Package: compcodeR (via r-universe)

November 6, 2026

Type Package

Title RNAseq data simulation, differential expression analysis and performance comparison of differential expression methods

Version 1.48.1

Description This package provides extensive functionality for comparing results obtained by different methods for differential expression analysis of RNAseq data. It also contains functions for simulating count data. Finally, it provides convenient interfaces to several packages for performing the differential expression analysis. These can also be used as templates for setting up and running a user-defined differential analysis workflow within the framework of the package.

Depends R (>= 4.0), sm

Imports knitr (>= 1.2), markdown, ROCR, lattice (>= 0.16), gplots, gtools, caTools, grid, KernSmooth, MASS, ggplot2, stringr, statip, edgeR, limma, vioplot, methods, stats, utils, ape, phylolm, matrixStats, grDevices, graphics, rmarkdown, shiny, shinydashboard

Suggests BiocStyle, EBSeq, DESeq2 (>= 1.1.31), genefilter, NOISeq, TCC, NBPSeq (>= 0.3.0), phytools, phangorn, testthat, ggtree, tidytree, statmod, covr, sva, tcltk

Enhances rpanel, DSS

License GPL (>= 2)

VignetteBuilder knitr

biocViews ImmunoOncology, RNASeq, DifferentialExpression

URL https://github.com/csoneson/compcodeR

BugReports https://github.com/csoneson/compcodeR/issues

Encoding UTF-8

Config/roxygen2/version 8.1.0

Config/pak/sysreqs cmake make libicu-dev libuv1-dev zlib1g-dev

Repository Bioconductor 3.23

Date/Publication 2026-09-10 08:18:26 UTC

RemoteUrl https://github.com/bioc/compcodeR

RemoteRef RELEASE_3_23

RemoteSha decbcc38b279dbec585627721f06c9f225053e53

Contents

check_compData . . . 3
check_compData_results . . . 3
check_phyloCompData . . . 4
checkDataObject . . . 5
checkTableConsistency . . . 5
compData . . . 7
compData-class . . . 10
convertcompDataToList . . . 12
convertListTocompData . . . 13
convertListTophyloCompData . . . 13
convertphyloCompDataToList . . . 14
DESeq2.createRmd . . . 15
DESeq2.length.createRmd . . . 17
DSS.createRmd . . . 19
EBSeq.createRmd . . . 20
edgeR.exact.createRmd . . . 22
edgeR.GLM.createRmd . . . 23
generateCodeHTMLs . . . 25
generateSyntheticData . . . 26
lengthNorm.limma.createRmd . . . 30
lengthNorm.sva.limma.createRmd . . . 33
listcreateRmd . . . 35
logcpm.limma.createRmd . . . 36
NBPSeq.createRmd . . . 37
NOISeq.prenorm.createRmd . . . 39
phyloCompData . . . 40
phyloCompData-class . . . 44
phylolm.createRmd . . . 45
runComparison . . . 47
runComparisonGUI . . . 51
runComparisonShiny . . . 52
runDiffExp . . . 54
show,compData-method . . . 56
show,phyloCompData-method . . . 57
sqrtcpm.limma.createRmd . . . 58
summarizeSyntheticDataSet . . . 59
TCC.createRmd . . . 60
ttest.createRmd . . . 61

voom.limma.createRmd 63
voom.ttest.createRmd 64

Index **66**

`check_compData` *Check the validity of a* `compData` *object*

Description

Check the validity of a `compData` object. An object that passes the check can be used as the input for the differential expression analysis methods interfaced by `compcodeR`.

Usage

```
check_compData(object)
```

Arguments

`object`	A `compData` object

Author(s)

Charlotte Soneson

Examples

```
mydata <- generateSyntheticData(dataset = "mydata", n.vars = 1000,
                                samples.per.cond = 5, n.diffexp = 100)
check_compData(mydata)
```

`check_compData_results`
Check the validity of a `compData` *result object*

Description

Check the validity of a `compData` object containing differential expression results. An object that passes the check can be used as the input for the method comparison functions in `compcodeR`.

Usage

```
check_compData_results(object)
```

Arguments

`object`	A `compData` object

Author(s)

Charlotte Soneson

Examples

```
tmpdir <- normalizePath(tempdir(), winslash = "/")
mydata <- generateSyntheticData(dataset = "mydata", n.vars = 1000,
                                samples.per.cond = 5, n.diffexp = 100,
                                output.file = file.path(tmpdir, "mydata.rds"))
## Check an object without differential expression results
check_compData_results(mydata)

runDiffExp(data.file = file.path(tmpdir, "mydata.rds"),
           result.extent = "voom.limma",
           Rmdfunction = "voom.limma.createRmd",
           output.directory = tmpdir, norm.method = "TMM")
resdata <- readRDS(file.path(tmpdir, "mydata_voom.limma.rds"))
## Check an object containing differential expression results
check_compData_results(resdata)
```

---

check_phyloCompData *Check the validity of a* phyloCompData *object*

Description

Check the validity of a phyloCompData object. An object that passes the check can be used as the input for the differential expression analysis methods interfaced by compcodeR.

Usage

```
check_phyloCompData(object)
```

Arguments

object	A phyloCompData object

Author(s)

Charlotte Soneson, Paul Bastide

Examples

```
mydata <- generateSyntheticData(dataset = "mydata", n.vars = 1000,
                                samples.per.cond = 5, n.diffexp = 100,
                                id.species = factor(1:10),
                                tree = ape::rphylo(10, 1, 0),
                                lengths.relmeans = "auto", lengths.dispersions = "auto")
check_phyloCompData(mydata)
```

checkDataObject *Check a list or a* compData *object for compatibility with the differential expression functions interfaced by* compcodeR

Description

Check if a list or a compData object contains the necessary slots for applying the differential expression functions interfaced by the compcodeR package. This function is provided for backward compatibility, see also check_compData and check_compData_results.

Usage

```
checkDataObject(data.obj)
```

Arguments

data.obj	A list containing data and condition information, or a compData object.

Author(s)

Charlotte Soneson

Examples

```
mydata.obj <- generateSyntheticData(dataset = "mydata", n.vars = 1000,
                                    samples.per.cond = 5, n.diffexp = 100)
checkDataObject(mydata.obj)
```

checkTableConsistency *Check consistency of input table to* runComparison

Description

Check that the dataset, nbr.samples, repl and de.methods columns of a data frame are consistent with the information provided in the input files (given in the input.files column of the data frame). If there are inconsistencies or missing information in any of the columns, replace the given information with the information in the input files.

Usage

```
checkTableConsistency(file.table)
```

Arguments

file.table	A data frame with columns named input.files and (optionally) datasets, nbr.samples, repl, de.methods.

Value

Returns a consistent file table defining the result files that will be used as the basis for a method comparison.

Author(s)

Charlotte Soneson

Examples

```
tmpdir <- normalizePath(tempdir(), winslash = "/")
mydata.obj <- generateSyntheticData(dataset = "mydata", n.vars = 1000,
                                    samples.per.cond = 5, n.diffexp = 100,
                                    output.file = file.path(tmpdir, "mydata.rds"))
runDiffExp(data.file = file.path(tmpdir, "mydata.rds"), result.extent = "voom.limma",
           Rmdfunction = "voom.limma.createRmd", output.directory = tmpdir,
           norm.method = "TMM")
runDiffExp(data.file = file.path(tmpdir, "mydata.rds"), result.extent = "edgeR.exact",
           Rmdfunction = "edgeR.exact.createRmd", output.directory = tmpdir,
           norm.method = "TMM",
           trend.method = "movingave", disp.type = "tagwise")

## A correct table
file.table <- data.frame(input.files = file.path(tmpdir,
                         c("mydata_voom.limma.rds", "mydata_edgeR.exact.rds")),
                         datasets = c("mydata", "mydata"),
                         nbr.samples = c(5, 5),
                         repl = c(1, 1),
                         stringsAsFactors = FALSE)
new.table <- checkTableConsistency(file.table)
new.table

## An incorrect table
file.table <- data.frame(input.files = file.path(tmpdir,
                         c("mydata_voom.limma.rds", "mydata_edgeR.exact.rds")),
                         datasets = c("mydata", "mydata"),
                         nbr.samples = c(5, 3),
                         repl = c(2, 1),
                         stringsAsFactors = FALSE)
new.table <- checkTableConsistency(file.table)
new.table

## A table with missing information
file.table <- data.frame(input.files = file.path(tmpdir,
                         c("mydata_voom.limma.rds", "mydata_edgeR.exact.rds")),
                         stringsAsFactors = FALSE)
new.table <- checkTableConsistency(file.table)
new.table
```

compData *Create a* compData *object*

Description

The compData class is used to store information about the experiment, such as the count matrix, sample and variable annotations, information regarding the generation of the data and results from applying a differential expression analysis to the data. This constructor function creates a compData object.

Usage

```
compData(
  count.matrix,
  sample.annotations,
  info.parameters,
  variable.annotations = data.frame(),
  filtering = "no info",
  analysis.date = "",
  package.version = "",
  method.names = list(),
  code = "",
  result.table = data.frame()
)
```

Arguments

count.matrix A count matrix, with genes as rows and observations as columns.

sample.annotations
A data frame, containing at least one column named 'condition', encoding the grouping of the observations into two groups. The row names should be the same as the column names of the count.matrix.

info.parameters
A list containing information regarding simulation parameters etc. The only mandatory entries are dataset and uID, but it may contain entries such as the ones listed below (see generateSyntheticData for more detailed information about each of these entries).

- dataset: an informative name or identifier of the data set (e.g., summarizing the simulation settings).
- samples.per.cond
- n.diffexp
- repl.id
- seqdepth
- minfact
- maxfact

- fraction.upregulated
- between.group.diffdisp
- filter.threshold.total
- filter.threshold.mediancpm
- fraction.non.overdispersed
- random.outlier.high.prob
- random.outlier.low.prob
- single.outlier.high.prob
- single.outlier.low.prob
- effect.size
- uID: a unique ID for the data set. In contrast to dataset, the uID is unique e.g. for each instance of replicated data sets generated with the same simulation settings.

variable.annotations

A data frame with variable annotations (with number of rows equal to the number of rows in count.matrix, that is, the number of variables in the data set). Not mandatory, but may contain columns such as the ones listed below. If present, the row names should be the same as the row names of the count.matrix.

- truedispersions.S1: the true dispersion for each gene in condition S1.
- truedispersions.S2: the true dispersion for each gene in condition S2.
- truemeans.S1: the true mean value for each gene in condition S1.
- truemeans.S2: the true mean value for each gene in condition S2.
- n.random.outliers.up.S1: the number of 'random' outliers with extremely high counts for each gene in condition S1.
- n.random.outliers.up.S2: the number of 'random' outliers with extremely high counts for each gene in condition S2.
- n.random.outliers.down.S1: the number of 'random' outliers with extremely low counts for each gene in condition S1.
- n.random.outliers.down.S2: the number of 'random' outliers with extremely low counts for each gene in condition S2.
- n.single.outliers.up.S1: the number of 'single' outliers with extremely high counts for each gene in condition S1.
- n.single.outliers.up.S2: the number of 'single' outliers with extremely high counts for each gene in condition S2.
- n.single.outliers.down.S1: the number of 'single' outliers with extremely low counts for each gene in condition S1.
- n.single.outliers.down.S2: the number of 'single' outliers with extremely low counts for each gene in condition S2.
- M.value: the M-value (observed log2 fold change between condition S1 and condition S2) for each gene.
- A.value: the A-value (observed average expression level across condition S1 and condition S2) for each gene.
- truelog2foldchanges: the true (simulated) log2 fold changes between condition S1 and condition S2.

- upregulation: a binary vector indicating which genes are simulated to be upregulated in condition S2 compared to condition S1.
- downregulation: a binary vector indicating which genes are simulated to be downregulated in condition S2 compared to condition S1.
- differential.expression: a binary vector indicating which genes are simulated to be differentially expressed in condition S2 compared to condition S1.

filtering A character string containing information about the filtering that has been applied to the data set.

analysis.date If a differential expression analysis has been performed, a character string detailing when it was performed.

package.version
If a differential expression analysis has been performed, a character string giving the version of the differential expression packages that were applied.

method.names If a differential expression analysis has been performed, a list with entries full.name and short.name, giving the full name of the differential expression method (may including version number and parameter settings) and a short name or abbreviation.

code If a differential expression analysis has been performed, a character string containing the code that was run to perform the analysis. The code should be in R markdown format, and can be written to an HTML file using the generateCodeHTMLs function.

result.table If a differential expression analysis has been performed, a data frame containing the results of the analysis. The number of rows should be equal to the number of rows in count.matrix and if present, the row names should be identical. The only mandatory column is score, which gives a score for each gene, where a higher score suggests a "more highly differentially expressed" gene. Different comparison functions use different columns of this table, if available. The list below gives the columns that are used by the interfaced methods.

- pvalue nominal p-values
- adjpvalue p-values adjusted for multiple comparisons
- logFC estimated log-fold changes between the two conditions
- score the score that will be used to rank the genes in order of significance. Note that high scores always signify differential expression, that is, a strong association with the predictor. For example, for methods returning a nominal p-value the score can be defined as 1 - pvalue.
- FDR false discovery rate estimates
- posterior.DE posterior probabilities of differential expression
- prob.DE conditional probabilities of differential expression
- lfdr local false discovery rates
- statistic test statistics from the differential expression analysis
- dispersion.S1 dispersion estimates in condition S1
- dispersion.S2 dispersion estimates in condition S2

Value

A compData object.

Author(s)

Charlotte Soneson

Examples

```
count.matrix <- round(matrix(1000*runif(4000), 1000))
sample.annotations <- data.frame(condition = c(1, 1, 2, 2))
info.parameters <- list(dataset = "mydata", uID = "123456")
cpd <- compData(count.matrix, sample.annotations, info.parameters)
```

---

compData-class *Class compData*

Description

The compData class is used to store information about the experiment, such as the count matrix, sample and variable annotations, information regarding the generation of the data and results from applying a differential expression analysis to the data.

Slots

count.matrix: The read count matrix, with genes as rows and samples as columns. Class matrix

sample.annotations: A data frame containing sample annotation information for all samples in the data set. Must contain at least a column named condition, encoding the division of the samples into two classes. The row names should be the same as the column names of count.matrix. Class data.frame

info.parameters: A list of parameters detailing the simulation process used to generate the data. Must contain at least two entries, named dataset (an informative name for the data set/simulation setting) and uID (a unique ID for the specific data set instance). Class list

filtering: A character string detailing the filtering process that has been applied to the data. Class character

variable.annotations: Contains information regarding the variables, such as the differential expression status, the true mean, dispersion and effect sizes. If present, the row names should be the same as those of count.matrix. Class data.frame

analysis.date: (If a differential expression analysis has been performed and the results are included in the compData object). Gives the date when the differential expression analysis was performed. Class character

package.version: (If a differential expression analysis has been performed and the results are included in the compData object). Gives the version numbers of the package(s) used for the differential expression analysis. Class character

`method.names`: (If a differential expression analysis has been performed and the results are included in the `compData` object). A list, containing the name of the method used for the differential expression analysis. The list should have two entries: `full.name` and `short.name`, where the `full.name` is the full (potentially long) name identifying the method, and `short.name` may be an abbreviation. Class `list`

`code`: (If a differential expression analysis has been performed and the results are included in the `compData` object). A character string containing the code that was used to run the differential expression analysis. The code should be in R markdown format. Class `character`

`result.table`: (If a differential expression analysis has been performed and the results are included in the `compData` object). Contains the results of the differential expression analysis, in the form of a data frame with one row per gene. Must contain at least one column named `score`, where a higher value corresponds to 'more strongly differentially expressed genes'. Class `data.frame`

Methods

count.matrix `signature(x="compData")`

count.matrix<- `signature(x="compData",value="matrix")`: Get or set the count matrix in a `compData` object. `value` should be a numeric matrix.

sample.annotations `signature(x="compData")`

sample.annotations<- `signature(x="compData",value="data.frame")`: Get or set the sample annotations data frame in a `compData` object. `value` should be a data frame with at least a column named 'condition'.

info.parameters `signature(x="compData")`

info.parameters<- `signature(x="compData",value="list")`: Get or set the list with info parameters in a `compData` object. `value` should be a list with at least elements named 'dataset' and 'uID'.

filtering `signature(x="compData")`

filtering<- `signature(x="compData",value="character")`: Get or set the information about the filtering in a `compData` object. `value` should be a character string describing the filtering that has been performed.

variable.annotations `signature(x="compData")`

variable.annotations<- `signature(x="compData",value="data.frame")`: Get or set the variable annotations data frame in a `compData` object. `value` should be a data frame.

analysis.date `signature(x="compData")`

analysis.date<- `signature(x="compData",value="character")`: Get or set the analysis date in a `compData` object. `value` should be a character string describing when the differential expression analysis of the data was performed.

package.version `signature(x="compData")`

package.version<- `signature(x="compData",value="character")`: Get or set the information about the package version in a `compData` object. `value` should be a character string detailing which packages and versions were used to perform the differential expression analysis of the data.

method.names `signature(x="compData")`

method.names<- `signature(x="compData",value="list")`: Get or set the method names in a `compData` object. `value` should be a list with slots `full.name` and `short.name`, giving the full name and an abbreviation for the method that was used to perform the analysis of the data.

code `signature(x="compData")`

code<- `signature(x="compData",value="character")`: Get or set the code slot in a `compData` object. `value` should be a character string in R markdown format, giving the code that was run to obtain the results from the differential expression analysis.

result.table `signature(x="compData")`

result.table<- `signature(x="compData",value="data.frame")`: Get or set the result table in a `compData` object. `value` should be a data frame with one row per gene, and at least a column named 'score'.

Construction

An object of the class `compData` can be constructed using the `compData` function.

Author(s)

Charlotte Soneson

---

convertcompDataToList *Convert a* `compData` *object to a list*

Description

Given a `compData` object, convert it to a list.

Usage

```
convertcompDataToList(cpd)
```

Arguments

cpd	A `compData` object

Author(s)

Charlotte Soneson

Examples

```
mydata.obj <- generateSyntheticData(dataset = "mydata", n.vars = 12500,
                                    samples.per.cond = 5, n.diffexp = 1250)
mydata.list <- convertcompDataToList(mydata.obj)
```

convertListTocompData *Convert a list with data and results to a* compData *object*

Description

Given a list with data and results (resulting e.g. from compcodeR version 0.1.0), convert it to a compData object.

Usage

```
convertListTocompData(inp.list)
```

Arguments

inp.list	A list with data and results, e.g. generated by compcodeR version 0.1.0.

Author(s)

Charlotte Soneson

Examples

```
convertListTocompData(list(count.matrix = matrix(round(1000*runif(4000)), 1000),
                           sample.annotations = data.frame(condition = c(1,1,2,2)),
                           info.parameters = list(dataset = "mydata",
                           uID = "123456")))
```

convertListTophyloCompData *Convert a list with data and results to a* phyloCompData *object*

Description

Given a list with data and results (resulting e.g. from compcodeR version 0.1.0), convert it to a phyloCompData object.

Usage

```
convertListTophyloCompData(inp.list)
```

Arguments

inp.list	A list with data and results, e.g. generated by compcodeR version 0.1.0.

Author(s)

Charlotte Soneson, Paul Bastide

Examples

```
tree <- ape::read.tree(
  text = "(((A1:0,A2:0,A3:0):1,B1:1):1,((C1:0,C2:0):1.5,(D1:0,D2:0):1.5):0.5);"
  )
count.matrix <- round(matrix(1000*runif(8000), 1000))
sample.annotations <- data.frame(condition = c(1, 1, 1, 1, 2, 2, 2, 2),
                                 id.species = c("A", "A", "A", "B", "C", "C", "D", "D"))
info.parameters <- list(dataset = "mydata", uID = "123456")
length.matrix <- round(matrix(1000*runif(8000), 1000))
colnames(count.matrix) <- colnames(length.matrix) <- rownames(sample.annotations) <- tree$tip.label
convertListTophyloCompData(list(count.matrix = count.matrix,
                                sample.annotations = sample.annotations,
                                info.parameters = list(dataset = "mydata",
                                                       uID = "123456"),
                                tree = tree,
                                length.matrix = length.matrix))
```

---

convertphyloCompDataToList

Convert a phyloCompData *object to a list*

Description

Given a phyloCompData object, convert it to a list.

Usage

```
convertphyloCompDataToList(cpd)
```

Arguments

cpd	A phyloCompData object

Author(s)

Charlotte Soneson, Paul Bastide

Examples

```
tree <- ape::read.tree(
  text = "(((A1:0,A2:0,A3:0):1,B1:1):1,((C1:0,C2:0):1.5,(D1:0,D2:0):1.5):0.5);"
  )
id.species <- factor(c("A", "A", "A", "B", "C", "C", "D", "D"))
names(id.species) <- tree$tip.label
mydata.obj <- generateSyntheticData(dataset = "mydata", n.vars = 1000,
                                    samples.per.cond = 4, n.diffexp = 100,
                                    tree = tree,
                                    id.species = id.species)
```

```
mydata.list <- convertcompDataToList(mydata.obj)
```

`DESeq2.createRmd` *Generate a* `.Rmd` *file containing code to perform differential expression analysis with DESeq2*

Description

A function to generate code that can be run to perform differential expression analysis of RNAseq data (comparing two conditions) using the DESeq2 package. The code is written to a `.Rmd` file. This function is generally not called by the user, the main interface for performing differential expression analysis is the `runDiffExp` function.

Usage

```
DESeq2.createRmd(
  data.path,
  result.path,
  codefile,
  fit.type,
  test,
  beta.prior = TRUE,
  independent.filtering = TRUE,
  cooks.cutoff = TRUE,
  impute.outliers = TRUE,
  nas.as.ones = FALSE
)
```

Arguments

`data.path`	The path to a .rds file containing the `compData` object that will be used for the differential expression analysis.
`result.path`	The path to the file where the result object will be saved.
`codefile`	The path to the file where the code will be written.
`fit.type`	The fitting method used to get the dispersion-mean relationship. Possible values are `"parametric"`, `"local"` and `"mean"`.
`test`	The test to use. Possible values are `"Wald"` and `"LRT"`.
`beta.prior`	Whether or not to put a zero-mean normal prior on the non-intercept coefficients. Default is `TRUE`.
`independent.filtering`	Whether or not to perform independent filtering of the data. With independent filtering=TRUE, the adjusted p-values for genes not passing the filter threshold are set to NA.

`cooks.cutoff` The cutoff value for the Cook's distance to consider a value to be an outlier. Set to Inf or FALSE to disable outlier detection. For genes with detected outliers, the p-value and adjusted p-value will be set to NA.

`impute.outliers` Whether or not the outliers should be replaced by a trimmed mean and the analysis rerun.

`nas.as.ones` Whether or not adjusted p values that are returned as `NA` by `DESeq2` should be set to `1`. This option is useful for comparisons with other methods. For more details, see section "I want to benchmark DESeq2 comparing to other DE tools" from the `DESeq2` vignette (available by running `vignette("DESeq2", package = "DESeq2")`). Default to `FALSE`.

Details

For more information about the methods and the interpretation of the parameters, see the `DESeq2` package and the corresponding publications.

Value

The function generates a `.Rmd` file containing the code for performing the differential expression analysis. This file can be executed using e.g. the `knitr` package.

Author(s)

Charlotte Soneson

References

Anders S and Huber W (2010): Differential expression analysis for sequence count data. Genome Biology 11:R106

Examples

```
try(
if (require(DESeq2)) {
tmpdir <- normalizePath(tempdir(), winslash = "/")
mydata.obj <- generateSyntheticData(dataset = "mydata", n.vars = 1000,
                                    samples.per.cond = 5, n.diffexp = 100,
                                    output.file = file.path(tmpdir, "mydata.rds"))
runDiffExp(data.file = file.path(tmpdir, "mydata.rds"), result.extent = "DESeq2",
           Rmdfunction = "DESeq2.createRmd",
           output.directory = tmpdir, fit.type = "parametric",
           test = "Wald")
})
```

DESeq2.length.createRmd *Generate a* .Rmd *file containing code to perform differential expression analysis with DESeq2 with custom model matrix*

Description

A function to generate code that can be run to perform differential expression analysis of RNAseq data (comparing two conditions) using the DESeq2 package. The code is written to a .Rmd file. This function is generally not called by the user, the main interface for performing differential expression analysis is the runDiffExp function.

Usage

```
DESeq2.length.createRmd(
  data.path,
  result.path,
  codefile,
  fit.type,
  test,
  beta.prior = TRUE,
  independent.filtering = TRUE,
  cooks.cutoff = TRUE,
  impute.outliers = TRUE,
  extra.design.covariates = NULL,
  nas.as.ones = FALSE
)
```

Arguments

data.path	The path to a .rds file containing the phyloCompData object that will be used for the differential expression analysis.
result.path	The path to the file where the result object will be saved.
codefile	The path to the file where the code will be written.
fit.type	The fitting method used to get the dispersion-mean relationship. Possible values are "parametric", "local" and "mean".
test	The test to use. Possible values are "Wald" and "LRT".
beta.prior	Whether or not to put a zero-mean normal prior on the non-intercept coefficients. Default is TRUE.
independent.filtering	Whether or not to perform independent filtering of the data. With independent filtering=TRUE, the adjusted p-values for genes not passing the filter threshold are set to NA.
cooks.cutoff	The cutoff value for the Cook's distance to consider a value to be an outlier. Set to Inf or FALSE to disable outlier detection. For genes with detected outliers, the p-value and adjusted p-value will be set to NA.

`impute.outliers`
: Whether or not the outliers should be replaced by a trimmed mean and the analysis rerun.

`extra.design.covariates`
: A vector containing the names of extra control variables to be passed to the design matrix of `DESeq2`. All the covariates need to be a column of the `sample.annotations` data frame from the `phyloCompData` object, with a matching column name. The covariates can be a numeric vector, or a factor. Note that "condition" factor column is always included, and should not be added here. See Details.

`nas.as.ones`
: Whether or not adjusted p values that are returned as `NA` by `DESeq2` should be set to `1`. This option is useful for comparisons with other methods. For more details, see section "I want to benchmark DESeq2 comparing to other DE tools" from the `DESeq2` vignette (available by running `vignette("DESeq2", package = "DESeq2")`). Default to `FALSE`.

Details

For more information about the methods and the interpretation of the parameters, see the `DESeq2` package and the corresponding publications.

The lengths matrix is used as a normalization factor and applied to the `DESeq2` model in the way explained in `normalizationFactors` (see examples of this function). The provided matrix will be multiplied by the default normalization factor obtained through the `estimateSizeFactors` function.

The `design` model used in the `DESeqDataSetFromMatrix` uses the "condition" column of the `sample.annotations` data frame from the `phyloCompData` object as well as all the covariates named in `extra.design.covariates`. For example, if `extra.design.covariates = c("var1", "var2")`, then `sample.annotations` must have two columns named "var1" and "var2", and the design formula in the `DESeqDataSetFromMatrix` function will be: `~ condition + var1 + var2`.

Value

The function generates a `.Rmd` file containing the code for performing the differential expression analysis. This file can be executed using e.g. the `knitr` package.

Author(s)

Charlotte Soneson, Paul Bastide, Mélina Gallopin

References

Anders S and Huber W (2010): Differential expression analysis for sequence count data. Genome Biology 11:R106

Love, M.I., Huber, W., Anders, S. (2014) Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2. Genome Biology, 15:550. 10.1186/s13059-014-0550-8.

Examples

```
try(
if (require(DESeq2)) {
tmpdir <- normalizePath(tempdir(), winslash = "/")
## Simulate data
mydata.obj <- generateSyntheticData(dataset = "mydata", n.vars = 1000,
                                    samples.per.cond = 5, n.diffexp = 100,
                                    id.species = 1:10,
                                    lengths.relmeans = rpois(1000, 1000),
                                    lengths.dispersions = rgamma(1000, 1, 1),
                                    output.file = file.path(tmpdir, "mydata.rds"))
## Add covariates
## Model fitted is count.matrix ~ condition + test_factor + test_reg
sample.annotations(mydata.obj)$test_factor <- factor(rep(1:2, each = 5))
sample.annotations(mydata.obj)$test_reg <- rnorm(10, 0, 1)
saveRDS(mydata.obj, file.path(tmpdir, "mydata.rds"))
## Diff Exp
runDiffExp(data.file = file.path(tmpdir, "mydata.rds"), result.extent = "DESeq2",
           Rmdfunction = "DESeq2.length.createRmd",
           output.directory = tmpdir, fit.type = "parametric",
           test = "Wald",
           extra.design.covariates = c("test_factor", "test_reg"))
})
```

`DSS.createRmd` *Generate a* `.Rmd` *file containing code to perform differential expression analysis with DSS*

Description

A function to generate code that can be run to perform differential expression analysis of RNAseq data (comparing two conditions) using the DSS package. The code is written to a `.Rmd` file. This function is generally not called by the user, the main interface for performing differential expression analysis is the `runDiffExp` function.

Usage

```
DSS.createRmd(data.path, result.path, codefile, norm.method, disp.trend)
```

Arguments

`data.path`	The path to a .rds file containing the `compData` object that will be used for the differential expression analysis.
`result.path`	The path to the file where the result object will be saved.
`codefile`	The path to the file where the code will be written.
`norm.method`	The between-sample normalization method used to compensate for varying library sizes and composition in the differential expression analysis. Possible values are `"quantile"`, `"total"` and `"median"`.

`disp.trend` A logical parameter indicating whether or not to include a trend in the dispersion estimation.

Details

For more information about the methods and the interpretation of the parameters, see the DSS package and the corresponding publications.

Author(s)

Charlotte Soneson

References

Wu H, Wang C and Wu Z (2013): A new shrinkage estimator for dispersion improves differential expression detection in RNA-seq data. Biostatistics 14(2), 232-243

Examples

```
try(
if (require(DSS)) {
tmpdir <- normalizePath(tempdir(), winslash = "/")
mydata.obj <- generateSyntheticData(dataset = "mydata", n.vars = 1000,
                                    samples.per.cond = 5, n.diffexp = 100,
                                    output.file = file.path(tmpdir, "mydata.rds"))
runDiffExp(data.file = file.path(tmpdir, "mydata.rds"), result.extent = "DSS",
           Rmdfunction = "DSS.createRmd",
           output.directory = tmpdir, norm.method = "quantile",
           disp.trend = TRUE)
})
```

EBSeq.createRmd *Generate a .Rmd file containing code to perform differential expression analysis with EBSeq*

Description

A function to generate code that can be run to perform differential expression analysis of RNAseq data (comparing two conditions) using the EBSeq package. The code is written to a .Rmd file. This function is generally not called by the user, the main interface for performing differential expression analysis is the `runDiffExp` function.

Usage

```
EBSeq.createRmd(data.path, result.path, codefile, norm.method)
```

Arguments

data.path	The path to a .rds file containing the compData object that will be used for the differential expression analysis.
result.path	The path to the file where the result object will be saved.
codefile	The path to the file where the code will be written.
norm.method	The between-sample normalization method used to compensate for varying library sizes and composition in the differential expression analysis. Possible values are "median" and "quantile".

Details

For more information about the methods and the meaning of the parameters, see the EBSeq package and the corresponding publications.

Value

The function generates a .Rmd file containing the differential expression code. This file can be executed using e.g. the knitr package.

Author(s)

Charlotte Soneson

References

Leng N, Dawson JA, Thomson JA, Ruotti V, Rissman AI, Smits BMG, Haag JD, Gould MN, Stewart RM and Kendziorski C (2013): EBSeq: An empirical Bayes hierarchical model for inference in RNA-seq experiments. Bioinformatics

Examples

```
try(
if (require(EBSeq)) {
tmpdir <- normalizePath(tempdir(), winslash = "/")
mydata.obj <- generateSyntheticData(dataset = "mydata", n.vars = 1000,
                                    samples.per.cond = 5, n.diffexp = 100,
                                    output.file = file.path(tmpdir, "mydata.rds"))
runDiffExp(data.file = file.path(tmpdir, "mydata.rds"), result.extent = "EBSeq",
           Rmdfunction = "EBSeq.createRmd",
           output.directory = tmpdir, norm.method = "median")
}
)
```

edgeR.exact.createRmd *Generate a* .Rmd *file containing code to perform differential expression analysis with the edgeR exact test*

Description

A function to generate code that can be run to perform differential expression analysis of RNAseq data (comparing two conditions) using the exact test functionality from the edgeR package. The code is written to a .Rmd file. This function is generally not called by the user, the main interface for performing differential expression analysis is the runDiffExp function.

Usage

```
edgeR.exact.createRmd(
  data.path,
  result.path,
  codefile,
  norm.method,
  trend.method,
  disp.type
)
```

Arguments

data.path	The path to a .rds file containing the compData object that will be used for the differential expression analysis.
result.path	The path to the file where the result object will be saved.
codefile	The path to the file where the code will be written.
norm.method	The between-sample normalization method used to compensate for varying library sizes and composition in the differential expression analysis. Possible values are "TMM", "RLE", "upperquartile" and "none".
trend.method	The method used to estimate the trend in the mean-dispersion relationship. Possible values are "none", "movingave" and "loess"
disp.type	The type of dispersion estimate used. Possible values are "common", "trended" and "tagwise".

Details

For more information about the methods and the interpretation of the parameters, see the edgeR package and the corresponding publications.

Value

The function generates a .Rmd file containing the code for performing the differential expression analysis. This file can be executed using e.g. the knitr package.

Author(s)

Charlotte Soneson

References

Robinson MD, McCarthy DJ and Smyth GK (2010): edgeR: a Bioconductor package for differential expression analysis of digital gene expression data. Bioinformatics 26, 139-140

Examples

```
tmpdir <- normalizePath(tempdir(), winslash = "/")
mydata.obj <- generateSyntheticData(dataset = "mydata", n.vars = 1000,
                                    samples.per.cond = 5, n.diffexp = 100,
                                    output.file = file.path(tmpdir, "mydata.rds"))
runDiffExp(data.file = file.path(tmpdir, "mydata.rds"), result.extent = "edgeR.exact",
           Rmdfunction = "edgeR.exact.createRmd",
           output.directory = tmpdir, norm.method = "TMM",
           trend.method = "movingave", disp.type = "tagwise")
```

---

edgeR.GLM.createRmd *Generate a* .Rmd *file containing code to perform differential expression analysis with the edgeR GLM approach*

Description

A function to generate code that can be run to perform differential expression analysis of RNAseq data (comparing two conditions) using the GLM functionality from the edgeR package. The code is written to a .Rmd file. This function is generally not called by the user, the main interface for performing differential expression analysis is the runDiffExp function.

Usage

```
edgeR.GLM.createRmd(
  data.path,
  result.path,
  codefile,
  norm.method,
  disp.type,
  disp.method,
  trended
)
```

Arguments

data.path	The path to a .rds file containing the compData object that will be used for the differential expression analysis.
result.path	The path to the file where the result object will be saved.

codefile	The path to the file where the code will be written.
norm.method	The between-sample normalization method used to compensate for varying library sizes and composition in the differential expression analysis. Possible values are "TMM", "RLE", "upperquartile" and "none".
disp.type	The type of dispersion estimate used. Possible values are "common", "trended" and "tagwise".
disp.method	The method used to estimate the dispersion. Possible values are "CoxReid", "Pearson" and "deviance".
trended	Logical parameter indicating whether or not a trended dispersion estimate should be used.

Details

For more information about the methods and the interpretation of the parameters, see the edgeR package and the corresponding publications.

Value

The function generates a .Rmd file containing the code for performing the differential expression analysis. This file can be executed using e.g. the knitr package.

Author(s)

Charlotte Soneson

References

Robinson MD, McCarthy DJ and Smyth GK (2010): edgeR: a Bioconductor package for differential expression analysis of digital gene expression data. Bioinformatics 26, 139-140

Examples

```
tmpdir <- normalizePath(tempdir(), winslash = "/")
mydata.obj <- generateSyntheticData(dataset = "mydata", n.vars = 1000,
                                    samples.per.cond = 5, n.diffexp = 100,
                                    output.file = file.path(tmpdir, "mydata.rds"))
runDiffExp(data.file = file.path(tmpdir, "mydata.rds"), result.extent = "edgeR.GLM",
           Rmdfunction = "edgeR.GLM.createRmd",
           output.directory = tmpdir, norm.method = "TMM",
           disp.type = "tagwise", disp.method = "CoxReid",
           trended = TRUE)
```

`generateCodeHTMLs` *Generate HTML file(s) containing code used to run differential expression analysis.*

Description

A function to extract the code used to generate differential expression results from saved `compData` result objects (typically obtained by `runDiffExp`), and to write the code to HTML files. This requires that the code was saved as a character string in R markdown format in the `code` slot of the result object, which is done automatically by `runDiffExp`. If the differential expression analysis was performed with functions outside `compcodeR`, the code has to be added manually to the result object.

Usage

```
generateCodeHTMLs(input.files, output.directory)
```

Arguments

`input.files`	A vector with paths to one or several `.rds` files containing `compData` objects with the results from differential expression analysis. One code HTML file is generated for each file in the vector.
`output.directory`	The path to the directory where the code HTML files will be saved.

Author(s)

Charlotte Soneson

Examples

```
tmpdir <- normalizePath(tempdir(), winslash = "/")
mydata.obj <- generateSyntheticData(dataset = "mydata", n.vars = 1000,
                                    samples.per.cond = 5, n.diffexp = 100,
                                    output.file = file.path(tmpdir, "mydata.rds"))
runDiffExp(data.file = file.path(tmpdir, "mydata.rds"), result.extent = "voom.limma",
           Rmdfunction = "voom.limma.createRmd", output.directory = tmpdir,
           norm.method = "TMM")
generateCodeHTMLs(file.path(tmpdir, "mydata_voom.limma.rds"), tmpdir)
```

`generateSyntheticData` *Generate synthetic count data sets*

Description

Generate synthetic count data sets, following the simulation strategy detailed in Soneson and Delorenzi (2013).

Usage

```
generateSyntheticData(
  dataset,
  n.vars,
  samples.per.cond,
  n.diffexp,
  repl.id = 1,
  seqdepth = 1e+07,
  minfact = 0.7,
  maxfact = 1.4,
  relmeans = "auto",
  dispersions = "auto",
  fraction.upregulated = 1,
  between.group.diffdisp = FALSE,
  filter.threshold.total = 1,
  filter.threshold.mediancpm = 0,
  fraction.non.overdispersed = 0,
  random.outlier.high.prob = 0,
  random.outlier.low.prob = 0,
  single.outlier.high.prob = 0,
  single.outlier.low.prob = 0,
  effect.size = 1.5,
  output.file = NULL,
  tree = NULL,
  prop.var.tree = 1,
  model.process = c("BM", "OU"),
  selection.strength = 0,
  id.condition = NULL,
  id.species = as.factor(rep(1, 2 * samples.per.cond)),
  check.id.species = TRUE,
  lengths.relmeans = NULL,
  lengths.dispersions = NULL,
  lengths.phylo = TRUE
)
```

Arguments

`dataset`	A name or identifier for the data set/simulation settings.

`n.vars` The initial number of genes in the simulated data set. Based on the filtering conditions (`filter.threshold.total` and `filter.threshold.mediancpm`), the number of genes in the final data set may be lower than this number.

`samples.per.cond`
The number of samples in each of the two conditions.

`n.diffexp` The number of genes simulated to be differentially expressed between the two conditions.

`repl.id` A replicate ID for the specific simulation instance. Useful for example when generating multiple count matrices with the same simulation settings.

`seqdepth` The base sequencing depth (total number of mapped reads). This number is multiplied by a value drawn uniformly between `minfact` and `maxfact` for each sample to generate data with different actual sequencing depths.

`minfact`, `maxfact`
The minimum and maximum for the uniform distribution used to generate factors that are multiplied with `seqdepth` to generate individual sequencing depths for the simulated samples.

`relmeans` A vector of mean values to use in the simulation of data from the Negative Binomial distribution, or `"auto"`. Note that these values may be scaled in order to comply with the given sequencing depth. With the default value (`"auto"`), the mean values are sampled from values estimated from the Pickrell and Cheung data sets. If `relmeans` is a vector, the provided values will be used as mean values in the simulation for the samples in the first condition. The mean values for the samples in the second condition are generated by combining the `relmeans` and `effect.size` arguments.

`dispersions` A vector or matrix of dispersions to use in the simulation of data from the Negative Binomial distribution, or `"auto"`. With the default value (`"auto"`), the dispersion values are sampled from values estimated from the Pickrell and Cheung data sets. If both `relmeans` and `dispersions` are set to `"auto"`, the means and dispersion values are sampled in pairs from the values in these data sets. If `dispersions` is a single vector, the provided dispersions will be used for simulating data from both conditions. If it is a matrix with two columns, the values in the first column are used for condition 1, and the values in the second column are used for condition 2.

`fraction.upregulated`
The fraction of the differentially expressed genes that is upregulated in condition 2 compared to condition 1.

`between.group.diffdisp`
Whether or not the dispersion should be allowed to be different between the conditions. Only applicable if `dispersions` is `"auto"`.

`filter.threshold.total`
The filter threshold on the total count for a gene across all samples. All genes for which the total count across all samples is less than the threshold will be filtered out.

`filter.threshold.mediancpm`
The filter threshold on the median count per million (cpm) for a gene across all samples. All genes for which the median cpm across all samples is less than the threshold will be filtered out.

fraction.non.overdispersed
: The fraction of the genes that should be simulated according to a Poisson distribution, without overdispersion. The non-overdispersed genes will be divided proportionally between the upregulated, downregulated and non-differentially expressed genes.

random.outlier.high.prob
: The fraction of 'random' outliers with unusually high counts.

random.outlier.low.prob
: The fraction of 'random' outliers with unusually low counts.

single.outlier.high.prob
: The fraction of 'single' outliers with unusually high counts.

single.outlier.low.prob
: The fraction of 'single' outliers with unusually low counts.

effect.size
: The strength of the differential expression, i.e., the effect size, between the two conditions. If this is a single number, the effect sizes will be obtained by simulating numbers from an exponential distribution (with rate 1) and adding the results to the `effect.size`. For genes that are upregulated in the second condition, the mean in the first condition is multiplied by the effect size. For genes that are downregulated in the second condition, the mean in the first condition is divided by the effect size. It is also possible to provide a vector of effect sizes (one for each gene), which will be used as provided. In this case, the `fraction.upregulated` and `n.diffexp` arguments will be ignored and the values will be derived from the `effect.size` vector.

output.file
: If not `NULL`, the path to the file where the data object should be saved. The extension should be `.rds`, if not it will be changed.

tree
: a dated phylogenetic tree of class `phylo` with 'samples.per.cond * 2' species.

prop.var.tree
: the proportion of the common variance explained by the tree for each gene. It can be a scalar, in which case the same parameter is used for all genes. Otherwise it needs to be a vector with length `n.vars`. Default to 1.

model.process
: the process to be used for phylogenetic simulations. One of "BM" or "OU", default to "BM".

selection.strength
: if the process is "OU", the selection strength parameter.

id.condition
: A named vector, indicating which species is in each condition. Default to first 'samples.per.cond' species in condition '1' and others in condition '2'.

id.species
: A factor giving the species for each sample. If a tree is used, should be a named vector with names matching the taxa of the tree. Default to `rep(1, 2*samples.per.cond)`, i.e. all the samples come from the same species.

check.id.species
: Should the species vector be checked against the tree lengths (if provided) ? If TRUE, the function checks that all the samples that share a factor value in `id.species` that their distance on the tree is zero, i.e. that they are on the same tip of the tree. Default to TRUE.

lengths.relmeans
: An optional vector of mean values to use in the simulation of lengths from the Negative Binomial distribution. Should be of length n.vars. Default to `NULL`: the

lengths are not taken into account for the simulation. If set to "auto", the mean length values are sampled from values estimated from the Stern & Crandall (2018) data set.

lengths.dispersions
: An optional vector of dispersions to use in the simulation of data from the Negative Binomial distribution. Should be of length n.vars. Default to NULL: the lengths are not taken into account for the simulation. If set to "auto", the dispersion length values are sampled from values estimated from the Stern & Crandall (2018) data set.

lengths.phylo
: If TRUE, the lengths are simulated according to a phylogenetic Poisson Log-Normal model on the tree, with a BM process. If FALSE, they are simulated according to an iid negative binomial distribution. In both cases, lengths.relmeans and lengths.dispersions are used. Default to TRUE if a tree is provided.

Details

In the comparison function, only results obtained for data sets with the same value of the dataset parameter will be compared. Hence, it is important to give the same value of this parameter e.g. to different replicates generated with the same simulation settings.

For more detailed information regarding the different types of outliers, see Soneson and Delorenzi (2013).

Mean and dispersion parameters (if relmeans and/or dispersions is set to "auto") are sampled from values estimated from the data sets by Pickrell et al (2010) and Cheung et al (2010). The data sets were downloaded from the ReCount web page (Frazee et al (2011)) and processed as detailed by Soneson and Delorenzi (2013).

To get the actual mean value for the Negative Binomial distribution used for the simulation of counts for a given sample, take the column truemeans.S1 (or truemeans.S2, if the sample is in condition S2) of the variable.annotations slot, divide by the sum of the same column and multiply with the base sequencing depth (provided in the info.parameters list) and the depth factor for the sample (given in the sample.annotations data frame). Thus, if you have a vector of mean values that you want to provide as the relmeans argument and make sure to use it 'as-is' in the simulation (for condition S1), make sure to set the seqdepth argument to the sum of the values in the relmeans vector, and to set minfact and maxfact equal to 1.

When the tree argument is provided (not NULL), then the "phylogenetic Poisson log-Normal" model is used for the simulations, possibly with varying gene lengths across species (both lengths.relmeans and lengths.dispersions must be specified or set to "auto".) Phylogenetic simulations use the rTrait function from package phylolm.

Value

A compData object. If output.file is not NULL, the object is saved in the given output.file (which should have an .rds extension).

Author(s)

Charlotte Soneson

References

Soneson C and Delorenzi M (2013): A comparison of methods for differential expression analysis of RNA-seq data. BMC Bioinformatics 14:91

Cheung VG, Nayak RR, Wang IX, Elwyn S, Cousins SM, Morley M and Spielman RS (2010): Polymorphic cis- and trans-regulation of human gene expression. PLoS Biology 8(9):e1000480

Frazee AC, Langmead B and Leek JT (2011): ReCount: a multi-experiment resource of analysis-ready RNA-seq gene count datasets. BMC Bioinformatics 12:449

Pickrell JK, Marioni JC, Pai AA, Degner JF, Engelhardt BE, Nkadori E, Veyrieras JB, Stephens M, Gilad Y and Pritchard JK (2010): Understanding mechanisms underlying human gene expression variation with RNA sequencing. Nature 464, 768-772

Robles JA, Qureshi SE, Stephen SJ, Wilson SR, Burden CJ and Taylor JM (2012): Efficient experimental design and analysis strategies for the detection of differential expression using RNA-sequencing. BMC Genomics 13:484

Stern DB and Crandall KA (2018): The Evolution of Gene Expression Underlying Vision Loss in Cave Animals. Molecular Biology and Evolution. 35:2005–2014.

Examples

```
## RNA-Seq data
mydata.obj <- generateSyntheticData(dataset = "mydata", n.vars = 1000,
                                    samples.per.cond = 5, n.diffexp = 100)

## Inter-species RNA-Seq data
library(ape)
tree <- read.tree(text = "(((A1:0,A2:0,A3:0):1,B1:1):1,((C1:0,C2:0):1.5,(D1:0,D2:0):1.5):0.5);")
id.species <- factor(c("A", "A", "A", "B", "C", "C", "D", "D"))
names(id.species) <- tree$tip.label
mydata.obj <- generateSyntheticData(dataset = "mydata", n.vars = 1000,
                                    samples.per.cond = 4, n.diffexp = 100,
                                    tree = tree,
                                    id.species = id.species,
                                    lengths.relmeans = "auto",
                                    lengths.dispersions = "auto")
```

`lengthNorm.limma.createRmd`

Generate a `.Rmd` *file containing code to perform differential expression analysis with length normalized counts + limma*

Description

A function to generate code that can be run to perform differential expression analysis of RNAseq data (comparing two conditions) by applying a length normalizing transformation followed by differential expression analysis with limma. The code is written to a `.Rmd` file. This function is generally not called by the user, the main interface for performing differential expression analysis is the `runDiffExp` function.

Usage

```
lengthNorm.limma.createRmd(
  data.path,
  result.path,
  codefile,
  norm.method,
  extra.design.covariates = NULL,
  length.normalization = "RPKM",
  data.transformation = "log2",
  trend = FALSE,
  block.factor = NULL
)
```

Arguments

data.path	The path to a .rds file containing the phyloCompData object that will be used for the differential expression analysis.
result.path	The path to the file where the result object will be saved.
codefile	The path to the file where the code will be written.
norm.method	The between-sample normalization method used to compensate for varying library sizes and composition in the differential expression analysis. The normalization factors are calculated using the calcNormFactors of the edgeR package. Possible values are "TMM", "RLE", "upperquartile" and "none"
extra.design.covariates	A vector containing the names of extra control variables to be passed to the design matrix of limma. All the covariates need to be a column of the sample.annotations data frame from the phyloCompData object, with a matching column name. The covariates can be a numeric vector, or a factor. Note that "condition" factor column is always included, and should not be added here. See Details.
length.normalization	one of "none" (no length correction), "TPM", or "RPKM" (default). See details.
data.transformation	one of "log2", "asin(sqrt)" or "sqrt". Data transformation to apply to the normalized data.
trend	should an intensity-trend be allowed for the prior variance? Default to FALSE.
block.factor	Name of the factor specifying a blocking variable, to be passed to duplicateCorrelation function of the limma package. All the factors need to be a sample.annotations from the phyloCompData object. Default to null (no block structure).

Details

For more information about the methods and the interpretation of the parameters, see the limma package and the corresponding publications.

The length.matrix field of the phyloCompData object is used to normalize the counts, using one of the following formulas:

- `length.normalization="none"` : $CPM_{gi} = \frac{N_{gi}+0.5}{NF_i \times \sum_g N_{gi}+1} \times 10^6$
- `length.normalization="TPM"` : $TPM_{gi} = \frac{(N_{gi}+0.5)/L_{gi}}{NF_i \times \sum_g N_{gi}/L_{gi}+1} \times 10^6$
- `length.normalization="RPKM"` : $RPKM_{gi} = \frac{(N_{gi}+0.5)/L_{gi}}{NF_i \times \sum_g N_{gi}+1} \times 10^9$

where N_{gi} is the count for gene g and sample i, where L_{gi} is the length of gene g in sample i, and NF_i is the normalization for sample i, normalized using `calcNormFactors` of the `edgeR` package.

The function specified by the `data.transformation` is then applied to the normalized count matrix.

The "$+0.5$" and "$+1$" are taken from Law et al 2014, and dropped from the normalization when the transformation is something else than `log2`.

The "$\times 10^6$" and "$\times 10^9$" factors are omitted when the `asin(sqrt)` transformation is taken, as $asin$ can only be applied to real numbers smaller than 1.

The `design` model used in the `lmFit` uses the "condition" column of the `sample.annotations` data frame from the `phyloCompData` object as well as all the covariates named in `extra.design.covariates`. For example, if `extra.design.covariates = c("var1", "var2")`, then `sample.annotations` must have two columns named "var1" and "var2", and the design formula in the `lmFit` function will be: `~ condition + var1 + var2`.

Value

The function generates a `.Rmd` file containing the code for performing the differential expression analysis. This file can be executed using e.g. the `knitr` package.

Author(s)

Charlotte Soneson, Paul Bastide, Mélina Gallopin

References

Smyth GK (2005): Limma: linear models for microarray data. In: 'Bioinformatics and Computational Biology Solutions using R and Bioconductor'. R. Gentleman, V. Carey, S. Dudoit, R. Irizarry, W. Huber (eds), Springer, New York, pages 397-420

Smyth, G. K., Michaud, J., and Scott, H. (2005). The use of within-array replicate spots for assessing differential expression in microarray experiments. Bioinformatics 21(9), 2067-2075.

Law, C.W., Chen, Y., Shi, W. et al. (2014) voom: precision weights unlock linear model analysis tools for RNA-seq read counts. Genome Biol 15, R29.

Musser, JM, Wagner, GP. (2015): Character trees from transcriptome data: Origin and individuation of morphological characters and the so-called "species signal". J. Exp. Zool. (Mol. Dev. Evol.) 324B: 588– 604.

Examples

```
try(
if (require(limma)) {
tmpdir <- normalizePath(tempdir(), winslash = "/")
## Simulate data
```

```
mydata.obj <- generateSyntheticData(dataset = "mydata", n.vars = 1000,
                                    samples.per.cond = 5, n.diffexp = 100,
                                    id.species = factor(1:10),
                                    lengths.relmeans = rpois(1000, 1000),
                                    lengths.dispersions = rgamma(1000, 1, 1),
                                    output.file = file.path(tmpdir, "mydata.rds"))
## Add covariates
## Model fitted is count.matrix ~ condition + test_factor + test_reg
sample.annotations(mydata.obj)$test_factor <- factor(rep(1:2, each = 5))
sample.annotations(mydata.obj)$test_reg <- rnorm(10, 0, 1)
saveRDS(mydata.obj, file.path(tmpdir, "mydata.rds"))
## Diff Exp
runDiffExp(data.file = file.path(tmpdir, "mydata.rds"), result.extent = "length.limma",
           Rmdfunction = "lengthNorm.limma.createRmd",
           output.directory = tmpdir, norm.method = "TMM",
           extra.design.covariates = c("test_factor", "test_reg"))
})
```

`lengthNorm.sva.limma.createRmd` *Generate a* `.Rmd` *file containing code to perform differential expression analysis with length normalized counts + SVA + limma*

Description

A function to generate code that can be run to perform differential expression analysis of RNAseq data (comparing two conditions) by applying a length normalizing transformation, followed by a surrogate variable analysis (SVA), and then a differential expression analysis with limma. The code is written to a `.Rmd` file. This function is generally not called by the user, the main interface for performing differential expression analysis is the `runDiffExp` function.

Usage

```
lengthNorm.sva.limma.createRmd(
  data.path,
  result.path,
  codefile,
  norm.method,
  extra.design.covariates = NULL,
  length.normalization = "RPKM",
  data.transformation = "log2",
  trend = FALSE,
  n.sv = "auto"
)
```

Arguments

data.path	The path to a .rds file containing the phyloCompData object that will be used for the differential expression analysis.
result.path	The path to the file where the result object will be saved.
codefile	The path to the file where the code will be written.
norm.method	The between-sample normalization method used to compensate for varying library sizes and composition in the differential expression analysis. The normalization factors are calculated using the calcNormFactors of the edgeR package. Possible values are "TMM", "RLE", "upperquartile" and "none"
extra.design.covariates	A vector containing the names of extra control variables to be passed to the design matrix of limma. All the covariates need to be a column of the sample.annotations data frame from the phyloCompData object, with a matching column name. The covariates can be a numeric vector, or a factor. Note that "condition" factor column is always included, and should not be added here. See Details.
length.normalization	one of "none" (no length correction), "TPM", or "RPKM" (default). See details.
data.transformation	one of "log2", "asin(sqrt)" or "sqrt". Data transformation to apply to the normalized data.
trend	should an intensity-trend be allowed for the prior variance? Default to FALSE.
n.sv	The number of surrogate variables to estimate (see sva). Default to "auto": will be estimated with num.sv.

Details

For more information about the methods and the interpretation of the parameters, see the sva and limma packages and the corresponding publications.

See the details section of lengthNorm.limma.createRmd for details on the normalization and the extra design covariates.

Value

The function generates a .Rmd file containing the code for performing the differential expression analysis. This file can be executed using e.g. the knitr package.

Author(s)

Charlotte Soneson, Paul Bastide, Mélina Gallopin

References

Smyth GK (2005): Limma: linear models for microarray data. In: 'Bioinformatics and Computational Biology Solutions using R and Bioconductor'. R. Gentleman, V. Carey, S. Dudoit, R. Irizarry, W. Huber (eds), Springer, New York, pages 397-420

Smyth, G. K., Michaud, J., and Scott, H. (2005). The use of within-array replicate spots for assessing differential expression in microarray experiments. Bioinformatics 21(9), 2067-2075.

Law, C.W., Chen, Y., Shi, W. et al. (2014) voom: precision weights unlock linear model analysis tools for RNA-seq read counts. Genome Biol 15, R29.

Musser, JM, Wagner, GP. (2015): Character trees from transcriptome data: Origin and individuation of morphological characters and the so-called "species signal". J. Exp. Zool. (Mol. Dev. Evol.) 324B: 588– 604.

Leek JT, Johnson WE, Parker HS, Jaffe AE, and Storey JD. (2012) The sva package for removing batch effects and other unwanted variation in high-throughput experiments. Bioinformatics DOI:10.1093/bioinformatics/bts034

Examples

```
try(
if (require(limma) && require(sva)) {
tmpdir <- normalizePath(tempdir(), winslash = "/")
## Simulate data
mydata.obj <- generateSyntheticData(dataset = "mydata", n.vars = 1000,
                                    samples.per.cond = 5, n.diffexp = 100,
                                    id.species = factor(1:10),
                                    lengths.relmeans = rpois(1000, 1000),
                                    lengths.dispersions = rgamma(1000, 1, 1),
                                    output.file = file.path(tmpdir, "mydata.rds"))
## Add covariates
## Model fitted is count.matrix ~ condition + test_factor + test_reg
sample.annotations(mydata.obj)$test_factor <- factor(rep(1:2, each = 5))
sample.annotations(mydata.obj)$test_reg <- rnorm(10, 0, 1)
saveRDS(mydata.obj, file.path(tmpdir, "mydata.rds"))
## Diff Exp
runDiffExp(data.file = file.path(tmpdir, "mydata.rds"), result.extent = "lengthNorm.sva.limma",
           Rmdfunction = "lengthNorm.sva.limma.createRmd",
           output.directory = tmpdir, norm.method = "TMM",
           extra.design.covariates = c("test_factor", "test_reg"))
})
```

`listcreateRmd` *List available *.createRmd functions*

Description

Print a list of all `*.createRmd` functions that are available in the search path. These functions can be used together with the `runDiffExp` function to perform differential expression analysis. Consult the help pages for the respective functions for more information.

Usage

```
listcreateRmd()
```

Author(s)

Charlotte Soneson

Examples

```
listcreateRmd()
```

`logcpm.limma.createRmd` *Generate a* `.Rmd` *file containing code to perform differential expression analysis with limma after log-transforming the counts per million (cpm)*

Description

A function to generate code that can be run to perform differential expression analysis of RNAseq data (comparing two conditions) using limma, after preprocessing the counts by computing the counts per million (cpm) and applying a logarithmic transformation. The code is written to a `.Rmd` file. This function is generally not called by the user, the main interface for performing differential expression analysis is the `runDiffExp` function.

Usage

```
logcpm.limma.createRmd(data.path, result.path, codefile, norm.method)
```

Arguments

`data.path`	The path to a .rds file containing the `compData` object that will be used for the differential expression analysis.
`result.path`	The path to the file where the result object will be saved.
`codefile`	The path to the file where the code will be written.
`norm.method`	The between-sample normalization method used to compensate for varying library sizes and composition in the differential expression analysis. The normalization factors are calculated using the `calcNormFactors` function from the `edgeR` package. Possible values are `"TMM"`, `"RLE"`, `"upperquartile"` and `"none"`

Details

For more information about the methods and the interpretation of the parameters, see the `edgeR` and `limma` packages and the corresponding publications.

Value

The function generates a `.Rmd` file containing the code for performing the differential expression analysis. This file can be executed using e.g. the `knitr` package.

Author(s)

Charlotte Soneson

References

Smyth GK (2005): Limma: linear models for microarray data. In: 'Bioinformatics and Computational Biology Solutions using R and Bioconductor'. R. Gentleman, V. Carey, S. Dudoit, R. Irizarry, W. Huber (eds), Springer, New York, pages 397-420

Robinson MD, McCarthy DJ and Smyth GK (2010): edgeR: a Bioconductor package for differential expression analysis of digital gene expression data. Bioinformatics 26, 139-140

Robinson MD and Oshlack A (2010): A scaling normalization method for differential expression analysis of RNA-seq data. Genome Biology 11:R25

Examples

```
tmpdir <- normalizePath(tempdir(), winslash = "/")
mydata.obj <- generateSyntheticData(dataset = "mydata", n.vars = 1000,
                                    samples.per.cond = 5, n.diffexp = 100,
                                    output.file = file.path(tmpdir, "mydata.rds"))
runDiffExp(data.file = file.path(tmpdir, "mydata.rds"), result.extent = "logcpm.limma",
           Rmdfunction = "logcpm.limma.createRmd",
           output.directory = tmpdir, norm.method = "TMM")
```

NBPSeq.createRmd *Generate a* .Rmd *file containing code to perform differential expression analysis with NBPSeq*

Description

A function to generate code that can be run to perform differential expression analysis of RNAseq data (comparing two conditions) using NBPSeq. The code is written to a .Rmd file. This function is generally not called by the user, the main interface for performing differential expression analysis is the runDiffExp function.

Usage

```
NBPSeq.createRmd(data.path, result.path, codefile, norm.method, disp.method)
```

Arguments

data.path	The path to a .rds file containing the compData object that will be used for the differential expression analysis.
result.path	The path to the file where the result object will be saved.
codefile	The path to the file where the code will be written.

`norm.method` The between-sample normalization method used to compensate for varying library sizes and composition in the differential expression analysis. The normalization factors are calculated using the `calcNormFactors` function from the `edgeR` package. Possible values are `"TMM"`, `"RLE"`, `"upperquartile"` and `"none"`.

`disp.method` The method to use to estimate the dispersion values. Possible values are `"NBP"` and `"NB2"`.

Details

For more information about the methods and the interpretation of the parameters, see the `NBPSeq` and `edgeR` packages and the corresponding publications.

Value

The function generates a `.Rmd` file containing the code for performing the differential expression analysis. This file can be executed using e.g. the `knitr` package.

Author(s)

Charlotte Soneson

References

Robinson MD, McCarthy DJ and Smyth GK (2010): edgeR: a Bioconductor package for differential expression analysis of digital gene expression data. Bioinformatics 26, 139-140

Robinson MD and Oshlack A (2010): A scaling normalization method for differential expression analysis of RNA-seq data. Genome Biology 11:R25

Di Y, Schafer DW, Cumbie JS, and Chang JH (2011): The NBP Negative Binomial Model for Assessing Differential Gene Expression from RNA-Seq. Statistical Applications in Genetics and Molecular Biology 10(1), 1-28

Examples

```
try(
if (require(NBPSeq)) {
tmpdir <- normalizePath(tempdir(), winslash = "/")
mydata.obj <- generateSyntheticData(dataset = "mydata", n.vars = 1000,
                                    samples.per.cond = 5, n.diffexp = 100,
                                    output.file = file.path(tmpdir, "mydata.rds"))
runDiffExp(data.file = file.path(tmpdir, "mydata.rds"), result.extent = "NBPSeq",
           Rmdfunction = "NBPSeq.createRmd",
           output.directory = tmpdir, norm.method = "TMM", disp.method = "NBP")
})
```

NOISeq.prenorm.createRmd *Generate a* .Rmd *file containing code to perform differential expression analysis with NOISeq*

Description

A function to generate code that can be run to perform differential expression analysis of RNAseq data (comparing two conditions) using NOISeq. The code is written to a .Rmd file. This function is generally not called by the user, the main interface for performing differential expression analysis is the runDiffExp function.

Usage

```
NOISeq.prenorm.createRmd(data.path, result.path, codefile, norm.method)
```

Arguments

data.path	The path to a .rds file containing the compData object that will be used for the differential expression analysis.
result.path	The path to the file where the result object will be saved.
codefile	The path to the file where the code will be written.
norm.method	The between-sample normalization method used to compensate for varying library sizes and composition in the differential expression analysis. The normalization factors are calculated using the calcNormFactors function from the edgeR package. Possible values are "TMM", "RLE", "upperquartile" and "none".

Details

For more information about the methods and the interpretation of the parameters, see the NOISeq package and the corresponding publications.

Value

The function generates a .Rmd file containing the code for performing the differential expression analysis. This file can be executed using e.g. the knitr package.

Author(s)

Charlotte Soneson

References

Robinson MD, McCarthy DJ and Smyth GK (2010): edgeR: a Bioconductor package for differential expression analysis of digital gene expression data. Bioinformatics 26, 139-140

Robinson MD and Oshlack A (2010): A scaling normalization method for differential expression analysis of RNA-seq data. Genome Biology 11:R25

Tarazona S, Furio-Tari P, Ferrer A and Conesa A (2012): NOISeq: Exploratory analysis and differential expression for RNA-seq data. R package

Tarazona S, Garcia-Alcalde F, Dopazo J, Ferrer A and Conesa A (2011): Differential expression in RNA-seq: a matter of depth. Genome Res 21(12), 2213-2223

Examples

```
try(
if (require(NOISeq)) {
tmpdir <- normalizePath(tempdir(), winslash = "/")
mydata.obj <- generateSyntheticData(dataset = "mydata", n.vars = 1000,
                                    samples.per.cond = 5, n.diffexp = 100,
                                    output.file = file.path(tmpdir, "mydata.rds"))
runDiffExp(data.file = file.path(tmpdir, "mydata.rds"), result.extent = "NOISeq",
           Rmdfunction = "NOISeq.prenorm.createRmd",
           output.directory = tmpdir, norm.method = "TMM")
})
```

---

phyloCompData *Create a* phyloCompData *object*

Description

The phyloCompData class extends the compData class with sequence length and phylogeny related information.

Usage

```
phyloCompData(
  count.matrix,
  sample.annotations,
  info.parameters,
  variable.annotations = data.frame(),
  filtering = "no info",
  analysis.date = "",
  package.version = "",
  method.names = list(),
  code = "",
  result.table = data.frame(),
  tree = list(),
  length.matrix = matrix(NA_integer_, 0, 0)
)
```

Arguments

`count.matrix` A count matrix, with genes as rows and observations as columns.

`sample.annotations`
A data frame, containing at least one column named 'condition', encoding the grouping of the observations into two groups, and one column named `id.species` of factors giving the species for each sample if the tree is specified. The row names should be the same as the column names of count.matrix. `Class data.frame`.

`info.parameters`
A list containing information regarding simulation parameters etc. The only mandatory entries are `dataset` and `uID`, but it may contain entries such as the ones listed below (see `generateSyntheticData` for more detailed information about each of these entries).

- `dataset`: an informative name or identifier of the data set (e.g., summarizing the simulation settings).
- `samples.per.cond`
- `n.diffexp`
- `repl.id`
- `seqdepth`
- `minfact`
- `maxfact`
- `fraction.upregulated`
- `between.group.diffdisp`
- `filter.threshold.total`
- `filter.threshold.mediancpm`
- `fraction.non.overdispersed`
- `random.outlier.high.prob`
- `random.outlier.low.prob`
- `single.outlier.high.prob`
- `single.outlier.low.prob`
- `effect.size`
- `uID`: a unique ID for the data set. In contrast to `dataset`, the `uID` is unique e.g. for each instance of replicated data sets generated with the same simulation settings.

`variable.annotations`
A data frame with variable annotations (with number of rows equal to the number of rows in `count.matrix`, that is, the number of variables in the data set). Not mandatory, but may contain columns such as the ones listed below. If present, the row names should be the same as the row names of the `count.matrix`.

- `truedispersions.S1`: the true dispersion for each gene in condition S1.
- `truedispersions.S2`: the true dispersion for each gene in condition S2.
- `truemeans.S1`: the true mean value for each gene in condition S1.
- `truemeans.S2`: the true mean value for each gene in condition S2.
- `n.random.outliers.up.S1`: the number of 'random' outliers with extremely high counts for each gene in condition S1.

- n.random.outliers.up.S2: the number of 'random' outliers with extremely high counts for each gene in condition S2.
- n.random.outliers.down.S1: the number of 'random' outliers with extremely low counts for each gene in condition S1.
- n.random.outliers.down.S2: the number of 'random' outliers with extremely low counts for each gene in condition S2.
- n.single.outliers.up.S1: the number of 'single' outliers with extremely high counts for each gene in condition S1.
- n.single.outliers.up.S2: the number of 'single' outliers with extremely high counts for each gene in condition S2.
- n.single.outliers.down.S1: the number of 'single' outliers with extremely low counts for each gene in condition S1.
- n.single.outliers.down.S2: the number of 'single' outliers with extremely low counts for each gene in condition S2.
- M.value: the M-value (observed log2 fold change between condition S1 and condition S2) for each gene.
- A.value: the A-value (observed average expression level across condition S1 and condition S2) for each gene.
- truelog2foldchanges: the true (simulated) log2 fold changes between condition S1 and condition S2.
- upregulation: a binary vector indicating which genes are simulated to be upregulated in condition S2 compared to condition S1.
- downregulation: a binary vector indicating which genes are simulated to be downregulated in condition S2 compared to condition S1.
- differential.expression: a binary vector indicating which genes are simulated to be differentially expressed in condition S2 compared to condition S1.

filtering — A character string containing information about the filtering that has been applied to the data set.

analysis.date — If a differential expression analysis has been performed, a character string detailing when it was performed.

package.version — If a differential expression analysis has been performed, a character string giving the version of the differential expression packages that were applied.

method.names — If a differential expression analysis has been performed, a list with entries full.name and short.name, giving the full name of the differential expression method (may including version number and parameter settings) and a short name or abbreviation.

code — If a differential expression analysis has been performed, a character string containing the code that was run to perform the analysis. The code should be in R markdown format, and can be written to an HTML file using the generateCodeHTMLs function.

result.table — If a differential expression analysis has been performed, a data frame containing the results of the analysis. The number of rows should be equal to the number of rows in count.matrix and if present, the row names should be identical. The

only mandatory column is score, which gives a score for each gene, where a higher score suggests a "more highly differentially expressed" gene. Different comparison functions use different columns of this table, if available. The list below gives the columns that are used by the interfaced methods.

- pvalue nominal p-values
- adjpvalue p-values adjusted for multiple comparisons
- logFC estimated log-fold changes between the two conditions
- score the score that will be used to rank the genes in order of significance. Note that high scores always signify differential expression, that is, a strong association with the predictor. For example, for methods returning a nominal p-value the score can be defined as 1 - pvalue.
- FDR false discovery rate estimates
- posterior.DE posterior probabilities of differential expression
- prob.DE conditional probabilities of differential expression
- lfdr local false discovery rates
- statistic test statistics from the differential expression analysis
- dispersion.S1 dispersion estimates in condition S1
- dispersion.S2 dispersion estimates in condition S2

tree — The phylogenetic tree describing the relationships between samples. The taxa names of the tree should be the same as the column names of the count.matrix.

length.matrix — The length matrix, with genes as rows and samples as columns. The column names of the length.matrix should be the same as the column names of the count.matrix.

Value

A phyloCompData object.

Author(s)

Charlotte Soneson, Paul Bastide

Examples

```
tree <- ape::read.tree(
  text = "(((A1:0,A2:0,A3:0):1,B1:1):1,((C1:0,C2:0):1.5,(D1:0,D2:0):1.5):0.5);"
  )
count.matrix <- round(matrix(1000*runif(8000), 1000))
sample.annotations <- data.frame(condition = c(1, 1, 1, 1, 2, 2, 2, 2),
                                 id.species = c("A", "A", "A", "B", "C", "C", "D", "D"))
info.parameters <- list(dataset = "mydata", uID = "123456")
length.matrix <- round(matrix(1000*runif(8000), 1000))
colnames(count.matrix) <- colnames(length.matrix) <- rownames(sample.annotations) <- tree$tip.label
cpd <- phyloCompData(count.matrix, sample.annotations, info.parameters,
                     tree = tree, length.matrix = length.matrix)
```

phyloCompData-class *Class phyloCompData*

Description

The phyloCompData class extends the compData class with sequence length and phylogeny related information.

Slots

tree: The phylogenetic tree describing the relationships between samples. The taxa names of the tree should be the same as the column names of the count.matrix. Class phylo.

length.matrix: The length matrix, with genes as rows and samples as columns. The column names of the length.matrix should be the same as the column names of the count.matrix. Class matrix.

sample.annotations: In addition to the columns described in the compData class, if the tree is specified, it should contain an extra column named id.species of factors giving the species for each sample. The row names should be the same as the column names of count.matrix. Class data.frame.

Methods

phylo.tree signature(x="phyloCompData")

phylo.tree<- signature(x="phyloCompData",value="phylo"): Get or set the tree in a phyloCompData object. value should be a phylo object.

length.matrix signature(x="phyloCompData")

length.matrix<- signature(x="phyloCompData",value="matrix"): Get or set the length matrix in a phyloCompData object. value should be a numeric matrix.

Construction

An object of the class phyloCompData can be constructed using the phyloCompData function.

Author(s)

Charlotte Soneson, Paul Bastide

phylolm.createRmd *Generate a* .Rmd *file containing code to perform differential expression analysis with* phylolm.

Description

A function to generate code that can be run to perform differential expression analysis of RNAseq data (comparing two conditions) using the phylolm package. The code is written to a .Rmd file. This function is generally not called by the user, the main interface for performing differential expression analysis is the runDiffExp function.

Usage

```
phylolm.createRmd(
  data.path,
  result.path,
  codefile,
  norm.method,
  model = "BM",
  measurement_error = TRUE,
  extra.design.covariates = NULL,
  length.normalization = "RPKM",
  data.transformation = "log2",
  ...
)
```

Arguments

data.path	The path to a .rds file containing the phyloCompData object that will be used for the differential expression analysis.
result.path	The path to the file where the result object will be saved.
codefile	The path to the file where the code will be written.
norm.method	The between-sample normalization method used to compensate for varying library sizes and composition in the differential expression analysis. The normalization factors are calculated using the calcNormFactors of the edgeR package. Possible values are "TMM", "RLE", "upperquartile" and "none"
model	The model for trait evolution on the tree. Default to "BM".
measurement_error	A logical value indicating whether there is measurement error. Default to TRUE.
extra.design.covariates	A vector containing the names of extra control variables to be passed to the design matrix of phyolm. All the covariates need to be a column of the sample.annotations data frame from the phyloCompData object, with a matching column name. The covariates can be a numeric vector, or a factor. Note that "condition" factor column is always included, and should not be added here. See Details.

`length.normalization`
: one of "none" (no correction), "TPM" or "RPKM" (default). See details.

`data.transformation`
: one of "log2", "asin(sqrt)" or "sqrt". Data transformation to apply to the normalized data.

`...`
: Further arguments to be passed to function `phylolm`.

Details

For more information about the methods and the interpretation of the parameters, see the `phylolm` package and the corresponding publications.

The `length.matrix` field of the `phyloCompData` object is used to normalize the counts, using one of the following formulas: * `length.normalization="none"` : $CPM_{gi} = \frac{N_{gi}+0.5}{NF_i \times \sum_g N_{gi}+1} \times 10^6$ * `length.normalization="TPM"` : $TPM_{gi} = \frac{(N_{gi}+0.5)/L_{gi}}{NF_i \times \sum_g N_{gi}/L_{gi}+1} \times 10^6$ * `length.normalization="RPKM"` : $RPKM_{gi} = \frac{(N_{gi}+0.5)/L_{gi}}{NF_i \times \sum_g N_{gi}+1} \times 10^9$

where N_{gi} is the count for gene g and sample i, where L_{gi} is the length of gene g in sample i, and NF_i is the normalization for sample i, normalized using `calcNormFactors` of the `edgeR` package.

The function specified by the `data.transformation` is then applied to the normalized count matrix.

The "$+0.5$" and "$+1$" are taken from Law et al 2014, and dropped from the normalization when the transformation is something else than `log2`.

The "$\times 10^6$" and "$\times 10^9$" factors are omitted when the `asin(sqrt)` transformation is taken, as $asin$ can only be applied to real numbers smaller than 1.

The `design` model used in the `phylolm` uses the "condition" column of the `sample.annotations` data frame from the `phyloCompData` object as well as all the covariates named in `extra.design.covariates`. For example, if `extra.design.covariates = c("var1", "var2")`, then `sample.annotations` must have two columns named "var1" and "var2", and the design formula in the `phylolm` function will be: `~ condition + var1 + var2`.

Value

The function generates a `.Rmd` file containing the code for performing the differential expression analysis. This file can be executed using e.g. the `knitr` package.

Author(s)

Charlotte Soneson, Paul Bastide, Mélina Gallopin

References

Ho, L. S. T. and Ane, C. 2014. "A linear-time algorithm for Gaussian and non-Gaussian trait evolution models". Systematic Biology 63(3):397-408.

Law, C.W., Chen, Y., Shi, W. et al. (2014) voom: precision weights unlock linear model analysis tools for RNA-seq read counts. Genome Biol 15, R29.

Musser, JM, Wagner, GP. (2015): Character trees from transcriptome data: Origin and individuation of morphological characters and the so-called "species signal". J. Exp. Zool. (Mol. Dev. Evol.) 324B: 588– 604.

Examples

```
try(
if (require(ape) && require(phylolm)) {
tmpdir <- normalizePath(tempdir(), winslash = "/")
set.seed(20200317)
tree <- rphylo(10, 0.1, 0)
mydata.obj <- generateSyntheticData(dataset = "mydata", n.vars = 1000,
                                    samples.per.cond = 5, n.diffexp = 100,
                                    tree = tree,
                                    id.species = 1:10,
                                    lengths.relmeans = rpois(1000, 1000),
                                    lengths.dispersions = rgamma(1000, 1, 1),
                                    output.file = file.path(tmpdir, "mydata.rds"))
## Add covariates
## Model fitted is count.matrix ~ condition + test_factor + test_reg
sample.annotations(mydata.obj)$test_factor <- factor(rep(1:2, each = 5))
sample.annotations(mydata.obj)$test_reg <- rnorm(10, 0, 1)
saveRDS(mydata.obj, file.path(tmpdir, "mydata.rds"))
## Diff Exp
runDiffExp(data.file = file.path(tmpdir, "mydata.rds"), result.extent = "DESeq2",
           Rmdfunction = "phylolm.createRmd",
           output.directory = tmpdir,
           norm.method = "TMM",
           extra.design.covariates = c("test_factor", "test_reg"),
           length.normalization = "RPKM")
})
```

`runComparison` *Run the performance comparison between differential expression methods.*

Description

The main function for performing comparisons among differential expression methods and generating a report in HTML format. It is assumed that all differential expression results have been generated in advance (using e.g. the function `runDiffExp`) and that the result `compData` object for each data set and each differential expression method is saved separately in files with the extension `.rds`. Note that the function can also be called via the `runComparisonGUI` function, which lets the user set parameters and select input files using a graphical user interface.

Usage

```
runComparison(
  file.table,
```

```
  parameters,
  output.directory,
  check.table = TRUE,
  out.width = NULL,
  save.result.table = FALSE,
  knit.results = TRUE
)
```

Arguments

file.table	A data frame with at least a column input.files, potentially also columns named datasets, nbr.samples, repl and de.methods.
parameters	A list containing parameters for the comparison study. The following entries are supported, and used by different comparison methods:

- incl.nbr.samples An array with sample sizes (number of samples per condition) to consider in the comparison. If set to NULL, all sample sizes will be included.
- incl.dataset A dataset name (corresponding to the dataset slot of the results or data objects), indicating the dataset that will be used for the comparison. Only one dataset can be chosen.
- incl.replicates An array with replicate numbers to consider in the comparison. If set to NULL, all replicates will be included.
- incl.de.methods An array with differential expression methods to be compared. If set to NULL, all differential expression methods will be included.
- fdr.threshold The adjusted p-value threshold for FDR calculations. Default 0.05.
- tpr.threshold The adjusted p-value threshold for TPR calculations. Default 0.05.
- mcc.threshold The adjusted p-value threshold for MCC calculations. Default 0.05.
- typeI.threshold The nominal p-value threshold for type I error calculations. Default 0.05.
- fdc.maxvar The maximal number of variables to include in false discovery curve plots. Default 1500.
- overlap.threshold The adjusted p-value for overlap analysis. Default 0.05.
- fracsign.threshold The adjusted p-value for calculation of the fraction/number of genes called significant. Default 0.05.
- nbrtpfp.threshold The adjusted p-value for calculation of the number of TP, FP, TN, FN genes. Default 0.05.
- ma.threshold The adjusted p-value threshold for coloring genes in MA plots. Default 0.05.
- signal.measure Either 'mean' or 'snr', determining how to define the signal strength for a gene which is expressed in only one condition.
- upper.limits,lower.limits Lists that can be used to manually set the upper and lower plot limits for boxplots of fdr, tpr, auc, mcc, fracsign, nbrtpfp and typeIerror.

- comparisons Array containing the comparison methods to be applied. The entries must be chosen among the following abbreviations:
  - "auc" - Compute the area under the ROC curve
  - "mcc" - Compute Matthew's correlation coefficient
  - "tpr" - Compute the true positive rate at a given adjusted p-value threshold (tpr.threshold)
  - "fdr" - Compute the false discovery rate at a given adjusted p-value threshold (fdr.threshold)
  - "fdrvsexpr" - Compute the false discovery rate as a function of the expression level.
  - "typeIerror" - Compute the type I error rate at a given nominal p-value threshold (typeI.threshold)
  - "fracsign" - Compute the fraction of genes called significant at a given adjusted p-value threshold (fracsign.threshold).
  - "nbrsign" - Compute the number of genes called significant at a given adjusted p-value threshold (fracsign.threshold).
  - "nbrtpfp" - Compute the number of true positives, false positives, true negatives and false negatives at a given adjusted p-value threshold (nbrtpfp.threshold).
  - "maplot" - Construct MA plots, depicting the average expression level and the log fold change for the genes and indicating the genes called differential expressed at a given adjusted p-value threshold (ma.threshold).
  - "fdcurvesall" - Construct false discovery curves for each of the included replicates.
  - "fdcurvesone" - Construct false discovery curves for a single replicate only
  - "rocall" - Construct ROC curves for each of the included replicates
  - "rocone" - Construct ROC curves for a single replicate only
  - "overlap" - Compute the overlap between collections of genes called differentially expressed by the different methods at a given adjusted p-value threshold (overlap.threshold)
  - "sorensen" - Compute the Sorensen index, quantifying the overlap between collections of genes called differentially expressed by the different methods, at a given adjusted p-value threshold (overlap.threshold)
  - "correlation" - Compute the Spearman correlation between gene scores assigned by different methods
  - "scorevsoutlier" - Visualize the distribution of the gene scores as a function of the number of outlier counts introduced for the genes
  - "scorevsexpr" - Visualize the gene scores as a function of the average expression level of the genes
  - "scorevssignal" - Visualize the gene score as a function of the 'signal strength' (see the signal.measure parameter above) for genes that are expressed in only one condition

output.directory

The directory where the results should be written. The subdirectory structure will be created automatically. If the directory already exists, it will be overwritten.

`check.table` Logical, should the input table be checked for consistency. Default `TRUE`.

`out.width` The width of the figures in the final report. Will be passed on to `knitr` when the HTML is generated.

`save.result.table` Logical, should the intermediate result table be saved for future use ? Default to `FALSE`.

`knit.results` Logical, should the Rmd be generated and knitted ? Default to `TRUE`. If `FALSE`, no comparison report is generated, and only the intermediate result table is saved (if `save.result.table=TRUE`).

Details

The input to `runComparison` is a data frame with at least a column named `input.files`, containing paths to `.rds` files containing result objects (of the class `compData`), such as those generated by `runDiffExp`. Other columns that can be included in the data frame are `datasets`, `nbr.samples`, `repl` and `de.methods`. They have to match the information contained in the corresponding result objects. If these columns are not present, they will be added to the data frame automatically.

Value

If `knit.results=TRUE`, the function will create a comparison report, named **compcodeR_report<timestamp>.html**, in the `output.directory`. It will also create subfolders named `compcodeR_code` and `compcodeR_figure`, where the code used to perform the differential expression analysis and the figures contained in the report, respectively, will be stored. Note that if these directories already exists, they will be overwritten. If `save.result.table=TRUE`, the function will also create a comparison report, named **compcodeR_result_table_<timestamp>.rds** in the `output.directory`, containing the result table.

Author(s)

Charlotte Soneson

Examples

```
tmpdir <- normalizePath(tempdir(), winslash = "/")
mydata.obj <- generateSyntheticData(dataset = "mydata", n.vars = 1000,
                                    samples.per.cond = 5, n.diffexp = 100,
                                    output.file = file.path(tmpdir, "mydata.rds"))
runDiffExp(data.file = file.path(tmpdir, "mydata.rds"), result.extent = "voom.limma",
           Rmdfunction = "voom.limma.createRmd", output.directory = tmpdir,
           norm.method = "TMM")
runDiffExp(data.file = file.path(tmpdir, "mydata.rds"), result.extent = "edgeR.exact",
           Rmdfunction = "edgeR.exact.createRmd", output.directory = tmpdir,
           norm.method = "TMM",
           trend.method = "movingave", disp.type = "tagwise")
file.table <- data.frame(input.files = file.path(tmpdir,
                         c("mydata_voom.limma.rds", "mydata_edgeR.exact.rds")),
                         stringsAsFactors = FALSE)
parameters <- list(incl.nbr.samples = 5, incl.replicates = 1, incl.dataset = "mydata",
                   incl.de.methods = NULL,
```

```
                    fdr.threshold = 0.05, tpr.threshold = 0.05, typeI.threshold = 0.05,
                    ma.threshold = 0.05, fdc.maxvar = 1500, overlap.threshold = 0.05,
                    fracsign.threshold = 0.05, mcc.threshold = 0.05,
                    nbrtpfp.threshold = 0.05,
                    comparisons = c("auc", "fdr", "tpr", "ma", "correlation"))
if (interactive()) {
 runComparison(file.table = file.table, parameters = parameters, output.directory = tmpdir)
}
```

`runComparisonGUI` *A GUI to the main function for running the performance comparison between differential expression methods.*

Description

This function provides a GUI to the main function for performing comparisons among differential expression methods and generating a report in HTML format (`runComparison`). It is assumed that all differential expression results have been generated in advance (using e.g. the function `runDiffExp`) and that the result `compData` object for each data set and each differential expression method is saved separately in files with the extension `.rds`. The function opens a graphical user interface where the user can set parameter values and choose the files to be used as the basis of the comparison. It is, however, possible to circumvent the GUI and call the comparison function `runComparison` directly.

Usage

```
runComparisonGUI(
  input.directories,
  output.directory,
  recursive,
  out.width = NULL,
  upper.limits = NULL,
  lower.limits = NULL
)
```

Arguments

`input.directories`
: A list of directories containing the result files (`*.rds`). All results in the provided directories will be available for inclusion in the comparison, and the selection is performed through a graphical user interface. All result objects saved in the files should be of the `compData` class, although list objects created by earlier versions of `compcodeR` are supported.

`output.directory`
: The directory where the results should be written. The subdirectory structure will be created automatically. If the directory already exists, it will be overwritten.

`recursive` A logical parameter indicating whether or not the search should be extended recursively to subfolders of the `input.directories`.

`out.width` The width of the figures in the final report. Will be passed on to `knitr` when the HTML is generated. Can be for example "800px" (see `knitr` documentation for more information)

`upper.limits, lower.limits`
Lists that can be used to manually set upper and lower limits for boxplots of fdr, tpr, auc, mcc, fracsign, nbrtpfp, nbrsign and typeIerror.

Details

This function requires that the `rpanel` package is installed. If this package can not be installed, please use the `runComparison` function directly.

Value

The function will create a comparison report, named **compcodeR_report<timestamp>.html**, in the `output.directory`. It will also create subfolders named `compcodeR_code` and `compcodeR_figure`, where the code used to perform the differential expression analysis and the figures contained in the report, respectively, will be saved. Note that if these directories already exist they will be overwritten.

Author(s)

Charlotte Soneson

Examples

```
if (interactive()) {
mydata.obj <- generateSyntheticData(dataset = "mydata", n.vars = 12500,
                                    samples.per.cond = 5, n.diffexp = 1250,
                                    output.file = "mydata.rds")
runDiffExp(data.file = "mydata.rds", result.extent = "voom.limma",
           Rmdfunction = "voom.limma.createRmd", output.directory = ".",
           norm.method = "TMM")
runDiffExp(data.file = "mydata.rds", result.extent = "ttest",
           Rmdfunction = "ttest.createRmd", output.directory = ".",
           norm.method = "TMM")
runComparisonGUI(input.directories = ".", output.directory = ".", recursive = FALSE)
}
```

`runComparisonShiny` *A shiny-based GUI to the main function for running the performance comparison between differential expression methods.*

Description

This function provides a GUI to the main function for performing comparisons among differential expression methods and generating a report in HTML format (runComparison). It is assumed that all differential expression results have been generated in advance (using e.g. the function runDiffExp) and that the result compData object for each data set and each differential expression method is saved separately in files with the extension .rds. The function opens a graphical user interface where the user can set parameter values and choose the files to be used as the basis of the comparison. It is, however, possible to circumvent the GUI and call the comparison function runComparison directly.

Usage

```
runComparisonShiny(
  input.directories,
  output.directory,
  recursive,
  out.width = NULL,
  upper.limits = NULL,
  lower.limits = NULL
)
```

Arguments

input.directories
: A list of directories containing the result files (*.rds). All results in the provided directories will be available for inclusion in the comparison, and the selection is performed through a graphical user interface. All result objects saved in the files should be of the compData class, although list objects created by earlier versions of compcodeR are supported.

output.directory
: The directory where the results should be written. The subdirectory structure will be created automatically. If the directory already exists, it will be overwritten.

recursive
: A logical parameter indicating whether or not the search should be extended recursively to subfolders of the input.directories.

out.width
: The width of the figures in the final report. Will be passed on to knitr when the HTML is generated. Can be for example "800px" (see knitr documentation for more information).

upper.limits, lower.limits
: Lists that can be used to manually set upper and lower limits for boxplots of fdr, tpr, auc, mcc, fracsign, nbrtpfp, nbrsign and typeIerror.

Value

The function will create a comparison report, named **compcodeR_report<timestamp>.html**, in the output.directory. It will also create subfolders named compcodeR_code and compcodeR_figure, where the code used to perform the differential expression analysis and the figures contained in the

report, respectively, will be saved. Note that if these directories already exist they will be overwritten.

Author(s)

Charlotte Soneson

Examples

```
if (interactive()) {
mydata.obj <- generateSyntheticData(dataset = "mydata", n.vars = 12500,
                                    samples.per.cond = 5, n.diffexp = 1250,
                                    output.file = "mydata.rds")
runDiffExp(data.file = "mydata.rds", result.extent = "voom.limma",
           Rmdfunction = "voom.limma.createRmd", output.directory = ".",
           norm.method = "TMM")
runDiffExp(data.file = "mydata.rds", result.extent = "ttest",
           Rmdfunction = "ttest.createRmd", output.directory = ".",
           norm.method = "TMM")
runComparisonShiny(input.directories = ".", output.directory = ".", recursive = FALSE)
}
```

---

runDiffExp *The main function to run differential expression analysis*

Description

The main function for running differential expression analysis (comparing two conditions), using one of the methods interfaced through compcodeR or a user-defined method. Note that the interface functions are provided for convenience and as templates for other, user-defined workflows, and there is no guarantee that the included differential expression code is kept up-to-date with the latest recommendations and best practices for running each of the interfaced methods, or that the chosen settings are suitable in all situations. The user should make sure that the analysis is performed in the way they intend, and check the code that was run, using e.g. the generateCodeHTMLs() function.

Usage

```
runDiffExp(
  data.file,
  result.extent,
  Rmdfunction,
  output.directory = ".",
  norm.path = TRUE,
  ...
)
```

Arguments

data.file	The path to a .rds file containing the data on which the differential expression analysis will be performed, for example a compData object returned from generateSyntheticData.
result.extent	The extension that will be added to the data file name in order to construct the result file name. This can be for example the differential expression method together with a version number.
Rmdfunction	A function that creates an Rmd file containing the code that should be run to perform the differential expression analysis. All functions available through compcodeR can be listed using the listcreateRmd function.
output.directory	The directory in which the result object will be saved.
norm.path	Logical, whether to include the full (absolute) path to the output object in the saved code.
...	Additional arguments that will be passed to the Rmdfunction, such as parameter choices for the differential expression method.

Author(s)

Charlotte Soneson

Examples

```
tmpdir <- normalizePath(tempdir(), winslash = "/")
mydata.obj <- generateSyntheticData(dataset = "mydata", n.vars = 1000,
                                    samples.per.cond = 5, n.diffexp = 100,
                                    output.file = file.path(tmpdir, "mydata.rds"))
listcreateRmd()
runDiffExp(data.file = file.path(tmpdir, "mydata.rds"), result.extent = "voom.limma",
           Rmdfunction = "voom.limma.createRmd",
           output.directory = tmpdir, norm.method = "TMM")

if (interactive()) {
## The following list covers the currently available
## differential expression methods:
runDiffExp(data.file = "mydata.rds", result.extent = "DESeq2",
           Rmdfunction = "DESeq2.createRmd",
           output.directory = ".", fit.type = "parametric",
           test = "Wald", beta.prior = TRUE,
           independent.filtering = TRUE, cooks.cutoff = TRUE,
           impute.outliers = TRUE)
runDiffExp(data.file = "mydata.rds", result.extent = "DSS",
           Rmdfunction = "DSS.createRmd",
           output.directory = ".", norm.method = "quantile",
           disp.trend = TRUE)
runDiffExp(data.file = "mydata.rds", result.extent = "EBSeq",
           Rmdfunction = "EBSeq.createRmd",
           output.directory = ".", norm.method = "median")
runDiffExp(data.file = "mydata.rds", result.extent = "edgeR.exact",
```

```
            Rmdfunction = "edgeR.exact.createRmd",
            output.directory = ".", norm.method = "TMM",
            trend.method = "movingave", disp.type = "tagwise")
    runDiffExp(data.file = "mydata.rds", result.extent = "edgeR.GLM",
            Rmdfunction = "edgeR.GLM.createRmd",
            output.directory = ".", norm.method = "TMM",
            disp.type = "tagwise", disp.method = "CoxReid",
            trended = TRUE)
    runDiffExp(data.file = "mydata.rds", result.extent = "logcpm.limma",
            Rmdfunction = "logcpm.limma.createRmd",
            output.directory = ".", norm.method = "TMM")
    runDiffExp(data.file = "mydata.rds", result.extent = "NBPSeq",
            Rmdfunction = "NBPSeq.createRmd",
            output.directory = ".", norm.method = "TMM",
            disp.method = "NBP")
    runDiffExp(data.file = "mydata.rds", result.extent = "NOISeq",
            Rmdfunction = "NOISeq.prenorm.createRmd",
            output.directory = ".", norm.method = "TMM")
    runDiffExp(data.file = "mydata.rds", result.extent = "sqrtcpm.limma",
            Rmdfunction = "sqrtcpm.limma.createRmd",
            output.directory = ".", norm.method = "TMM")
    runDiffExp(data.file = "mydata.rds", result.extent = "TCC",
            Rmdfunction = "TCC.createRmd",
            output.directory = ".", norm.method = "tmm",
            test.method = "edger", iteration = 3,
            normFDR = 0.1, floorPDEG = 0.05)
    runDiffExp(data.file = "mydata.rds", result.extent = "ttest",
            Rmdfunction = "ttest.createRmd",
            output.directory = ".", norm.method = "TMM")
    runDiffExp(data.file = "mydata.rds", result.extent = "voom.limma",
            Rmdfunction = "voom.limma.createRmd",
            output.directory = ".", norm.method = "TMM")
    runDiffExp(data.file = "mydata.rds", result.extent = "voom.ttest",
            Rmdfunction = "voom.ttest.createRmd",
            output.directory = ".", norm.method = "TMM")
    }
```

---

show,compData-method *Show method for* compData *object*

Description

Show method for compData object.

Usage

```
## S4 method for signature 'compData'
show(object)
```

Arguments

object	A compData object

Author(s)

Charlotte Soneson

Examples

```
mydata <- generateSyntheticData(dataset = "mydata", n.vars = 12500,
                                samples.per.cond = 5, n.diffexp = 1250)
mydata
```

---

show,phyloCompData-method

Show method for phyloCompData *object*

Description

Show method for phyloCompData object.

Usage

```
## S4 method for signature 'phyloCompData'
show(object)
```

Arguments

object	A phyloCompData object

Author(s)

Charlotte Soneson, Paul Bastide

Examples

```
mydata <- generateSyntheticData(dataset = "mydata", n.vars = 1000,
                                samples.per.cond = 5, n.diffexp = 100,
                                id.species = factor(1:10),
                                tree = ape::rphylo(10, 1, 0),
                                lengths.relmeans = "auto", lengths.dispersions = "auto")
mydata
```

`sqrtcpm.limma.createRmd` *Generate a* `.Rmd` *file containing code to perform differential expression analysis with limma after square root-transforming the counts per million (cpm)*

Description

A function to generate code that can be run to perform differential expression analysis of RNAseq data (comparing two conditions) using limma, after preprocessing the counts by computing the counts per million (cpm) and applying a square-root transformation. The code is written to a `.Rmd` file. This function is generally not called by the user, the main interface for performing differential expression analysis is the `runDiffExp` function.

Usage

```
sqrtcpm.limma.createRmd(data.path, result.path, codefile, norm.method)
```

Arguments

`data.path`	The path to a .rds file containing the `compData` object that will be used for the differential expression analysis.
`result.path`	The path to the file where the result object will be saved.
`codefile`	The path to the file where the code will be written.
`norm.method`	The between-sample normalization method used to compensate for varying library sizes and composition in the differential expression analysis. The normalization factors are calculated using the `calcNormFactors` function from the `edgeR` package. Possible values are `"TMM"`, `"RLE"`, `"upperquartile"` and `"none"`.

Details

For more information about the methods and the interpretation of the parameters, see the `edgeR` and `limma` packages and the corresponding publications.

Value

The function generates a `.Rmd` file containing the code for performing the differential expression analysis. This file can be executed using e.g. the `knitr` package.

Author(s)

Charlotte Soneson

References

Smyth GK (2005): Limma: linear models for microarray data. In: 'Bioinformatics and Computational Biology Solutions using R and Bioconductor'. R. Gentleman, V. Carey, S. Dudoit, R. Irizarry, W. Huber (eds), Springer, New York, pages 397-420

Robinson MD, McCarthy DJ and Smyth GK (2010): edgeR: a Bioconductor package for differential expression analysis of digital gene expression data. Bioinformatics 26, 139-140

Robinson MD and Oshlack A (2010): A scaling normalization method for differential expression analysis of RNA-seq data. Genome Biology 11:R25

Examples

```
tmpdir <- normalizePath(tempdir(), winslash = "/")
mydata.obj <- generateSyntheticData(dataset = "mydata", n.vars = 1000,
                                    samples.per.cond = 5, n.diffexp = 100,
                                    output.file = file.path(tmpdir, "mydata.rds"))
runDiffExp(data.file = file.path(tmpdir, "mydata.rds"), result.extent = "sqrtcpm.limma",
           Rmdfunction = "sqrtcpm.limma.createRmd",
           output.directory = tmpdir, norm.method = "TMM")
```

---

summarizeSyntheticDataSet

Summarize a synthetic data set by some diagnostic plots

Description

Summarize a synthetic data set (generated by `generateSyntheticData`) by some diagnostic plots.

Usage

```
summarizeSyntheticDataSet(data.set, output.filename)
```

Arguments

`data.set`	A data set, either a `compData` object or a path to an `.rds` file where such an object is stored.
`output.filename`	The filename of the resulting html report (including the path).

Author(s)

Charlotte Soneson

Examples

```
tmpdir <- normalizePath(tempdir(), winslash = "/")
mydata.obj <- generateSyntheticData(dataset = "mydata", n.vars = 1000,
                                    samples.per.cond = 5, n.diffexp = 100,
                                    output.file = file.path(tmpdir, "mydata.rds"))
if (interactive()) {
  summarizeSyntheticDataSet(data.set = file.path(tmpdir, "mydata.rds"),
                            output.filename = file.path(tmpdir, "mydata_check.html"))
}
```

`TCC.createRmd` *Generate a* `.Rmd` *file containing code to perform differential expression analysis with TCC*

Description

A function to generate code that can be run to perform differential expression analysis of RNAseq data (comparing two conditions) using the TCC package. The code is written to a `.Rmd` file. This function is generally not called by the user, the main interface for performing differential expression analysis is the `runDiffExp` function.

Usage

```
TCC.createRmd(
  data.path,
  result.path,
  codefile,
  norm.method,
  test.method,
  iteration = 3,
  normFDR = 0.1,
  floorPDEG = 0.05
)
```

Arguments

`data.path`	The path to a .rds file containing the `compData` object that will be used for the differential expression analysis.
`result.path`	The path to the file where the result object will be saved.
`codefile`	The path to the file where the code will be written.
`norm.method`	The between-sample normalization method used to compensate for varying library sizes and composition in the differential expression analysis. Possible values are `"tmm"`, and `"deseq"`.
`test.method`	The method used in TCC to find differentially expressed genes. Possible values are `"edger"`, `"deseq"` and `"bayseq"`.

iteration	The number of iterations used to find the normalization factors. Default value is 3.
normFDR	The FDR cutoff for calling differentially expressed genes in the computation of the normalization factors. Default value is 0.1.
floorPDEG	The minimum value to be eliminated as potential differentially expressed genes before performing step 3 in the TCC algorithm. Default value is 0.05.

Details

For more information about the methods and the interpretation of the parameters, see the TCC package and the corresponding publications.

Author(s)

Charlotte Soneson

References

Kadota K, Nishiyama T, and Shimizu K. A normalization strategy for comparing tag count data. Algorithms Mol Biol. 7:5, 2012.

Sun J, Nishiyama T, Shimizu K, and Kadota K. TCC: an R package for comparing tag count data with robust normalization strategies. BMC Bioinformatics 14:219, 2013.

Examples

```
try(
if (require(TCC)) {
tmpdir <- normalizePath(tempdir(), winslash = "/")
mydata.obj <- generateSyntheticData(dataset = "mydata", n.vars = 1000,
                                    samples.per.cond = 5, n.diffexp = 100,
                                    output.file = file.path(tmpdir, "mydata.rds"))
runDiffExp(data.file = file.path(tmpdir, "mydata.rds"), result.extent = "TCC",
           Rmdfunction = "TCC.createRmd",
           output.directory = tmpdir, norm.method = "tmm",
           test.method = "edger")
})
```

---

ttest.createRmd *Generate a* .Rmd *file containing code to perform differential expression analysis with a t-test*

Description

A function to generate code that can be run to perform differential expression analysis of RNAseq data (comparing two conditions) using a t-test, applied to the normalized counts. The code is written to a .Rmd file. This function is generally not called by the user, the main interface for performing differential expression analysis is the runDiffExp function.

Usage

```
ttest.createRmd(data.path, result.path, codefile, norm.method)
```

Arguments

data.path	The path to a .rds file containing the compData object that will be used for the differential expression analysis.
result.path	The path to the file where the result object will be saved.
codefile	The path to the file where the code will be written.
norm.method	The between-sample normalization method used to compensate for varying library sizes and composition in the differential expression analysis. The normalization factors are calculated using the calcNormFactors function from the edgeR package. Possible values are "TMM", "RLE", "upperquartile" and "none"

Details

For more information about the methods and the interpretation of the parameters, see the edgeR package and the corresponding publications.

Value

The function generates a .Rmd file containing the code for performing the differential expression analysis. This file can be executed using e.g. the knitr package.

Author(s)

Charlotte Soneson

References

Robinson MD, McCarthy DJ and Smyth GK (2010): edgeR: a Bioconductor package for differential expression analysis of digital gene expression data. Bioinformatics 26, 139-140

Robinson MD and Oshlack A (2010): A scaling normalization method for differential expression analysis of RNA-seq data. Genome Biology 11:R25

Examples

```
try(
if (require(genefilter)) {
tmpdir <- normalizePath(tempdir(), winslash = "/")
mydata.obj <- generateSyntheticData(dataset = "mydata", n.vars = 1000,
                                    samples.per.cond = 5, n.diffexp = 100,
                                    output.file = file.path(tmpdir, "mydata.rds"))
runDiffExp(data.file = file.path(tmpdir, "mydata.rds"), result.extent = "ttest",
           Rmdfunction = "ttest.createRmd",
           output.directory = tmpdir, norm.method = "TMM")
})
```

`voom.limma.createRmd` *Generate a* `.Rmd` *file containing code to perform differential expression analysis with voom+limma*

Description

A function to generate code that can be run to perform differential expression analysis of RNAseq data (comparing two conditions) by applying the voom transformation (from the limma package) followed by differential expression analysis with limma. The code is written to a `.Rmd` file. This function is generally not called by the user, the main interface for performing differential expression analysis is the `runDiffExp` function.

Usage

```
voom.limma.createRmd(data.path, result.path, codefile, norm.method)
```

Arguments

`data.path`	The path to a .rds file containing the `compData` object that will be used for the differential expression analysis.
`result.path`	The path to the file where the result object will be saved.
`codefile`	The path to the file where the code will be written.
`norm.method`	The between-sample normalization method used to compensate for varying library sizes and composition in the differential expression analysis. The normalization factors are calculated using the `calcNormFactors` of the `edgeR` package. Possible values are `"TMM"`, `"RLE"`, `"upperquartile"` and `"none"`

Details

For more information about the methods and the interpretation of the parameters, see the `limma` package and the corresponding publications.

Value

The function generates a `.Rmd` file containing the code for performing the differential expression analysis. This file can be executed using e.g. the `knitr` package.

Author(s)

Charlotte Soneson

References

Smyth GK (2005): Limma: linear models for microarray data. In: 'Bioinformatics and Computational Biology Solutions using R and Bioconductor'. R. Gentleman, V. Carey, S. Dudoit, R. Irizarry, W. Huber (eds), Springer, New York, pages 397-420

Law CW, Chen Y, Shi W and Smyth GK (2014): voom: precision weights unlock linear model analysis tools for RNA-seq read counts. Genome Biology 15, R29

Examples

```
tmpdir <- normalizePath(tempdir(), winslash = "/")
mydata.obj <- generateSyntheticData(dataset = "mydata", n.vars = 1000,
                                    samples.per.cond = 5, n.diffexp = 100,
                                    output.file = file.path(tmpdir, "mydata.rds"))
runDiffExp(data.file = file.path(tmpdir, "mydata.rds"), result.extent = "voom.limma",
           Rmdfunction = "voom.limma.createRmd",
           output.directory = tmpdir, norm.method = "TMM")
```

`voom.ttest.createRmd` *Generate a* `.Rmd` *file containing code to perform differential expression analysis with voom+t-test*

Description

A function to generate code that can be run to perform differential expression analysis of RNAseq data (comparing two conditions) by applying the voom transformation (from the `limma` package) followed by differential expression analysis with a t-test. The code is written to a `.Rmd` file. This function is generally not called by the user, the main interface for performing differential expression analysis is the `runDiffExp` function.

Usage

```
voom.ttest.createRmd(data.path, result.path, codefile, norm.method)
```

Arguments

`data.path`	The path to a .rds file containing the `compData` object that will be used for the differential expression analysis.
`result.path`	The path to the file where the result object will be saved.
`codefile`	The path to the file where the code will be written.
`norm.method`	The between-sample normalization method used to compensate for varying library sizes and composition in the differential expression analysis. The normalization factors are calculated using the `calcNormFactors` function from the `edgeR` package. Possible values are `"TMM"`, `"RLE"`, `"upperquartile"` and `"none"`.

Details

For more information about the methods and the interpretation of the parameters, see the `limma` and `edgeR` packages and the corresponding publications.

Value

The function generates a `.Rmd` file containing the code for performing the differential expression analysis. This file can be executed using e.g. the `knitr` package.

Author(s)

Charlotte Soneson

References

Smyth GK (2005): Limma: linear models for microarray data. In: 'Bioinformatics and Computational Biology Solutions using R and Bioconductor'. R. Gentleman, V. Carey, S. Dudoit, R. Irizarry, W. Huber (eds), Springer, New York, pages 397-420

Law CW, Chen Y, Shi W and Smyth GK (2014): voom: precision weights unlock linear model analysis tools for RNA-seq read counts. Genome Biology 15, R29

Examples

```
try(
if (require(genefilter)) {
tmpdir <- normalizePath(tempdir(), winslash = "/")
mydata.obj <- generateSyntheticData(dataset = "mydata", n.vars = 1000,
                                    samples.per.cond = 5, n.diffexp = 100,
                                    output.file = file.path(tmpdir, "mydata.rds"))
runDiffExp(data.file = file.path(tmpdir, "mydata.rds"), result.extent = "voom.ttest",
           Rmdfunction = "voom.ttest.createRmd",
           output.directory = tmpdir, norm.method = "TMM")
})
```

Index

check_compData, 3, *5*
check_compData_results, 3, *5*
check_phyloCompData, 4
checkDataObject, 5
checkTableConsistency, 5
compData, 7, *12*, *29*, *40*, *44*, *59*
compData-class, 10
convertcompDataToList, 12
convertListTocompData, 13
convertListTophyloCompData, 13
convertphyloCompDataToList, 14

DESeq2.createRmd, 15
DESeq2.length.createRmd, 17
DESeqDataSetFromMatrix, *18*
DSS.createRmd, 19
duplicateCorrelation, *31*

EBSeq.createRmd, 20
edgeR.exact.createRmd, 22
edgeR.GLM.createRmd, 23
estimateSizeFactors, *18*

generateCodeHTMLs, *9*, 25, *42*
generateSyntheticData, 26, *55*, *59*

lengthNorm.limma.createRmd, 30, *34*
lengthNorm.sva.limma.createRmd, 33
listcreateRmd, 35, *55*
lmFit, *32*
logcpm.limma.createRmd, 36

NBPSeq.createRmd, 37
NOISeq.prenorm.createRmd, 39
normalizationFactors, *18*
num.sv, *34*

phylo, *28*
phyloCompData, *18*, *31*, *32*, *34*, 40, *40*, *44–46*
phyloCompData-class, 44
phylolm, *45*, *46*
phylolm.createRmd, 45

rTrait, *29*
runComparison, *5*, 47, *50–53*
runComparisonGUI, *47*, 51
runComparisonShiny, 52
runDiffExp, *15*, *17*, *19*, *20*, *22*, *23*, *25*, *30*, *33*, *35–37*, *39*, *45*, *47*, *50*, *51*, *53*, 54, *58*, *60*, *61*, *63*, *64*

show,compData-method, 56
show,phyloCompData-method, 57
sqrtcpm.limma.createRmd, 58
summarizeSyntheticDataSet, 59
sva, *34*

TCC.createRmd, 60
ttest.createRmd, 61

voom.limma.createRmd, 63
voom.ttest.createRmd, 64