Package ‘bulkAnalyseR’

July 22, 2025

Title Interactive Shiny App for Bulk Sequencing Data

Version 1.1.0

Maintainer Ilias Moutsopoulos <im383@cam.ac.uk>

Description Given an expression matrix from a bulk sequencing experiment, pre-processes it and creates a shiny app for interactive data analysis and visualisation. The app contains quality checks, differential expression analysis, volcano and cross plots, enrichment analysis and gene regulatory network inference, and can be customised to contain more panels by the user.

License GPL-2

Encoding UTF-8

URL https://github.com/Core-Bioinformatics/bulkAnalyseR

BugReports https://github.com/Core-Bioinformatics/bulkAnalyseR/issues

RoxygenNote 7.2.3

Depends R (>= 4.0)

Imports ggplot2, shiny, gprofiler2, edgeR, DESeq2, stats, ggrepel, utils, RColorBrewer, ComplexHeatmap, circlize, grid, shinyWidgets, shinyjqui, dplyr, magrittr, ggforce, rlang, glue, preprocessCore, matrixStats, noisyr, tibble, ggnewscale, ggrastr, GENIE3, visNetwork, DT, scales, shinyjs, tidyr, shinyLP, UpSetR, stringr, ggVennDiagram

Suggests rmarkdown, knitr, shinythemes, BiocManager, AnnotationDbi, org.Hs.eg.db, org.Mm.eg.db, readxl, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

NeedsCompilation no

Author Ilias Moutsopoulos [aut, cre], Eleanor Williams [aut, ctb], Irina Mohorianu [aut, ctb]

Repository CRAN

Date/Publication 2022-12-15 12:20:02 UTC

Contents

`calculate_condition_mean_sd_per_gene`
Calculate statistics for each gene of an expression matrix given a grouping

---

Description

This function calculates the mean and standard deviation of the expression of each gene in an expression matrix, grouped by the conditions supplied.

Usage

```
calculate_condition_mean_sd_per_gene(expression.matrix, condition)
```

Arguments

`expression.matrix`	the expression matrix; rows correspond to genes and columns correspond to samples; usually preprocessed by `preprocessExpressionMatrix`; a list (of the same length as modality) can be provided if #' `length(modality)` > 1
`condition`	the condition to group the columns of the expression matrix by; must be a factor of the same length as ncol(expression.matrix)

Value

A tibble in long format, with the mean and standard deviation of each gene in each condition. The standard deviation is increased to the minimum value in the expression matrix (the noise threshold) if it is lower, in order to avoid sensitivity to small changes.

Examples

```
expression.matrix.preproc <- as.matrix(read.csv(
  system.file("extdata", "expression_matrix_preprocessed.csv", package = "bulkAnalyseR"),
   row.names = 1
))[1:500,]

condition <- factor(rep(c("0h", "12h", "36h"), each = 2))
tbl <- calculate_condition_mean_sd_per_gene(expression.matrix.preproc[1:10, ], condition)
tbl
```

`crossPanel` *Generate the cross plot panel of the shiny app*

Description

These are the UI and server components of the cross plot panel of the shiny app. It is generated by including 'Cross' in the panels.default argument of `generateShinyApp`.

Usage

```
crossPanelUI(id, metadata, show = TRUE)

crossPanelServer(id, expression.matrix, metadata, anno)
```

Arguments

`id`	the input slot that will be used to access the value
`metadata`	a data frame containing metadata for the samples contained in the expression.matrix; must contain at minimum two columns: the first column must contain the column names of the expression.matrix, while the last column is assumed to contain the experimental conditions that will be tested for differential expression; a list (of the same length as modality) can be provided if #' `length(modality)` > 1
`show`	whether to show the panel or not; default is TRUE; there for compatibility with specifying panels to show
`expression.matrix`	the expression matrix; rows correspond to genes and columns correspond to samples; usually preprocessed by `preprocessExpressionMatrix`; a list (of the same length as modality) can be provided if #' `length(modality)` > 1
`anno`	annotation data frame containing a match between the row names of the expression.matrix (usually ENSEMBL IDs) and the gene names that should be rendered within the app and in output files; this object is created by `generateShinyApp` using the org.db specified

Value

The UI and Server components of the shiny module, that can be used within the UI and Server definitions of a shiny app.

cross_plot — *Create a cross plot comparing differential expression (DE) results*

Description

This function creates a cross plot visualising the differences in log2(fold-change) between two DE analyses.

Usage

```
cross_plot(
  DEtable1,
  DEtable2,
  DEtable1Subset,
  DEtable2Subset,
  df = NULL,
  lfc.threshold = NULL,
  raster = FALSE,
  mask = FALSE,
  labnames = c("not DE", "DE both", "DE comparison 1", "DE comparison 2"),
  cols.chosen = c("grey", "purple", "dodgerblue", "lightcoral"),
  labels.per.region = 5,
  fix.axis.ratio = TRUE,
  add.guide.lines = TRUE,
  add.labels.custom = FALSE,
  genes.to.label = NULL,
  seed = 0,
  label.force = 1
)
```

Arguments

`DEtable1, DEtable2, DEtable1Subset, DEtable2Subset`	tables of DE results, usually generated by `DEanalysis_edger`; the first two should contain all genes, while the second two should only contain DE genes
`df`	Optionally, pre-computed cross plot table, from cross_plot_prep
`lfc.threshold`	the log2(fold-change) threshold to determine whether a gene is DE
`raster`	whether to rasterize non-DE genes with ggraster to reduce memory usage; particularly useful when saving plots to files
`mask`	whether to hide genes that were not called DE in either comparison; default is FALSE
`labnames, cols.chosen`	the legend labels and colours for the 4 categories of genes ("not DE", "DE both", "DE comparison 1", "DE comparison 2")

`labels.per.region`	how many labels to show in each region of the plot; the plot is split in 8 regions using the axes and major diagonals, and the points closest to the origin in each region are labelled; default is 5, set to 0 for no labels
`fix.axis.ratio`	whether to ensure the x and y axes have the same units, resulting in a square plot; default is TRUE
`add.guide.lines`	whether to add vertical and horizontal guide lines to the plot to highlight the thresholds; default is TRUE
`add.labels.custom`	whether to add labels to user-specified genes; the parameter genes.to.label must also be specified; default is FALSE
`genes.to.label`	a vector of gene names to be labelled in the plot; if names are present those are shown as the labels (but the values are the ones matched - this is to allow custom gene names to be presented)
`seed`	the random seed to be used for reproducibility; only used for ggrepel::geom_label_repel if labels are present
`label.force`	passed to the force argument of ggrepel::geom_label_repel; higher values make labels overlap less (at the cost of them being further away from the points they are labelling)

Value

The cross plot as a ggplot object.

Examples

```
expression.matrix.preproc <- as.matrix(read.csv(
  system.file("extdata", "expression_matrix_preprocessed.csv", package = "bulkAnalyseR"),
  row.names = 1
))[1:500, 1:4]

anno <- AnnotationDbi::select(
  getExportedValue('org.Mm.eg.db', 'org.Mm.eg.db'),
  keys = rownames(expression.matrix.preproc),
  keytype = 'ENSEMBL',
  columns = 'SYMBOL'
) %>%
  dplyr::distinct(ENSEMBL, .keep_all = TRUE) %>%
  dplyr::mutate(NAME = ifelse(is.na(SYMBOL), ENSEMBL, SYMBOL))

edger <- DEanalysis_edger(
  expression.matrix = expression.matrix.preproc,
  condition = rep(c("0h", "12h"), each = 2),
  var1 = "0h",
  var2 = "12h",
  anno = anno
)
deseq <- DEanalysis_edger(
  expression.matrix = expression.matrix.preproc,
  condition = rep(c("0h", "12h"), each = 2),
  var1 = "0h",
  var2 = "12h",
  anno = anno
)
cross_plot(
  DEtable1 = edger,
  DEtable2 = deseq,
  DEtable1Subset = dplyr::filter(edger, abs(log2FC) > 1, pvalAdj < 0.05),
  DEtable2Subset = dplyr::filter(deseq, abs(log2FC) > 1, pvalAdj < 0.05),
  labels.per.region = 0
)
```

`DEanalysis` *Perform differential expression (DE) analysis on an expression matrix*

Description

This function performs DE analysis on an expression using edgeR or DESeq2, given a vector of sample conditions.

Usage

```
DEanalysis_edger(expression.matrix, condition, var1, var2, anno)

DEanalysis_deseq2(expression.matrix, condition, var1, var2, anno)
```

Arguments

`expression.matrix`	the expression matrix; rows correspond to genes and columns correspond to samples; usually preprocessed by `preprocessExpressionMatrix`; a list (of the same length as modality) can be provided if #' `length(modality)` > 1
`condition`	a vector of the same length as the number of columns of expression.matrix, containing the sample conditions; this is usually the last column of the metadata
`var1, var2`	conditions (contained in condition) to perform DE between; note that DESeq2 requires at least two replicates per condition
`anno`	annotation data frame containing a match between the row names of the expression.matrix (usually ENSEMBL IDs) and the gene names that should be rendered within the app and in output files; this object is created by `generateShinyApp` using the org.db specified

Value

A tibble with the differential expression results for all genes. Columns are

- gene_id (usually ENSEMBL ID matching one of the rows of the expression matrix)
- gene_name (name matched through the annotation)

- log2exp (average log2(expression) of the gene across samples)
- log2FC (log2(fold-change) of the gene between conditions)
- pval (p-value of the gene being called DE)
- pvalAdj (adjusted p-value using the Benjamini Hochberg correction)

Examples

```
expression.matrix.preproc <- as.matrix(read.csv(
  system.file("extdata", "expression_matrix_preprocessed.csv", package = "bulkAnalyseR"),
  row.names = 1
))[1:100, 1:4]

anno <- AnnotationDbi::select(
  getExportedValue('org.Mm.eg.db', 'org.Mm.eg.db'),
  keys = rownames(expression.matrix.preproc),
  keytype = 'ENSEMBL',
  columns = 'SYMBOL'
) %>%
  dplyr::distinct(ENSEMBL, .keep_all = TRUE) %>%
  dplyr::mutate(NAME = ifelse(is.na(SYMBOL), ENSEMBL, SYMBOL))

edger <- DEanalysis_edger(
  expression.matrix = expression.matrix.preproc,
  condition = rep(c("0h", "12h"), each = 2),
  var1 = "0h",
  var2 = "12h",
  anno = anno
)
deseq <- DEanalysis_edger(
  expression.matrix = expression.matrix.preproc,
  condition = rep(c("0h", "12h"), each = 2),
  var1 = "0h",
  var2 = "12h",
  anno = anno
)
# DE genes with log2(fold-change) > 1 in both pipelines
intersect(
  dplyr::filter(edger, abs(log2FC) > 1, pvalAdj < 0.05)$gene_name,
  dplyr::filter(deseq, abs(log2FC) > 1, pvalAdj < 0.05)$gene_name
)
```

`DEpanel` *Generate the DE panel of the shiny app*

Description

These are the UI and server components of the DE panel of the shiny app. It is generated by including 'DE' in the panels.default argument of `generateShinyApp`.

Usage

```
DEpanelUI(id, metadata, show = TRUE)

DEpanelServer(id, expression.matrix, metadata, anno)
```

Arguments

`id`	the input slot that will be used to access the value
`metadata`	a data frame containing metadata for the samples contained in the expression.matrix; must contain at minimum two columns: the first column must contain the column names of the expression.matrix, while the last column is assumed to contain the experimental conditions that will be tested for differential expression; a list (of the same length as modality) can be provided if #' `length(modality)` > 1
`show`	whether to show the panel or not; default is TRUE; there for compatibility with specifying panels to show
`expression.matrix`	the expression matrix; rows correspond to genes and columns correspond to samples; usually preprocessed by `preprocessExpressionMatrix`; a list (of the same length as modality) can be provided if #' `length(modality)` > 1
`anno`	annotation data frame containing a match between the row names of the expression.matrix (usually ENSEMBL IDs) and the gene names that should be rendered within the app and in output files; this object is created by `generateShinyApp` using the org.db specified

Value

The UI and Server components of the shiny module, that can be used within the UI and Server definitions of a shiny app.

`DEplotPanel` *Generate the DE plot plot panel of the shiny app*

Description

These are the UI and server components of the DE plot panel of the shiny app. It is generated by including 'DEplot' in the panels.default argument of `generateShinyApp`.

Usage

```
DEplotPanelUI(id, show = TRUE)

DEplotPanelServer(id, DEresults, anno)
```

Arguments

`id`	the input slot that will be used to access the value
`show`	whether to show the panel or not; default is TRUE; there for compatibility with specifying panels to show
`DEresults`	differential expression results output from DEpanelServer; a reactive list with slots 'DEtable' (all genes), 'DEtableSubset' (only DE genes), 'lfcThreshold' and 'pvalThreshold'
`anno`	annotation data frame containing a match between the row names of the expression.matrix (usually ENSEMBL IDs) and the gene names that should be rendered within the app and in output files; this object is created by `generateShinyApp` using the org.db specified

Value

The UI and Server components of the shiny module, that can be used within the UI and Server definitions of a shiny app.

`DEsummaryPanel` *Generate the DE summary panel of the shiny app*

Description

These are the UI and server components of the Heatmap panel of the shiny app. It is generated by including 'DEsummary' in the panels.default argument of `generateShinyApp`.

Usage

```
DEsummaryPanelUI(id, metadata, show = TRUE)

DEsummaryPanelServer(id, expression.matrix, metadata, DEresults, anno)
```

Arguments

`id`	the input slot that will be used to access the value
`metadata`	a data frame containing metadata for the samples contained in the expression.matrix; must contain at minimum two columns: the first column must contain the column names of the expression.matrix, while the last column is assumed to contain the experimental conditions that will be tested for differential expression; a list (of the same length as modality) can be provided if #' `length(modality)` > 1
`show`	whether to show the panel or not; default is TRUE; there for compatibility with specifying panels to show
`expression.matrix`	the expression matrix; rows correspond to genes and columns correspond to samples; usually preprocessed by `preprocessExpressionMatrix`; a list (of the same length as modality) can be provided if #' `length(modality)` > 1

`DEresults`	differential expression results output from DEpanelServer; a reactive list with slots 'DEtable' (all genes), 'DEtableSubset' (only DE genes), 'lfcThreshold' and 'pvalThreshold'
`anno`	annotation data frame containing a match between the row names of the expression.matrix (usually ENSEMBL IDs) and the gene names that should be rendered within the app and in output files; this object is created by `generateShinyApp` using the org.db specified

Value

The UI and Server components of the shiny module, that can be used within the UI and Server definitions of a shiny app.

`determine_uds` *Determine the pattern between two intervals*

Description

This function checks if the two input intervals oferlap and outputs the corresponding pattern (up, down, or straight) based on that.

Usage

```
determine_uds(min1, max1, min2, max2)
```

Arguments

`min1, max1, min2, max2`	the endpoints of the two intervals

Value

A single character (one of "U", "D", "S") representing the pattern

Examples

```
determine_uds(10, 20, 15, 25) # overlap
determine_uds(10, 20, 25, 35) # no overlap
```

`enrichmentPanel` *Generate the enrichment panel of the shiny app*

Description

These are the UI and server components of the enrichment panel of the shiny app. It is generated by including 'Enrichment' in the panels.default argument of `generateShinyApp`.

Usage

```
enrichmentPanelUI(id, show = TRUE)

enrichmentPanelServer(id, DEresults, organism, seed = 13)
```

Arguments

`id`	the input slot that will be used to access the value
`show`	whether to show the panel or not; default is TRUE; there for compatibility with specifying panels to show
`DEresults`	differential expression results output from DEpanelServer; a reactive list with slots 'DEtable' (all genes), 'DEtableSubset' (only DE genes), 'lfcThreshold' and 'pvalThreshold'
`organism`	organism name to be passed on to `gprofiler2::gost`; organism names are constructed by concatenating the first letter of the name and the family name; default is NA - enrichment is not included to ensure compatibility with datasets that have non-standard gene names; a vector (of the same length as modality) can be provided if `length(modality)` > 1
`seed`	the random seed to be set for the jitter plot, to avoid seemingly different plots for the same inputs

Value

The UI and Server components of the shiny module, that can be used within the UI and Server definitions of a shiny app.

`expression_heatmap` *Create heatmap of an expression matrix*

Description

This function creates a heatmap to visualise an expression matrix

Usage

```
expression_heatmap(
  expression.matrix.subset,
  top.annotation.ids = NULL,
  metadata,
  type = c("Z-score", "Log2 Expression", "Expression"),
  show.column.names = TRUE
)
```

Arguments

`expression.matrix.subset`	a subset of rows from the expression matrix; rows correspond to genes and columns correspond to samples
`top.annotation.ids`	a vector of column indices denoting which columns of the metadata should become heatmap annotations
`metadata`	a data frame containing metadata for the samples contained in the expression.matrix; must contain at minimum two columns: the first column must contain the column names of the expression.matrix, while the last column is assumed to contain the experimental conditions that will be tested for differential expression; a list (of the same length as modality) can be provided if #' `length(modality) > 1`
`type`	type of rescaling; one of "Expression" (defautl, does nothing), "Log2 Expression" (returns log2(x + 1) for every value), "Mean Scaled" (each row is scaled by its average), "Z-score" (each row is centered and scaled to mean = 0 and sd = 1)
`show.column.names`	whether to show the column names below the heatmap; default is TRUE

Value

The heatmap as detailed in the ComplexHeatmap package.

Examples

```
expression.matrix.preproc <- as.matrix(read.csv(
  system.file("extdata", "expression_matrix_preprocessed.csv", package = "bulkAnalyseR"),
  row.names = 1
))[1:500,]

metadata <- data.frame(
  srr = colnames(expression.matrix.preproc),
  timepoint = rep(c("0h", "12h", "36h"), each = 2)
)
print(expression_heatmap(head(expression.matrix.preproc), NULL, metadata))
```

`find_regulators_with_recurring_edges` *Find recurring regulators*

Description

This function finds regulators that appear as the same network edge in more than one of the input networks.

Usage

```
find_regulators_with_recurring_edges(weightMatList, plotConnections)
```

Arguments

`weightMatList`	a list of (weighted) adjacency matrices; each list element must be an adjacency matrix with regulators in rows, targets in columns
`plotConnections`	the number of connections to subset to

Value

A vector containing the names of the recurring regulators

Examples

```
weightMat1 <- matrix(
  c(0.1, 0.4, 0.8, 0.3), nrow = 2, ncol = 2,
  dimnames = list("regulators" = c("r1", "r2"), "targets" = c("t1", "t2"))
)
weightMat2 <- matrix(
  c(0.1, 0.2, 0.8, 0.3), nrow = 2, ncol = 2,
  dimnames = list("regulators" = c("r1", "r2"), "targets" = c("t1", "t2"))
)
find_regulators_with_recurring_edges(list(weightMat1, weightMat2), 2)
```

`generateShinyApp` *Generate all files required for an autonomous shiny app*

Description

This function creates an app.R file and all required objects to run the app in .rda format in the target directory. A basic argument check is performed to avoid input data problems. The app directory is standalone and can be used on another platform, as long as bulkAnalyseR is installed there. It is recommended to run `preprocessExpressionMatrix` before this function.

Usage

```
generateShinyApp(
  shiny.dir = "shiny_bulkAnalyseR",
  app.title = "Visualisation of RNA-Seq data",
  theme = "flatly",
  modality = "RNA",
  expression.matrix,
  metadata,
  organism = NA,
  org.db = NA,
  panels.default = c("Landing", "SampleSelect", "QC", "GRN", "DE", "DEplot", "DEsummary",
    "Enrichment", "GRNenrichment", "Cross", "Patterns"),
  panels.extra = tibble::tibble(name = NULL, UIfun = NULL, UIvars = NULL, serverFun =
    NULL, serverVars = NULL),
  data.extra = list(),
  packages.extra = c(),
  cis.integration = tibble::tibble(reference.expression.matrix = NULL, reference.org.db =
   NULL, reference.coord = NULL, comparison.coord = NULL, reference.table.name = NULL,
    comparison.table.name = NULL),
  trans.integration = tibble::tibble(reference.expression.matrix = NULL, reference.org.db
    = NULL, comparison.expression.matrix = NULL, comparison.org.db = NULL,
    reference.table.name = NULL, comparison.table.name = NULL),
  custom.integration = tibble::tibble(reference.expression.matrix = NULL,
   reference.org.db = NULL, comparison.table = NULL, reference.table.name = NULL,
    comparison.table.name = NULL)
)
```

Arguments

`shiny.dir`	directory to store the shiny app; if a non-empty directory with that name already exists an error is generated
`app.title`	title to be displayed within the app
`theme`	shiny theme to be used in the app; default is 'flatly'
`modality`	name of the modality, or a vector of modalities to be included in the app
`expression.matrix`	the expression matrix; rows correspond to genes and columns correspond to samples; usually preprocessed by `preprocessExpressionMatrix`; a list (of the same length as modality) can be provided if #' `length(modality)` > 1
`metadata`	a data frame containing metadata for the samples contained in the expression.matrix; must contain at minimum two columns: the first column must contain the column names of the expression.matrix, while the last column is assumed to contain the experimental conditions that will be tested for differential expression; a list (of the same length as modality) can be provided if #' `length(modality)` > 1
`organism`	organism name to be passed on to `gprofiler2::gost`; organism names are constructed by concatenating the first letter of the name and the family name; default is NA - enrichment is not included to ensure compatibility with datasets

that have non-standard gene names; a vector (of the same length as modality) can be provided if `length(modality) > 1`

`org.db`	database for annotations to transform ENSEMBL IDs to gene names; a list of bioconductor packaged databases can be found with `BiocManager::available("^org\.")`; default in NA, in which case the row names of the expression matrix are used directly - it is recommended to provide ENSEMBL IDs if the database for your model organism is available; a vector (of the same length as modality) can be provided if `length(modality) > 1`
`panels.default`	argument to control which of the default panels will be included in the app; default is all, but the enrichment panel will not appear unless organism is also supplied; note that the 'DE' panel is required for 'DEplot', 'DEsummary', 'Enrichment', and 'GRNenrichment'; a list (of the same length as modality) can be provided if `length(modality) > 1`
`panels.extra`, `data.extra`, `packages.extra`	functionality to add new user-created panels to the app to extend functionality or change the default behaviour of existing panels; a data frame of the modality, panel UI and server names and default parameters should be passed to panels.extra (see example); the names of any packages required should be passed to the packages.extra argument; extra data should be a single list and passed to the data.extra argument
`cis.integration`	functionality to integrate extra cis-regulatory information into GRN panel. Tibble containing names of reference expression matrix, tables of coordinates for elements corresponding to rows of reference expression matrix (reference.coord), tables of coordinates to compare against reference.coord (comparison.coord) and names for comparison tables. See vignettes for more details about inputs.
`trans.integration`	functionality to integrate extra trans-regulatory information into GRN panel. Tibble containing names of reference expression matrix, (reference.expression.matrix), comparison expression matrix (comparison.expression.matrix). Organism database names for each expression matrix and names for each table are also required. See vignettes for more details about inputs.
`custom.integration`	functionality to integrate custom information related to rows of reference expression matrix. Tibble containing names of reference expression matrix, tables (comparison.table) with Reference_ID and Reference_Name (matching ENSEMBL and NAME columns of reference organism database) and Comparison_ID and Comparison_Name plus a Category column containing extra information. Names for the reference expression matrix and comparison table (comparison.table.name) are also required. See vignettes for more details about inputs.

Value

The path to shiny.dir (invisibly).

Examples

```
expression.matrix.preproc <- as.matrix(read.csv(
    system.file("extdata", "expression_matrix_preprocessed.csv", package = "bulkAnalyseR"),
    row.names = 1
  ))
  metadata <- data.frame(
    srr = colnames(expression.matrix.preproc),
    timepoint = rep(c("0h", "12h", "36h"), each = 2)
  )
  app.dir <- generateShinyApp(
    shiny.dir = paste0(tempdir(), "/shiny_Yang2019"),
    app.title = "Shiny app for the Yang 2019 data",
    modality = "RNA",
    expression.matrix = expression.matrix.preproc,
    metadata = metadata,
    organism = "mmusculus",
    org.db = "org.Mm.eg.db"
  )
  # runApp(app.dir)

  # Example of an app with a second copy of the QC panel
  app.dir.qc2 <- generateShinyApp(
    shiny.dir = paste0(tempdir(), "/shiny_Yang2019_QC2"),
    app.title = "Shiny app for the Yang 2019 data",
    expression.matrix = expression.matrix.preproc,
    metadata = metadata,
    organism = "mmusculus",
    org.db = "org.Mm.eg.db",
    panels.extra = tibble::tibble(
      name = "RNA2",
      UIfun = "modalityPanelUI",
      UIvars = "'RNA2', metadata[[1]], NA, 'QC'",
      serverFun = "modalityPanelServer",
      serverVars = "'RNA2', expression.matrix[[1]], metadata[[1]], anno[[1]], NA, 'QC'"
    )
  )
  # runApp(app.dir.qc2)

  # clean up tempdir
  unlink(paste0(normalizePath(tempdir()), "/", dir(tempdir())), recursive = TRUE)
```

genes_barplot   *Create a bar plot of expression for selected genes across samples in an experiment*

Description

This function creates a clustered bar plot between all samples in the expression matrix for the selection of genes.

Usage

```
genes_barplot(sub.expression.matrix, log.transformation = TRUE)
```

Arguments

`sub.expression.matrix`	subset of the expression matrix containing only selected genes
`log.transformation`	whether expression should be shown on log (default) or linear scale

Value

The bar plot as a ggplot object.

Examples

```
expression.matrix.preproc <- as.matrix(read.csv(
  system.file("extdata", "expression_matrix_preprocessed.csv", package = "bulkAnalyseR"),
  row.names = 1
))[1:500,]

print(genes_barplot(head(expression.matrix.preproc,5)))
```

`get_link_list_rename` *Convert the adjacency matrix to network links*

Description

This function converts an adjacency matrix to a data frame of network links, subset to the most important ones.

Usage

```
get_link_list_rename(weightMat, plotConnections)
```

Arguments

`weightMat`	the (weighted) adjacency matrix - regulators in rows, targets in columns
`plotConnections`	the number of connections to subset to

Value

A data frame with fields from, to and value, describing the edges of the network

Examples

```
weightMat <- matrix(
  c(0.1, 0.4, 0.8, 0.3), nrow = 2, ncol = 2,
  dimnames = list("regulators" = c("r1", "r2"), "targets" = c("t1", "t2"))
)
get_link_list_rename(weightMat, 2)
```

`GRNCisPanel` *Generate the GRN cis integration panel of the shiny app*

Description

These are the UI and server components of the GRN cis integration panel of the shiny app. It is generated by including at least 1 row in the cis.integration parameter of `generateShinyApp`.

Usage

```
GRNCisPanelUI(id, reference.table.name, comparison.table.name)

GRNCisPanelServer(
  id,
  expression.matrix,
  anno,
  coord.table.reference,
  coord.table.comparison,
  seed = 13
)
```

Arguments

`id`	the input slot that will be used to access the value
`reference.table.name`	Name for reference expression matrix and coordinate table
`comparison.table.name`	Name for comparison coordinate table
`expression.matrix`	the expression matrix; rows correspond to genes and columns correspond to samples; usually preprocessed by `preprocessExpressionMatrix`; a list (of the same length as modality) can be provided if #' `length(modality) > 1`
`anno`	annotation data frame containing a match between the row names of the expression.matrix (usually ENSEMBL IDs) and the gene names that should be rendered within the app and in output files; this object is created by `generateShinyApp` using the org.db specified
`coord.table.reference`	Table of coordinates corresponding to rows of expression.matrix
`coord.table.comparison`	Table of coordinates to compare against coord.table.reference
`seed`	Random seed to create reproducible GRNs

Value

The UI and Server components of the shiny module, that can be used within the UI and Server definitions of a shiny app.

GRNCustomPanel *Generate the GRN custom integration panel of the shiny app*

Description

These are the UI and server components of the GRN custom integration panel of the shiny app. It is generated by including at least 1 row in the custom.integration parameter of `generateShinyApp`.

Usage

```
GRNCustomPanelUI(id, title = "GRN with custom integration", show = TRUE)

GRNCustomPanelServer(
  id,
  expression.matrix,
  anno,
  comparison.table,
  DEresults = NULL,
  seed = 13
)
```

Arguments

`id`	the input slot that will be used to access the value
`title`	Name for custom panel instance
`show`	whether to show the panel or not; default is TRUE; there for compatibility with specifying panels to show
`expression.matrix`	the expression matrix; rows correspond to genes and columns correspond to samples; usually preprocessed by `preprocessExpressionMatrix`; a list (of the same length as modality) can be provided if #' `length(modality) > 1`
`anno`	annotation data frame containing a match between the row names of the expression.matrix (usually ENSEMBL IDs) and the gene names that should be rendered within the app and in output files; this object is created by `generateShinyApp` using the org.db specified
`comparison.table`	Table linking rows of expression.matrix to custom information, for example miRNAs or transcription factors.
`DEresults`	differential expression results output from DEpanelServer; a reactive list with slots 'DEtable' (all genes), 'DEtableSubset' (only DE genes), 'lfcThreshold' and 'pvalThreshold'
`seed`	Random seed to create reproducible GRNs

Value

The UI and Server components of the shiny module, that can be used within the UI and Server definitions of a shiny app.

`GRNpanel` *Generate the GRN panel of the shiny app*

Description

These are the UI and server components of the GRN panel of the shiny app. It is generated by including 'GRN' in the panels.default argument of `generateShinyApp`.

Usage

```
GRNpanelUI(id, metadata, show = TRUE)

GRNpanelServer(id, expression.matrix, metadata, anno)
```

Arguments

`id`	the input slot that will be used to access the value
`metadata`	a data frame containing metadata for the samples contained in the expression.matrix; must contain at minimum two columns: the first column must contain the column names of the expression.matrix, while the last column is assumed to contain the experimental conditions that will be tested for differential expression; a list (of the same length as modality) can be provided if #' `length(modality)` > 1
`show`	whether to show the panel or not; default is TRUE; there for compatibility with specifying panels to show
`expression.matrix`	the expression matrix; rows correspond to genes and columns correspond to samples; usually preprocessed by `preprocessExpressionMatrix`; a list (of the same length as modality) can be provided if #' `length(modality)` > 1
`anno`	annotation data frame containing a match between the row names of the expression.matrix (usually ENSEMBL IDs) and the gene names that should be rendered within the app and in output files; this object is created by `generateShinyApp` using the org.db specified

Value

The UI and Server components of the shiny module, that can be used within the UI and Server definitions of a shiny app.

GRNTransPanel *Generate the GRN trans integration panel of the shiny app*

Description

These are the UI and server components of the GRN trans integration panel of the shiny app. It is generated by including at least 1 row in the trans.integration parameter of generateShinyApp.

Usage

```
GRNTransPanelUI(id, reference.table.name, comparison.table.name)

GRNTransPanelServer(
  id,
  expression.matrix,
  anno,
  anno.comparison,
  expression.matrix.comparison,
  tablenames,
  seed = 13
)
```

Arguments

`id`	the input slot that will be used to access the value
`expression.matrix`	the expression matrix; rows correspond to genes and columns correspond to samples; usually preprocessed by preprocessExpressionMatrix; a list (of the same length as modality) can be provided if #' `length(modality) > 1`
`anno`	annotation data frame containing a match between the row names of the expression.matrix (usually ENSEMBL IDs) and the gene names that should be rendered within the app and in output files; this object is created by generateShinyApp using the org.db specified
`anno.comparison`	annotation data frame containing a match between the row names of the comparison expression matrix and the names that should be rendered within the app and in output files. The structure matches the anno table created in generateShinyApp using the org.db specified
`expression.matrix.comparison`	Additional expression matrix to integrate. Column names must match column names from expression.matrix.
`tablenames`, `reference.table.name`, `comparison.table.name`	Names for reference and comparison expression tables.
`seed`	Random seed to create reproducible GRNs

Value

The UI and Server components of the shiny module, that can be used within the UI and Server definitions of a shiny app.

---

infer_GRN *Perform GRN inference*

Description

This function performs Gene Regulatory Network inference on a subset of the expression matrix, for a set of potential targets

Usage

```
infer_GRN(
  expression.matrix,
  metadata,
  anno,
  seed = 13,
  targets,
  condition,
  samples,
  inference_method
)
```

Arguments

`expression.matrix`	the expression matrix; rows correspond to genes and columns correspond to samples; usually preprocessed by `preprocessExpressionMatrix`; a list (of the same length as modality) can be provided if #' `length(modality)` > 1
`metadata`	a data frame containing metadata for the samples contained in the expression.matrix; must contain at minimum two columns: the first column must contain the column names of the expression.matrix, while the last column is assumed to contain the experimental conditions that will be tested for differential expression; a list (of the same length as modality) can be provided if #' `length(modality)` > 1
`anno`	annotation data frame containing a match between the row names of the expression.matrix (usually ENSEMBL IDs) and the gene names that should be rendered within the app and in output files; this object is created by `generateShinyApp` using the org.db specified
`seed`	the random seed to be set when running GRN inference, to ensure reproducibility of outputs
`targets`	the target genes of interest around which the GRN is built; must be row names of the expression matrix
`condition`	name of the metadata column to select samples from

`samples`	names of the sample groups to select; must appear in `metadata[[condition]]`
`inference_method`	method used for GRN inference; only supported method is currently GENIE3.

Value

The adjacency matrix of the inferred network

Examples

```
expression.matrix.preproc <- as.matrix(read.csv(
  system.file("extdata", "expression_matrix_preprocessed.csv", package = "bulkAnalyseR"),
  row.names = 1
))[1:500, ]

metadata <- data.frame(
  srr = colnames(expression.matrix.preproc),
  timepoint = rep(c("0h", "12h", "36h"), each = 2)
)

anno <- AnnotationDbi::select(
  getExportedValue('org.Mm.eg.db', 'org.Mm.eg.db'),
  keys = rownames(expression.matrix.preproc),
  keytype = 'ENSEMBL',
  columns = 'SYMBOL'
) %>%
  dplyr::distinct(ENSEMBL, .keep_all = TRUE) %>%
  dplyr::mutate(NAME = ifelse(is.na(SYMBOL), ENSEMBL, SYMBOL))

res <- infer_GRN(
  expression.matrix = expression.matrix.preproc,
  metadata = metadata,
  anno = anno,
  seed = 13,
  targets = c("Hecw2", "Akr1cl"),
  condition = "timepoint",
  samples = "0h",
  inference_method = "GENIE3"
)
```

`jaccard_heatmap` *Create a heatmap of the Jaccard similarity index (JSI) between samples of an experiment*

Description

This function creates a JSI heatmap between all samples in the expression matrix using the specified number of most abundant genes as input. Metadata columns are used as annotations.

Usage

```
jaccard_heatmap(
  expression.matrix,
  metadata,
  top.annotation.ids = NULL,
  n.abundant = NULL,
  show.values = TRUE,
  show.row.column.names = TRUE
)
```

Arguments

`expression.matrix`	the expression matrix; rows correspond to genes and columns correspond to samples; usually preprocessed by `preprocessExpressionMatrix`; a list (of the same length as modality) can be provided if #' `length(modality) > 1`
`metadata`	a data frame containing metadata for the samples contained in the expression.matrix; must contain at minimum two columns: the first column must contain the column names of the expression.matrix, while the last column is assumed to contain the experimental conditions that will be tested for differential expression; a list (of the same length as modality) can be provided if #' `length(modality) > 1`
`top.annotation.ids`	a vector of column indices denoting which columns of the metadata should become heatmap annotations
`n.abundant`	number of most abundant genes to use for the JSI calculation
`show.values`	whether to show the JSI values within the heatmap squares
`show.row.column.names`	whether to show the row and column names below the heatmap; default is TRUE

Value

The JSI heatmap as detailed in the ComplexHeatmap package.

Examples

```
expression.matrix.preproc <- as.matrix(read.csv(
  system.file("extdata", "expression_matrix_preprocessed.csv", package = "bulkAnalyseR"),
  row.names = 1
))[1:500,]

metadata <- data.frame(
  srr = colnames(expression.matrix.preproc),
  timepoint = rep(c("0h", "12h", "36h"), each = 2)
)
print(jaccard_heatmap(expression.matrix.preproc, metadata, n.abundant = 100))
```

`jaccard_index` *Calculate the Jaccard similarity index (JSI) between two vectors*

Description

Calculate the Jaccard similarity index (JSI) between two vectors

Usage

```
jaccard_index(a, b)
```

Arguments

`a, b`	two vectors

Value

The JSI of the two vectors, a single value between 0 and 1.

Examples

```
jaccard_index(1:4, 2:6)
```

`landingPanel` *Generate the landing page panel of the shiny app*

Description

These are the UI and server components of the landing page panel of the shiny app. It is generated by including 'Landing' in the panels.default argument of `generateShinyApp`.

Usage

```
landingPanelUI(id, show = TRUE)

landingPanelServer(id)
```

Arguments

`id`	the input slot that will be used to access the value
`show`	whether to show the panel or not; default is TRUE; there for compatibility with specifying panels to show

Value

The UI and Server components of the shiny module, that can be used within the UI and Server definitions of a shiny app.

make_heatmap_matrix — *Create a matrix of the average expression of each gene in each condition*

Description

This function reshapes the tibble output of calculate_condition_mean_sd_per_gene into a matrix of average expression by condition. Its output can be used by expression_heatmap.

Usage

```
make_heatmap_matrix(tbl, genes = NULL)
```

Arguments

tbl	the output of calculate_condition_mean_sd_per_gene
genes	gene names to use for the output; if NULL (the default), all genes will be used

Value

A matrix of averaged expression per gene in each condition.

Examples

```
expression.matrix.preproc <- as.matrix(read.csv(
  system.file("extdata", "expression_matrix_preprocessed.csv", package = "bulkAnalyseR"),
   row.names = 1
))[1:500,]

condition <- factor(rep(c("0h", "12h", "36h"), each = 2))
tbl <- calculate_condition_mean_sd_per_gene(expression.matrix.preproc[1:10, ], condition)
heatmat <- make_heatmap_matrix(tbl)
heatmat
```

make_pattern_matrix — *Create a matrix of the patterns between conditions*

Description

This function determines the patterns between different conditions of each gene. It should be applied to the output of calculate_condition_mean_sd_per_gene.

Usage

```
make_pattern_matrix(tbl, n_sd = 2)
```

Arguments

`tbl`	the output of `calculate_condition_mean_sd_per_gene`
`n_sd`	number of standard deviations from the mean to use to construct the intervals; default is 2

Value

A matrix of single character patterns between conditions. The last column is named pattern and is a concatenation of all other columns.

Examples

```
expression.matrix.preproc <- as.matrix(read.csv(
  system.file("extdata", "expression_matrix_preprocessed.csv", package = "bulkAnalyseR"),
  row.names = 1
))[1:500,]

condition <- factor(rep(c("0h", "12h", "36h"), each = 2))
tbl <- calculate_condition_mean_sd_per_gene(expression.matrix.preproc[1:10, ], condition)
patmat <- make_pattern_matrix(tbl)
patmat
```

`ma_plot` *Create an MA plot visualising differential expression (DE) results*

Description

This function creates an MA plot to visualise the results of a DE analysis.

`ma_enhance` is called indirectly by `ma_plot` to add extra features.

Usage

```
ma_plot(
  genes.de.results,
  pval.threshold = 0.05,
  lfc.threshold = 1,
  alpha = 0.1,
  ylims = NULL,
  add.colours = TRUE,
  add.expression.colour.gradient = TRUE,
  add.guide.lines = TRUE,
  add.labels.auto = TRUE,
  add.labels.custom = FALSE,
  ...
)

ma_enhance(
  p,
  df,
  pval.threshold,
  lfc.threshold,
  alpha,
  add.colours,
  point.colours = c("#bfbfbf", "orange", "red", "blue"),
  raster = FALSE,
  add.expression.colour.gradient,
 colour.gradient.scale = list(left = c("#99e6ff", "#000066"), right = c("#99e6ff",
    "#000066")),
  colour.gradient.breaks = waiver(),
  colour.gradient.limits = NULL,
  add.guide.lines,
  guide.line.colours = c("green", "blue"),
  add.labels.auto,
  add.labels.custom,
  annotation = NULL,
  n.labels.auto = c(5, 5, 5),
  genes.to.label = NULL,
  seed = 0,
  label.force = 1
)
```

Arguments

`genes.de.results`	the table of DE genes, usually generated by `DEanalysis_edger`
`pval.threshold, lfc.threshold`	the p-value and/or log2(fold-change) thresholds to determine whether a gene is DE
`alpha`	the transparency of points; ignored for DE genes if add.expression.colour.gradient is TRUE; default is 0.1
`ylims`	a single value to create (symmetric) y-axis limits; by default inferred from the data
`add.colours`	whether to colour genes based on their log2(fold-change) and -log10(p-value); default is TRUE
`add.expression.colour.gradient`	whether to add a colour gradient for DE genes to present their log2(expression); default is TRUE
`add.guide.lines`	whether to add vertical and horizontal guide lines to the plot to highlight the thresholds; default is TRUE
`add.labels.auto`	whether to automatically label genes with the highest |log2(fold-change)| and expression; default is TRUE

`add.labels.custom`	whether to add labels to user-specified genes; the parameter genes.to.label must also be specified; default is FALSE
`...`	parameters passed on to `ma_enhance`
`p`	MA plot as a ggplot object (usually passed by `ma_plot`)
`df`	data frame of DE results for all genes (usually passed by `ma_plot`)
`point.colours`	a vector of 4 colours to colour genes with both pval and lfc under thresholds, just pval under threshold, just lfc under threshold, both pval and lfc over threshold (DE genes) respectively; only used if add.colours is TRUE
`raster`	whether to rasterize non-DE genes with ggraster to reduce memory usage; particularly useful when saving plots to files
`colour.gradient.scale`	a vector of two colours to create a colour gradient for colouring the DE genes based on expression; a named list with components left and right can be supplied to use two different colour scales; only used if add.expression.colour.gradient is TRUE
`colour.gradient.breaks`, `colour.gradient.limits`	parameters to customise the legend of the colour gradient scale; especially useful if creating multiple plots or a plot with two scales; only used if add.expression.colour.gradient is TRUE
`guide.line.colours`	a vector with two colours to be used to colour the guide lines; the first colour is used for the p-value and log2(fold-change) thresholds and the second for double those values
`annotation`	annotation data frame containing a match between the gene field of df (usually ENSEMBL IDs) and the gene names that should be shown in the plot labels; not necessary if df already contains gene names
`n.labels.auto`	a integer vector of length 3 denoting the number of genes that should be automatically labelled; the first entry corresponds to DE genes with the lowest p-value, the second to those with highest absolute log2(fold-change) and the third to those with highest expression; a single integer can also be specified, to be used for all 3 entries; default is 5
`genes.to.label`	a vector of gene names to be labelled in the plot; if names are present those are shown as the labels (but the values are the ones matched - this is to allow custom gene names to be presented)
`seed`	the random seed to be used for reproducibility; only used for ggrepel::geom_label_repel if labels are present
`label.force`	passed to the force argument of ggrepel::geom_label_repel; higher values make labels overlap less (at the cost of them being further away from the points they are labelling)

Value

The MA plot as a ggplot object.

The enhanced MA plot as a ggplot object.

Examples

```
expression.matrix.preproc <- as.matrix(read.csv(
  system.file("extdata", "expression_matrix_preprocessed.csv", package = "bulkAnalyseR"),
  row.names = 1
))[1:500, 1:4]

anno <- AnnotationDbi::select(
  getExportedValue('org.Mm.eg.db', 'org.Mm.eg.db'),
  keys = rownames(expression.matrix.preproc),
  keytype = 'ENSEMBL',
  columns = 'SYMBOL'
) %>%
  dplyr::distinct(ENSEMBL, .keep_all = TRUE) %>%
  dplyr::mutate(NAME = ifelse(is.na(SYMBOL), ENSEMBL, SYMBOL))

edger <- DEanalysis_edger(
  expression.matrix = expression.matrix.preproc,
  condition = rep(c("0h", "12h"), each = 2),
  var1 = "0h",
  var2 = "12h",
  anno = anno
)
mp <- ma_plot(edger)
print(mp)
```

`modalityPanel` *Generate an app panel for a modality*

Description

These are the UI and server components of a modality panel of the shiny app. Different modalities can be included by specifying their inputs in `generateShinyApp`.

Usage

```
modalityPanelUI(id, metadata, organism, panels.default)

modalityPanelServer(
  id,
  expression.matrix,
  metadata,
  anno,
  organism,
  panels.default
)
```

Arguments

`id`	the input slot that will be used to access the value
`metadata`	a data frame containing metadata for the samples contained in the expression.matrix; must contain at minimum two columns: the first column must contain the column names of the expression.matrix, while the last column is assumed to contain the experimental conditions that will be tested for differential expression; a list (of the same length as modality) can be provided if #' `length(modality) > 1`
`organism`	organism name to be passed on to `gprofiler2::gost`; organism names are constructed by concatenating the first letter of the name and the family name; default is NA - enrichment is not included to ensure compatibility with datasets that have non-standard gene names; a vector (of the same length as modality) can be provided if `length(modality) > 1`
`panels.default`	argument to control which of the default panels will be included in the app; default is all, but the enrichment panel will not appear unless organism is also supplied; note that the 'DE' panel is required for 'DEplot', 'DEsummary', 'Enrichment', and 'GRNenrichment'; a list (of the same length as modality) can be provided if `length(modality) > 1`
`expression.matrix`	the expression matrix; rows correspond to genes and columns correspond to samples; usually preprocessed by `preprocessExpressionMatrix`; a list (of the same length as modality) can be provided if #' `length(modality) > 1`
`anno`	annotation data frame containing a match between the row names of the expression.matrix (usually ENSEMBL IDs) and the gene names that should be rendered within the app and in output files; this object is created by `generateShinyApp` using the org.db specified

Value

The UI and Server components of the shiny module, that can be used within the UI and Server definitions of a shiny app.

`noisyr_counts_with_plot`
Apply a modified noisyR counts pipeline printing a plot

Description

This function is identical to the noisyr::noisyr_counts function, with the addition of the option to print a line plot of the similarity against expression for all samples.

Usage

```
noisyr_counts_with_plot(
  expression.matrix,
  n.elements.per.window = NULL,
  optimise.window.length.logical = FALSE,
  similarity.threshold = 0.25,
  method.chosen = "Boxplot-IQR",
  ...,
  output.plot = FALSE
)
```

Arguments

`expression.matrix`	the expression matrix; rows correspond to genes and columns correspond to samples
`n.elements.per.window`	number of elements to have in a window passed to calculate_expression_similarity_counts(); default 10% of the number of rows
`optimise.window.length.logical`	whether to call optimise_window_length to try and optimise the value of n.elements.per.window
`similarity.threshold, method.chosen`	parameters passed on to `calculate_noise_threshold`; they can be single values or vectors; if they are vectors optimal values are computed by calling `calculate_noise_threshold_r` and minimising the coefficient of variation across samples; all possible values for method.chosen can be viewed by `get_methods_calculate_noise_threshold`
`...`	optional arguments passed on to `noisyr::noisyr_counts()`
`output.plot`	whether to create an expression-similarity plot for the noise analysis (printed to the console); default is FALSE

Value

The denoised expression matrix.

Examples

```
expression.matrix <- as.matrix(read.csv(
  system.file("extdata", "expression_matrix.csv", package = "bulkAnalyseR"),
  row.names = 1
))[1:10, 1:4]
expression.matrix.denoised <- noisyr_counts_with_plot(expression.matrix)
```

`patternPanel` *Generate the expression patterns panel of the shiny app*

Description

These are the UI and server components of the expression patterns panel of the shiny app. It is generated by including 'Patterns' in the panels.default argument of `generateShinyApp`.

Usage

```
patternPanelUI(id, metadata, show = TRUE)

patternPanelServer(id, expression.matrix, metadata, anno)
```

Arguments

`id`	the input slot that will be used to access the value
`metadata`	a data frame containing metadata for the samples contained in the expression.matrix; must contain at minimum two columns: the first column must contain the column names of the expression.matrix, while the last column is assumed to contain the experimental conditions that will be tested for differential expression; a list (of the same length as modality) can be provided if #' `length(modality)` > 1
`show`	whether to show the panel or not; default is TRUE; there for compatibility with specifying panels to show
`expression.matrix`	the expression matrix; rows correspond to genes and columns correspond to samples; usually preprocessed by `preprocessExpressionMatrix`; a list (of the same length as modality) can be provided if #' `length(modality)` > 1
`anno`	annotation data frame containing a match between the row names of the expression.matrix (usually ENSEMBL IDs) and the gene names that should be rendered within the app and in output files; this object is created by `generateShinyApp` using the org.db specified

Value

The UI and Server components of the shiny module, that can be used within the UI and Server definitions of a shiny app.

`plot_GRN` *Plot a GRN*

Description

This function creates a network plot of a GRN.

Usage

```
plot_GRN(
  weightMat,
  anno,
  plotConnections,
  plot_position_grid,
  n_networks,
  recurring_regulators
)
```

Arguments

`weightMat`	the (weighted) adjacency matrix - regulators in rows, targets in columns
`anno`	annotation data frame containing a match between the row names of the expression.matrix (usually ENSEMBL IDs) and the gene names that should be rendered within the app and in output files; this object is created by `generateShinyApp` using the org.db specified
`plotConnections`	the number of connections to subset to
`plot_position_grid`, `n_networks`	the position of the plot in the grid (1-4) and the number of networks shown (1-4); these are solely used for hiding unwanted plots in the shiny app
`recurring_regulators`	targets to be highlighted; usually the result of `find_regulators_with_recurring_edges`

Value

A network plot. See visNetwork package for more details.

Examples

```
weightMat1 <- matrix(
  c(0.1, 0.4, 0.8, 0.3), nrow = 2, ncol = 2,
  dimnames = list("regulators" = c("r1", "r2"), "targets" = c("t1", "t2"))
)
weightMat2 <- matrix(
  c(0.1, 0.2, 0.8, 0.3), nrow = 2, ncol = 2,
  dimnames = list("regulators" = c("r1", "r2"), "targets" = c("t1", "t2"))
)
anno <- tibble::tibble(ENSEMBL = c("r1", "r2", "t1", "t2"), NAME = ENSEMBL)
recurring_regulators <- find_regulators_with_recurring_edges(list(weightMat1, weightMat2), 2)
plot_GRN(weightMat1, anno, 2, 1, 1, recurring_regulators)
plot_GRN(weightMat2, anno, 2, 1, 1, recurring_regulators)
```

`plot_line_pattern` *Create a line plot of average expression across conditions*

Description

This function creates a line plot of average expression across conditions for a selection of genes, usually to visualise an expression pattern.

Usage

```
plot_line_pattern(
  tbl,
  genes = NULL,
  type = c("Mean Scaled", "Log2 Expression", "Expression"),
  show.legend = FALSE
)
```

Arguments

`tbl`	the output of `calculate_condition_mean_sd_per_gene`
`genes`	gene names to use for the output; if NULL (the default), all genes will be used
`type`	whether the expression values should be scaled using their mean (default), log-transformed, or not adjusted for the plot
`show.legend`	whether to show the gene names in the legend; should be avoided in many genes are plotted

Value

A matrix of average gene expression per gene in each condition.

Examples

```
expression.matrix.preproc <- as.matrix(read.csv(
  system.file("extdata", "expression_matrix_preprocessed.csv", package = "bulkAnalyseR"),
   row.names = 1
))[1:500,]

condition <- factor(rep(c("0h", "12h", "36h"), each = 2))
tbl <- calculate_condition_mean_sd_per_gene(expression.matrix.preproc[1:10, ], condition)
plot_line_pattern(tbl)
```

`plot_pca` *Create a principal component analysis (PCA) plot the samples of an experiment*

Description

This function creates a PCA plot between all samples in the expression matrix using the specified number of most abundant genes as input. A metadata column is used as annotation.

Usage

```
plot_pca(
  expression.matrix,
  metadata,
  annotation.id,
  n.abundant = NULL,
  show.labels = FALSE,
  show.ellipses = TRUE,
  label.force = 1
)
```

Arguments

`expression.matrix`	the expression matrix; rows correspond to genes and columns correspond to samples; usually preprocessed by `preprocessExpressionMatrix`; a list (of the same length as modality) can be provided if #' `length(modality)` > 1
`metadata`	a data frame containing metadata for the samples contained in the expression.matrix; must contain at minimum two columns: the first column must contain the column names of the expression.matrix, while the last column is assumed to contain the experimental conditions that will be tested for differential expression; a list (of the same length as modality) can be provided if #' `length(modality)` > 1
`annotation.id`	a column index denoting which column of the metadata should be used to colour the points and draw confidence ellipses
`n.abundant`	number of most abundant genes to use for the JSI calculation
`show.labels`	whether to label the points with the sample names
`show.ellipses`	whether to draw confidence ellipses
`label.force`	passed to the force argument of ggrepel::geom_label_repel; higher values make labels overlap less (at the cost of them being further away from the points they are labelling)

Value

The PCA plot as a ggplot object.

Examples

```
expression.matrix.preproc <- as.matrix(read.csv(
  system.file("extdata", "expression_matrix_preprocessed.csv", package = "bulkAnalyseR"),
  row.names = 1
))[1:500,]

metadata <- data.frame(
  srr = colnames(expression.matrix.preproc),
  timepoint = rep(c("0h", "12h", "36h"), each = 2)
)
plot_pca(expression.matrix.preproc, metadata, 2)
```

`plot_upset` *Visualise the overlap of edges between different networks*

Description

This function creates an UpSet plot of the intersections and specific differences of the edges in the input networks.

Usage

```
plot_upset(weightMatList, plotConnections)
```

Arguments

`weightMatList`	a list of (weighted) adjacency matrices; each list element must be an adjacency matrix with regulators in rows, targets in columns
`plotConnections`	the number of connections to subset to

Value

An UpSet plot. See UpSetR package for more details.

Examples

```
weightMat1 <- matrix(
  c(0.1, 0.4, 0.8, 0.3), nrow = 2, ncol = 2,
  dimnames = list("regulators" = c("r1", "r2"), "targets" = c("t1", "t2"))
)
weightMat2 <- matrix(
  c(0.1, 0.2, 0.8, 0.3), nrow = 2, ncol = 2,
  dimnames = list("regulators" = c("r1", "r2"), "targets" = c("t1", "t2"))
)
plot_upset(list(weightMat1, weightMat2), 2)
```

`preprocessExpressionMatrix` *Pre-process the expression matrix before generating the shiny app*

Description

This function denoises the expression matrix using the noisyR package and then normalises it. It is recommended to use this function before using `generateShinyApp`.

Usage

```
preprocessExpressionMatrix(
  expression.matrix,
  denoise = TRUE,
  output.plot = FALSE,
  normalisation.method = c("quantile", "rpm", "tmm", "deseq2", "median"),
  n_million = 1,
  ...
)
```

Arguments

`expression.matrix`	the expression matrix; rows correspond to genes and columns correspond to samples

`denoise`	whether to use noisyR to denoise the expression matrix; proceeding without denoising data is not recommended
`output.plot`	whether to create an expression-similarity plot for the noise analysis (printed to the console); default is FALSE
`normalisation.method`	the normalisation method to be used; default is quantile; any unrecognised input will result in no normalisation being applied, but proceeding with un-normalised data is not recommended; currently supported normalisation methods are: **quantile** Quantile normalisation using the `normalize.quantiles` function from the `preprocessCore` package; **rpm** RPM (reads per million) normalisation, where each sample is scaled by 1 (or more using the n_million parameter) million and divided by the total number of reads in that sample; **tmm** Trimmed Mean of M values normalisation using the `calcNormFactors` function from the `edgeR` package; **deseq2** Size factor normalisation using the `estimateSizeFactorsForMatrix` function from the `DESeq2` package; **median** Normalisation using the median, where each sample is scaled by the median expression in the sample divided by the total number of reads in that sample
`n_million`	scaling factor for RPM normalisation; default is 1 million
`...`	optional arguments passed on to `noisyr::noisyr_counts()`

Value

The denoised, normalised expression matrix; some rows (genes) may have been removed by noisyR.

Examples

```
expression.matrix <- as.matrix(read.csv(
  system.file("extdata", "expression_matrix.csv", package = "bulkAnalyseR"),
  row.names = 1
))[1:10, 1:4]
expression.matrix.preproc <- preprocessExpressionMatrix(expression.matrix)
```

`preprocess_miRTarBase` *Creates a comparison table for miRTarBase to be used for custom integration*

Description

This function downloads the miRTarBase database for the organism of choice, filters it according to user-specified values and formats ready for custom integration in `generateShinyApp`.

Usage

```
preprocess_miRTarBase(
  download.dir = ".",
  download.method = "auto",
  mirtarbase.file = NULL,
  organism.code,
  org.db,
  support.type = c(),
  validation.method = c(),
  reference = c("mRNA", "miRNA"),
  print.support.types = FALSE,
  print.validation.methods = FALSE
)
```

Arguments

`download.dir`	Directory where miRTarBase database will be downloaded.
`download.method`	Method for downloading miRTarBase file through download.file, see download.file documentation for options for your operating system.
`mirtarbase.file`	Path to pre-downloaded miRTarBase file for your organism. If this is left NULL then the file will be downloaded.
`organism.code`	Three letter code for the organism of choice. See miRTarBase website for options. For human, enter 'hsa' and for mouse, 'mmu'.
`org.db`	database for annotations to transform ENSEMBL IDs to gene names; a list of bioconductor packaged databases can be found with `BiocManager::available("^org\.")`.
`support.type`	Subset of entries of the 'Support Type' field in miRTarBase. Only these values will be kept. To find the options available for your organism of choice, run the function once with `print.support.types = TRUE`.
`validation.method`	Subset of entries of 'Experiments' field in miRTarBase. Only these values will be kept. To find the options available for your organism of choice, run the function once with `print.validation.methods = TRUE`.
`reference`	Should the reference category be mRNA or miRNA? The reference category chosen here must match the reference category chosen in `custom.integration` in `generateShinyApp`. Default in mRNA.
`print.support.types`, `print.validation.methods`	Should options for Support Type and Experiments be displayed? Default is FALSE.

Value

A dataframe with Reference_ID/Name and Comparison_ID/Name columns which can be supplied to `custom.integration` in `generateShinyApp`

Examples

```
comparison.table <- preprocess_miRTarBase(
  mirtarbase.file = system.file("extdata", "mmu_MTI_sub.xls", package = "bulkAnalyseR"),
  organism.code = "mmu",
  org.db = "org.Mm.eg.db",
  support.type = "Functional MTI",
  validation.method = "Luciferase reporter assay",
  reference = "miRNA")
```

QCpanel *Generate the QC panel of the shiny app*

Description

These are the UI and server components of the QC panel of the shiny app. It is generated by including 'QC' in the panels.default argument of `generateShinyApp`.

Usage

```
QCpanelUI(id, metadata, show = TRUE)

QCpanelServer(id, expression.matrix, metadata, anno)
```

Arguments

`id`	the input slot that will be used to access the value
`metadata`	a data frame containing metadata for the samples contained in the expression.matrix; must contain at minimum two columns: the first column must contain the column names of the expression.matrix, while the last column is assumed to contain the experimental conditions that will be tested for differential expression; a list (of the same length as modality) can be provided if #' `length(modality)` > 1
`show`	whether to show the panel or not; default is TRUE; there for compatibility with specifying panels to show
`expression.matrix`	the expression matrix; rows correspond to genes and columns correspond to samples; usually preprocessed by `preprocessExpressionMatrix`; a list (of the same length as modality) can be provided if #' `length(modality)` > 1
`anno`	annotation data frame containing a match between the row names of the expression.matrix (usually ENSEMBL IDs) and the gene names that should be rendered within the app and in output files; this object is created by `generateShinyApp` using the org.db specified

Value

The UI and Server components of the shiny module, that can be used within the UI and Server definitions of a shiny app.

qc_density_plot — *Create a density plot of log2 expression across samples of an experiment*

Description

This function creates a density plot between all samples in the expression matrix. Metadata columns are used to group samples.

Usage

```
qc_density_plot(expression.matrix, metadata, annotation.id)
```

Arguments

`expression.matrix`	the expression matrix; rows correspond to genes and columns correspond to samples; usually preprocessed by `preprocessExpressionMatrix`; a list (of the same length as modality) can be provided if #' `length(modality) > 1`
`metadata`	a data frame containing metadata for the samples contained in the expression.matrix; must contain at minimum two columns: the first column must contain the column names of the expression.matrix, while the last column is assumed to contain the experimental conditions that will be tested for differential expression; a list (of the same length as modality) can be provided if #' `length(modality) > 1`
`annotation.id`	name of metadata column on which to group samples

Value

The density plot as a ggplot object.

Examples

```
expression.matrix.preproc <- as.matrix(read.csv(
  system.file("extdata", "expression_matrix_preprocessed.csv", package = "bulkAnalyseR"),
  row.names = 1
))[1:500,]

metadata <- data.frame(
  srr = colnames(expression.matrix.preproc),
  timepoint = rep(c("0h", "12h", "36h"), each = 2)
)
print(qc_density_plot(expression.matrix.preproc, metadata, 'timepoint'))
```

qc_violin_plot *Create a violin/box plot of expression across samples of an experiment*

Description

This function creates a combined violin and box plot between all samples in the expression matrix. Metadata columns are used to colour samples.

Usage

```
qc_violin_plot(
  expression.matrix,
  metadata,
  annotation.id,
  log.transformation = TRUE
)
```

Arguments

`expression.matrix`	the expression matrix; rows correspond to genes and columns correspond to samples; usually preprocessed by `preprocessExpressionMatrix`; a list (of the same length as modality) can be provided if #' `length(modality)` > 1
`metadata`	a data frame containing metadata for the samples contained in the expression.matrix; must contain at minimum two columns: the first column must contain the column names of the expression.matrix, while the last column is assumed to contain the experimental conditions that will be tested for differential expression; a list (of the same length as modality) can be provided if #' `length(modality)` > 1
`annotation.id`	name of metadata column on which to group samples
`log.transformation`	whether expression should be shown on log (default) or linear scale

Value

The violin/box plot as a ggplot object.

Examples

```
expression.matrix.preproc <- as.matrix(read.csv(
  system.file("extdata", "expression_matrix_preprocessed.csv", package = "bulkAnalyseR"),
  row.names = 1
))[1:500,]

metadata <- data.frame(
  srr = colnames(expression.matrix.preproc),
  timepoint = rep(c("0h", "12h", "36h"), each = 2)
)
```

```
print(qc_violin_plot(expression.matrix.preproc, metadata, 'timepoint'))
```

`rescale_matrix` *Rescale a matrix*

Description

This function rescales the rows of a matrix according to the specified type.

Usage

```
rescale_matrix(
  mat,
  type = c("Expression", "Log2 Expression", "Mean Scaled", "Z-score")
)
```

Arguments

`mat`	the matrix to rescale
`type`	type of rescaling; one of "Expression" (defautl, does nothing), "Log2 Expression" (returns log2(x + 1) for every value), "Mean Scaled" (each row is scaled by its average), "Z-score" (each row is centered and scaled to mean = 0 and sd = 1)

Value

The rescaled matrix.

Examples

```
mat = matrix(1:10, nrow = 2, ncol = 5)
rescale_matrix(mat, type = "Expression")
rescale_matrix(mat, type = "Log2 Expression")
rescale_matrix(mat, type = "Mean Scaled")
rescale_matrix(mat, type = "Z-score")
```

`sampleSelectPanel` *Generate the sample select panel of the shiny app*

Description

These are the UI and server components of the sample selection panel of the shiny app. It is generated by including 'SampleSelect' in the panels.default argument of `generateShinyApp`.

Usage

```
sampleSelectPanelUI(id, metadata, show = TRUE)

sampleSelectPanelServer(id, expression.matrix, metadata, modality = "RNA")
```

Arguments

`id`	the input slot that will be used to access the value
`metadata`	a data frame containing metadata for the samples contained in the expression.matrix; must contain at minimum two columns: the first column must contain the column names of the expression.matrix, while the last column is assumed to contain the experimental conditions that will be tested for differential expression; a list (of the same length as modality) can be provided if #' `length(modality)` > 1
`show`	whether to show the panel or not; default is TRUE; there for compatibility with specifying panels to show
`expression.matrix`	the expression matrix; rows correspond to genes and columns correspond to samples; usually preprocessed by `preprocessExpressionMatrix`; a list (of the same length as modality) can be provided if #' `length(modality)` > 1
`modality`	the modality, needs to be passed when used within another shiny module for namespacing reasons

Value

The UI and Server components of the shiny module, that can be used within the UI and Server definitions of a shiny app.

`scatter_plot` *Create a scatter plot of expression between two samples of an experiment*

Description

This function creates a scatter plot between two samples.

Usage

```
scatter_plot(
  sub.expression.matrix,
  anno,
  genes.to.highlight = c(),
  log.transformation = TRUE
)
```

Arguments

`sub.expression.matrix`	subset of the expression matrix containing only the two selected samples
`anno`	annotation data frame containing a match between the row names of the expression.matrix (usually ENSEMBL IDs) and the gene names that should be rendered within the app and in output files; this object is created by `generateShinyApp` using the org.db specified
`genes.to.highlight`	vector of gene names to highlight. These should match entries in the anno NAME column.
`log.transformation`	whether expression should be shown on log (default) or linear scale

Value

The scatter plot as a ggplot object.

Examples

```
expression.matrix.preproc <- as.matrix(read.csv(
  system.file("extdata", "expression_matrix_preprocessed.csv", package = "bulkAnalyseR"),
  row.names = 1
))[,1:2]

print(scatter_plot(expression.matrix.preproc, c()))
```

`volcano_plot` *Create a volcano plot visualising differential expression (DE) results*

Description

This function creates a volcano plot to visualise the results of a DE analysis.

`volcano_enhance` is called indirectly by `volcano_plot` to add extra features.

Usage

```
volcano_plot(
  genes.de.results,
  pval.threshold = 0.05,
  lfc.threshold = 1,
  alpha = 0.1,
  xlims = NULL,
  log10pval.cap = TRUE,
  add.colours = TRUE,
  add.expression.colour.gradient = TRUE,
  add.guide.lines = TRUE,
  add.labels.auto = TRUE,
  add.labels.custom = FALSE,
  ...
)

volcano_enhance(
  vp,
  df,
  pval.threshold,
  lfc.threshold,
  alpha,
  add.colours,
  point.colours = c("#bfbfbf", "orange", "red", "blue"),
  raster = FALSE,
  add.expression.colour.gradient,
 colour.gradient.scale = list(left = c("#99e6ff", "#000066"), right = c("#99e6ff",
    "#000066")),
  colour.gradient.breaks = waiver(),
  colour.gradient.limits = NULL,
  add.guide.lines,
  guide.line.colours = c("green", "blue"),
  add.labels.auto,
  add.labels.custom,
  annotation = NULL,
  n.labels.auto = c(5, 5, 5),
  genes.to.label = NULL,
  seed = 0,
  label.force = 1
)
```

Arguments

`genes.de.results`	the table of DE genes, usually generated by `DEanalysis_edger`
`pval.threshold, lfc.threshold`	the p-value and/or log2(fold-change) thresholds to determine whether a gene is DE

`alpha`	the transparency of points; ignored for DE genes if add.expression.colour.gradient is TRUE; default is 0.1
`xlims`	a single value to create (symmetric) x-axis limits; by default inferred from the data
`log10pval.cap`	whether to cap the log10(p-value at -10); any p-values lower that 10^(-10) are set to the cap for plotting
`add.colours`	whether to colour genes based on their log2(fold-change) and -log10(p-value); default is TRUE
`add.expression.colour.gradient`	whether to add a colour gradient for DE genes to present their log2(expression); default is TRUE
`add.guide.lines`	whether to add vertical and horizontal guide lines to the plot to highlight the thresholds; default is TRUE
`add.labels.auto`	whether to automatically label genes with the highest |log2(fold-change)| and expression; default is TRUE
`add.labels.custom`	whether to add labels to user-specified genes; the parameter genes.to.label must also be specified; default is FALSE
`...`	parameters passed on to `volcano_enhance`
`vp`	volcano plot as a ggplot object (usually passed by `volcano_plot`)
`df`	data frame of DE results for all genes (usually passed by `volcano_plot`)
`point.colours`	a vector of 4 colours to colour genes with both pval and lfc under thresholds, just pval under threshold, just lfc under threshold, both pval and lfc over threshold (DE genes) respectively; only used if add.colours is TRUE
`raster`	whether to rasterize non-DE genes with ggraster to reduce memory usage; particularly useful when saving plots to files
`colour.gradient.scale`	a vector of two colours to create a colour gradient for colouring the DE genes based on expression; a named list with components left and right can be supplied to use two different colour scales; only used if add.expression.colour.gradient is TRUE
`colour.gradient.breaks`, `colour.gradient.limits`	parameters to customise the legend of the colour gradient scale; especially useful if creating multiple plots or a plot with two scales; only used if add.expression.colour.gradient is TRUE
`guide.line.colours`	a vector with two colours to be used to colour the guide lines; the first colour is used for the p-value and log2(fold-change) thresholds and the second for double those values
`annotation`	annotation data frame containing a match between the gene field of df (usually ENSEMBL IDs) and the gene names that should be shown in the plot labels; not necessary if df already contains gene names

`n.labels.auto`	a integer vector of length 3 denoting the number of genes that should be automatically labelled; the first entry corresponds to DE genes with the lowest p-value, the second to those with highest absolute log2(fold-change) and the third to those with highest expression; a single integer can also be specified, to be used for all 3 entries; default is 5
`genes.to.label`	a vector of gene names to be labelled in the plot; if names are present those are shown as the labels (but the values are the ones matched - this is to allow custom gene names to be presented)
`seed`	the random seed to be used for reproducibility; only used for ggrepel::geom_label_repel if labels are present
`label.force`	passed to the force argument of ggrepel::geom_label_repel; higher values make labels overlap less (at the cost of them being further away from the points they are labelling)

Value

The volcano plot as a ggplot object.

The enhanced volcano plot as a ggplot object.

Examples

```
expression.matrix.preproc <- as.matrix(read.csv(
  system.file("extdata", "expression_matrix_preprocessed.csv", package = "bulkAnalyseR"),
  row.names = 1
))[1:500, 1:4]

anno <- AnnotationDbi::select(
  getExportedValue('org.Mm.eg.db', 'org.Mm.eg.db'),
  keys = rownames(expression.matrix.preproc),
  keytype = 'ENSEMBL',
  columns = 'SYMBOL'
) %>%
  dplyr::distinct(ENSEMBL, .keep_all = TRUE) %>%
  dplyr::mutate(NAME = ifelse(is.na(SYMBOL), ENSEMBL, SYMBOL))

edger <- DEanalysis_edger(
  expression.matrix = expression.matrix.preproc,
  condition = rep(c("0h", "12h"), each = 2),
  var1 = "0h",
  var2 = "12h",
  anno = anno
)
vp <- volcano_plot(edger)
print(vp)
```

Index