Package 'PolySTest'

November 29, 2024

Title PolySTest: Detection of differentially regulated features. Combined statistical testing for data with few replicates and missing values

Version 1.0.2

Description The complexity of high-throughput quantitative omics experiments often leads to low replicates numbers and many missing values. We implemented a new test to simultaneously consider missing values and quantitative changes, which we combined with well-performing statistical tests for high confidence detection of differentially regulated features. The package contains functions to run the test and to visualize the results.

License GPL-2

Encoding UTF-8

biocViews MassSpectrometry, Proteomics, Software, DifferentialExpression

BugReports https://github.com/computproteomics/PolySTest/issues

URL https://github.com/computproteomics/PolySTest

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Imports fdrtool (>= 1.2.15), limma (>= 3.61.3), matrixStats (>= 0.57.0), qvalue (>= 2.22.0), shiny (>= 1.5.0), SummarizedExperiment (>= 1.20.0), knitr (>= 1.33), plotly (>= 4.9.4), heatmaply (>= 1.1.1), circlize (>= 0.4.12), UpSetR (>= 1.4.0), gplots (>= 3.1.1), S4Vectors (>= 0.30.0), parallel (>= 4.1.0), grDevices (>= 4.1.0), graphics (>= 4.1.0), stats (>= 4.1.0), utils (>= 4.1.0)

Suggests testthat (>= 3.0.0), BiocStyle

VignetteBuilder knitr

Config/testthat/edition 3

Depends R (>= 4.4.0)

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

git_branch RELEASE_3_20

git_last_commit dc89cc3

git_last_commit_date 2024-11-26

Repository Bioconductor 3.20

Date/Publication 2024-11-28

Author Veit Schwämmle [aut, cre] (<https://orcid.org/0000-0002-9708-6722>)

Maintainer Veit Schwämmle <veits@bmb.sdu.dk>

Contents

`check_for_polystest` *Check SummarizedExperiment for PolySTest Requirements*

Description

Performs checks on a SummarizedExperiment object to ensure it is properly formatted for PolySTest analysis. It checks for specific metadata properties, the number of assays, and the distribution of conditions and replicates.

Usage

```
check_for_polystest(se)
```

Arguments

se	A SummarizedExperiment object.

Value

Invisible TRUE if checks pass; otherwise, warnings or errors are thrown.

Examples

```
data(liver_example)
check_for_polystest(liver_example)
```

---

check_stat_names *Check Statistical Test and Comparison Names*

Description

Verifies if the provided test names and comparison names are correct and have been executed within the given SummarizedExperiment object. It checks for the presence of specific metadata related to the tests and comparisons to ensure that the requested analyses have been carried out.

Usage

```
check_stat_names(fulldata, compNames, testNames)
```

Arguments

fulldata	A SummarizedExperiment object containing the dataset and metadata for statistical analyses.
compNames	A character vector of comparison names to be verified against the SummarizedExperiment metadata. If set to "all", the function checks for the presence of any comparison names in the metadata.
testNames	A character vector of statistical test names to be verified against the SummarizedExperiment metadata.

Details

This function is used to ensure that the requested statistical tests and comparisons have been correctly named and executed prior to further analysis. It verifies that the names provided match those stored within the metadata of the SummarizedExperiment object. If the names do not match or the necessary metadata is missing, the function stops execution and returns an error message indicating the issue.

Value

The function return the updated comparison names if the checks pass.

Examples

```
data(liver_example)
compNames <- "all"
testNames <- c("limma", "t_test")
check_stat_names(liver_example, compNames, testNames)
```

`create_pairwise_comparisons` *Create All Pairwise Comparisons*

Description

This function generates a matrix of all pairwise comparisons between the provided conditions. Optionally, a reference condition can be specified, and comparisons will be made between the reference condition and all other conditions.

Usage

```
create_pairwise_comparisons(conditions, refCond)
```

Arguments

`conditions`	A character vector of condition names.
`refCond`	An integer indicating the index of the reference condition within the `conditions` vector. If `refCond` is greater than 0, comparisons are made between the reference condition and all other conditions. If `refCond` is 0, all possible pairwise comparisons are made. Default is 0.

Value

A matrix with two columns, representing all possible pairwise comparisons between the specified conditions. If a reference condition is specified, it appears in the first column of every row.

Examples

```
conditions <- c("Cond1", "Cond2", "Cond3")
# Generate all pairwise comparisons
allComps <- create_pairwise_comparisons(conditions, refCond = 0)
allComps

# Generate comparisons with a specific condition as reference
refComps <- create_pairwise_comparisons(conditions, refCond = 1)
refComps
```

`FindFCandQlim` *Find Optimal Fold-Change and Q-Value Thresholds*

Description

Identifies the optimal fold-change (FC) and q-value thresholds that maximize the number of significant features identified across different statistical tests. It processes a matrix of q-values, applying various fold-change thresholds, and computes the distribution of significant features for each q-value threshold to determine the optimal combination of thresholds.

Usage

```
FindFCandQlim(Qvalue, LogRatios)
```

Arguments

`Qvalue`	A matrix of q-values obtained from statistical tests such as PolySTest, limma, rank products, and permutation tests. NA values in the matrix are treated as non-significant. The matrix should have the same number of rows as the LogRatios matrix, and the number of columns should be a multiple of the number of conditions, given by the number of different statistical tests.
`LogRatios`	A matrix of log2 fold-change ratios for the same comparisons.

Details

The function first replaces NA values in the Qvalue matrix with 1 to denote non-significant results. It then identifies the smallest q-value and calculates a range of q-values around this minimum. For each fold-change threshold, the function adjusts the Qvalue matrix and calculates the number of significant features for each q-value threshold. The combination yielding the highest mean distribution of significant features is considered optimal.

Value

A numeric vector containing two elements: the optimal fold-change threshold and the optimal q-value threshold, which together maximize the number of significant features detected.

Examples

```
# Example Qvalue and LogRatios matrices
Qvalue <- matrix(seq(0.01, 0.12, 0.01), nrow = 4, ncol = 3)
LogRatios <- matrix(c(
    1.2, 0.8, 1.5, -0.5, 0.2, 0.9, -1.1, 0.7,
    1.8, -0.9, 0.3, 1.1
), nrow = 4, ncol = 3)
# Find optimal thresholds
thresholds <- FindFCandQlim(Qvalue, LogRatios)
print(thresholds)
```

FindFCandQlimAlternative

Function to determine "optimal" fold-change and q-value thresholds using a higher criticism method and an FC threshold based on a standard deviation of one

Description

Function to determine "optimal" fold-change and q-value thresholds using a higher criticism method and an FC threshold based on a standard deviation of one

Usage

```
FindFCandQlimAlternative(Pvalue, LogRatios)
```

Arguments

Pvalue	A matrix of p-values for each condition
LogRatios	A matrix of log ratios for each condition

Value

A vector containing the mean fold-change threshold and mean q-value threshold

Examples

```
Pvalue <- matrix(seq(0.01, 0.12, 0.01), nrow = 4, ncol = 3)
LogRatios <- matrix(c(
    1.2, 0.8, 1.5, -0.5, 0.2, 0.9, -1.1, 0.7, 1.8,
    -0.9, 0.3, 1.1
), nrow = 4, ncol = 3)
thresholds <- FindFCandQlimAlternative(Pvalue, LogRatios)
print(thresholds)
```

get_numthreads *Set number of threads (default is 4)*

Description

This function sets the number of threads for parallel processing. If the environment variable SHINY_THREADS is set, the number of threads is set to the value of SHINY_THREADS.

Usage

```
get_numthreads(threads = NULL)
```

Arguments

threads	An integer indicating the number of threads to use.

Value

An integer indicating the number of threads to use.

Examples

```
get_numthreads(threads = 4)
get_numthreads()
```

`limma_paired` *Perform paired limma analysis*

Description

This function performs paired limma analysis on MAData.

Usage

```
limma_paired(MAData, NumCond, NumReps)
```

Arguments

`MAData`	A ratio matrix of gene expression data. The rows are genes and the columns are samples. Replicates must be grouped together.
`NumCond`	The number of ratios to check vs zero level
`NumReps`	The number of replicates per condition.

Value

A list containing the p-values and q-values.

Examples

```
MAData <- matrix(rnorm(600), nrow = 100)
NumCond <- 3
NumReps <- 2
limma_res <- limma_paired(MAData, NumCond, NumReps)
head(limma_res$qlvalues)
```

limma_unpaired *Perform unpaired limma analysis*

Description

This function performs unpaired limma analysis on Data.

Usage

```
limma_unpaired(Data, NumCond, NumReps, RRCateg)
```

Arguments

Data	A matrix of quantitative expression data.
NumCond	The number of conditions in the experiment.
NumReps	The number of replicates per condition.
RRCateg	A matrix specifying the conditons to be compared (each comparison given as separate column).

Value

A list containing the following results:

- plvalues: The p-values from limma tests.
- qlvalues: The q-values from limma tests.
- Sds: The standard deviations of the Bayesian linear model. @details This function performs unpaired limma analysis on Data. It calculates the p-values and q-values for each row, indicating the significance of the difference between the two datasets.RRCateg is a matrix specifying the conditons to be compared (each comparison given as separate column). It thus has always 2 rows and n columns, where n is the number of comparisons. The function returns a list containing the p-values and q-values for each comparison, as well as the standard deviations of the Bayesian linear model.

Examples

```
dataMatrix <- matrix(rnorm(900), ncol = 9)
NumCond <- 3
NumReps <- 3
colnames(dataMatrix) <- rep(c("A", "B", "C"), each = 3)
#  Specifying comparisons
RRCateg <- matrix(c(1, 2, 2, 3), nrow = 2, ncol = 2)
# Run function
results <- limma_unpaired(dataMatrix, NumCond, NumReps, RRCateg)
print(results$plvalues)
```

liver_example *Example data set liver_example for PolySTest*

Description

This data set is a SummarizedExperiment object containing liver proteomics of mice fed with four different diets. The data contains 3 replicates per diet (condition). The rowData of the object contains the output from running the PolySTest statistical tests on the data.

Format

A SummarizedExperiment object

Value

A SummarizedExperiment object

Source

Protein expressions from the livers of mice fed with different diets: high fat, TTA component, Fish oil and TTA + fish oil

Examples

```
data(liver_example)
```

MissingStats *Calculates the p-values and q-values for missing values in a data frame with columns as samples and rows as features.*

Description

Calculates the p-values and q-values for missing values in a data frame with columns as samples and rows as features.

Usage

```
MissingStats(Data, NumCond, NumReps)
```

Arguments

Data	A matrix containing the expression data with rows as features and columns as samples
NumCond	The number of conditions in the dataset
NumReps	The number of replicates for each condition (needs to be same for all conditions)

Value

A list containing the calculated p-values and q-values (Benjamini-Hochberg) for missingness statistics

Examples

```
Data <- matrix(rnorm(120), nrow = 10)
# Introduce some missingness
Data[sample(1:120, 40)] <- NA
NumCond <- 4
NumReps <- 3
res_misstest <- MissingStats(Data, NumCond, NumReps)
head(res_misstest$qNAvalues)
```

`MissingStatsDesign` *Function of calculation of Miss tests. This happens between full groups and thus does not have distinction for pairwise testing.*

Description

Function of calculation of Miss tests. This happens between full groups and thus does not have distinction for pairwise testing.

Usage

```
MissingStatsDesign(Data, RRCateg, NumCond, NumReps)
```

Arguments

`Data`	A matrix containing the expression data with rows as features and columns as samples
`RRCateg`	A matrix containing the indices of the conditions to compare (each row is a pairing)
`NumCond`	The number of conditions in the dataset
`NumReps`	The number of replicates for each condition (needs to be same for all conditions)

Value

A list containing the calculated p-values and q-values (Benjamini-Hochbeg) for missingness statistics

Examples

```
Data <- matrix(rnorm(120), nrow = 10)
# Introduce some missingness
Data[sample(seq_len(120), 40)] <- NA
RRCateg <- matrix(c(1, 2, 1, 3, 1, 4, 2, 3, 2, 4, 3, 4), nrow = 2, ncol = 6)
NumCond <- 4
NumReps <- 3
res_misstest <- MissingStatsDesign(Data, RRCateg, NumCond, NumReps)
head(res_misstest$qNAvalues)
```

MissValPDistr *Calculate the distribution of missing values*

Description

This function calculates the distribution of missing values for a given number of repetitions and percentage of missing values.

Usage

```
MissValPDistr(NumReps, PercNA)
```

Arguments

NumReps	An integer indicating the number of repetitions.
PercNA	A numeric value indicating the percentage of missing values.

Value

A numeric vector containing the distribution of missing values.

Examples

```
MissValPDistr(10, 0.2)
```

permtest_paired *Perform permutation tests*

Description

This function performs permutation tests on the given tMAData The permutation tests determine an empirical null distribution of t-values for the p-value calculation

Usage

```
permtest_paired(tMAData)
```

Arguments

tMAData	A matrix of data for running permutation tests

Value

A list containing the p-values and q-values (Benjamini-Hochberg)

Examples

```
tMAData <- matrix(rnorm(100), nrow = 10)
tout <- permtest_paired(tMAData)
head(tout$qPermutvalues)
```

perm_unpaired *Perform unpaired permutation tests*

Description

This function performs permutation testing for unpaired data. The permutation testing is based on comparing the t-values of the real data with the t-values of the permuted data.

Usage

```
perm_unpaired(tData, trefData)
```

Arguments

tData	The data matrix for the test group (features are rows).
trefData	The data matrix for the reference group (features are rows).

Details

The function adds columns from the randomized full set to reach the minimum number of permutation columns (NumPermCols) replicates. It randomizes the sign as well to avoid tendencies to one or the other side. In the unpaired case, it also normalizes by the mean of the entire sample to avoid strange effects. The function then performs permutation testing using parallel computing, and calculates the p-values and q-values based on the permutation results. Both groups needs to consist of the same number of samples (columns).

Value

A list containing the p-values and q-values for the permutation test.

Examples

```
tData <- matrix(rnorm(1000), nrow = 100)
trefData <- matrix(rnorm(1000), nrow = 100)
result <- perm_unpaired(tData, trefData)
```

plotExpression *Plot Expression Profiles*

Description

This function plots expression profiles for selected features across different conditions and comparisons. It supports both scaling and unscaled profiles. It adds a circular plot to compare the different statistical tests

Usage

```
plotExpression(
  fulldata,
  compNames = "all",
 testNames = c("PolySTest", "limma", "Miss Test", "rank products", "permutation test",
    "t-test"),
  sel_prots = "all",
  profiles_scale = TRUE,
  qlim = 0.05,
  fclim = c(0, 0)
)
```

Arguments

fulldata	A SummarizedExperiment object containing the data.
compNames	A character vector of comparison names. "all" selects all comparisons.
testNames	A character vector of test names used in the analysis. Default values are "PolySTest", "limma", "Miss test", "rank products", "permutation test", and "t-test".
sel_prots	A numeric vector with the indices of the selected features. Default is "all". These will still be filterd
profiles_scale	Logical indicating if profiles should be scaled. Default is TRUE.
qlim	A numeric value indicating the q-value limit for significance.
fclim	A numeric vector of length 2 indicating fold-change limits.

Value

Plots expression profiles for the selected features.

Examples

```
data(liver_example)
compNames <- c("HF.Rep._vs_TTA.Rep.")
plotExpression(liver_example)
```

---

plotHeatmaply *Heatmap Visualization with Heatmaply*

Description

This function generates a heatmap for selected features across comparisons using the heatmaply package. It provides options for scaling and saving the plot to a file.

Usage

```
plotHeatmaply(
  fulldata,
  sel_prots = "all",
  heatmap_scale = "none",
  file = NULL,
  ...
)
```

Arguments

fulldata	A SummarizedExperiment object containing the dataset.
sel_prots	Character vector specifying selected features to include in the heatmap or "all" to include all proteins.
heatmap_scale	Character, indicating if and how the data should be scaled. Possible values are "none", "row", or "column".
file	Optional character string specifying the path to save the heatmap plot. If NULL, the plot is rendered interactively.
...	Arguments passed further to heatmaply function

Value

A plotly object if file is NULL. Otherwise, the heatmap is saved to the specified file.

Examples

```
data(liver_example)
plotHeatmaply(
    fulldata = liver_example, sel_prots = "all",
    heatmap_scale = "row"
)
```

---

plotPvalueDistr *Plot P-Value Distributions*

Description

Generates histograms of p-value distributions for each test and comparison.

Usage

```
plotPvalueDistr(
  fulldata,
  compNames = "all",
 testNames = c("limma", "Miss Test", "rank products", "permutation test", "t-test"),
 testCols = c("#33AAAA", "#33AA33", "#AA3333", "#AA33AA", "#AAAA33", "#3333AA"),
  ...
)
```

Arguments

fulldata	A SummarizedExperiment object containing the dataset and FDR/q-values from PolySTest.
compNames	A character vector of comparison names. "all" selects all comparisons.
testNames	A character vector of test names used in the analysis. Default values are "PolySTest", "limma", "Miss test", "rank products", "permutation test", and "t-test".
testCols	A character vector of colors for each test. Defaults to c("#33AAAA", "#33AA33", "#AA3333", "#AA33AA", "#AAAA33", "#3333AA").
...	Additional arguments passed to hist().

Value

Creates histograms of p-value distributions for the specified tests

Examples

```
# Assuming `fulldata` is a properly prepared `SummarizedExperiment` object
data(liver_example)
plotPvalueDistr(liver_example,
    compNames = c("HF.Rep._vs_TTA.Rep."),
    testCols = rainbow(5)
)
```

`plotRegNumber` *Plot Number of Regulated Features*

Description

This function plots the mfrnumber of regulated features across comparisons for different statistical tests. It shows how the number of significant features varies with different FDR thresholds.

Usage

```
plotRegNumber(
  fulldata,
  compNames = "all",
  testNames = c("PolySTest", "limma", "Miss Test", "rank products", "permutation test",
    "t-test"),
  qlim = 0.05,
  fclim = c(0, 0),
  TestCols = c("#33AAAA", "#33AA33", "#AA3333", "#AA33AA", "#AAAA33", "#3333AA"),
  ...
)
```

Arguments

`fulldata`	A `SummarizedExperiment` object containing the dataset.
`compNames`	A character vector of comparison names. "all" selects all comparisons.
`testNames`	A character vector of test names used in the analysis. Default values are "PolySTest", "limma", "Miss test", "rank products", "permutation test", and "t-test".
`qlim`	Numeric, q-value (FDR) threshold.
`fclim`	Numeric vector, fold-change limits.
`TestCols`	Character vector, colors to use for each test in the plot.
`...`	Arguments passed further to plot/lines calls

Value

Invisible. The function generates plots.

Examples

```
data(liver_example)
plotRegNumber(fulldata = liver_example, NumComps = 3)
```

`plotUpset` *Plot UpSet*

Description

Visualizes the intersections of significant features across multiple comparisons using an UpSet plot. Summarizes all comparisons from all tests

Usage

```
plotUpset(fulldata, qlim = 0.05, fclim = c(0, 0))
```

Arguments

`fulldata`	A `SummarizedExperiment` object containing the data.
`qlim`	A numeric value, the q-value threshold for significance.
`fclim`	A numeric vector, specifying fold change limits for filtering.

Value

An UpSet plot visualizing the intersections of significant features.

Examples

```
data(liver_example)

plotUpset(liver_example, qlim = 0.05)
```

`plotVolcano` *Plot Volcano Plots for PolySTest results*

Description

This function creates volcano plots for all specified statistical tests and comparisons using data from a `SummarizedExperiment` object. It highlights selected proteins and applies fold-change and q-value limits for visualization.

Usage

```
plotVolcano(
  fulldata,
  compNames = "all",
 testNames = c("PolySTest", "limma", "Miss Test", "rank products", "permutation test",
    "t-test"),
  sel_prots = "all",
  qlim = 0.05,
  fclim = c(0, 0),
 testCols = c("#33AAAA", "#33AA33", "#AA3333", "#AA33AA", "#AAAA33", "#3333AA"),
  ...
)
```

Arguments

fulldata	A SummarizedExperiment object containing the data.
compNames	A character vector of comparison names.
testNames	A character vector of test names including "PolySTest", "limma", "Miss test", "rank products", "permutation test", and "t-test".
sel_prots	A numeric vector indicating selected features to be visualized differently or "all" to select all features. Default is "all".
qlim	A numeric value setting the q-value limit for the plots. Default is 0.05.
fclim	A numeric vector of length two setting the fold-change limits for the plots. Default is c(0,0).
testCols	A character vector of colors for each test. Default is a predefined set of colors.
...	Additional arguments passed to the plot function.

Value

Creates volcano plots for the specified tests and comparisons.

Examples

```
data(liver_example)
compNames <- c("HF.Rep._vs_TTA.Rep.")
plotVolcano(liver_example, compNames)
```

PolySTest_paired *PolySTest for Paired Tests*

Description

Integrates various statistical tests for analyzing paired data within a SummarizedExperiment framework. It compares pairs of conditions specified by the user to calculate p-values and q-values for each comparison.

Usage

```
PolySTest_paired(
  fulldata,
  allComps,
 statTests = c("limma", "Miss_Test", "t_test", "rank_products", "permutation_test")
)
```

Arguments

fulldata	A SummarizedExperiment or derived object containing the quantitative data required for PolySTest analysis.
allComps	A matrix specifying pairs of conditions to compare. Each row represents a pair for comparison.
statTests	A character vector specifying the statistical tests to be applied. Available tests include "limma", "Miss_Test", "t-test", "rank_products", and "permutation_test". The function will perform each specified test and integrate the results.

Details

Executes specified statistical tests on the dataset contained in fulldata using the condition pairs outlined in allComps. Calculates p-values and q-values for each genomic feature (e.g., genes, proteins) included in the analysis. Results are added to the rowData of the SummarizedExperiment object, enhancing it with detailed statistics from the paired tests.

Value

A SummarizedExperiment object augmented with p-values and q-values in its rowData, reflecting the outcomes of the specified statistical analyses.

Examples

```
library(SummarizedExperiment)

# Mock quantitative data and metadata for samples
quantData <- matrix(rnorm(2000), nrow = 200, ncol = 10)
colnames(quantData) <- c(
    paste("Sample", 1:5, "_Condition_A", sep = ""),
    paste("Sample", 1:5, "_Condition_B", sep = "")
)
rownames(quantData) <- paste("Gene", 1:200)
sampleMetadata <- data.frame(Condition = rep(c("A", "B"), each = 5))

# Creating the SummarizedExperiment object
fulldata <- SummarizedExperiment(
    assays = list(quant = quantData),
    colData = sampleMetadata
)
metadata(fulldata) <- list(NumReps = 5, NumCond = 2)

# Specifying pairs of conditions to compare
allComps <- matrix(c("A", "B"), ncol = 2, byrow = TRUE)

# Specify statistical tests to apply
statTests <- c("limma", "t_test", "rank_products")
```

```
# Running PolySTest for paired comparisons
results <- PolySTest_paired(fulldata, allComps, statTests)
```

`PolySTest_unpaired` *PolySTest for unpaired tests*

Description

Combining the power of different statistical tests

Usage

```
PolySTest_unpaired(
  fulldata,
  allComps,
 statTests = c("limma", "Miss_Test", "t_test", "rank_products", "permutation_test")
)
```

Arguments

`fulldata`	A SummarizedExperiment or derived object that contains the quantitative data as required for PolySTest
`allComps`	A matrix containing the reference matrix specifying the pairs of conditions to compare (each comparison given as separate row)
`statTests`	A character vector specifying the statistical tests to be used. The available tests are: "limma", "Miss_Test", "t-test", "rank_products", and "permutation_test"

Details

This function performs unpaired statistical tests on the data in 'fulldata' using the pairs of conditions specified in 'allComps'. It calculates the p-values and q-values for each row for the statistical tests used. The statistical tests available are: limma, Miss_Test, t-test, rank_products, and a permutation_test based on t values. The function returns a SummarizedExperiment object with added columns for p-values and q-values in rowData.

Value

SummarizedExperiment with added columns for p-values and q-values in rowData

Examples

```
# Creating mock quantitative data and sample metadata
library(SummarizedExperiment)
quantData <- matrix(rnorm(2000), nrow = 200, ncol = 10)
colnames(quantData) <- c(
    paste("Sample", seq_len(5), "_Condition_A", sep = ""),
    paste("Sample", seq_len(5), "_Condition_B", sep = "")
)
rownames(quantData) <- paste("Gene", seq_len(200))
sampleMetadata <- data.frame(Condition = rep(c("A", "B"), each = 5))
```

```
# Creating the SummarizedExperiment object
fulldata <- SummarizedExperiment(
    assays = list(quant = quantData),
    colData = sampleMetadata
)
metadata(fulldata) <- list(NumReps = 5, NumCond = 2)

# Specifying pairs of conditions to compare
allComps <- matrix(c("A", "B"), ncol = 2, byrow = TRUE)

# Running the PolySTest_unpaired function
results <- PolySTest_unpaired(fulldata, allComps)
```

RPStats *RPStats Function*

Description

This function calculates the p-values for the RP (Rank Product) statistic based on the input tRPMAData and the number of replicates (NumReps). The statistcs is one-sided, i.e. it only detects up-regulation.

Usage

```
RPStats(tRPMAData, NumReps)
```

Arguments

tRPMAData	A matrix containing the expression data with rows as features and columns as replicates.
NumReps	The number of replicates for each feature.

Value

A numeric vector containing the p-values for the RP statistic.

Examples

```
tRPMAData <- matrix(c(1, 2, 3, 4, 5, 6), nrow = 2, ncol = 3)
NumReps <- 3
RPStats(tRPMAData, NumReps)
```

rp_unpaired *Perform unpaired rank products test*

Description

Perform unpaired rank products test

Usage

```
rp_unpaired(tData, trefData)
```

Arguments

tData	The data matrix for the test group (features are rows).
trefData	The data matrix for the reference group (features are rows).

Details

This function calculates the p-values and q-values using rankd product statistics. The function requires having the same number of samples per group. The function uses parallel computing to speed up the calculations.

Value

A list containing the p-values and q-values.

Examples

```
tData <- matrix(rnorm(1000), nrow = 100)
trefData <- matrix(rnorm(1000), nrow = 100)
rp_unpaired(tData, trefData)
```

set_mfrow *Set Graphics Layout*

Description

Automatically sets the mfrow parameter for par() based on the total number of plots and the maximum number of columns desired.

Usage

```
set_mfrow(num_total, max_col)
```

Arguments

num_total	The total number of plots to display.
max_col	The maximum number of columns for the layout.

Value

Sets the mfrow parameter for par().

Examples

```
# This will set the layout to 2 rows of 3 columns
set_mfrow(num_total = 6, max_col = 3)
for (i in 1:6) hist(1:10)
par(mfrow = c(1, 1))
```

StatsForPermutTest *Calculate statistics for permutation test*

Description

This function calculates the statistics for a permutation test based on the input data.

Usage

```
StatsForPermutTest(Data, Paired)
```

Arguments

Data	A matrix or data frame containing the data for the test.
Paired	A logical value indicating whether the test is paired or not.

Value

A numeric vector containing the calculated statistics.

Examples

```
Data <- matrix(rnorm(100), ncol = 10)
StatsForPermutTest(Data, Paired = FALSE)
```

---

ttest_paired *Perform paired t-tests*

Description

This function performs row-wise paired t-tests on a data frame containing log-fold changes

Usage

```
ttest_paired(tMAData)
```

Arguments

tMAData	A matrix of data for running row-wise t-tests

Value

A list containing the p-values and q-values (qvalue package)

Examples

```
tMAData <- matrix(rnorm(1000), nrow = 100)
tout <- ttest_paired(tMAData)
head(tout$qtvalues)
```

---

ttest_unpaired *Perform unpaired t-tests on two datasets*

Description

Perform unpaired t-tests on two datasets

Usage

```
ttest_unpaired(tData, trefData)
```

Arguments

tData	A matrix or data frame with the quantitative features (via rows) of the first group
trefData	A matrix or data frame with the quantitative features (via rows) of the second group

Details

This function performs unpaired t-tests between corresponding rows of two datasets. It calculates the p-values and q-values for each row, indicating the significance of the difference between the two datasets. We require providing the same number of samples (columns) per group.

Value

A list containing the p-values and q-values for each row

Examples

```
tData <- matrix(rnorm(1000), nrow = 100)
trefData <- matrix(rnorm(1000), nrow = 100)
result <- ttest_unpaired(tData, trefData)
print(result$ptvalues)
print(result$qtvalues)
```

`update_conditions_with_lcs` *Update Conditions with Longest Common Subsequence*

Description

This function iterates over a set of conditions and updates each condition with the longest common subsequence of column names associated with that condition.

Usage

```
update_conditions_with_lcs(fulldata, default = NULL)
```

Arguments

`fulldata`	A SummarizedExperiment or derived object that contains the quantitative data as specified for PolySTest
`default`	A vector of the length of the number of conditions suggesting their names

Value

SummarizedExperiment with updated conditions

Examples

```
library(SummarizedExperiment)
se <- SummarizedExperiment(assays = list(count = matrix(rnorm(200),
    ncol = 10
)))
metadata(se) <- list(NumCond = 2, NumReps = 5)
rownames(colData(se)) <- paste0(
    rep(c("CondA_Rep", "CondB_Rep"), 5),
    rep(seq_len(5), each = 2)
)
default_conditions <- c("Condition_A", "Condition_B")
updated_conditions <- update_conditions_with_lcs(se, default_conditions)
print(colData(updated_conditions))
```

Index