```

Format

A data frame with 3846 rows (KO entries) and 10 variables:

Orthology_Entry Character. KEGG Orthology ID (e.g., "K00844").

Module_Type Character. Type of metabolic module (e.g., "Pathway modules").

Level_2 Character. Broad metabolic category (e.g., "Carbohydrate metabolism").

Level_3 Character. Specific metabolic subcategory (e.g., "Central carbohydrate metabolism").

Module_Entry Character. KEGG Module ID (e.g., "M00001").

Module_Name Character. Full name of the metabolic module (e.g., "Glycolysis (Embden-Meyerhof pathway), glucose => pyruvate").

Definition Character. KO composition of the module, truncated in display (e.g., "(K00844,K12407,...)").

Orthology_Symbol Character. Short symbol for the KO (e.g., "HK" for hexokinase).

Orthology_Name Character. Full enzyme name with EC number (e.g., "hexokinase `[EC:2.7.1.1]`").

KO_Symbol Character. Combined KO ID and symbol (e.g., "K00844; HK").

`KO_Sample_wide` *KEGG Orthology (KO) Abundance/Presence Across Microbial Samples or Genomes*

Description

A test dataset (wide-format) showing the relative abundance of KEGG Orthology (KO) entries across multiple microbial samples. Values represent normalized abundance metrics (e.g., TPM, RPKM, relative percentage, presence/absence).

Usage

```
KO_Sample_wide
```

Format

A data frame with 2495 rows (KO entries) and 5 variables:

KO Character. KEGG Orthology ID (e.g., "K00001").

Marinobacter salarius Numeric. Abundance in Genome "Marinobacter salarius".

Pseudooceanicola nanhaiensis Numeric. Abundance in Genome "Pseudooceanicola nanhaiensis".

Alteromonas australica Numeric. Abundance in Genome "Alteromonas australica".

Henriciella pelagia Numeric. Abundance in Genome "Henriciella pelagia".

`mclink` *Metabolic Pathway Coverage Analysis*

Description

Analyzes metabolic pathway completeness/abundance from (meta)genome KO presence/abundance data. Can use either built-in KEGG datasets or user-provided data frames. Output includes pathway coverage metrics and detected KOs in each pathway/module.

The distill analysis of KEGG Module coverage is calculated based on the abundance or presence of KOs in a given module, as per the KEGG Module Definition. In detail, the coverage of a KEGG Module is determined by first dividing a set of KOs into distinct steps. The coverage for each step is then calculated separately, and summarized as the coverage of this KEGG Module. When calculating coverage, spaces or plus signs connecting KO numbers are interpreted as AND operators, while commas are interpreted as OR operators. For instance, to calculate the completeness for the module M00020: “K00058 K00831 (K01079, K02203, K22305)”:

1. Convert the abundance table of KOs into a 0-1 matrix.
2. Consider K00058 as step 1, K00831 as step 2, and (K01079, K02203, K22305) as step 3.
3. Calculate the `maximum` (or `minimum`, `mean`) value of step 3 (K01079, K02203, K22305). If presence of any KO indicates completeness, calculate the `maximum` value; If all KOs must be resent for completeness, calculate the `minimum` value. For a moderate approach, calculate the `mean` value.
4. Use this value along with the values of step 1 and step 2 to calculate the average value, representing the completeness of the module M00020.

Usage

```
mclink(
  ref = NULL,
  data = NULL,
  table_feature = "completeness",
  plus_scale_method = "mean",
  comma_scale_method = "max",
  out_dir = NULL,
  split_by_pathway = FALSE
)
```

Arguments

`ref`	Pathway information data frame. When `NULL` (default), uses the built-in `KO_pathway_ref` dataset. Must contain the same columns as the built-in dataset if providing custom data.
`data`	Sample KO abundance data frame. When `NULL` (default), uses the built-in `KO_Sample_wide` dataset.
`table_feature`	Analysis type, either: • "completeness" (binary presence/absence, default) • "abundance" (weighted by KO abundance)
`plus_scale_method`	Scaling method for plus-separated KOs (K1+K2+...) (Enzyme subunits/Protein complexes): • "mean" - Moderate approach (default), calculates average value of all components • "min" - Rigorous/conservative estimate, uses lowest value (all components must be present) • "max" - Liberal estimate, uses highest value (any component indicates completeness)
`comma_scale_method`	Scaling method for comma-separated KOs (K1,K2,...) (Gene isoforms/Alternative pathways): • "max" - For completeness analysis (default), any component indicates functional pathway • "sum" - For abundance analysis, sums all functionally equivalent variants
`out_dir`	Output directory path. If `NULL` (default), results are only returned as R objects without writing files.
`split_by_pathway`	Logical. If `TRUE`, splits results by pathway/module. Requires non-NULL `out_dir`. Default: `FALSE`.

Value

A list containing:

- coverage - Data frame with pathway coverage metrics
- detected_KOs - List of detected KOs per pathway/module
- log - log of the analysis process

If `out_dir` is specified, results are also written as TSV files.

Examples

```
data(KO_pathway_ref)
data(KO_Sample_wide)
selected_modules <- c("M00176","M00165","M00173","M00374","M00375","M00376","M00377")
KO_pathway_ref_selected <- KO_pathway_ref[KO_pathway_ref$Module_Entry %in% selected_modules, ]
```

```
mc_list =
  mclink(ref = KO_pathway_ref_selected,
         data = KO_Sample_wide,
         table_feature = "completeness",
         plus_scale_method = "min",
         comma_scale_method = "max")
mc_coverage = mc_list[["coverage"]]
mc_detected_KOs = mc_list[["detected_KOs"]]
mc_log = mc_list[["log"]]
print(head(mc_coverage))
```

`merge_module_name` *Merge Module Information with Module Table*

Description

Merges pathway information with a module table to create a sample-by-module matrix with proper module names. Ensures all modules are represented in the output.

Usage

```
merge_module_name(pathway_infor, module_table)
```

Arguments

`pathway_infor`	Data frame containing pathway information, see examples.
`module_table`	Data frame containing module data with: - Module_Entry: Matching module identifiers - Orthology_Entry: KO identifiers - Definition: Module definitions - Sample columns with abundance values

Value

A data frame where: - Rows are module names (from Module_Name) - Columns are samples - All modules from pathway_infor are represented - Original row names are replaced with descriptive module names

---

`process_all_modules` *Process All Modules in Pathway Information*

Description

Processes all metabolic modules in pathway information, handling each module's structure, definition, and bracket components. Aggregates results across all modules.

Usage

```
process_all_modules(
  pathway_infor,
  Sample_KO,
  plus_scale_method,
  comma_scale_method,
  verbose = TRUE
)
```

Arguments

`pathway_infor`	Data frame containing pathway information, see examples.
`Sample_KO`	Data frame containing KO (KEGG Orthology) sample data
`plus_scale_method`	Scaling method for plus-separated KOs ("mean", "min", or "max")
`comma_scale_method`	Scaling method for comma-separated KOs ("sum" or "max")
`verbose`	Logical controlling console output: • `TRUE` (default): Print progress messages • `FALSE`: Silent mode

Value

A list with two components:

- `data`: A data frame of processed results for all modules, with unique rows to avoid duplicates.
- `log`: A character vector of timestamped log messages.

`process_all_pathways` *Process All Pathways Analysis*

Description

Processes module sample data across all pathways with specified scaling methods. Handles different comparison methods and outputs results by pathway.

Usage

```
process_all_pathways(
  pathway_infor,
  Module_Sample,
  out_DIR_Module_Sample_by_pathway,
  compare_method = c("log", "avg", "round"),
  plus_scale_method,
  comma_scale_method
)
```

Arguments

`pathway_infor`	Data frame containing pathway information, see examples.
`Module_Sample`	Data frame of module sample data to process
`out_DIR_Module_Sample_by_pathway`	Output directory for pathway-specific results
`compare_method`	Comparison method to use: "log" (log10 transform), "avg" (average calculation), or "round" (simple rounding)
`plus_scale_method`	Scaling method for plus-separated KOs ("mean", "min", or "max")
`comma_scale_method`	Scaling method for comma-separated KOs ("sum" or "max")

Value

Main outputs are written to: - Combined module file (All_modules.*.tsv) - Pathway-specific files (via create_sub_module_sample)

`process_module_brackets`

Process Module Brackets Recursively

Description

Recursively processes nested brackets in module definitions to calculate pathway completeness or abundance scores. Handles complex pathway structures with multiple nesting levels.

Usage

```
process_module_brackets(
  module_abundance = sub_Sample_KO_pathway,
  module_steps_str = module_steps_str,
  bracket_count = 1,
  step_count = 1,
  module_name = "Module",
  raw_module_steps = module_steps_str,
  plus_scale_method,
  comma_scale_method,
  abundance_log = list()
)
```

Arguments

`module_abundance`	Data frame containing KO abundance data for the module
`module_steps_str`	String representation of module steps/structure

`bracket_count`	Counter for tracking nested bracket levels
`step_count`	Counter for tracking processing steps
`module_name`	Name of the module being processed
`raw_module_steps`	Original unprocessed module steps string
`plus_scale_method`	Scaling method for plus-separated KOs ("mean", "min", or "max")
`comma_scale_method`	Scaling method for comma-separated KOs ("sum" or "max")
`abundance_log`	A character vector of timestamped log messages

Value

A list with two components:

- `data`: Data frame containing processed abundance values with: - Rows for each bracket level and final step - Consistent sample columns as input
- `log`: A character vector of timestamped log messages.

`process_module_definition`

Process Module Definition String

Description

Cleans and processes module definition strings by removing various patterns and formatting elements to extract core KO relationships. Handles special cases like negative KOs and parenthetical expressions.

Usage

```
process_module_definition(sub_Sample_KO_pathway)
```

Arguments

`sub_Sample_KO_pathway`	Data frame containing module definitions in a ’Definition’ column

Value

A list with log and character vector of cleaned module definition strings. The vector contains: - Removed negative KO indicators - Simplified parentheses - Normalized space

---

`process_module_loop_comma`

Process Module Components with Comma Handling

Description

Processes a vector of KOs, applying different handling methods depending on whether they contain commas or not. Useful for processing complex pathway definitions with alternative KOs.

Usage

```
process_module_loop_comma(
  KO_vector,
  module_abundance,
  process_step_comma,
  process_step_direct,
  aggregrate_rowname = "step_1",
  step_count = 1,
  comma_scale_method
)
```

Arguments

`KO_vector`	Character vector of KO identifiers to process
`module_abundance`	Data frame containing KO abundance data
`process_step_comma`	Function to handle comma-separated KOs (alternative forms)
`process_step_direct`	Function to handle individual KOs
`aggregrate_rowname`	Base name for row aggregation (default: 'step_1')
`step_count`	Counter for processing steps (default: 1)
`comma_scale_method`	Scaling method for comma-separated KOs ("sum" or "max")

Value

List containing: - abundance_table: Processed abundance Data frame - step_count: Updated step counter - abundance_log: log

---

`process_module_loop_plu_comma` *Process Module Components with Plus and Comma Handling*

Description

Processes a vector of KOs, applying different handling methods depending on whether they contain plus signs, commas, or both. Handles complex pathway definitions with both required components (plus-separated) and alternative forms (comma-separated).

Usage

```
process_module_loop_plu_comma(
  KO_vector,
  module_abundance,
  process_step_plus,
  process_step_comma,
  process_step_direct,
  aggregrate_rowname = "step_1",
  step_count = 1,
  plus_scale_method,
  comma_scale_method
)
```

Arguments

`KO_vector`	Character vector of KO identifiers to process
`module_abundance`	Data frame containing KO abundance data
`process_step_plus`	Function to handle plus-separated KOs (required components)
`process_step_comma`	Function to handle comma-separated KOs (alternative forms)
`process_step_direct`	Function to handle individual KOs
`aggregrate_rowname`	Base name for row aggregation (default: 'step_1')
`step_count`	Counter for processing steps (default: 1)
`plus_scale_method`	Scaling method for plus-separated KOs ("mean", "min", or "max")
`comma_scale_method`	Scaling method for comma-separated KOs ("sum" or "max")

Value

List containing: - abundance_table: Processed abundance values - step_count: Updated step counter - abundance_log: log

---

`process_module_loop_plus`

Process Module Components with Plus Sign Handling

Description

Processes a vector of KOs, applying different handling methods depending on whether they contain plus signs or not. Handles pathway definitions with required components (plus-separated KOs representing complex subunits).

Usage

```
process_module_loop_plus(
  KO_vector,
  module_abundance,
  process_step_plus,
  process_step_direct,
  aggregrate_rowname = "step_1",
  step_count = 1,
  plus_scale_method
)
```

Arguments

`KO_vector`	Character vector of KO identifiers to process
`module_abundance`	Data frame containing KO abundance data
`process_step_plus`	Function to handle plus-separated KOs (required components)
`process_step_direct`	Function to handle individual KOs
`aggregrate_rowname`	Base name for row aggregation (default: 'step_1')
`step_count`	Counter for processing steps (default: 1)
`plus_scale_method`	Scaling method for plus-separated KOs ("mean", "min", or "max")

Value

List containing: - abundance_table: Processed abundance values - step_count: Updated step counter - abundance_log: log

---

`process_module_step` *Process Module Steps with Complex KO String Handling*

Description

Processes a KO string containing various combinations of KOs separated by different operators (commas, plus signs, or spaces). Handles complex pathway definitions with multiple types of relationships between KOs.

Usage

```
process_module_step(
  module_abundance,
  KO_string = "K03388,K03389+K03390+K14083,K14126+K14127,K14128",
  aggregrate_rowname = "bracket_1",
  step_count = 1,
  plus_scale_method,
  comma_scale_method
)
```

Arguments

`module_abundance`	Data frame containing KO abundance data
`KO_string`	String representation of KO relationships (default: "K03388,K03389+K03390+K14083,K14126+K14127
`aggregrate_rowname`	Base name for row aggregation (default: 'bracket_1')
`step_count`	Counter for processing steps (default: 1)
`plus_scale_method`	Scaling method for plus-separated KOs ("mean", "min", or "max")
`comma_scale_method`	Scaling method for comma-separated KOs ("sum" or "max")

Value

List containing: - abundance_table: Processed abundance values - step_count: Updated step counter - abundance_log: log

---

`process_module_structure` *Process Module Structure Data*

Description

Filters and merges pathway information with sample KO data for a specific module. Returns a merged dataframe containing KO abundance data and pathway definitions.

Usage

```
process_module_structure(pathway_infor, Sample_KO, module)
```

Arguments

`pathway_infor`	Data frame containing pathway information, see examples.
`Sample_KO`	Dataframe containing KO abundance data with KO IDs as row names
`module`	Character string of the module ID to process (e.g. "M00563")

Value

A merged dataframe containing: - KO abundance data for the specified module - Corresponding pathway information - Empty dataframe if no matching KOs found

---

`process_step_comma` *Process Comma-Separated KOs with Specified Scaling Method*

Description

Handles comma-separated KOs by applying the specified scaling method (sum or max). Processes multiple KOs separated by commas and aggregates them into a single row.

Usage

```
process_step_comma(
  module_abundance,
  KOs = c("K14126,K14127,K14128"),
  aggregrate_rowname,
  step_count = 1,
  comma_scale_method
)
```

Arguments

`module_abundance`	Data frame containing KO abundance data with required columns: Orthology_Entry, Module_Entry, Definition
`KOs`	Character vector of comma-separated KO IDs (default: "K14126,K14127,K14128")
`aggregrate_rowname`	Base name for row aggregation (default: 'step_1')
`step_count`	Processing step counter (default: 1)
`comma_scale_method`	Scaling method for comma-separated KOs ("sum" or "max")

Value

List containing: - abundance_table: Processed data with aggregated values - step_count: Updated step counter - abundance_log: log

---

`process_step_direct` *Direct KO Processing Without Special Handling*

Description

Processes KO abundances directly without any special scaling or aggregation. Simply extracts the specified KOs from the abundance table while maintaining the original module metadata.

Usage

```
process_step_direct(module_abundance, KOs = c("K14126", "K14128", "K14127"))
```

Arguments

`module_abundance`	Data frame containing KO abundance data with required columns: Orthology_Entry, Module_Entry, Definition
`KOs`	Character vector of KO IDs to extract (default: c("K14126","K14128","K14127"))

Value

List containing: - abundance_table: A subset of the input data frame containing only the specified KOs, with original module metadata preserved - abundance_log: log

---

`process_step_plus` *Process Plus-Separated KOs with Specified Scaling Method*

Description

Handles plus-separated KOs by applying the specified scaling method (mean, min, or max). Processes multiple KOs separated by plus signs and aggregates them into a single row. Note: For mean calculation, uses the sum of existing KO abundances divided by total number of KOs (including those with zero abundance in all samples).

Usage

```
process_step_plus(
  module_abundance,
  KOs = c("K14126+K14127+K14128"),
  aggregrate_rowname,
  step_count = 1,
  plus_scale_method
)
```

Arguments

`module_abundance`	Data frame containing KO abundance data with required columns: Orthology_Entry, Module_Entry, Definition
`KOs`	Character string of plus-separated KO IDs (default: "K14126+K14127+K14128")
`aggregrate_rowname`	Base name for row aggregation (default: 'step_1')
`step_count`	Processing step counter (default: 1)
`plus_scale_method`	Scaling method for plus-separated KOs ("mean", "min", or "max")

Value

List containing: - abundance_table: Processed data with aggregated values - step_count: Updated step counter - abundance_log: log

---

`process_step_space` *Process Space-Separated KOs with Mean Calculation*

Description

Handles space-separated KOs by calculating the mean abundance across all specified KOs. Processes multiple KOs separated by spaces and aggregates them into a single row. Note: For mean calculation, uses the sum of KO abundances divided by total number of KOs, including those with zero abundance in all samples.

Usage

```
process_step_space(
  module_abundance,
  KOs = c("K14126 K14127 K14128"),
  aggregrate_rowname,
  step_count = 1
)
```

Arguments

`module_abundance`	Data frame containing KO abundance data with required columns: Orthology_Entry, Module_Entry, Definition
`KOs`	Character string of space-separated KO IDs (default: "K14126 K14127 K14128")
`aggregrate_rowname`	Base name for row aggregation (default: 'step_1')
`step_count`	Processing step counter (default: 1)

Value

List containing: - abundance_table: Processed data with mean values - step_count: Updated step counter - abundance_log: log

---

`read_and_process_KO_table` *Read and Process KO Sample Table with Pathway Information*

Description

This function reads a KO sample table (wide format), validates its contents against pathway information, and returns a filtered table containing only KOs present in both datasets.

Usage

```
read_and_process_KO_table(in_KO_Sample_wide, pathway_infor)
```

Arguments

`in_KO_Sample_wide`	Character string specifying the path to the input KO sample table file. Should be a tab-delimited file with KO identifiers as row names.
`pathway_infor`	Data frame containing pathway information, see examples.

Value

A list with log and a data frame.The data frame contains the filtered sample data, with only rows that match KOs in the pathway information. Includes an added 'Orthology_Entry' column containing the row names.

`read_and_process_pathway_infor`

Read and process Pathway information dataframe

Description

This function reads a tab-delimited file containing KEGG pathway information, performs data validation and cleaning, and returns a processed data frame.

Usage

```
read_and_process_pathway_infor(in_KO_pathway_ref)
```

Arguments

`in_KO_pathway_ref`	Character string specifying the path to the input file. The file should be a tab-delimited text file containing KEGG pathway information, with a header row and at least one column named "Module_Entry".

Value

A list with log and a data frame containing the processed pathway information after removing empty/NA entries. The data frame will have the same columns as the input file. Returns NULL if the file cannot be read.

`remove_outer_brackets` *Remove Outer Parentheses from String*

Description

This function checks if a string is wrapped in outer parentheses and removes them if present.

Usage

```
remove_outer_brackets(s)
```

Arguments

`s`	A character string to be processed

Value

The input string with outer parentheses removed (if they existed), or the original string if no outer parentheses were found.

Index

∗ **datasets**
    KO_pathway_ref, 7
    KO_Sample_wide, 7

add_rows_if_not_exists, 2
ata_cal, 3

convert_abundance_to_presence, 4
create_sub_module_sample, 4

escape_special_chars, 5
extract_inner_brackets, 6

group_ko_by_module, 6

KO_pathway_ref, 7, *9*
KO_Sample_wide, 7, *9*

mclink, 8
merge_module_name, 10

process_all_modules, 10
process_all_pathways, 11
process_module_brackets, 12
process_module_definition, 13
process_module_loop_comma, 14
process_module_loop_plu_comma, 15
process_module_loop_plus, 16
process_module_step, 17
process_module_structure, 18
process_step_comma, 18
process_step_direct, 19
process_step_plus, 20
process_step_space, 21

read_and_process_KO_table, 21
read_and_process_pathway_infor, 22
remove_outer_brackets, 23