Package 'benchdamic'

April 12, 2022

Type Package

Title Benchmark of differential abundance methods on microbiome data

Version 1.0.0

Description Starting from a microbiome dataset (16S or WMS with absolute count values) it is possible to perform several analysis to assess the performances of many differential abundance detection methods. A basic and standardized version of the main differential abundance analysis methods is supplied but the user can also add his method to the benchmark.
The analyses focus on 4 main aspects: i) the goodness of fit of each method's distributional assumptions on the observed count data, ii) the ability to control the false discovery rate, iii) the within and between method concordances, iv) the truthfulness of the findings if any apriori knowledge is given. Several graphical functions are available for result visualization.

License Artistic-2.0

Encoding UTF-8

Depends R (>= 4.1.0)

Imports stats, stats4, utils, methods, phyloseq, BiocParallel, zinbwave, edgeR, DESeq2, limma, ALDEx2, corncob, SummarizedExperiment, MAST, Seurat, metagenomeSeq, MGLM, ggplot2, RColorBrewer, plyr, ffpe, reshape2, ggdendro, graphics, cowplot

Suggests knitr, rmarkdown, HMP16SData, curatedMetagenomicData, BiocStyle, testthat

VignetteBuilder knitr

LazyData TRUE

RoxygenNote 7.1.2

biocViews Metagenomics, Microbiome, DifferentialExpression, MultipleComparison, Normalization, Preprocessing, Software

BugReports https://github.com/mcalgaro93/benchdamic/issues

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

git_branch RELEASE_3_14

git_last_commit 5175773

git_last_commit_date 2021-10-26

Date/Publication 2022-04-12

Author Matteo Calgaro [aut, cre]

Maintainer Matteo Calgaro <mcalgaro93@gmail.com>

R topics documented:

`addKnowledge` *addKnowledge*

Description

Add a priori knowledge for each feature tested by a method.

Usage

```
addKnowledge(method, priorKnowledge, enrichmentCol, namesCol = NULL)
```

Arguments

method	Output of differential abundance detection method in which DA information is extracted by the getDA function.
priorKnowledge	data.frame (with feature names as row.names) containing feature level metadata.
enrichmentCol	name of the column containing information for enrichment analysis.
namesCol	name of the column containing new names for features (default namesCol = NULL).

Value

A data.frame with a new column containing information for enrichment analysis.

See Also

createEnrichment.

Examples

```
data("ps_plaque_16S")
data("microbial_metabolism")

# Extract genera from the phyloseq tax_table slot
genera <- phyloseq::tax_table(ps_plaque_16S)[, "GENUS"]
# Genera as rownames of microbial_metabolism data.frame
rownames(microbial_metabolism) <- microbial_metabolism$Genus
# Match OTUs to their metabolism
priorInfo <- data.frame(genera,
    "Type" =  microbial_metabolism[genera, "Type"])
# Unmatched genera becomes "Unknown"
unknown_metabolism <- is.na(priorInfo$Type)
priorInfo[unknown_metabolism, "Type"] <- "Unknown"
priorInfo$Type <- factor(priorInfo$Type)
# Add a more informative names column
priorInfo[, "newNames"] <- paste0(rownames(priorInfo), priorInfo[, "GENUS"])

# DA Analysis
# Add scaling factors
ps_plaque_16S <- norm_edgeR(object = ps_plaque_16S, method = "TMM")
# DA analysis
da.limma <- DA_limma(
    object = ps_plaque_16S,
    design = ~ 1 + HMP_BODY_SUBSITE,
    coef = 2,
    norm = "TMM"
)

DA <- getDA(method = da.limma, slot = "pValMat", colName = "adjP",
    type = "pvalue", direction = "logFC", threshold_pvalue = 0.05,
    threshold_logfc = 1, top = NULL)
```

```
# Add a priori information
DA_info <- addKnowledge(method = DA, priorKnowledge = priorInfo,
    enrichmentCol = "Type", namesCol = "newNames")
```

---

areaCAT *areaCAT*

Description

Compute the area between the bisector and the concordance curve.

Usage

```
areaCAT(concordance, plotIt = FALSE)
```

Arguments

concordance	A long format data.frame produced by createConcordance function.
plotIt	Plot the concordance (default plotIt = FALSE).

Value

A long format data.frame object with several columns:

- comparison which indicates the comparison number;
- n_features which indicates the total number of taxa in the comparison dataset;
- method1 which contains the first method name;
- method2 which contains the first method name;
- rank;
- concordance which is defined as the cardinality of the intersection of the top rank elements of each list, divided by rank, i.e. , $(L_{1:rank} \bigcap M_{1:rank})/(rank)$, where L and M represent the lists of the extracted statistics of method1 and method2 respectively;
- heightOver which is the distance between the bisector and the concordance value;
- areaOver which is the cumulative sum of the heightOver value.

See Also

createConcordance and plotConcordance

Examples

```
data(ps_plaque_16S)

# Balanced design for independent samples
my_splits <- createSplits(
    object = ps_plaque_16S, varName =
    "HMP_BODY_SUBSITE", balanced = TRUE, N = 10 # N = 100 suggested
)

# Initialize some limma based methods
my_limma <- set_limma(design = ~ HMP_BODY_SUBSITE, coef = 2,
    norm = c("TMM", "CSSmedian"))

# Set the normalization methods according to the DA methods
my_norm <- setNormalizations(fun = c("norm_edgeR", "norm_CSS"),
    method = c("TMM", "median"))

# Run methods on split datasets
results <- runSplits(split_list = my_splits, method_list = my_limma,
    normalization_list = my_norm, object = ps_plaque_16S)

# Concordance for p-values
concordance_pvalues <- createConcordance(
    object = results, slot = "pValMat", colName = "rawP", type = "pvalue"
)

# Add area over the concordance curve
concordance_area <- areaCAT(concordance = concordance_pvalues)
```

`checkNormalization` *checkNormalization*

Description

Check if the normalization function's name and the method's name to compute normalization/scaling factors are correctly matched.

Usage

```
checkNormalization(fun, method, ...)
```

Arguments

`fun`	a character with the name of normalization function (e.g. "norm_edgeR", "norm_DESeq2", "norm_CSS"...).
`method`	a character with the normalization method (e.g. "TMM", "upperquartile"... if the `fun` is "norm_edgeR").
`...`	other arguments if needed (e.g. for `norm_edgeR` normalizations).

Value

a list object containing the normalization method and its parameters.

See Also

setNormalizations, norm_edgeR, norm_DESeq2, norm_CSS, norm_TSS

Examples

```
# Check if TMM normalization belong to "norm_edgeR"
check_TMM_normalization <- checkNormalization(fun = "norm_edgeR",
    method = "TMM")
```

---

createColors *createColors*

Description

Produce a qualitative set of colors.

Usage

```
createColors(variable)
```

Arguments

variable	character vector or factor variable.

Value

A named vector containing the color codes.

Examples

```
# Given qualitative variable
cond <- factor(c("A", "A", "B", "B", "C", "D"),
    levels = c("A", "B", "C", "D"))

# Associate a color to each level (or unique value, if not a factor)
cond_colors <- createColors(cond)
```

createConcordance *createConcordance*

Description

Compute the between and within method concordances comparing the lists of extracted statistics from the outputs of the differential abundance detection methods.

Usage

```
createConcordance(object, slot = "pValMat", colName = "rawP", type = "pvalue")
```

Arguments

object	Output of differential abundance detection methods. pValMat, statInfo matrices, and method's name must be present (See vignette for detailed information).
slot	A character vector with 1 or number-of-methods-times repeats of the slot names where to extract values for each method (default slot = "pValMat").
colName	A character vector with 1 or number-of-methods-times repeats of the column name of the slot where to extract values for each method (default colName = "rawP").
type	A character vector with 1 or number-of-methods-times repeats of the value type of the column selected where to extract values for each method. Two values are possible: "pvalue" or "logfc" (default type = "pvalue").

Value

A long format data.frame object with several columns:

- comparison which indicates the comparison number;
- n_features which indicates the total number of taxa in the comparison dataset;
- method1 which contains the first method name;
- method2 which contains the first method name;
- rank;
- concordance which is defined as the cardinality of the intersection of the top rank elements of each list, divided by rank, i.e. , $(L_{1:rank} \bigcap M_{1:rank})/(rank)$, where L and M represent the lists of the extracted statistics of method1 and method2 respectively (averaged values between subset1 and subset2).

See Also

extractStatistics and areaCAT.

Examples

```
data(ps_plaque_16S)

# Balanced design for independent samples
my_splits <- createSplits(
    object = ps_plaque_16S, varName =
    "HMP_BODY_SUBSITE", balanced = TRUE, N = 10 # N = 100 suggested
)

# Initialize some limma based methods
my_limma <- set_limma(design = ~ HMP_BODY_SUBSITE, coef = 2,
    norm = c("TMM", "CSSmedian"))

# Set the normalization methods according to the DA methods
my_norm <- setNormalizations(fun = c("norm_edgeR", "norm_CSS"),
    method = c("TMM", "median"))

# Run methods on split datasets
results <- runSplits(split_list = my_splits, method_list = my_limma,
    normalization_list = my_norm, object = ps_plaque_16S)

# Concordance for p-values
concordance_pvalues <- createConcordance(
    object = results, slot = "pValMat", colName = "rawP", type = "pvalue"
)

# Concordance for log fold changes
concordance_logfc <- createConcordance(
    object = results, slot = "statInfo", colName = "logFC", type = "logfc"
)

# Concordance for log fold changes in the first method and p-values in the
# other
concordance_logfc_pvalues <- createConcordance(
    object = results, slot = c("statInfo", "pValMat"),
    colName = c("logFC", "rawP"), type = c("logfc", "pvalue")
)
```

`createEnrichment` *createEnrichment*

Description

Create a `data.frame` object with several information to perform enrichment analysis.

Usage

```
createEnrichment(
  object,
  priorKnowledge,
```

```
  enrichmentCol,
  namesCol = NULL,
  slot = "pValMat",
  colName = "adjP",
  type = "pvalue",
  direction = NULL,
  threshold_pvalue = 1,
  threshold_logfc = 0,
  top = NULL,
  alternative = "greater",
  verbose = FALSE
)
```

Arguments

object	Output of differential abundance detection methods. pValMat, statInfo matrices, and method's name must be present (See vignette for detailed information).
priorKnowledge	data.frame (with feature names as row.names) containing feature level metadata.
enrichmentCol	name of the column containing information for enrichment analysis.
namesCol	name of the column containing new names for features (default namesCol = NULL).
slot	A character vector with 1 or number-of-methods-times repeats of the slot names where to extract values for each method (default slot = "pValMat").
colName	A character vector with 1 or number-of-methods-times repeats of the column name of the slot where to extract values for each method (default colName = "rawP").
type	A character vector with 1 or number-of-methods-times repeats of the value type of the column selected where to extract values for each method. Two values are possible: "pvalue" or "logfc" (default type = "pvalue").
direction	A character vector with 1 or number-of-methods-times repeats of the statInfo's column name containing information about the signs of differential abundance (usually log fold changes) for each method (default direction = NULL).
threshold_pvalue	A single or a numeric vector of thresholds for p-values. If present, features with p-values lower than threshold_pvalue are considered differentially abundant. Set threshold_pvalue = 1 to not filter by p-values.
threshold_logfc	A single or a numeric vector of thresholds for log fold changes. If present, features with log fold change absolute values higher than threshold_logfc are considered differentially abundant. Set threshold_logfc = 0 to not filter by log fold change values.
top	If not null, the top number of features, ordered by p-values or log fold change values, are considered as differentially abundant (default top = NULL).
alternative	indicates the alternative hypothesis and must be one of "two.sided", "greater" or "less". You can specify just the initial letter. Only used in the 2×2 case.
verbose	Boolean to display the kind of extracted values (default verbose = FALSE).

Value

a list of objects for each method. Each list contains:

- `data` a `data.frame` object with DA directions, statistics, and feature names;
- `tables` a list of 2x2 contingency tables;
- `tests` the list of Fisher exact tests' p-values for each contingency table;
- `summaries` a list with the first element of each contingency table and its p-value (for graphical purposes);

See Also

`addKnowledge`, `extractDA`, and `enrichmentTest`.

Examples

```
data("ps_plaque_16S")
data("microbial_metabolism")

# Extract genera from the phyloseq tax_table slot
genera <- phyloseq::tax_table(ps_plaque_16S)[, "GENUS"]
# Genera as rownames of microbial_metabolism data.frame
rownames(microbial_metabolism) <- microbial_metabolism$Genus
# Match OTUs to their metabolism
priorInfo <- data.frame(genera,
    "Type" =  microbial_metabolism[genera, "Type"])
# Unmatched genera becomes "Unknown"
unknown_metabolism <- is.na(priorInfo$Type)
priorInfo[unknown_metabolism, "Type"] <- "Unknown"
priorInfo$Type <- factor(priorInfo$Type)
# Add a more informative names column
priorInfo[, "newNames"] <- paste0(rownames(priorInfo), priorInfo[, "GENUS"])

# Add some normalization/scaling factors to the phyloseq object
my_norm <- setNormalizations(fun = c("norm_edgeR", "norm_CSS"),
    method = c("TMM", "median"))
ps_plaque_16S <- runNormalizations(normalization_list = my_norm,
    object = ps_plaque_16S)

# Initialize some limma based methods
my_limma <- set_limma(design = ~ 1 + HMP_BODY_SUBSITE, coef = 2,
    norm = c("TMM", "CSSmedian"))

# Perform DA analysis
Plaque_16S_DA <- runDA(method_list = my_limma, object = ps_plaque_16S)

# Enrichment analysis
enrichment <- createEnrichment(object = Plaque_16S_DA,
    priorKnowledge = priorInfo, enrichmentCol = "Type", namesCol = "GENUS",
    slot = "pValMat", colName = "adjP", type = "pvalue", direction = "logFC",
    threshold_pvalue = 0.1, threshold_logfc = 1, top = 10, verbose = TRUE)
```

createMocks *createMocks*

Description

Given the number of samples of the dataset from which the mocks should be created, this function produces a `data.frame` object with as many rows as the number of mocks and as many columns as the number of samples. If an odd number of samples is given, the lower even integer will be considered in order to obtain a balanced design for the mocks.

Usage

```
createMocks(nsamples, N = 1000)
```

Arguments

`nsamples`	an integer representing the total number of samples.
`N`	number of mock comparison to generate.

Value

a `data.frame` containing `N` rows and `nsamples` columns (if even). Each cell of the data frame contains the "grp1" or "grp2" characters which represent the mock groups pattern.

Examples

```
# Generate the pattern for 100 mock comparisons for an experiment with 30
# samples
mocks <- createMocks(nsamples = 30, N = 100)
head(mocks)
```

createPositives *createPositives*

Description

Inspect the list of p-values or/and log fold changes from the output of the differential abundance detection methods and count the True Positives (TP) and the False Positives (FP).

Usage

```
createPositives(
  object,
  priorKnowledge,
  enrichmentCol,
  namesCol = NULL,
  slot = "pValMat",
  colName = "adjP",
  type = "pvalue",
  direction = NULL,
  threshold_pvalue = 1,
  threshold_logfc = 0,
  top = NULL,
  alternative = "greater",
  verbose = FALSE,
  TP,
  FP
)
```

Arguments

object	Output of differential abundance detection methods. pValMat, statInfo matrices, and method's name must be present (See vignette for detailed information).
priorKnowledge	data.frame (with feature names as row.names) containing feature level metadata.
enrichmentCol	name of the column containing information for enrichment analysis.
namesCol	name of the column containing new names for features (default namesCol = NULL).
slot	A character vector with 1 or number-of-methods-times repeats of the slot names where to extract values for each method (default slot = "pValMat").
colName	A character vector with 1 or number-of-methods-times repeats of the column name of the slot where to extract values for each method (default colName = "rawP").
type	A character vector with 1 or number-of-methods-times repeats of the value type of the column selected where to extract values for each method. Two values are possible: "pvalue" or "logfc" (default type = "pvalue").
direction	A character vector with 1 or number-of-methods-times repeats of the statInfo's column name containing information about the signs of differential abundance (usually log fold changes) for each method (default direction = NULL).
threshold_pvalue	A single or a numeric vector of thresholds for p-values. If present, features with p-values lower than threshold_pvalue are considered differentially abundant. Set threshold_pvalue = 1 to not filter by p-values.
threshold_logfc	A single or a numeric vector of thresholds for log fold changes. If present, features with log fold change absolute values higher than threshold_logfc are

	considered differentially abundant. Set `threshold_logfc = 0` to not filter by log fold change values.
`top`	If not null, the `top` number of features, ordered by p-values or log fold change values, are considered as differentially abundant (default `top = NULL`).
`alternative`	indicates the alternative hypothesis and must be one of `"two.sided"`, `"greater"` or `"less"`. You can specify just the initial letter. Only used in the 2×2 case.
`verbose`	Boolean to display the kind of extracted values (default `verbose = FALSE`).
`TP`	A list of length-2 vectors. The entries in the vector are the direction ("UP Abundant", "DOWN Abundant", or "non-DA") in the first position, and the level of the enrichment variable (`enrichmentCol`) which is expected in that direction, in the second position.
`FP`	A list of length-2 vectors. The entries in the vector are the direction ("UP Abundant", "DOWN Abundant", or "non-DA") in the first position, and the level of the enrichment variable (`enrichmentCol`) which is not expected in that direction, in the second position.

Value

a `data.frame` object which contains the number of TPs and FPs features for each method and for each threshold of the `top` argument.

See Also

`getPositives`, `plotPositives`.

Examples

```
data("ps_plaque_16S")
data("microbial_metabolism")

# Extract genera from the phyloseq tax_table slot
genera <- phyloseq::tax_table(ps_plaque_16S)[, "GENUS"]
# Genera as rownames of microbial_metabolism data.frame
rownames(microbial_metabolism) <- microbial_metabolism$Genus
# Match OTUs to their metabolism
priorInfo <- data.frame(genera,
    "Type" =  microbial_metabolism[genera, "Type"])
# Unmatched genera becomes "Unknown"
unknown_metabolism <- is.na(priorInfo$Type)
priorInfo[unknown_metabolism, "Type"] <- "Unknown"
priorInfo$Type <- factor(priorInfo$Type)
# Add a more informative names column
priorInfo[, "newNames"] <- paste0(rownames(priorInfo), priorInfo[, "GENUS"])

# Add some normalization/scaling factors to the phyloseq object
my_norm <- setNormalizations(fun = c("norm_edgeR", "norm_CSS"),
    method = c("TMM", "median"))
ps_plaque_16S <- runNormalizations(normalization_list = my_norm,
    object = ps_plaque_16S)
# Initialize some limma based methods
```

```
my_limma <- set_limma(design = ~ 1 + HMP_BODY_SUBSITE, coef = 2,
    norm = c("TMM", "CSSmedian"))

# Perform DA analysis
Plaque_16S_DA <- runDA(method_list = my_limma, object = ps_plaque_16S)

# Count TPs and FPs, from the top 1 to the top 20 features.
# As direction is supplied, features are ordered by "logFC" absolute values.
positives <- createPositives(object = Plaque_16S_DA,
priorKnowledge = priorInfo, enrichmentCol = "Type", namesCol = "newNames",
slot = "pValMat", colName = "rawP", type = "pvalue", direction = "logFC",
threshold_pvalue = 1, threshold_logfc = 0, top = 1:20,
alternative = "greater", verbose = FALSE,
TP = list(c("DOWN Abundant", "Anaerobic"), c("UP Abundant", "Aerobic")),
FP = list(c("DOWN Abundant", "Aerobic"), c("UP Abundant", "Anaerobic")))

# Plot the TP-FP differences for each threshold
plotPositives(positives = positives)
```

`createSplits` *createSplits*

Description

Given the phyloseq object from which the random splits should be created, this function produces a list of 2 `data.frame` objects: `Subset1` and `Subset2` with as many rows as the number of splits and as many columns as the half of the number of samples.

Usage

```
createSplits(object, varName = NULL, paired = NULL, balanced = TRUE, N = 1000)
```

Arguments

`object`	a `phyloseq` object.
`varName`	name of a factor variable with 2 levels.
`paired`	name of the unique subject identifier variable. If specified, paired samples will remain in the same split. (default = NULL).
`balanced`	If `TRUE` a balanced design will be created for the splits. (Ignored if paired is supplied).
`N`	number of splits to generate.

Value

A list of 2 `data.frame` objects: `Subset1` and `Subset2` containing `N` rows and half of the total number of samples columns. Each cell contains a unique sample identifier.

Examples

```
data(ps_plaque_16S)
set.seed(123)

# Balanced design for repeated measures
splits_df <- createSplits(
    object = ps_plaque_16S, varName =
        "HMP_BODY_SUBSITE", paired = "RSID", balanced = TRUE, N = 100
)

# Balanced design for independent samples
splits_df <- createSplits(
    object = ps_plaque_16S, varName =
        "HMP_BODY_SUBSITE", balanced = TRUE, N = 100
)

# Unbalanced design
splits_df <- createSplits(
    object = ps_plaque_16S, varName =
        "HMP_BODY_SUBSITE", balanced = FALSE, N = 100
)
```

`createTIEC` *createTIEC*

Description

Extract the list of p-values from the outputs of the differential abundance detection methods to compute several statistics to study the ability to control the type I error.

Usage

```
createTIEC(object)
```

Arguments

`object`	Output of the differential abundance tests on mock comparisons. Must follow a specific structure with comparison, method, matrix of p-values, and method's name (See vignette for detailed information).

Value

A `list` of `data.frames`:

- `df_pval`3 columns per number_of_features x methods x comparisons rows data.frame. The three columns are called Comparison, pval, and method;
- `df_FPR`5 columns per methods x comparisons rows data.frame. For each set of method and comparison, the proportion of false discoveries, considering 3 threshold (0.01, 0.05, 0.1) are reported;

- df_QQcontains the coordinates to draw the QQ-plot to compare the mean observed p-value distribution across comparisons, with the theoretical uniform distribution;
- df_KS5 columns and methods x comparisons rows data.frame. For each set of method and comparison, the Kolmogorov-Smirnov test statistics and p-values are reported in KS and KS_pval columns respectively.

See Also

createMocks

Examples

```
# Load some data
data(ps_stool_16S)

# Generate the patterns for 10 mock comparison for an experiment
# (N = 1000 is suggested)
mocks <- createMocks(nsamples = phyloseq::nsamples(ps_stool_16S), N = 10)
head(mocks)

# Add some normalization/scaling factors to the phyloseq object
my_norm <- setNormalizations(fun = c("norm_edgeR", "norm_CSS"),
    method = c("TMM", "median"))
ps_stool_16S <- runNormalizations(normalization_list = my_norm,
    object = ps_stool_16S)

# Initialize some limma based methods
my_limma <- set_limma(design = ~ group, coef = 2,
    norm = c("TMM", "CSSmedian"))

# Run methods on mock datasets
results <- runMocks(mocks = mocks, method_list = my_limma,
    object = ps_stool_16S)

# Prepare results for Type I Error Control
TIEC_summary <- createTIEC(results)

# Plot the results
plotFPR(df_FPR = TIEC_summary$df_FPR)
plotQQ(df_QQ = TIEC_summary$df_QQ, zoom = c(0, 0.1))
plotKS(df_KS = TIEC_summary$df_KS)
```

DA_ALDEx2 *DA_ALDEx2*

Description

Fast run for the ALDEx2's differential abundance detection method.

Usage

```
DA_ALDEx2(
  object,
  pseudo_count = FALSE,
  conditions = NULL,
  mc.samples = 128,
  test = c("t", "wilcox"),
  denom = "iqlr",
  norm = c("TMM", "TMMwsp", "RLE", "upperquartile", "posupperquartile", "none",
    "ratio", "poscounts", "iterate", "TSS", "CSSmedian", "CSSdefault"),
  verbose = TRUE
)
```

Arguments

object	phyloseq object.
pseudo_count	add 1 to all counts if TRUE (default pseudo_count = FALSE).
conditions	A character vector. A description of the data structure used for testing. Typically, a vector of group labels. For aldex.glm, use a model.matrix.
mc.samples	An integer. The number of Monte Carlo samples to use when estimating the underlying distributions. Since we are estimating central tendencies, 128 is usually sufficient.
test	A character string. Indicates which tests to perform. "t" runs Welch's t and Wilcoxon tests. "kw" runs Kruskal-Wallace and glm tests. "glm" runs a generalized linear model using a model.matrix. "corr" runs a correlation test using cor.test.
denom	A character string. Indicates which features to retain as the denominator for the Geometric Mean calculation. Using "iqlr" accounts for data with systematic variation and centers the features on the set features that have variance that is between the lower and upper quartile of variance. Using "zero" is a more extreme case where there are many non-zero features in one condition but many zeros in another. In this case the geometric mean of each group is calculated using the set of per-group non-zero features.
norm	name of the normalization method used to compute the normalization factors to use in the differential abundance analysis. If norm is equal to "TMM", "TMMwsp", "RLE", "upperquartile", "posupperquartile", "CSSmedian", "CSSdefault", "TSS" the scaling factors are automatically transformed into normalization factors.
verbose	an optional logical value. If TRUE, information about the steps of the algorithm is printed. Default verbose = TRUE.

Value

A list object containing the matrix of p-values 'pValMat', the matrix of summary statistics for each tag 'statInfo', and a suggested 'name' of the final object considering the parameters passed to the function.

See Also

aldex for the Dirichlet-Multinomial model estimation. Several and more complex tests are present in the ALDEx2 framework.

Examples

```
set.seed(1)
# Create a very simple phyloseq object
counts <- matrix(rnbinom(n = 300, size = 3, prob = 0.5), nrow = 50, ncol = 6)
metadata <- data.frame("Sample" = c("S1", "S2", "S3", "S4", "S5", "S6"),
                       "group" = as.factor(c("A", "A", "A", "B", "B", "B")))
ps <- phyloseq::phyloseq(phyloseq::otu_table(counts, taxa_are_rows = TRUE),
                         phyloseq::sample_data(metadata))
# No use of scaling factors
ps_NF <- norm_edgeR(object = ps, method = "none")
# The phyloseq object now contains the scaling factors:
scaleFacts <- phyloseq::sample_data(ps_NF)[, "NF.none"]
head(scaleFacts)
# Differential abundance
DA_ALDEx2(ps_NF, conditions = "group", test = "t", denom = "iqlr",
    norm = "none")
```

DA_corncob *DA_corncob*

Description

Fast run for corncob differential abundance detection method.

Usage

```
DA_corncob(
  object,
  pseudo_count = FALSE,
  formula,
  phi.formula,
  formula_null,
  phi.formula_null,
  test,
  boot = FALSE,
  coefficient = NULL,
  norm = c("TMM", "TMMwsp", "RLE", "upperquartile", "posupperquartile", "none",
    "ratio", "poscounts", "iterate", "TSS", "CSSmedian", "CSSdefault"),
  verbose = TRUE
)
```

Arguments

object	phyloseq object.
pseudo_count	add 1 to all counts if TRUE (default pseudo_count = FALSE).
formula	an object of class formula without the response: a symbolic description of the model to be fitted to the abundance.
phi.formula	an object of class formula without the response: a symbolic description of the model to be fitted to the dispersion.
formula_null	Formula for mean under null, without response
phi.formula_null	Formula for overdispersion under null, without response
test	Character. Hypothesis testing procedure to use. One of "Wald" or "LRT" (likelihood ratio test).
boot	Boolean. Defaults to FALSE. Indicator of whether or not to use parametric bootstrap algorithm. (See pbWald and pbLRT).
coefficient	The coefficient of interest as a single word formed by the variable name and the non reference level. (e.g.: 'ConditionDisease' if the reference level for the variable 'Condition' is 'control').
norm	name of the normalization method used to compute the normalization factors to use in the differential abundance analysis. If norm is equal to "TMM", "TMMwsp", "RLE", "upperquartile", "posupperquartile", "CSSmedian", "CSSdefault", "TSS" the scaling factors are automatically transformed into normalization factors.
verbose	an optional logical value. If TRUE, information about the steps of the algorithm is printed. Default verbose = TRUE.

Value

A list object containing the matrix of p-values 'pValMat', the matrix of summary statistics for each tag 'statInfo', and a suggested 'name' of the final object considering the parameters passed to the function.

See Also

bbdml and differentialTest for differential abundance and differential variance evaluation.

Examples

```
set.seed(1)
# Create a very simple phyloseq object
counts <- matrix(rnbinom(n = 60, size = 3, prob = 0.5), nrow = 10, ncol = 6)
metadata <- data.frame("Sample" = c("S1", "S2", "S3", "S4", "S5", "S6"),
                       "group" = as.factor(c("A", "A", "A", "B", "B", "B")))
ps <- phyloseq::phyloseq(phyloseq::otu_table(counts, taxa_are_rows = TRUE),
                         phyloseq::sample_data(metadata))
# No use of scaling factors
ps_NF <- norm_edgeR(object = ps, method = "none")
# The phyloseq object now contains the scaling factors:
```

```
scaleFacts <- phyloseq::sample_data(ps_NF)[, "NF.none"]
head(scaleFacts)
# Differential abundance
DA_corncob(object = ps_NF, formula = ~ group, phi.formula = ~ group,
    formula_null = ~ 1, phi.formula_null = ~ group, coefficient = "groupB",
    norm = "none", test = "Wald")
```

DA_DESeq2 *DA_DESeq2*

Description

Fast run for DESeq2 differential abundance detection method.

Usage

```
DA_DESeq2(
  object,
  pseudo_count = FALSE,
  design = NULL,
  contrast = NULL,
  alpha = 0.05,
  norm = c("TMM", "TMMwsp", "RLE", "upperquartile", "posupperquartile", "none",
    "ratio", "poscounts", "iterate", "TSS", "CSSmedian", "CSSdefault"),
  weights,
  verbose = TRUE
)
```

Arguments

object	phyloseq object.
pseudo_count	add 1 to all counts if TRUE (default pseudo_count = FALSE).
design	(Required). A formula which specifies the design of the experiment, taking the form formula(~ x + y + z). That is, a formula with right-hand side only. By default, the functions in this package and DESeq2 will use the last variable in the formula (e.g. z) for presenting results (fold changes, etc.) and plotting. When considering your specification of experimental design, you will want to re-order the levels so that the NULL set is first. For example, the following line of code would ensure that Enterotype 1 is used as the reference sample class in tests by setting it to the first of the factor levels using the relevel function: sample_data(entill)$Enterotype <-relevel(sample_data(entill)$Enterotype,"1")
contrast	character vector with exactly three elements: the name of a factor in the design formula, the name of the numerator level for the fold change, and the name of the denominator level for the fold change.
alpha	the significance cutoff used for optimizing the independent filtering (by default 0.05). If the adjusted p-value cutoff (FDR) will be a value other than 0.05, alpha should be set to that value.

norm	name of the normalization method used to compute the normalization factors to use in the differential abundance analysis. If norm is equal to "TMM", "TMMwsp", "RLE", "upperquartile", "posupperquartile", "CSSmedian", "CSSdefault", "TSS" the scaling factors are automatically transformed into normalization factors.
weights	an optional numeric matrix giving observational weights.
verbose	an optional logical value. If TRUE, information about the steps of the algorithm is printed. Default verbose = TRUE.

Value

A list object containing the matrix of p-values ‘pValMat‘, the dispersion estimates ‘dispEsts‘, the matrix of summary statistics for each tag ‘statInfo‘, and a suggested ‘name‘ of the final object considering the parameters passed to the function.

See Also

phyloseq_to_deseq2 for phyloseq to DESeq2 object conversion, DESeq and results for the differential abundance method.

Examples

```
set.seed(1)
# Create a very simple phyloseq object
counts <- matrix(rnbinom(n = 60, size = 3, prob = 0.5), nrow = 10, ncol = 6)
metadata <- data.frame("Sample" = c("S1", "S2", "S3", "S4", "S5", "S6"),
                       "group" = as.factor(c("A", "A", "A", "B", "B", "B")))
ps <- phyloseq::phyloseq(phyloseq::otu_table(counts, taxa_are_rows = TRUE),
                         phyloseq::sample_data(metadata))
# Calculate the poscounts normalization factors
ps_NF <- norm_DESeq2(object = ps, method = "poscounts")
# The phyloseq object now contains the normalization factors:
scaleFacts <- phyloseq::sample_data(ps_NF)[, "NF.poscounts"]
head(scaleFacts)
# Differential abundance
DA_DESeq2(object = ps_NF, pseudo_count = FALSE, design = ~ group, contrast =
             c("group", "B", "A"), norm = "poscounts")
```

DA_edgeR *DA_edgeR*

Description

Fast run for edgeR differential abundance detection method.

Usage

```
DA_edgeR(
  object,
  pseudo_count = FALSE,
  group_name = NULL,
  design = NULL,
  robust = FALSE,
  coef = 2,
  norm = c("TMM", "TMMwsp", "RLE", "upperquartile", "posupperquartile", "none",
    "ratio", "poscounts", "iterate", "TSS", "CSSmedian", "CSSdefault"),
  weights,
  verbose = TRUE
)
```

Arguments

object	phyloseq object.
pseudo_count	add 1 to all counts if TRUE (default pseudo_count = FALSE).
group_name	character giving the name of the column containing information about experimental group/condition for each sample/library.
design	character or formula to specify the model matrix.
robust	logical, should the estimation of prior.df be robustified against outliers?
coef	integer or character index vector indicating which coefficients of the linear model are to be tested equal to zero.
norm	name of the normalization method used to compute the scaling factors to use in the differential abundance analysis. If norm is equal to "ratio", "poscounts", or "iterate" the normalization factors are automatically transformed into scaling factors.
weights	an optional numeric matrix giving observational weights.
verbose	an optional logical value. If TRUE, information about the steps of the algorithm is printed. Default verbose = TRUE.

Value

A list object containing the matrix of p-values pValMat, the dispersion estimates dispEsts, the matrix of summary statistics for each tag statInfo, and a suggested name of the final object considering the parameters passed to the function.

See Also

DGEList for the edgeR DEG object creation, estimateDisp and estimateGLMRobustDisp for dispersion estimation, and glmQLFit and glmQLFTest for the quasi-likelihood negative binomial model fit.

Examples

```
set.seed(1)
# Create a very simple phyloseq object
counts <- matrix(rnbinom(n = 60, size = 3, prob = 0.5), nrow = 10, ncol = 6)
metadata <- data.frame("Sample" = c("S1", "S2", "S3", "S4", "S5", "S6"),
                       "group" = as.factor(c("A", "A", "A", "B", "B", "B")))
ps <- phyloseq::phyloseq(phyloseq::otu_table(counts, taxa_are_rows = TRUE),
                         phyloseq::sample_data(metadata))

# Calculate the TMM scaling factors
ps_NF <- norm_edgeR(object = ps, method = "TMM")
# The phyloseq object now contains the scaling factors:
scaleFacts <- phyloseq::sample_data(ps_NF)[, "NF.TMM"]
head(scaleFacts)

# Differential abundance
DA_edgeR(object = ps_NF, pseudo_count = FALSE, group_name = "group",
         design = ~ group, coef = 2, robust = FALSE, norm = "TMM")
```

DA_limma *DA_limma*

Description

Fast run for limma voom differential abundance detection method.

Usage

```
DA_limma(
  object,
  pseudo_count = FALSE,
  design = NULL,
  coef = 2,
  norm = c("TMM", "TMMwsp", "RLE", "upperquartile", "posupperquartile", "none",
    "ratio", "poscounts", "iterate", "TSS", "CSSmedian", "CSSdefault"),
  weights,
  verbose = TRUE
)
```

Arguments

object	phyloseq object.
pseudo_count	add 1 to all counts if TRUE (default pseudo_count = FALSE).
design	character name of the metadata columns, formula, or design matrix with rows corresponding to samples and columns to coefficients to be estimated.
coef	integer or character index vector indicating which coefficients of the linear model are to be tested equal to zero.

norm	name of the normalization method used to compute the scaling factors to use in the differential abundance analysis. If norm is equal to "ratio", "poscounts", or "iterate" the normalization factors are automatically transformed into scaling factors.
weights	an optional numeric matrix giving observational weights.
verbose	an optional logical value. If TRUE, information about the steps of the algorithm is printed. Default verbose = TRUE.

Value

A list object containing the matrix of p-values ‘pValMat‘, the matrix of summary statistics for each tag ‘statInfo‘, and a suggested ‘name‘ of the final object considering the parameters passed to the function.

See Also

voom for the mean-variance relationship estimation, lmFit for the linear model framework.

Examples

```
set.seed(1)
# Create a very simple phyloseq object
counts <- matrix(rnbinom(n = 60, size = 3, prob = 0.5), nrow = 10, ncol = 6)
metadata <- data.frame("Sample" = c("S1", "S2", "S3", "S4", "S5", "S6"),
                       "group" = as.factor(c("A", "A", "A", "B", "B", "B")))
ps <- phyloseq::phyloseq(phyloseq::otu_table(counts, taxa_are_rows = TRUE),
                         phyloseq::sample_data(metadata))
# Calculate the TMM scaling factors
ps_NF <- norm_edgeR(object = ps, method = "TMM")
# The phyloseq object now contains the scaling factors:
scaleFacts <- phyloseq::sample_data(ps_NF)[, "NF.TMM"]
head(scaleFacts)
# Differential abundance
DA_limma(object = ps_NF, pseudo_count = FALSE, design = ~ group, coef = 2,
    norm = "TMM")
```

DA_MAST *DA_MAST*

Description

Fast run for MAST differential abundance detection method.

Usage

```
DA_MAST(
  object,
  pseudo_count = FALSE,
  rescale = c("median", "default"),
  design,
  coefficient = NULL,
  norm = c("TMM", "TMMwsp", "RLE", "upperquartile", "posupperquartile", "none",
    "ratio", "poscounts", "iterate", "TSS", "CSSmedian", "CSSdefault"),
  verbose = TRUE
)
```

Arguments

object	phyloseq object.
pseudo_count	add 1 to all counts if TRUE (default pseudo_count = FALSE).
rescale	Rescale count data, per million if 'default', or per median library size if 'median' ('median' is suggested for metagenomics data).
design	The model for the count distribution. Can be the variable name, or a character similar to "~ 1 + group", or a formula, or a 'model.matrix' object.
coefficient	The coefficient of interest as a single word formed by the variable name and the non reference level. (e.g.: 'ConditionDisease' if the reference level for the variable 'Condition' is 'control').
norm	name of the normalization method used to compute the normalization factors to use in the differential abundance analysis. If norm is equal to "TMM", "TMMwsp", "RLE", "upperquartile", "posupperquartile", "CSSmedian", "CSSdefault", "TSS" the scaling factors are automatically transformed into normalization factors.
verbose	an optional logical value. If TRUE, information about the steps of the algorithm is printed. Default verbose = TRUE.

Value

A list object containing the matrix of p-values 'pValMat', the matrix of summary statistics for each tag 'statInfo', and a suggested 'name' of the final object considering the parameters passed to the function.

See Also

zlm for the Truncated Gaussian Hurdle model estimation.

Examples

```
set.seed(1)
# Create a very simple phyloseq object
counts <- matrix(rnbinom(n = 60, size = 3, prob = 0.5), nrow = 10, ncol = 6)
metadata <- data.frame("Sample" = c("S1", "S2", "S3", "S4", "S5", "S6"),
                       "group" = as.factor(c("A", "A", "A", "B", "B", "B")))
```

```
ps <- phyloseq::phyloseq(phyloseq::otu_table(counts, taxa_are_rows = TRUE),
                         phyloseq::sample_data(metadata))
# No use of scaling factors
ps_NF <- norm_edgeR(object = ps, method = "none")
# The phyloseq object now contains the scaling factors:
scaleFacts <- phyloseq::sample_data(ps_NF)[, "NF.none"]
head(scaleFacts)
# Differential abundance
DA_MAST(object = ps_NF, pseudo_count = FALSE, rescale = "median",
    design = ~ group, norm = "none", coefficient = "groupB")
```

DA_metagenomeSeq *DA_metagenomeSeq*

Description

Fast run for the metagenomeSeq's differential abundance detection method.

Usage

```
DA_metagenomeSeq(
  object,
  pseudo_count = FALSE,
  design = NULL,
  coef = 2,
  norm = c("TMM", "TMMwsp", "RLE", "upperquartile", "posupperquartile", "none",
    "ratio", "poscounts", "iterate", "TSS", "CSSmedian", "CSSdefault"),
  verbose = TRUE
)
```

Arguments

object	phyloseq object.
pseudo_count	add 1 to all counts if TRUE (default pseudo_count = FALSE).
design	The model for the count distribution. Can be the variable name, or a character similar to "~ 1 + group", or a formula, or a 'model.matrix' object.
coef	integer or character index vector indicating which coefficients of the linear model are to be tested equal to zero.
norm	name of the normalization method used to compute the scaling factors to use in the differential abundance analysis. If norm is equal to "ratio", "poscounts", or "iterate" the normalization factors are automatically transformed into scaling factors.
verbose	an optional logical value. If TRUE, information about the steps of the algorithm is printed. Default verbose = TRUE.

Value

A list object containing the matrix of p-values ‘pValMat‘, the matrix of summary statistics for each tag ‘statInfo‘, and a suggested ‘name‘ of the final object considering the parameters passed to the function.

See Also

`fitZig` for the Zero-Inflated Gaussian regression model estimation and `MRfulltable` for results extraction.

Examples

```
set.seed(1)
# Create a very simple phyloseq object
counts <- matrix(rnbinom(n = 60, size = 3, prob = 0.5), nrow = 10, ncol = 6)
metadata <- data.frame("Sample" = c("S1", "S2", "S3", "S4", "S5", "S6"),
                       "group" = as.factor(c("A", "A", "A", "B", "B", "B")))
ps <- phyloseq::phyloseq(phyloseq::otu_table(counts, taxa_are_rows = TRUE),
                         phyloseq::sample_data(metadata))
# Calculate the CSSdefault scaling factors
ps_NF <- norm_CSS(object = ps, method = "default")
# The phyloseq object now contains the scaling factors:
scaleFacts <- phyloseq::sample_data(ps_NF)[, "NF.CSSdefault"]
head(scaleFacts)
# Differential abundance
DA_metagenomeSeq(object = ps_NF, pseudo_count = FALSE, design = ~ group,
    coef = 2, norm = "CSSdefault")
```

DA_Seurat *DA_Seurat*

Description

Fast run for Seurat differential abundance detection method.

Usage

```
DA_Seurat(
  object,
  pseudo_count = FALSE,
  test.use = "wilcox",
  contrast,
  norm = c("TMM", "TMMwsp", "RLE", "upperquartile", "posupperquartile", "none",
    "ratio", "poscounts", "iterate", "TSS", "CSSmedian", "CSSdefault"),
  verbose = TRUE
)
```

Arguments

`object`	phyloseq object.
`pseudo_count`	add 1 to all counts if TRUE (default `pseudo_count = FALSE`).
`test.use`	Denotes which test to use. Available options are: • "wilcox" : Identifies differentially expressed genes between two groups of cells using a Wilcoxon Rank Sum test (default) • "bimod" : Likelihood-ratio test for single cell gene expression, (McDavid et al., Bioinformatics, 2013) • "roc" : Identifies 'markers' of gene expression using ROC analysis. For each gene, evaluates (using AUC) a classifier built on that gene alone, to classify between two groups of cells. An AUC value of 1 means that expression values for this gene alone can perfectly classify the two groupings (i.e. Each of the cells in cells.1 exhibit a higher level than each of the cells in cells.2). An AUC value of 0 also means there is perfect classification, but in the other direction. A value of 0.5 implies that the gene has no predictive power to classify the two groups. Returns a 'predictive power' (abs(AUC-0.5) * 2) ranked matrix of putative differentially expressed genes. • "t" : Identify differentially expressed genes between two groups of cells using the Student's t-test. • "negbinom" : Identifies differentially expressed genes between two groups of cells using a negative binomial generalized linear model. Use only for UMI-based datasets • "poisson" : Identifies differentially expressed genes between two groups of cells using a poisson generalized linear model. Use only for UMI-based datasets • "LR" : Uses a logistic regression framework to determine differentially expressed genes. Constructs a logistic regression model predicting group membership based on each feature individually and compares this to a null model with a likelihood ratio test. • "MAST" : Identifies differentially expressed genes between two groups of cells using a hurdle model tailored to scRNA-seq data. Utilizes the MAST package to run the DE testing. • "DESeq2" : Identifies differentially expressed genes between two groups of cells based on a model using DESeq2 which uses a negative binomial distribution (Love et al, Genome Biology, 2014).This test does not support pre-filtering of genes based on average difference (or percent detection rate) between cell groups. However, genes may be pre-filtered based on their minimum detection rate (min.pct) across both cell groups. To use this method, please install DESeq2, using the instructions at https://bioconductor.org/packages/release/bioc/html/D
`contrast`	character vector with exactly three elements: the name of a factor in the design formula, the name of the numerator level for the fold change, and the name of the denominator level for the fold change.
`norm`	name of the normalization method used to compute the normalization factors to use in the differential abundance analysis. If `norm` is equal to "TMM", "TMMwsp", "RLE", "upperquartile", "posupperquartile", "CSSmedian", "CSSdefault", "TSS" the scaling factors are automatically transformed into normalization factors.

verbose — an optional logical value. If TRUE, information about the steps of the algorithm is printed. Default verbose = TRUE.

Value

A list object containing the matrix of p-values ‘pValMat‘, the matrix of summary statistics for each tag ‘statInfo‘, and a suggested ‘name‘ of the final object considering the parameters passed to the function.

See Also

CreateSeuratObject to create the Seurat object, AddMetaData to add metadata information, NormalizeData to compute the normalization for the counts, FindVariableFeatures to estimate the mean-variance trend, ScaleData to scale and center features in the dataset, and FindMarkers to perform differential abundance analysis.

Examples

```
set.seed(1)
# Create a very simple phyloseq object
counts <- matrix(rnbinom(n = 60, size = 3, prob = 0.5), nrow = 10, ncol = 6)
metadata <- data.frame("Sample" = c("S1", "S2", "S3", "S4", "S5", "S6"),
                       "group" = as.factor(c("A", "A", "A", "B", "B", "B")))
ps <- phyloseq::phyloseq(phyloseq::otu_table(counts, taxa_are_rows = TRUE),
                         phyloseq::sample_data(metadata))
# No use of scaling factors
ps_NF <- norm_edgeR(object = ps, method = "none")
# The phyloseq object now contains the scaling factors:
scaleFacts <- phyloseq::sample_data(ps_NF)[, "NF.none"]
head(scaleFacts)
# Differential abundance
DA_Seurat(object = ps_NF, contrast = c("group","B","A"), norm = "none")
```

---

enrichmentTest *enrichmentTest*

Description

Perform the Fisher exact test for all the possible 2x2 contingency tables, considering differential abundance direction and enrichment variable.

Usage

```
enrichmentTest(method, enrichmentCol, alternative = "greater")
```

Arguments

method	Output of differential abundance detection method in which DA information is extracted by the getDA function and the information related to enrichment is appropriately added through the addKnowledge.
enrichmentCol	name of the column containing information for enrichment analysis.
alternative	indicates the alternative hypothesis and must be one of "two.sided", "greater" or "less". You can specify just the initial letter. Only used in the 2×2 case.

Value

a list of objects:

- data a data.frame object with DA directions, statistics, and feature names;
- tables a list of 2x2 contingency tables;
- tests the list of Fisher exact tests' p-values for each contingency table;
- summaries a list with the first element of each contingency table and its p-value (for graphical purposes);

See Also

extractDA, addKnowledge, and createEnrichment

Examples

```
data("ps_plaque_16S")
data("microbial_metabolism")

# Extract genera from the phyloseq tax_table slot
genera <- phyloseq::tax_table(ps_plaque_16S)[, "GENUS"]
# Genera as rownames of microbial_metabolism data.frame
rownames(microbial_metabolism) <- microbial_metabolism$Genus
# Match OTUs to their metabolism
priorInfo <- data.frame(genera,
    "Type" =  microbial_metabolism[genera, "Type"])
# Unmatched genera becomes "Unknown"
unknown_metabolism <- is.na(priorInfo$Type)
priorInfo[unknown_metabolism, "Type"] <- "Unknown"
priorInfo$Type <- factor(priorInfo$Type)
# Add a more informative names column
priorInfo[, "newNames"] <- paste0(rownames(priorInfo), priorInfo[, "GENUS"])

# DA Analysis
# Add scaling factors
ps_plaque_16S <- norm_edgeR(object = ps_plaque_16S, method = "TMM")
# DA analysis
da.limma <- DA_limma(
    object = ps_plaque_16S,
    design = ~ 1 + HMP_BODY_SUBSITE,
    coef = 2,
    norm = "TMM"
```

```
)

DA <- getDA(method = da.limma, slot = "pValMat", colName = "adjP",
    type = "pvalue", direction = "logFC", threshold_pvalue = 0.05,
    threshold_logfc = 1, top = NULL)
# Add a priori information
DA_info <- addKnowledge(method = DA, priorKnowledge = priorInfo,
    enrichmentCol = "Type", namesCol = "newNames")
# Create contingency tables and compute F tests
DA_info_enriched <- enrichmentTest(method = DA_info, enrichmentCol = "Type",
    alternative = "greater")
```

`extractDA` *extractDA*

Description

Inspect the list of p-values or/and log fold changes from the output of differential abundance detection methods.

Usage

```
extractDA(
  object,
  slot = "pValMat",
  colName = "adjP",
  type = "pvalue",
  direction = NULL,
  threshold_pvalue = 1,
  threshold_logfc = 0,
  top = NULL,
  verbose = FALSE
)
```

Arguments

`object`	Output of differential abundance detection methods. `pValMat`, `statInfo` matrices, and method's `name` must be present (See vignette for detailed information).
`slot`	A character vector with 1 or number-of-methods-times repeats of the slot names where to extract values for each method (default `slot = "pValMat"`).
`colName`	A character vector with 1 or number-of-methods-times repeats of the column name of the slot where to extract values for each method (default `colName = "rawP"`).
`type`	A character vector with 1 or number-of-methods-times repeats of the value type of the column selected where to extract values for each method. Two values are possible: `"pvalue"` or `"logfc"` (default `type = "pvalue"`).

direction
: A character vector with 1 or number-of-methods-times repeats of the `statInfo`'s column name containing information about the signs of differential abundance (usually log fold changes) for each method (default `direction = NULL`).

threshold_pvalue
: A single or a numeric vector of thresholds for p-values. If present, features with p-values lower than `threshold_pvalue` are considered differentially abundant. Set `threshold_pvalue = 1` to not filter by p-values.

threshold_logfc
: A single or a numeric vector of thresholds for log fold changes. If present, features with log fold change absolute values higher than `threshold_logfc` are considered differentially abundant. Set `threshold_logfc = 0` to not filter by log fold change values.

top
: If not null, the `top` number of features, ordered by p-values or log fold change values, are considered as differentially abundant (default `top = NULL`).

verbose
: Boolean to display the kind of extracted values (default `verbose = FALSE`).

Value

A `data.frame` with several columns for each method:

- `stat` which contains the p-values or the absolute log fold change values;
- `direction` which is present if `direction` was supplied, it contains the information about directionality of differential abundance (usually log fold changes);
- `DA` which can contain several values according to thresholds and inputs. `"DA"` or `"non-DA"` if `direction = NULL`, `"UP Abundant"`, `"DOWN Abundant"`, or `"non-DA"` otherwise.

See Also

`getDA`, `extractStatistics`

Examples

```
data("ps_plaque_16S")
# Add scaling factors
my_norm <- setNormalizations(fun = c("norm_edgeR", "norm_CSS"),
    method = c("TMM", "median"))
ps_plaque_16S <- runNormalizations(normalization_list = my_norm,
    object = ps_plaque_16S)
# Perform DA analysis
my_methods <- set_limma(design = ~ 1 + HMP_BODY_SUBSITE, coef = 2,
    norm = c("TMM", "CSSmedian"))
Plaque_16S_DA <- runDA(method_list = my_methods, object = ps_plaque_16S)
# Top 10 features (ordered by 'direction') are DA
DA_1 <- extractDA(
    object = Plaque_16S_DA, slot = "pValMat", colName = "adjP",
    type = "pvalue", direction = "logFC", threshold_pvalue = 1,
    threshold_logfc = 0, top = 10
)
# Features with p-value < 0.05 and |logFC| > 1 are DA
```

```
DA_2 <- extractDA(
    object = Plaque_16S_DA, slot = "pValMat", colName = "adjP",
    type = "pvalue", direction = "logFC", threshold_pvalue = 0.05,
    threshold_logfc = 1, top = NULL
)
```

`extractStatistics` *extractStatistics*

Description

Extract the list of p-values or/and log fold changes from the outputs of the differential abundance detection methods.

Usage

```
extractStatistics(
  object,
  slot = "pValMat",
  colName = "rawP",
  type = "pvalue",
  direction = NULL,
  verbose = FALSE
)
```

Arguments

`object`	Output of differential abundance detection methods. `pValMat`, `statInfo` matrices, and method's `name` must be present (See vignette for detailed information).
`slot`	A character vector with 1 or number-of-methods-times repeats of the slot names where to extract values for each method (default `slot = "pValMat"`).
`colName`	A character vector with 1 or number-of-methods-times repeats of the column name of the slot where to extract values for each method (default `colName = "rawP"`).
`type`	A character vector with 1 or number-of-methods-times repeats of the value type of the column selected where to extract values for each method. Two values are possible: `"pvalue"` or `"logfc"` (default `type = "pvalue"`).
`direction`	A character vector with 1 or number-of-methods-times repeats of the `statInfo`'s column name containing information about the signs of differential abundance (usually log fold changes) for each method (default `direction = NULL`).
`verbose`	Boolean to display the kind of extracted values (default `verbose = FALSE`).

Value

A vector or a `data.frame` for each method. If `direction = NULL`, the `colname` column values, transformed according to `type` (not tranformed if `type = "pvalue"`, `-abs(value)` if `type = "logfc"`), of the `slot` are reported , otherwise the `direction` column of the `statInfo` matrix is added to the output.

See Also

getStatistics

Examples

```
data("ps_plaque_16S")
# Add scaling factors
my_norm <- setNormalizations(fun = c("norm_edgeR", "norm_CSS"),
    method = c("TMM", "median"))
ps_plaque_16S <- runNormalizations(normalization_list = my_norm,
    object = ps_plaque_16S)
# Perform DA analysis
my_methods <- set_limma(design = ~ 1 + HMP_BODY_SUBSITE, coef = 2,
    norm = c("TMM", "CSSmedian"))
Plaque_16S_DA <- runDA(method_list = my_methods, object = ps_plaque_16S)
### Extract statistics for concordance analysis:
# Only p-values
extracted_pvalues <- extractStatistics(
    object = Plaque_16S_DA, slot =
        "pValMat", colName = "rawP", type = "pvalue"
)
# Only transformed log fold changes -abs(logFC)
extracted_abslfc <- extractStatistics(
    object = Plaque_16S_DA, slot =
        "statInfo", colName = "logFC", type = "logfc"
)
# Only transformed log fold changes for a method and p-values for the other
extracted_abslfc_pvalues <- extractStatistics(
    object = Plaque_16S_DA,
    slot = c("statInfo", "pValMat"), colName = c("logFC", "rawP"), type =
        c("logfc", "pvalue")
)
### Extract statistics for enrichment analysis:
# p-values and log fold changes
extracted_pvalues_and_lfc <- extractStatistics(
    object = Plaque_16S_DA,
    slot = "pValMat", colName = "rawP", type = "pvalue", direction = "logFC"
)
# transformed log fold changes and untouched log fold changes
extracted_abslfc_and_lfc <- extractStatistics(
    object = Plaque_16S_DA,
    slot = "statInfo", colName = "logFC", type = "logfc", direction =
        "logFC"
)
# Only transformed log fold changes for a method and p-values for the other
extracted_mix <- extractStatistics(
    object = Plaque_16S_DA,
    slot = c("statInfo", "pValMat"), colName = c("logFC", "rawP"), type =
        c("logfc", "pvalue"), direction = "logFC"
)
```

fitDM *fitDM*

Description

Fit a Dirichlet-Multinomial (DM) distribution for each taxon of the count data. The model estimation procedure is performed by MGLM `MGLMreg` function without assuming the presence of any group in the samples (design matrix equal to a column of ones.)

Usage

```
fitDM(counts, verbose = TRUE)
```

Arguments

`counts`	a phyloseq object or a matrix of counts with features (OTUs, ASVs, genes) by row and samples by column.
`verbose`	an optional logical value. If `TRUE` information on the steps of the algorithm is printed. Default `verbose = TRUE`.

Value

A data frame containing the continuity corrected logarithms of the average fitted values for each row of the matrix of counts in the `Y` column, and the estimated probability to observe a zero in the `Y0` column.

Examples

```
# Generate some random counts
counts = matrix(rnbinom(n = 60, size = 3, prob = 0.5), nrow = 10, ncol = 6)

# Fit model on the counts matrix
DM <- fitDM(counts)
head(DM)
```

fitHURDLE *fitHURDLE*

Description

Fit a truncated gaussian hurdle model for each taxon of the count data. The hurdle model estimation procedure is performed by MAST `zlm` function without assuming the presence of any group in the samples (design matrix equal to a column of ones.)

Usage

```
fitHURDLE(counts, scale = "default", verbose = TRUE)
```

Arguments

counts	a phyloseq object or a matrix of counts with features (OTUs, ASVs, genes) by row and samples by column.
scale	Character vector, either median or default to choose between the median of the library size or one million to scale raw counts.
verbose	an optional logical value. If TRUE information on the steps of the algorithm is printed. Default verbose = TRUE.

Value

A data frame containing the continuity corrected logarithms of the average fitted values for each row of the matrix of counts in the Y column, and the estimated probability to observe a zero in the Y0 column.

Examples

```
# Generate some random counts
counts = matrix(rnbinom(n = 60, size = 3, prob = 0.5), nrow = 10, ncol = 6)

# Fit model on the counts matrix
HURDLE <- fitHURDLE(counts, scale = "median")
head(HURDLE)
```

---

fitModels *fitModels*

Description

A wrapper function that fits the specified models for each taxon of the count data and computes the mean difference (MD) and zero probability difference (ZPD) between estimated and observed values.

Usage

```
fitModels(
  counts,
  models = c("NB", "ZINB", "DM", "ZIG", "HURDLE"),
  scale_ZIG = c("default", "median"),
  scale_HURDLE = c("default", "median"),
  verbose = TRUE
)
```

Arguments

counts	a phyloseq object or a matrix of counts with features (OTUs, ASVs, genes) by row and samples by column.
models	character vector which assumes the values NB, ZINB, DM, ZIG, and HURDLE.
scale_ZIG	character vector, either median or default to choose between the median of the library size or one thousand to scale normalization factors for the zero-inflated gaussian model.
scale_HURDLE	character vector, either median or default to choose between the median of the library size or one million to scale raw counts for the truncated gaussian hurdle model.
verbose	an optional logical value. If TRUE information on the steps of the algorithm is printed. Default verbose = TRUE.

Value

list of data.frame objects for each model. The first two columns contain the properly transformed observed values for mean and zero proportion, while the third and the fourth columns contain the estimated values for the mean and the zero rate respectively.

See Also

fitNB, fitZINB, fitDM, fitZIG, and fitHURDLE for the model estimations, prepareObserved for raw counts preparation, and meanDifferences for the Mean Difference (MD) and Zero Probability Difference (ZPD) computations.

Examples

```
# Generate some random counts
counts <- matrix(rnbinom(n = 60, size = 3, prob = 0.5), nrow = 10, ncol = 6)
# Estimate the counts assuming several distributions
GOF <- fitModels(
    counts = counts, models = c(
        "NB", "ZINB",
        "DM", "ZIG", "HURDLE"
    ), scale_ZIG = c("median", "default"), scale_HURDLE =
        c("median", "default")
)

head(GOF)
```

---

fitNB *fitNB*

Description

Fit a Negative Binomial (NB) distribution for each taxon of the count data. The NB estimation procedure is performed by edgeR `glmFit` function, using `TMM` normalized counts, tag-wise dispersion estimation, and not assuming the presence of any group in the samples (design matrix equal to a column of ones.)

Usage

```
fitNB(counts, verbose = TRUE)
```

Arguments

`counts`	a phyloseq object or a matrix of counts with features (OTUs, ASVs, genes) by row and samples by column.
`verbose`	an optional logical value. If `TRUE` information on the steps of the algorithm is printed. Default `verbose = TRUE`.

Value

A data frame containing the continuity corrected logarithms of the average fitted values for each row of the 'counts' matrix in the 'Y' column, and the estimated probability to observe a zero in the 'Y0' column.

Examples

```
# Generate some random counts
counts = matrix(rnbinom(n = 60, size = 3, prob = 0.5), nrow = 10, ncol = 6)

# Fit model on the matrix of counts
NB <- fitNB(counts)
head(NB)
```

`fitZIG` *fitZIG*

Description

Fit a Zero-Inflated Gaussian (ZIG) distribution for each taxon of the count data. The model estimation procedure is performed by metagenomeSeq `fitZig` function without assuming the presence of any group in the samples (design matrix equal to a column of ones.)

Usage

```
fitZIG(counts, scale = "default", verbose = TRUE)
```

Arguments

counts	a phyloseq object or a matrix of counts with features (OTUs, ASVs, genes) by row and samples by column.
scale	Character vector, either median or default to choose between the median of the library size or one thousand to scale normalization factors.
verbose	an optional logical value. If TRUE information on the steps of the algorithm is printed. Default verbose = TRUE.

Value

A data frame containing the continuity corrected logarithms of the average fitted values for each row of the matrix of counts in the Y column, and the estimated probability to observe a zero in the Y0 column.

Examples

```
# Generate some random counts
counts = matrix(rnbinom(n = 60, size = 3, prob = 0.5), nrow = 10, ncol = 6)

# Fit model on the counts matrix
ZIG <- fitZIG(counts, scale = "median")
head(ZIG)
```

---

fitZINB *fitZINB*

Description

Fit a Zero-Inflated Negative Binomial (ZINB) distribution for each taxon of the countdata. The ZINB estimation procedure is performed by zinbwave zinbFit function with commondispersion = FALSE, regularization parameter epsilon = 1e10, and not assuming the presence of any group in the samples (design matrix equal to a column of ones.)

Usage

```
fitZINB(counts, verbose = TRUE)
```

Arguments

counts	a phyloseq object or a matrix of counts with features (OTUs, ASVs, genes) by row and samples by column.
verbose	an optional logical value. If TRUE information on the steps of the algorithm is printed. Default verbose = TRUE.

Value

A data frame containing the continuity corrected logarithms of the average fitted values for each row of the matrix of counts in the `Y` column, and the estimated probability to observe a zero in the `Y0` column.

Examples

```
# Generate some random counts
counts = matrix(rnbinom(n = 60, size = 3, prob = 0.5), nrow = 10, ncol = 6)

# Fit model on the counts matrix
ZINB <- fitZINB(counts)
head(ZINB)
```

`getDA` *getDA*

Description

Inspect the list of p-values or/and log fold changes from the output of a differential abundance detection method.

Usage

```
getDA(
  method,
  slot = "pValMat",
  colName = "rawP",
  type = "pvalue",
  direction = NULL,
  threshold_pvalue = 1,
  threshold_logfc = 0,
  top = NULL,
  verbose = FALSE
)
```

Arguments

`method`	Output of a differential abundance detection method. `pValMat`, `statInfo` matrices, and method's `name` must be present (See vignette for detailed information).
`slot`	The slot name where to extract values (default `slot = "pValMat"`).
`colName`	The column name of the slot where to extract values (default `colName = "rawP"`).
`type`	The value type of the column selected where to extract values. Two values are possible: `"pvalue"` or `"logfc"` (default `type = "pvalue"`).
`direction`	`statInfo`'s column name containing information about the signs of differential abundance (usually log fold changes) (default `direction = NULL`).

threshold_pvalue
: Threshold value for p-values. If present, features with p-values lower than threshold_pvalue are considered differentially abundant. Set threshold_pvalue = 1 to not filter by p-values.

threshold_logfc
: Threshold value for log fold changes. If present, features with log fold change absolute values higher than threshold_logfc are considered differentially abundant. Set threshold_logfc = 0 to not filter by log fold change values.

top
: If not null, the top number of features, ordered by p-values or log fold change values, are considered as differentially abundant (default top = NULL).

verbose
: Boolean to display the kind of extracted values (default verbose = FALSE).

Value

A data.frame with several columns:

- stat which contains the p-values or the absolute log fold change values;
- direction which is present if method was a data.frame object, it contains the information about directionality of differential abundance (usually log fold changes);
- DA which can contain several values according to thresholds and inputs. "DA" or "non-DA" if method object was a vector, "UP Abundant", "DOWN Abundant", or "non-DA" if method was a data.frame.

See Also

getStatistics, extractDA

Examples

```
data("ps_plaque_16S")
# Add scaling factors
ps_plaque_16S <- norm_edgeR(object = ps_plaque_16S, method = "TMM")
# DA analysis
da.limma <- DA_limma(
    object = ps_plaque_16S,
    design = ~ 1 + HMP_BODY_SUBSITE,
    coef = 2,
    norm = "TMM"
)
# features with p-value < 0.1 as DA
getDA(
    method = da.limma, slot = "pValMat", colName = "rawP", type = "pvalue",
    direction = NULL, threshold_pvalue = 0.1, threshold_logfc = 0,
    top = NULL
)
# top 10 feature with highest logFC are DA
getDA(
    method = da.limma, slot = "pValMat", colName = "rawP", type = "pvalue",
    direction = "logFC", threshold_pvalue = 1, threshold_logfc = 0, top = 10
)
```

```
# features with p-value < 0.1 and |logFC| > 1 are DA
getDA(
    method = da.limma, slot = "pValMat", colName = "rawP", type = "pvalue",
    direction = "logFC", threshold_pvalue = 0.1, threshold_logfc = 1, top =
        NULL
)
# top 10 features with |logFC| > 1 are DA
getDA(
    method = da.limma, slot = "pValMat", colName = "rawP", type = "pvalue",
    direction = "logFC", threshold_pvalue = 1, threshold_logfc = 1, top = 10
)
```

`getPositives` *getPositives*

Description

Inspect the list of p-values or/and log fold changes from the output of a differential abundance detection method and count the True Positives (TP) and the False Positives (FP).

Usage

```
getPositives(method, enrichmentCol, TP, FP)
```

Arguments

`method`	Output of differential abundance detection method in which DA information is extracted by the `getDA` function, information related to enrichment is appropriately added through the `addKnowledge` function and the Fisher exact tests is performed for the contingency tables by the `enrichmentTests` function.
`enrichmentCol`	name of the column containing information for enrichment analysis.
`TP`	A list of length-2 vectors. The entries in the vector are the direction ("UP Abundant", "DOWN Abundant", or "non-DA") in the first position, and the level of the enrichment variable (`enrichmentCol`) which is expected in that direction, in the second position.
`FP`	A list of length-2 vectors. The entries in the vector are the direction ("UP Abundant", "DOWN Abundant", or "non-DA") in the first position, and the level of the enrichment variable (`enrichmentCol`) which is not expected in that direction, in the second position.

Value

A named vector containing the number of TPs and FPs.

See Also

`createPositives`.

Examples

```
data("ps_plaque_16S")
data("microbial_metabolism")
# Extract genera from the phyloseq tax_table slot
genera <- phyloseq::tax_table(ps_plaque_16S)[, "GENUS"]
# Genera as rownames of microbial_metabolism data.frame
rownames(microbial_metabolism) <- microbial_metabolism$Genus
# Match OTUs to their metabolism
priorInfo <- data.frame(genera,
    "Type" = microbial_metabolism[genera, "Type"]
)
# Unmatched genera becomes "Unknown"
unknown_metabolism <- is.na(priorInfo$Type)
priorInfo[unknown_metabolism, "Type"] <- "Unknown"
priorInfo$Type <- factor(priorInfo$Type)
# Add a more informative names column
priorInfo[, "newNames"] <- paste0(rownames(priorInfo), priorInfo[, "GENUS"])

# DA Analysis
# Add scaling factors
ps_plaque_16S <- norm_edgeR(object = ps_plaque_16S, method = "TMM")
# DA analysis
da.limma <- DA_limma(
    object = ps_plaque_16S,
    design = ~ 1 + HMP_BODY_SUBSITE,
    coef = 2,
    norm = "TMM"
)

DA <- getDA(
    method = da.limma, slot = "pValMat", colName = "adjP",
    type = "pvalue", direction = "logFC", threshold_pvalue = 0.05,
    threshold_logfc = 1, top = NULL
)
# Add a priori information
DA_info <- addKnowledge(
    method = DA, priorKnowledge = priorInfo,
    enrichmentCol = "Type", namesCol = "newNames"
)
# Create contingency tables and compute F tests
DA_info_enriched <- enrichmentTest(
    method = DA_info, enrichmentCol = "Type",
    alternative = "greater"
)
# Count True and False Positives
DA_TP_FP <- getPositives(
    method = DA_info_enriched, enrichmentCol = "Type",
    TP = list(c("UP Abundant", "Aerobic"), c("DOWN Abundant", "Anaerobic")),
    FP = list(c("UP Abundant", "Anaerobic"), c("DOWN Abundant", "Aerobic"))
)
```

getStatistics *getStatistics*

Description

Extract the list of p-values or/and log fold changes from the output of a differential abundance detection method.

Usage

```
getStatistics(
  method,
  slot = "pValMat",
  colName = "rawP",
  type = "pvalue",
  direction = NULL,
  verbose = FALSE
)
```

Arguments

method	Output of a differential abundance detection method. pValMat, statInfo matrices, and method's name must be present (See vignette for detailed information).
slot	The slot name where to extract values (default slot = "pValMat").
colName	The column name of the slot where to extract values (default colName = "rawP").
type	The value type of the column selected where to extract values. Two values are possible: "pvalue" or "logfc" (default type = "pvalue").
direction	statInfo's column name containing information about the signs of differential abundance (usually log fold changes) (default direction = NULL).
verbose	Boolean to display the kind of extracted values (default verbose = FALSE).

Value

A vector or a data.frame. If direction = NULL, the colname column values, transformed according to type (not tranformed if type = "pvalue", -abs(value) if type = "logfc"), of the slot are reported, otherwise the direction column of the statInfo matrix is added to the output.

See Also

extractStatistics

Examples

```
data("ps_plaque_16S")
# Add scaling factors
ps_plaque_16S <- norm_edgeR(object = ps_plaque_16S, method = "TMM")
# DA analysis
da.limma <- DA_limma(
    object = ps_plaque_16S,
    design = ~ 1 + HMP_BODY_SUBSITE,
    coef = 2,
    norm = "TMM"
)
# get p-values
getStatistics(
    method = da.limma, slot = "pValMat", colName = "rawP",
    type = "pvalue", direction = NULL
)
# get negative abs(logFC) values
getStatistics(
    method = da.limma, slot = "statInfo", colName = "logFC",
    type = "logfc", direction = NULL
)
# get p-values and logFC
getStatistics(
    method = da.limma, slot = "pValMat", colName = "rawP",
    type = "pvalue", direction = "logFC"
)
```

---

iterative_ordering *iterativeOrdering*

Description

Turn the chosen columns (factor) of the input `data.frame` into ordered factors. For each factor, the order is given by the number of elements in each level of that factor.

Usage

```
iterative_ordering(df, var_names, i = 1, decreasing = TRUE)
```

Arguments

`df`	a `data.frame` object.
`var_names`	character vector containing the names of one or more columns of `df`.
`i`	iteration index (default `i = 1`).
`decreasing`	logical value or a vector of them. Each value should be associated to a `var_name` vector's element. Should the sort order be increasing or decreasing?

Value

the input data.frame with the var_names variables as ordered factors.

See Also

plotMutualFindings

Examples

```
# From a dataset with some factor columns
mpg <- data.frame(ggplot2::mpg)
# Order the levels of the desired factors based on the cardinality of each
# level.
ordered_mpg <- iterative_ordering(df = mpg,
   var_names = c("manufacturer", "model"),
   decreasing = c(TRUE, TRUE))
# Now the levels of the factors are ordered in a decreasing way
levels(ordered_mpg$manufacturer)
levels(ordered_mpg$model)
```

---

meanDifferences *meanDifferences*

Description

Compute the differences between the estimated and the observed continuity corrected logarithms of the average count values (MD), and between the estimated average probability to observe a zero and the the observed zero rate (ZPD).

Usage

```
meanDifferences(estimated, observed)
```

Arguments

estimated	a two column data.frame, output of fitNB, fitZINB, fitDM, fitZIG, or fitHURDLE functions. More in general, a data frame containing the continuity corrected logarithm for the average of the fitted values for each row of a matrix of counts in the Y column, and the estimated probability to observe a zero in the Y0 column.
observed	a two column data.frame, output of prepareObserved function. More in general, a data frame containing the continuity corrected logarithm for the average of the observed values for each row of a matrix of counts in the Y column, and the estimated proportion of zeroes in the Y0 column.

Value

a data.frame containing the differences between the estimated and the observed continuity corrected logarithms of the average count values in the MD column, and between the estimated average probability to observe a zero and the the observed zero rate in the ZPD column.

See Also

prepareObserved.

Examples

```
# Randomly generate the observed and estimated data.frames
observed <- data.frame(Y = rpois(10, 5), Y0 = runif(10, 0, 1))
estimated <- data.frame(Y = rpois(10, 5), Y0 = runif(10, 0, 1))

# Compute the mean differences between estimated and observed data.frames
meanDifferences(estimated, observed)
```

---

microbial_metabolism *(Data) Microbial metabolism*

Description

Aerobic, Anaerobic, or Facultative Anaerobic microbes by genus (NYC-HANES study).

Usage

```
data(microbial_metabolism)
```

Format

A data.frame object

---

norm_CSS *norm_CSS*

Description

Calculate scaling factors from a phyloseq object to scale the raw library sizes. Inherited from metagenomeSeq 'calcNormFactors' function which performs the Cumulative Sum Scaling normalization.

Usage

```
norm_CSS(object, method = "default", verbose = TRUE)
```

Arguments

object	phyloseq object containing the counts to be normalized.
method	normalization scaling parameter (default method = "default"). If "median", the median of the normalization factors is used as scaling (Paulson et al. 2013).
verbose	an optional logical value. If TRUE, information about the steps of the algorithm is printed. Default verbose = TRUE.

Value

A new column containing the CSS scaling factors is added to the phyloseq `sample_data` slot.

See Also

`calcNormFactors` for details. `setNormalizations` and `runNormalizations` to fastly set and run normalizations.

Examples

```
set.seed(1)
# Create a very simple phyloseq object
counts <- matrix(rnbinom(n = 60, size = 3, prob = 0.5), nrow = 10, ncol = 6)
metadata <- data.frame("Sample" = c("S1", "S2", "S3", "S4", "S5", "S6"),
                       "group" = as.factor(c("A", "A", "A", "B", "B", "B")))
ps <- phyloseq::phyloseq(phyloseq::otu_table(counts, taxa_are_rows = TRUE),
                         phyloseq::sample_data(metadata))

# Calculate the scaling factors
ps_NF <- norm_CSS(object = ps, method = "median")
# The phyloseq object now contains the scaling factors:
scaleFacts <- phyloseq::sample_data(ps_NF)[, "NF.CSSmedian"]
head(scaleFacts)

# VERY IMPORTANT: to convert scaling factors to normalization factors
# multiply them by the library sizes and renormalize.
normFacts = scaleFacts * phyloseq::sample_sums(ps_stool_16S)
# Renormalize: multiply to 1
normFacts = normFacts/exp(colMeans(log(normFacts)))
```

---

norm_DESeq2 *norm_DESeq2*

Description

Calculate normalization factors from a phyloseq object to scale the raw library sizes. Inherited from DESeq2 `estimateSizeFactors` function.

Usage

```
norm_DESeq2(
  object,
  method = c("ratio", "poscounts", "iterate"),
  verbose = TRUE,
  ...
)
```

Arguments

object	phyloseq object containing the counts to be normalized.
method	Method for estimation: either "ratio", "poscounts", or "iterate". "ratio" uses the standard median ratio method introduced in DESeq. The size factor is the median ratio of the sample over a "pseudosample": for each gene, the geometric mean of all samples. "poscounts" and "iterate" offer alternative estimators, which can be used even when all features contain a sample with a zero (a problem for the default method, as the geometric mean becomes zero, and the ratio undefined). The "poscounts" estimator deals with a feature with some zeros, by calculating a modified geometric mean by taking the n-th root of the product of the non-zero counts. This evolved out of use cases with Paul McMurdie's phyloseq package for metagenomic samples. The "iterate" estimator iterates between estimating the dispersion with a design of ~1, and finding a size factor vector by numerically optimizing the likelihood of the ~1 model.
verbose	an optional logical value. If TRUE, information about the steps of the algorithm is printed. Default verbose = TRUE.
...	other parameters for DESeq2 estimateSizeFactors function.

Value

A new column containing the chosen DESeq2-based normalization factors is added to the phyloseq sample_data slot.

See Also

estimateSizeFactors for details. setNormalizations and runNormalizations to fastly set and run normalizations.

Examples

```
set.seed(1)
# Create a very simple phyloseq object
counts <- matrix(rnbinom(n = 60, size = 3, prob = 0.5), nrow = 10, ncol = 6)
metadata <- data.frame("Sample" = c("S1", "S2", "S3", "S4", "S5", "S6"),
                       "group" = as.factor(c("A", "A", "A", "B", "B", "B")))
ps <- phyloseq::phyloseq(phyloseq::otu_table(counts, taxa_are_rows = TRUE),
                         phyloseq::sample_data(metadata))

# Calculate the normalization factors
ps_NF <- norm_DESeq2(object = ps, method = "poscounts")
# The phyloseq object now contains the normalization factors:
normFacts <- phyloseq::sample_data(ps_NF)[, "NF.poscounts"]
head(normFacts)

# VERY IMPORTANT: to convert normalization factors to scaling factors divide
# them by the library sizes and renormalize.
scaleFacts = normFacts / phyloseq::sample_sums(ps_stool_16S)
# Renormalize: multiply to 1
scaleFacts = scaleFacts/exp(mean(log(scaleFacts)))
```

norm_edgeR *norm_edgeR*

Description

Calculate scaling factors from a phyloseq object to scale the raw library sizes. Inherited from edgeR `calcNormFactors` function.

Usage

```
norm_edgeR(
  object,
 method = c("TMM", "TMMwsp", "RLE", "upperquartile", "posupperquartile", "none"),
  refColumn = NULL,
  logratioTrim = 0.3,
  sumTrim = 0.05,
  doWeighting = TRUE,
  Acutoff = -1e+10,
  p = 0.75,
  verbose = TRUE,
  ...
)
```

Arguments

`object`	a phyloseq object containing the counts to be normalized.
`method`	normalization method to be used. Choose between `TMM`, `TMMwsp`, `RLE`, `upperquartile`, `posupperquartile` or `none`.
`refColumn`	column to use as reference for `method="TMM"`. Can be a column number or a numeric vector of length `nrow(object)`.
`logratioTrim`	the fraction (0 to 0.5) of observations to be trimmed from each tail of the distribution of log-ratios (M-values) before computing the mean. Used by `method="TMM"` for each pair of samples.
`sumTrim`	the fraction (0 to 0.5) of observations to be trimmed from each tail of the distribution of A-values before computing the mean. Used by `method="TMM"` for each pair of samples.
`doWeighting`	logical, whether to use (asymptotic binomial precision) weights when computing the mean M-values. Used by `method="TMM"` for each pair of samples.
`Acutoff`	minimum cutoff applied to A-values. Count pairs with lower A-values are ignored. Used by `method="TMM"` for each pair of samples.
`p`	numeric value between 0 and 1 specifying which quantile of the counts should be used by `method="upperquartile"`.
`verbose`	an optional logical value. If `TRUE`, information about the steps of the algorithm is printed. Default `verbose = TRUE`.
`...`	other arguments are not currently used.

Value

A new column containing the chosen edgeR-based scaling factors is added to the phyloseq `sample_data` slot. The effective library sizes to use in downstream analysis must be multiplied by the normalization factors.

See Also

`calcNormFactors` for details.

`setNormalizations` and `runNormalizations` to fastly set and run normalizations.

Examples

```
set.seed(1)
# Create a very simple phyloseq object
counts <- matrix(rnbinom(n = 60, size = 3, prob = 0.5), nrow = 10, ncol = 6)
metadata <- data.frame("Sample" = c("S1", "S2", "S3", "S4", "S5", "S6"),
                       "group" = as.factor(c("A", "A", "A", "B", "B", "B")))
ps <- phyloseq::phyloseq(phyloseq::otu_table(counts, taxa_are_rows = TRUE),
                         phyloseq::sample_data(metadata))

# Calculate the scaling factors
ps_NF <- norm_edgeR(object = ps, method = "TMM")

# The phyloseq object now contains the scaling factors:
scaleFacts <- phyloseq::sample_data(ps_NF)[, "NF.TMM"]
head(scaleFacts)

# VERY IMPORTANT: to convert scaling factors to normalization factors
# multiply them by the library sizes and renormalize.
normFacts = scaleFacts * phyloseq::sample_sums(ps_stool_16S)
# Renormalize: multiply to 1
normFacts = normFacts/exp(colMeans(log(normFacts)))
```

---

norm_TSS *norm_TSS*

Description

Calculate the raw library sizes from a phyloseq object. If used to divide counts, known as Total Sum Scaling normalization (TSS).

Usage

```
norm_TSS(object, method = "TSS", verbose = TRUE)
```

Arguments

object	phyloseq object containing the counts to be normalized.
method	normalization method to be used.
verbose	an optional logical value. If TRUE, information about the steps of the algorithm is printed. Default verbose = TRUE.

Value

A new column containing the TSS scaling factors is added to the phyloseq sample_data slot.

See Also

setNormalizations and runNormalizations to fastly set and run normalizations.

Examples

```
set.seed(1)
# Create a very simple phyloseq object
counts <- matrix(rnbinom(n = 60, size = 3, prob = 0.5), nrow = 10, ncol = 6)
metadata <- data.frame("Sample" = c("S1", "S2", "S3", "S4", "S5", "S6"),
                       "group" = as.factor(c("A", "A", "A", "B", "B", "B")))
ps <- phyloseq::phyloseq(phyloseq::otu_table(counts, taxa_are_rows = TRUE),
                         phyloseq::sample_data(metadata))

# Calculate the scaling factors
ps_NF <- norm_TSS(object = ps)
# The phyloseq object now contains the scaling factors:
scaleFacts <- phyloseq::sample_data(ps_NF)[, "NF.TSS"]
head(scaleFacts)

# VERY IMPORTANT: to convert scaling factors to normalization factors
# multiply them by the library sizes and renormalize.
normFacts = scaleFacts * phyloseq::sample_sums(ps_stool_16S)
# Renormalize: multiply to 1
normFacts = normFacts/exp(colMeans(log(normFacts)))
```

---

plotConcordance *plotConcordance*

Description

Produce a list of graphical outputs summarizing the between and within method concordance.

Usage

```
plotConcordance(concordance, threshold = NULL, cols = NULL)
```

Arguments

concordance	A long format data.frame produced by createConcordance function.
threshold	The threshold for rank (x-axis upper limit if all methods have a higher number of computed statistics).
cols	A named vector containing the color hex codes.

Value

A 2 elements list of ggplot2 class objects:

- concordanceDendrogram which contains the vertically directioned dendrogram for the methods involved in the concordance analysis;
- concordanceHeatmap which contains the heatmap of between and within method concordances.

See Also

createConcordance

Examples

```
data(ps_plaque_16S)
# Balanced design for independent samples
my_splits <- createSplits(
    object = ps_plaque_16S, varName = "HMP_BODY_SUBSITE", balanced = TRUE,
    N = 10 # N = 100 suggested
)

# Initialize some limma based methods
my_limma <- set_limma(design = ~ HMP_BODY_SUBSITE, coef = 2,
    norm = c("TMM", "CSSmedian"))

# Set the normalization methods according to the DA methods
my_norm <- setNormalizations(fun = c("norm_edgeR", "norm_CSS"),
    method = c("TMM", "median"))

# Run methods on split datasets
Plaque_16S_splitsDA <- runSplits(split_list = my_splits,
    method_list = my_limma, normalization_list = my_norm,
    object = ps_plaque_16S)

# Concordance for p-values
concordance_pvalues <- createConcordance(
    object = Plaque_16S_splitsDA, slot =
        "pValMat", colName = "rawP", type = "pvalue"
)

# plot concordances from rank 1 to 50.
plotConcordance(
    concordance = concordance_pvalues,
```

```
    threshold = 50
)
```

---

plotContingency *plotContingency*

Description

Plot of the contingency tables for the chosen method. The top-left cells are colored, according to Fisher exact tests' p-values, if the number of features in those cells are enriched.

Usage

```
plotContingency(enrichment, method, levels_to_plot)
```

Arguments

enrichment	enrichment object produced by createEnrichment function.
method	name of the method to plot.
levels_to_plot	A character vector containing the levels of the enrichment variable to plot.

Value

a ggplot2 object.

See Also

createEnrichment, plotEnrichment, and plotMutualFindings.

Examples

```
data("ps_plaque_16S")
data("microbial_metabolism")

# Extract genera from the phyloseq tax_table slot
genera <- phyloseq::tax_table(ps_plaque_16S)[, "GENUS"]
# Genera as rownames of microbial_metabolism data.frame
rownames(microbial_metabolism) <- microbial_metabolism$Genus
# Match OTUs to their metabolism
priorInfo <- data.frame(genera,
    "Type" = microbial_metabolism[genera, "Type"]
)
# Unmatched genera becomes "Unknown"
unknown_metabolism <- is.na(priorInfo$Type)
priorInfo[unknown_metabolism, "Type"] <- "Unknown"
priorInfo$Type <- factor(priorInfo$Type)
# Add a more informative names column
priorInfo[, "newNames"] <- paste0(rownames(priorInfo), priorInfo[, "GENUS"])
```

```
# DA analysis
# Add scaling factors
ps_plaque_16S <- norm_edgeR(object = ps_plaque_16S, method = "TMM")
ps_plaque_16S <- norm_CSS(object = ps_plaque_16S, method = "median")

# Perform DA analysis
Plaque_16S_DA <- list()
Plaque_16S_DA <- within(Plaque_16S_DA, {
    # DA analysis
    da.limma <- DA_limma(
        object = ps_plaque_16S,
        design = ~ 1 + HMP_BODY_SUBSITE,
        coef = 2,
        norm = "TMM"
    )
    da.limma.css <- DA_limma(
        object = ps_plaque_16S,
        design = ~ 1 + HMP_BODY_SUBSITE,
        coef = 2,
        norm = "CSSmedian"
    )
})

enrichment <- createEnrichment(
    object = Plaque_16S_DA,
    priorKnowledge = priorInfo, enrichmentCol = "Type", namesCol = "GENUS",
    slot = "pValMat", colName = "adjP", type = "pvalue", direction = "logFC",
    threshold_pvalue = 0.1, threshold_logfc = 1, top = 10, verbose = TRUE
)
# Contingency tables
plotContingency(enrichment = enrichment, method = "limma.TMM")
# Barplots
plotEnrichment(enrichment, enrichmentCol = "Type")
# Mutual findings
plotMutualFindings(
    enrichment = enrichment, enrichmentCol = "Type",
    n_methods = 1
)
```

`plotEnrichment` *plotEnrichment*

Description

Summary plot for the number of differentially abundant (DA) features and their association with enrichment variable. If some features are UP-Abundant or DOWN-Abundant (or just DA), several bars will be represented in the corresponding direction. Significance thresholds are reported over/under each bar, according to the Fisher exact tests.

Usage

```
plotEnrichment(enrichment, enrichmentCol, levels_to_plot)
```

Arguments

enrichment	enrichment object produced by createEnrichment function.
enrichmentCol	name of the column containing information for enrichment analysis.
levels_to_plot	A character vector containing the levels of the enrichment variable to plot.

Value

a ggplot2 object.

See Also

createEnrichment, plotContingency, and plotMutualFindings.

Examples

```
data("ps_plaque_16S")
data("microbial_metabolism")

# Extract genera from the phyloseq tax_table slot
genera <- phyloseq::tax_table(ps_plaque_16S)[, "GENUS"]
# Genera as rownames of microbial_metabolism data.frame
rownames(microbial_metabolism) <- microbial_metabolism$Genus
# Match OTUs to their metabolism
priorInfo <- data.frame(genera,
    "Type" = microbial_metabolism[genera, "Type"]
)
# Unmatched genera becomes "Unknown"
unknown_metabolism <- is.na(priorInfo$Type)
priorInfo[unknown_metabolism, "Type"] <- "Unknown"
priorInfo$Type <- factor(priorInfo$Type)
# Add a more informative names column
priorInfo[, "newNames"] <- paste0(rownames(priorInfo), priorInfo[, "GENUS"])

# DA analysis
# Add scaling factors
ps_plaque_16S <- norm_edgeR(object = ps_plaque_16S, method = "TMM")
ps_plaque_16S <- norm_CSS(object = ps_plaque_16S, method = "median")

# Perform DA analysis
Plaque_16S_DA <- list()
Plaque_16S_DA <- within(Plaque_16S_DA, {
    # DA analysis
    da.limma <- DA_limma(
        object = ps_plaque_16S,
        design = ~ 1 + HMP_BODY_SUBSITE,
        coef = 2,
        norm = "TMM"
```

```
    )
    da.limma.css <- DA_limma(
        object = ps_plaque_16S,
        design = ~ 1 + HMP_BODY_SUBSITE,
        coef = 2,
        norm = "CSSmedian"
    )
})

enrichment <- createEnrichment(
    object = Plaque_16S_DA,
    priorKnowledge = priorInfo, enrichmentCol = "Type", namesCol = "GENUS",
    slot = "pValMat", colName = "adjP", type = "pvalue", direction = "logFC",
    threshold_pvalue = 0.1, threshold_logfc = 1, top = 10, verbose = TRUE
)
# Contingency tables
plotContingency(enrichment = enrichment, method = "limma.TMM")
# Barplots
plotEnrichment(enrichment, enrichmentCol = "Type")
# Mutual findings
plotMutualFindings(
    enrichment = enrichment, enrichmentCol = "Type",
    n_methods = 1
)
```

`plotFPR` *plotFPR*

Description

Draw the boxplots of the proportions of p-values lower than 0.01, 0.05, and 0.1 thresholds for each method.

Usage

```
plotFPR(df_FPR, cols = NULL)
```

Arguments

`df_FPR`	a `data.frame` produced by the `createTIEC` function, containing the FPR values.
`cols`	named vector of colors.

Value

A ggplot object.

Examples

```
# Load some data
data(ps_stool_16S)

# Generate the patterns for 10 mock comparison for an experiment
# (N = 1000 is suggested)
mocks <- createMocks(nsamples = phyloseq::nsamples(ps_stool_16S), N = 10)
head(mocks)

# Add some normalization/scaling factors to the phyloseq object
my_norm <- setNormalizations(fun = c("norm_edgeR", "norm_CSS"),
    method = c("TMM", "median"))
ps_stool_16S <- runNormalizations(normalization_list = my_norm,
    object = ps_stool_16S)

# Initialize some limma based methods
my_limma <- set_limma(design = ~ group, coef = 2,
    norm = c("TMM", "CSSmedian"))

# Run methods on mock datasets
results <- runMocks(mocks = mocks, method_list = my_limma,
    object = ps_stool_16S)

# Prepare results for Type I Error Control
TIEC_summary <- createTIEC(results)

# Plot the results
plotFPR(df_FPR = TIEC_summary$df_FPR)
plotQQ(df_QQ = TIEC_summary$df_QQ, zoom = c(0, 0.1))
plotKS(df_KS = TIEC_summary$df_KS)
```

`plotKS` *plotKS*

Description

Draw the boxplots of the Kolmogorov-Smirnov test statistics for the p-value distributions across the mock comparisons.

Usage

```
plotKS(df_KS, cols = NULL)
```

Arguments

`df_KS`	a `data.frame` produced by the `createTIEC` function containing the KS statistics and their p-values.
`cols`	named vector of colors.

Value

A ggplot object.

Examples

```
# Load some data
data(ps_stool_16S)

# Generate the patterns for 10 mock comparison for an experiment
# (N = 1000 is suggested)
mocks <- createMocks(nsamples = phyloseq::nsamples(ps_stool_16S), N = 10)
head(mocks)

# Add some normalization/scaling factors to the phyloseq object
my_norm <- setNormalizations(fun = c("norm_edgeR", "norm_CSS"),
    method = c("TMM", "median"))
ps_stool_16S <- runNormalizations(normalization_list = my_norm,
    object = ps_stool_16S)

# Initialize some limma based methods
my_limma <- set_limma(design = ~ group, coef = 2,
    norm = c("TMM", "CSSmedian"))

# Run methods on mock datasets
results <- runMocks(mocks = mocks, method_list = my_limma,
    object = ps_stool_16S)

# Prepare results for Type I Error Control
TIEC_summary <- createTIEC(results)

# Plot the results
plotFPR(df_FPR = TIEC_summary$df_FPR)
plotQQ(df_QQ = TIEC_summary$df_QQ, zoom = c(0, 0.1))
plotKS(df_KS = TIEC_summary$df_KS)
```

`plotMD` *plotMD*

Description

A function to plot mean difference (MD) and zero probability difference (ZPD) values between estimated and observed values.

Usage

```
plotMD(data, difference = NULL, split = TRUE)
```

Arguments

data	a list, output of the fitModels function or a 'data.frame' object with Model, Y, Y0, MD, and ZPD columns containing the model name, the observed values for the mean and the zero proportion and the differences between observed and estimated values.
difference	character vector, either MD or ZPD to plot the differences between estimated and observed mean counts or the differences between estimated zero probability and observed zero proportion.
split	Display each model mean differences in different facets (default split = TRUE). If FALSE, points are not displayed for more clear representation.

Value

a ggplot object.

See Also

fitModels and RMSE for the model estimations and the RMSE computations respectively. plotRMSE for the graphical evaluation of the RMSE values.

Examples

```
# Generate some random counts
counts = matrix(rnbinom(n = 60, size = 3, prob = 0.5), nrow = 10, ncol = 6)

# Estimate the counts assuming several distributions
GOF <- fitModels(
    counts = counts, models = c(
        "NB", "ZINB",
        "DM", "ZIG", "HURDLE"
    ), scale_ZIG = c("median", "default"), scale_HURDLE =
        c("median", "default")
)

# Plot the results
plotMD(data = GOF, difference = "MD", split = TRUE)
plotMD(data = GOF, difference = "ZPD", split = TRUE)
```

---

plotMutualFindings *plotMutualFindings*

Description

Plot and filter the features which are considered differentially abundant, simultaneously, by a specified number of methods.

Usage

```
plotMutualFindings(enrichment, enrichmentCol, levels_to_plot, n_methods = 1)
```

Arguments

enrichment	enrichment object produced by createEnrichment function.
enrichmentCol	name of the column containing information for enrichment analysis.
levels_to_plot	A character vector containing the levels of the enrichment variable to plot.
n_methods	minimum number of method that mutually find the features.

Value

a ggplot2 object.

See Also

createEnrichment, plotEnrichment, and plotContingency.

Examples

```
data("ps_plaque_16S")
data("microbial_metabolism")

# Extract genera from the phyloseq tax_table slot
genera <- phyloseq::tax_table(ps_plaque_16S)[, "GENUS"]
# Genera as rownames of microbial_metabolism data.frame
rownames(microbial_metabolism) <- microbial_metabolism$Genus
# Match OTUs to their metabolism
priorInfo <- data.frame(genera,
    "Type" = microbial_metabolism[genera, "Type"]
)
# Unmatched genera becomes "Unknown"
unknown_metabolism <- is.na(priorInfo$Type)
priorInfo[unknown_metabolism, "Type"] <- "Unknown"
priorInfo$Type <- factor(priorInfo$Type)
# Add a more informative names column
priorInfo[, "newNames"] <- paste0(rownames(priorInfo), priorInfo[, "GENUS"])

# DA analysis
# Add scaling factors
ps_plaque_16S <- norm_edgeR(object = ps_plaque_16S, method = "TMM")
ps_plaque_16S <- norm_CSS(object = ps_plaque_16S, method = "median")

# Perform DA analysis
Plaque_16S_DA <- list()
Plaque_16S_DA <- within(Plaque_16S_DA, {
    # DA analysis
    da.limma <- DA_limma(
        object = ps_plaque_16S,
        design = ~ 1 + HMP_BODY_SUBSITE,
```

```
        coef = 2,
        norm = "TMM"
    )
    da.limma.css <- DA_limma(
        object = ps_plaque_16S,
        design = ~ 1 + HMP_BODY_SUBSITE,
        coef = 2,
        norm = "CSSmedian"
    )
})

enrichment <- createEnrichment(
    object = Plaque_16S_DA,
    priorKnowledge = priorInfo, enrichmentCol = "Type", namesCol = "GENUS",
    slot = "pValMat", colName = "adjP", type = "pvalue", direction = "logFC",
    threshold_pvalue = 0.1, threshold_logfc = 1, top = 10, verbose = TRUE
)
# Contingency tables
plotContingency(enrichment = enrichment, method = "limma.TMM")
# Barplots
plotEnrichment(enrichment, enrichmentCol = "Type")
# Mutual findings
plotMutualFindings(
    enrichment = enrichment, enrichmentCol = "Type",
    n_methods = 1
)
```

`plotPositives` *plotPositives*

Description

Plot the difference between the number of true positives (TP) and false positives (FP) for each method and for each 'top' threshold provided by the `createPositives()` function.

Usage

```
plotPositives(positives, cols = NULL)
```

Arguments

`positives`	`data.frame` object produced by `createPositives()` function.
`cols`	named vector of cols (default `cols = NULL`).

Value

a `ggplot2` object.

See Also

getPositives, createPositives.

Examples

```
data("ps_plaque_16S")
data("microbial_metabolism")

# Extract genera from the phyloseq tax_table slot
genera <- phyloseq::tax_table(ps_plaque_16S)[, "GENUS"]
# Genera as rownames of microbial_metabolism data.frame
rownames(microbial_metabolism) <- microbial_metabolism$Genus
# Match OTUs to their metabolism
priorInfo <- data.frame(genera,
    "Type" = microbial_metabolism[genera, "Type"]
)
# Unmatched genera becomes "Unknown"
unknown_metabolism <- is.na(priorInfo$Type)
priorInfo[unknown_metabolism, "Type"] <- "Unknown"
priorInfo$Type <- factor(priorInfo$Type)
# Add a more informative names column
priorInfo[, "newNames"] <- paste0(rownames(priorInfo), priorInfo[, "GENUS"])

# DA analysis
# Add scaling factors
ps_plaque_16S <- norm_edgeR(object = ps_plaque_16S, method = "TMM")
ps_plaque_16S <- norm_CSS(object = ps_plaque_16S, method = "median")

# Perform DA analysis
Plaque_16S_DA <- list()
Plaque_16S_DA <- within(Plaque_16S_DA, {
    # DA analysis
    da.limma <- DA_limma(
        object = ps_plaque_16S,
        design = ~ 1 + HMP_BODY_SUBSITE,
        coef = 2,
        norm = "TMM"
    )
    da.limma.css <- DA_limma(
        object = ps_plaque_16S,
        design = ~ 1 + HMP_BODY_SUBSITE,
        coef = 2,
        norm = "CSSmedian"
    )
})

# Count TPs and FPs, from the top 1 to the top 20 features.
# As direction is supplied, features are ordered by "logFC" absolute values.
positives <- createPositives(
    object = Plaque_16S_DA,
    priorKnowledge = priorInfo, enrichmentCol = "Type",
    namesCol = "newNames", slot = "pValMat", colName = "rawP",
```

```
    type = "pvalue", direction = "logFC", threshold_pvalue = 1,
    threshold_logfc = 0, top = 1:20, alternative = "greater",
    verbose = FALSE,
    TP = list(c("DOWN Abundant", "Anaerobic"), c("UP Abundant", "Aerobic")),
    FP = list(c("DOWN Abundant", "Aerobic"), c("UP Abundant", "Anaerobic"))
)
# Plot the TP-FP differences for each threshold
plotPositives(positives = positives)
```

---

plotQQ *plotQQ*

Description

Draw the average QQ-plots across the mock comparisons.

Usage

```
plotQQ(df_QQ, cols = NULL, zoom = c(0, 0.1))
```

Arguments

df_QQ	Coordinates to draw the QQ-plot to compare the mean observed p-value distribution across comparisons, with the theoretical uniform distribution.
cols	named vector of colors.
zoom	2-dimesional vector containing the starting and the final coordinates (default: c(0,0.1))

Value

A ggplot object.

Examples

```
# Load some data
data(ps_stool_16S)

# Generate the patterns for 10 mock comparison for an experiment
# (N = 1000 is suggested)
mocks <- createMocks(nsamples = phyloseq::nsamples(ps_stool_16S), N = 10)
head(mocks)

# Add some normalization/scaling factors to the phyloseq object
my_norm <- setNormalizations(fun = c("norm_edgeR", "norm_CSS"),
    method = c("TMM", "median"))
ps_stool_16S <- runNormalizations(normalization_list = my_norm,
    object = ps_stool_16S)

# Initialize some limma based methods
```

```
my_limma <- set_limma(design = ~ group, coef = 2,
    norm = c("TMM", "CSSmedian"))

# Run methods on mock datasets
results <- runMocks(mocks = mocks, method_list = my_limma,
    object = ps_stool_16S)

# Prepare results for Type I Error Control
TIEC_summary <- createTIEC(results)

# Plot the results
plotFPR(df_FPR = TIEC_summary$df_FPR)
plotQQ(df_QQ = TIEC_summary$df_QQ, zoom = c(0, 0.1))
plotKS(df_KS = TIEC_summary$df_KS)
```

---

plotRMSE *plotRMSE*

Description

A function to plot RMSE values computed for mean difference (MD) and zero probability difference (ZPD) values between estimated and observed values.

Usage

```
plotRMSE(data, difference = NULL, plotIt = TRUE)
```

Arguments

data	a list, output of the fitModels function or a 'data.frame' object with Model, Y, Y0, MD, and ZPD columns containing the model name, the observed values for the mean and the zero proportion and the differences between observed and estimated values.
difference	character vector, either MD or ZPD to plot the differences between estimated and observed mean counts or the differences between estimated zero probability and observed zero proportion.
plotIt	logical. Should plotting be done? (default plotIt = TRUE)

Value

a ggplot object.

See Also

fitModels and RMSE for the model estimations and the RMSE computations respectively. plotMD for the graphical evaluation.

Examples

```
# Generate some random counts
counts = matrix(rnbinom(n = 60, size = 3, prob = 0.5), nrow = 10, ncol = 6)

# Estimate the counts assuming several distributions
GOF <- fitModels(
    counts = counts, models = c(
        "NB", "ZINB",
        "DM", "ZIG", "HURDLE"
    ), scale_ZIG = c("median", "default"), scale_HURDLE =
        c("median", "default")
)

# Plot the RMSE results
plotRMSE(data = GOF, difference = "MD")
plotRMSE(data = GOF, difference = "ZPD")
```

`prepareObserved` *prepareObserved*

Description

Continuity corrected logarithms of the average counts and fraction of zeroes by feature.

Usage

```
prepareObserved(counts, scale = NULL)
```

Arguments

`counts`	a phyloseq object or a matrix of counts with features (OTUs, ASVs, genes) by row and samples by column.
`scale`	If specified it refers to the character vector used in `fitHURDLE` function. Either `median` or `default` to choose between the median library size or one million as scaling factors for raw counts.

Value

A data frame containing the continuity corrected logarithm for the raw count mean values for each taxon of the matrix of counts in the `Y` column and the observed zero rate in the `Y0` column. If `scale` is specified the continuity corrected logarithm for the mean CPM (`scale = "default"`) or the mean counts per median library size (`scale = "median"`) is computed instead.

See Also

`meanDifferences`

Examples

```
# Generate some random counts
counts <- matrix(rnbinom(n = 60, size = 3, prob = 0.5), nrow = 10, ncol = 6)

observed1 <- prepareObserved(counts)
# For the comparison with HURDLE model
observed2 <- prepareObserved(counts, scale = "median")
```

---

ps_plaque_16S *(Data) 60 Gingival Plaque samples of 16S rRNA (HMP 2012)*

Description

A demonstrative purpose dataset containing microbial abundances for a total of 88 OTUs. The 60 Gingival Plaque paired samples belong to the Human Microbiome Project. This particular subset contains 30 Supragingival and 30 Subgingival Plaque samples from the `SEX = "Male"`, `RUN_CENTER = "WUCG"`, and `VISITNO = "1"` samples. It is possible to obtain the same dataset after basic filters (remove taxa with zero counts) and collapsing the counts to the genus level; `HMP16SData` Bioconductor package was used to download the data.

Usage

```
data(ps_plaque_16S)
```

Format

An object of class `phyloseq`

---

ps_stool_16S *(Data) 33 Stool samples of 16S rRNA (HMP 2012)*

Description

A demonstrative purpose dataset containing microbial abundances for a total of 71 OTUs. The 32 Stool samples belong to the Human Microbiome Project. This particular subset contains the `SEX = "Male"`, `RUN_CENTER = "BI"`, and `VISITNO = "1"` samples. It is possible to obtain the same dataset after basic filters (remove taxa with zero counts) and collapsing the counts to the genus level; `HMP16Data` Bioconductor package was used to download the data.

Usage

```
data(ps_stool_16S)
```

Format

An object of class `phyloseq`

RMSE *RMSE*

Description

Computes the Root Mean Square Error (RMSE) from a vector of differences.

Usage

```
RMSE(differences)
```

Arguments

differences	a vector of differences.

Value

RMSE value

See Also

prepareObserved and meanDifferences.

Examples

```
# Generate the data.frame of Mean Differences and Zero Probability Difference
MD_df <- data.frame(MD = rpois(10, 5), ZPD = runif(10, -1, 1))

# Calculate RMSE for MD and ZPD values
RMSE(MD_df[, "MD"])
RMSE(MD_df[, "ZPD"])
```

runDA *runDA*

Description

Run the differential abundance detection methods.

Usage

```
runDA(method_list, object, weights = NULL, verbose = TRUE)
```

Arguments

method_list	a list object containing the methods and their parameters.
object	a phyloseq object.
weights	an optional numeric matrix giving observational weights.
verbose	an optional logical value. If TRUE, information about the steps of the algorithm is printed. Default verbose = TRUE.

Value

A named list containing the results for each method.

Examples

```
set.seed(1)
# Create a very simple phyloseq object
counts <- matrix(rnbinom(n = 60, size = 3, prob = 0.5), nrow = 10, ncol = 6)
metadata <- data.frame("Sample" = c("S1", "S2", "S3", "S4", "S5", "S6"),
    "group" = as.factor(c("A", "A", "A", "B", "B", "B")))
ps <- phyloseq::phyloseq(phyloseq::otu_table(counts, taxa_are_rows = TRUE),
    phyloseq::sample_data(metadata))

# Set some simple normalizations
my_norm <- setNormalizations()

# Add them to the phyloseq object
ps <- runNormalizations(normalization_list = my_norm, object = ps)

# Set some limma instances
my_methods <- set_limma(design = ~ group, coef = 2,
    norm = c("TMM", "poscounts", "CSSmedian"))

# Run the methods
results <- runDA(method_list = my_methods, object = ps)
```

---

runMocks *runMocks*

Description

Run the differential abundance detection methods on mock datasets.

Usage

```
runMocks(mocks, method_list, object, weights = NULL, verbose = TRUE)
```

Arguments

mocks	a data.frame containing N rows and nsamples columns (if even). Each cell of the data frame contains the "grp1" or "grp2" characters which represent the mock groups pattern. Produced by the createMocks function.
method_list	a list object containing the methods and their parameters.
object	a phyloseq object.
weights	an optional numeric matrix giving observational weights.
verbose	an optional logical value. If TRUE, information about the steps of the algorithm is printed. Default verbose = TRUE.

Value

A named list containing the results for each method.

Examples

```
# Load some data
data(ps_stool_16S)

# Generate the pattern for 10 mock comparisons
# (N = 1000 is suggested)
mocks <- createMocks(nsamples = phyloseq::nsamples(ps_stool_16S), N = 10)
head(mocks)

# Add some normalization/scaling factors to the phyloseq object
my_norm <- setNormalizations(fun = c("norm_edgeR", "norm_CSS"),
    method = c("TMM", "median"))
ps_stool_16S <- runNormalizations(normalization_list = my_norm,
    object = ps_stool_16S)

# Initialize some limma based methods
my_limma <- set_limma(design = ~ group, coef = 2,
    norm = c("TMM", "CSSmedian"))

# Run methods on mock datasets
results <- runMocks(mocks = mocks, method_list = my_limma,
    object = ps_stool_16S)
```

---

runNormalizations *runNormalizations*

Description

Add normalization/scaling factors to a phyloseq object

Usage

```
runNormalizations(normalization_list, object, verbose = TRUE)
```

Arguments

normalization_list	a list object containing the normalization methods and their parameters.
object	a phyloseq object.
verbose	an optional logical value. If TRUE, information about the steps of the algorithm is printed. Default verbose = TRUE.

Value

A phyloseq object containing the normalization/scaling factors.

See Also

setNormalizations

Examples

```
set.seed(1)
# Create a very simple phyloseq object
counts <- matrix(rnbinom(n = 60, size = 3, prob = 0.5), nrow = 10, ncol = 6)
metadata <- data.frame("Sample" = c("S1", "S2", "S3", "S4", "S5", "S6"),
                       "group" = as.factor(c("A", "A", "A", "B", "B", "B")))
ps <- phyloseq::phyloseq(phyloseq::otu_table(counts, taxa_are_rows = TRUE),
                         phyloseq::sample_data(metadata))

# Set some simple normalizations
my_normalizations <- setNormalizations()

# Add them to the phyloseq object
ps <- runNormalizations(normalization_list = my_normalizations, object = ps)
```

---

runSplits *runSplits*

Description

Run the differential abundance detection methods on split datasets.

Usage

```
runSplits(split_list, method_list, normalization_list, object, verbose = TRUE)
```

Arguments

split_list	A list of 2 data.frame objects: Subset1 and Subset2 produced by the createSplits function.
method_list	a list object containing the methods and their parameters.
normalization_list	a list object containing the normalization method names and their parameters produced by setNormalizations.
object	a phyloseq object.
verbose	an optional logical value. If TRUE, information about the steps of the algorithm is printed. Default verbose = TRUE.

Value

A named list containing the results for each method.

Examples

```
data(ps_plaque_16S)

# Balanced design for independent samples
my_splits <- createSplits(
    object = ps_plaque_16S, varName =
    "HMP_BODY_SUBSITE", balanced = TRUE, N = 10 # N = 100 suggested
)

# Initialize some limma based methods
my_limma <- set_limma(design = ~ HMP_BODY_SUBSITE, coef = 2,
    norm = c("TMM", "CSSmedian"))

# Set the normalization methods according to the DA methods
my_norm <- setNormalizations(fun = c("norm_edgeR", "norm_CSS"),
    method = c("TMM", "median"))

# Run methods on split datasets
results <- runSplits(split_list = my_splits, method_list = my_limma,
    normalization_list = my_norm, object = ps_plaque_16S)
```

---

setNormalizations *setNormalizations*

Description

Set the methods and parameters to compute normalization/scaling factors.

Usage

```
setNormalizations(
  fun = c("norm_edgeR", "norm_DESeq2", "norm_CSS", "norm_edgeR"),
  method = c("TMM", "poscounts", "median", "none")
)
```

Arguments

fun	a character with the name of normalization function (e.g. "norm_edgeR", "norm_DESeq2", "norm_CSS"...).
method	a character with the normalization method (e.g. "TMM", "upperquartile"... if the fun is "norm_edgeR").

Value

a list object containing the normalization methods and their parameters.

See Also

runNormalizations, norm_edgeR, norm_DESeq2, norm_CSS, norm_TSS

Examples

```
# Set a TMM normalization
my_TMM_normalization <- setNormalizations(fun = "norm_edgeR", method = "TMM")

# Set some simple normalizations
my_normalizations <- setNormalizations()

# Add a custom normalization
my_normalizations <- c(my_normalizations,
    myNormMethod1 = list("myNormMethod", "parameter1", "parameter2"))
```

---

set_ALDEx2 *set_ALDEx2*

Description

Set the parameters for ALDEx2 differential abundance detection method.

Usage

```
set_ALDEx2(
  pseudo_count = FALSE,
  conditions = NULL,
  mc.samples = 128,
  test = "t",
  denom = "iqlr",
```

```
  norm = "TSS",
  expand = TRUE
)
```

Arguments

pseudo_count	add 1 to all counts if TRUE (default pseudo_count = FALSE).
conditions	A character vector. A description of the data structure used for testing. Typically, a vector of group labels. For aldex.glm, use a model.matrix.
mc.samples	An integer. The number of Monte Carlo samples to use when estimating the underlying distributions. Since we are estimating central tendencies, 128 is usually sufficient.
test	A character string. Indicates which tests to perform. "t" runs Welch's t and Wilcoxon tests. "kw" runs Kruskal-Wallace and glm tests. "glm" runs a generalized linear model using a model.matrix. "corr" runs a correlation test using cor.test.
denom	A character string. Indicates which features to retain as the denominator for the Geometric Mean calculation. Using "iqlr" accounts for data with systematic variation and centers the features on the set features that have variance that is between the lower and upper quartile of variance. Using "zero" is a more extreme case where there are many non-zero features in one condition but many zeros in another. In this case the geometric mean of each group is calculated using the set of per-group non-zero features.
norm	name of the normalization method used to compute the normalization factors to use in the differential abundance analysis. If norm is equal to "TMM", "TMMwsp", "RLE", "upperquartile", "posupperquartile", "CSSmedian", "CSSdefault", "TSS" the scaling factors are automatically transformed into normalization factors.
expand	logical, if TRUE create all combinations of input parameters (default expand = TRUE)

Value

A named list containing the set of parameters for DA_ALDEx2 method.

See Also

DA_ALDEx2

Examples

```
# Set some basic combinations of parameters for ALDEx2
base_ALDEx2 <- set_ALDEx2(conditions = "group")
# Set a specific set of normalization for ALDEx2 (even of other
# packages!)
setNorm_ALDEx2 <- set_ALDEx2(conditions = "group", norm = c("TSS", "TMM"))
# Set many possible combinations of parameters for ALDEx2
all_ALDEx2 <- set_ALDEx2(conditions = "group", denom = c("iqlr", "zero"),
    test = c("t", "wilcox"))
```

set_corncob *set_corncob*

Description

Set the parameters for corncob differential abundance detection method.

Usage

```
set_corncob(
  pseudo_count = FALSE,
  formula = NULL,
  phi.formula = NULL,
  formula_null = NULL,
  phi.formula_null = NULL,
  test = c("Wald", "LRT"),
  boot = FALSE,
  coefficient = NULL,
  norm = "TSS",
  expand = TRUE
)
```

Arguments

pseudo_count	add 1 to all counts if TRUE (default pseudo_count = FALSE).
formula	an object of class formula without the response: a symbolic description of the model to be fitted to the abundance.
phi.formula	an object of class formula without the response: a symbolic description of the model to be fitted to the dispersion.
formula_null	Formula for mean under null, without response
phi.formula_null	Formula for overdispersion under null, without response
test	Character. Hypothesis testing procedure to use. One of "Wald" or "LRT" (likelihood ratio test).
boot	Boolean. Defaults to FALSE. Indicator of whether or not to use parametric bootstrap algorithm. (See pbWald and pbLRT).
coefficient	The coefficient of interest as a single word formed by the variable name and the non reference level. (e.g.: 'ConditionDisease' if the reference level for the variable 'Condition' is 'control').
norm	name of the normalization method used to compute the normalization factors to use in the differential abundance analysis. If norm is equal to "TMM", "TMMwsp", "RLE", "upperquartile", "posupperquartile", "CSSmedian", "CSSdefault", "TSS" the scaling factors are automatically transformed into normalization factors.
expand	logical, if TRUE create all combinations of input parameters (default expand = TRUE)

Value

A named list containing the set of parameters for DA_corncob method.

See Also

DA_corncob

Examples

```
# Set some basic combinations of parameters for corncob
base_corncob <- set_corncob(formula = ~ group, phi.formula = ~ group,
    formula_null = ~ 1, phi.formula_null = ~ group, coefficient = "groupB")
# Set a specific set of normalization for corncob (even of other packages!)
setNorm_corncob <- set_corncob(formula = ~ group, phi.formula = ~ group,
    formula_null = ~ 1, phi.formula_null = ~ group, coefficient = "groupB",
    norm = c("TMM", "TSS", "poscounts"))
# Set many possible combinations of parameters for corncob
all_corncob <- set_corncob(pseudo_count = c(TRUE, FALSE), formula = ~ group,
    phi.formula = ~ group, formula_null = ~ 1, phi.formula_null = ~ group,
    coefficient = "groupB", boot = c(TRUE, FALSE))
```

---

set_DESeq2 *set_DESeq2*

Description

Set the parameters for DESeq2 differential abundance detection method.

Usage

```
set_DESeq2(
  pseudo_count = FALSE,
  design = NULL,
  contrast = NULL,
  alpha = 0.05,
  norm = c("ratio", "poscounts", "iterate"),
  weights_logical = FALSE,
  expand = TRUE
)
```

Arguments

pseudo_count	add 1 to all counts if TRUE (default pseudo_count = FALSE).
design	(Required). A formula which specifies the design of the experiment, taking the form formula(~ x + y + z). That is, a formula with right-hand side only. By default, the functions in this package and DESeq2 will use the last variable in the formula (e.g. z) for presenting results (fold changes, etc.) and plotting. When considering your specification of experimental design, you will want to

re-order the levels so that the NULL set is first. For example, the following line of code would ensure that Enterotype 1 is used as the reference sample class in tests by setting it to the first of the factor levels using the relevel function:

```
sample_data(entill)$Enterotype <-relevel(sample_data(entill)$Enterotype,"1")
```

contrast — character vector with exactly three elements: the name of a factor in the design formula, the name of the numerator level for the fold change, and the name of the denominator level for the fold change.

alpha — the significance cutoff used for optimizing the independent filtering (by default 0.05). If the adjusted p-value cutoff (FDR) will be a value other than 0.05, alpha should be set to that value.

norm — name of the normalization method used to compute the normalization factors to use in the differential abundance analysis. If norm is equal to "TMM", "TMMwsp", "RLE", "upperquartile", "posupperquartile", "CSSmedian", "CSSdefault", "TSS" the scaling factors are automatically transformed into normalization factors.

weights_logical — logical vector, if TRUE a matrix of observational weights will be used for differential abundance analysis (default weights_logical = FALSE).

expand — logical, if TRUE create all combinations of input parameters (default expand = TRUE).

Value

A named list containing the set of parameters for DA_DESeq2 method.

See Also

DA_DESeq2

Examples

```
# Set some basic combinations of parameters for DESeq2
base_DESeq2 <- set_DESeq2(design = ~ group, contrast = c("group", "B", "A"))
# Set a specific set of normalization for DESeq2 (even of other packages!)
setNorm_DESeq2 <- set_DESeq2(design = ~ group, contrast =
    c("group", "B", "A"), norm = c("TMM", "poscounts"))
# Set many possible combinations of parameters for edgeR
all_DESeq2 <- set_DESeq2(pseudo_count = c(TRUE, FALSE), design = ~ group,
    contrast = c("group", "B", "A"), weights_logical = c(TRUE,FALSE))
```

---

set_edgeR *set_edgeR*

Description

Set the parameters for edgeR differential abundance detection method.

Usage

```
set_edgeR(
  pseudo_count = FALSE,
  group_name = NULL,
  design = NULL,
  robust = FALSE,
  coef = 2,
  norm = c("TMM", "TMMwsp", "RLE", "upperquartile", "posupperquartile", "none"),
  weights_logical = FALSE,
  expand = TRUE
)
```

Arguments

pseudo_count	add 1 to all counts if TRUE (default pseudo_count = FALSE).
group_name	character giving the name of the column containing information about experimental group/condition for each sample/library.
design	character or formula to specify the model matrix.
robust	logical, should the estimation of prior.df be robustified against outliers?
coef	integer or character index vector indicating which coefficients of the linear model are to be tested equal to zero.
norm	name of the normalization method used to compute the scaling factors to use in the differential abundance analysis. If norm is equal to "ratio", "poscounts", or "iterate" the normalization factors are automatically transformed into scaling factors.
weights_logical	logical vector, if true a matrix of observation weights must be supplied (default weights_logical = FALSE).
expand	logical, if TRUE create all combinations of input parameters (default expand = TRUE).

Value

A named list containing the set of parameters for DA_edgeR method.

See Also

DA_edgeR

Examples

```
# Set some basic combinations of parameters for edgeR
base_edgeR <- set_edgeR(group_name = "group", design = ~ group, coef = 2)

# Set a specific set of normalization for edgeR (even of other packages!)
setNorm_edgeR <- set_edgeR(group_name = "group", design = ~ group, coef = 2,
    norm = c("TMM", "poscounts"))
```

```
# Set many possible combinations of parameters for edgeR
all_edgeR <- set_edgeR(pseudo_count = c(TRUE, FALSE), group_name = "group",
    design = ~ group, robust = c(TRUE, FALSE), coef = 2,
    weights_logical = c(TRUE,FALSE))
```

---

set_limma *set_limma*

Description

Set the parameters for limma differential abundance detection method.

Usage

```
set_limma(
  pseudo_count = FALSE,
  design = NULL,
  coef = 2,
  norm = c("TMM", "TMMwsp", "RLE", "upperquartile", "posupperquartile", "none"),
  weights_logical = FALSE,
  expand = TRUE
)
```

Arguments

pseudo_count	add 1 to all counts if TRUE (default pseudo_count = FALSE).
design	character name of the metadata columns, formula, or design matrix with rows corresponding to samples and columns to coefficients to be estimated.
coef	integer or character index vector indicating which coefficients of the linear model are to be tested equal to zero.
norm	name of the normalization method used to compute the scaling factors to use in the differential abundance analysis. If norm is equal to "ratio", "poscounts", or "iterate" the normalization factors are automatically transformed into scaling factors.
weights_logical	logical vector, if TRUE a matrix of observational weights will be used for differential abundance analysis (default weights_logical = FALSE).
expand	logical, if TRUE create all combinations of input parameters (default expand = TRUE).

Value

A named list containing the set of parameters for DA_limma method.

See Also

DA_limma

Examples

```
# Set some basic combinations of parameters for limma
base_limma <- set_limma(design = ~ group, coef = 2)
# Set a specific set of normalization for limma (even of other packages!)
setNorm_limma <- set_limma(design = ~ group, coef = 2,
    norm = c("TMM", "poscounts"))
# Set many possible combinations of parameters for limma
all_limma <- set_limma(pseudo_count = c(TRUE, FALSE), design = ~ group,
    coef = 2, weights_logical = c(TRUE,FALSE))
```

---

set_MAST *set_MAST*

Description

Set the parameters for MAST differential abundance detection method.

Usage

```
set_MAST(
  pseudo_count = FALSE,
  rescale = c("median", "default"),
  design = NULL,
  coefficient = NULL,
  norm = "TSS",
  expand = TRUE
)
```

Arguments

pseudo_count	add 1 to all counts if TRUE (default pseudo_count = FALSE).
rescale	Rescale count data, per million if 'default', or per median library size if 'median' ('median' is suggested for metagenomics data).
design	The model for the count distribution. Can be the variable name, or a character similar to "~ 1 + group", or a formula, or a 'model.matrix' object.
coefficient	The coefficient of interest as a single word formed by the variable name and the non reference level. (e.g.: 'ConditionDisease' if the reference level for the variable 'Condition' is 'control').
norm	name of the normalization method used to compute the normalization factors to use in the differential abundance analysis. If norm is equal to "TMM", "TMMwsp", "RLE", "upperquartile", "posupperquartile", "CSSmedian", "CSSdefault", "TSS" the scaling factors are automatically transformed into normalization factors.
expand	logical, if TRUE create all combinations of input parameters (default expand = TRUE)

Value

A named list containing the set of parameters for DA_MAST method.

See Also

DA_MAST

Examples

```
# Set some basic combinations of parameters for MAST
base_MAST <- set_MAST(design = ~ group, coefficient = "groupB")
# Set a specific set of normalization for MAST (even of other packages!)
setNorm_MAST <- set_MAST(design = ~ group, coefficient = "groupB",
    norm = c("TSS", "poscounts", "TMM"))
# Set many possible combinations of parameters for MAST
all_MAST <- set_MAST(pseudo_count = c(TRUE, FALSE), rescale = c("median",
    "default"), design = ~ group, coefficient = "groupB", norm = c("TSS",
    "poscounts"))
```

---

set_metagenomeSeq *set_metagenomeSeq*

Description

Set the parameters for metagenomeSeq differential abundance detection method.

Usage

```
set_metagenomeSeq(
  pseudo_count = FALSE,
  design = NULL,
  coef = 2,
  norm = c("CSSmedian", "CSSdefault"),
  expand = TRUE
)
```

Arguments

pseudo_count	add 1 to all counts if TRUE (default pseudo_count = FALSE).
design	The model for the count distribution. Can be the variable name, or a character similar to "~ 1 + group", or a formula, or a ‘model.matrix‘ object.
coef	integer or character index vector indicating which coefficients of the linear model are to be tested equal to zero.
norm	name of the normalization method used to compute the scaling factors to use in the differential abundance analysis. If norm is equal to "ratio", "poscounts", or "iterate" the normalization factors are automatically transformed into scaling factors.
expand	logical, if TRUE create all combinations of input parameters (default expand = TRUE)

Value

A named list containing the set of parameters for DA_metagenomeSeq method.

See Also

DA_metagenomeSeq

Examples

```
# Set some basic combinations of parameters for metagenomeSeq
base_mgs <- set_metagenomeSeq(design = ~ group, coef = 2)
# Set a specific set of normalization for metagenomeSeq (even of other
# packages!)
setNorm_mgs <- set_metagenomeSeq(design = ~ group, coef = 2,
    norm = c("CSSmedian", "TMM"))
# Set many possible combinations of parameters for metagenomeSeq
all_mgs <- set_metagenomeSeq(pseudo_count = c(TRUE, FALSE), design = ~ group,
    coef = 2, norm = c("CSSmedian", "CSSdefault", "TMM", "TSS"))
```

---

set_Seurat *set_Seurat*

Description

Set the parameters for Seurat differential abundance detection method.

Usage

```
set_Seurat(
  pseudo_count = FALSE,
  test.use = c("wilcox", "t"),
  contrast = NULL,
  norm = "TSS",
  expand = TRUE
)
```

Arguments

pseudo_count	add 1 to all counts if TRUE (default pseudo_count = FALSE).
test.use	Denotes which test to use. Available options are: • "wilcox" : Identifies differentially expressed genes between two groups of cells using a Wilcoxon Rank Sum test (default) • "bimod" : Likelihood-ratio test for single cell gene expression, (McDavid et al., Bioinformatics, 2013)

- "roc" : Identifies 'markers' of gene expression using ROC analysis. For each gene, evaluates (using AUC) a classifier built on that gene alone, to classify between two groups of cells. An AUC value of 1 means that expression values for this gene alone can perfectly classify the two groupings (i.e. Each of the cells in cells.1 exhibit a higher level than each of the cells in cells.2). An AUC value of 0 also means there is perfect classification, but in the other direction. A value of 0.5 implies that the gene has no predictive power to classify the two groups. Returns a 'predictive power' (abs(AUC-0.5) * 2) ranked matrix of putative differentially expressed genes.
- "t" : Identify differentially expressed genes between two groups of cells using the Student's t-test.
- "negbinom" : Identifies differentially expressed genes between two groups of cells using a negative binomial generalized linear model. Use only for UMI-based datasets
- "poisson" : Identifies differentially expressed genes between two groups of cells using a poisson generalized linear model. Use only for UMI-based datasets
- "LR" : Uses a logistic regression framework to determine differentially expressed genes. Constructs a logistic regression model predicting group membership based on each feature individually and compares this to a null model with a likelihood ratio test.
- "MAST" : Identifies differentially expressed genes between two groups of cells using a hurdle model tailored to scRNA-seq data. Utilizes the MAST package to run the DE testing.
- "DESeq2" : Identifies differentially expressed genes between two groups of cells based on a model using DESeq2 which uses a negative binomial distribution (Love et al, Genome Biology, 2014).This test does not support pre-filtering of genes based on average difference (or percent detection rate) between cell groups. However, genes may be pre-filtered based on their minimum detection rate (min.pct) across both cell groups. To use this method, please install DESeq2, using the instructions at https://bioconductor.org/packages/release/bioc/html/D

`contrast` character vector with exactly three elements: the name of a factor in the design formula, the name of the numerator level for the fold change, and the name of the denominator level for the fold change.

`norm` name of the normalization method used to compute the normalization factors to use in the differential abundance analysis. If `norm` is equal to "TMM", "TMMwsp", "RLE", "upperquartile", "posupperquartile", "CSSmedian", "CSSdefault", "TSS" the scaling factors are automatically transformed into normalization factors.

`expand` logical, if TRUE create all combinations of input parameters (default `expand = TRUE`)

Value

A named list containing the set of parameters for `DA_Seurat` method.

See Also

`DA_Seurat`

Examples

```
# Set some basic combinations of parameters for Seurat
base_Seurat <- set_Seurat(contrast = c("group", "B", "A"))
# Set a specific set of normalization for Seurat (even of other packages!)
setNorm_Seurat <- set_Seurat(contrast = c("group", "B", "A"), norm = c("TSS",
"TMM", "poscounts"))
# Set many possible combinations of parameters for Seurat
all_Seurat <- set_Seurat(pseudo_count = c(TRUE, FALSE),
    test.use = c("wilcox", "t", "negbinom", "poisson"),
    contrast = c("group", "B", "A"), norm = c("TSS", "TMM"))
```

`weights_ZINB` *weights_ZINB*

Description

Computes the observational weights of the counts under a zero-inflated negative binomial (ZINB) model. For each count, the ZINB distribution is parametrized by three parameters: the mean value and the dispersion of the negative binomial distribution, and the probability of the zero component.

Usage

```
weights_ZINB(
  object,
  design,
  K = 0,
  commondispersion = TRUE,
  zeroinflation = TRUE,
  verbose = FALSE,
  ...
)
```

Arguments

`object`	phyloseq object containing the counts and the sample data.
`design`	character name of the metadata columns, formula, or design matrix with rows corresponding to samples and columns to coefficients to be estimated (the user needs to explicitly include the intercept in the design).
`K`	integer. Number of latent factors.
`commondispersion`	Whether or not a single dispersion for all features is estimated (default TRUE).
`zeroinflation`	Whether or not a ZINB model should be fitted. If FALSE, a negative binomial model is fitted instead.
`verbose`	Print helpful messages.
`...`	Additional parameters to describe the model, see `zinbModel`.

Value

A matrix of weights.

See Also

zinbFit for zero-inflated negative binomial parameters' estimation and computeObservationalWeights for weights extraction.

Examples

```
set.seed(1)
# Create a very simple phyloseq object
counts <- matrix(rnbinom(n = 60, size = 3, prob = 0.5), nrow = 10, ncol = 6)
metadata <- data.frame("Sample" = c("S1", "S2", "S3", "S4", "S5", "S6"))
ps <- phyloseq::phyloseq(phyloseq::otu_table(counts, taxa_are_rows = TRUE),
    phyloseq::sample_data(metadata))
# Calculate the ZINB weights
zinbweights <- weights_ZINB(object = ps, K = 0, design = "~ 1")
```

Index