Package ‘metaseqR2’

April 24, 2024

Type Package

Title An R package for the analysis and result reporting of RNA-Seq data by combining multiple statistical algorithms

Depends R (>= 4.0.0), DESeq2, limma, locfit, splines

Imports ABSSeq, Biobase, BiocGenerics, BiocParallel, biomaRt, Biostrings, corrplot, DSS, DT, EDASeq, edgeR, genefilter, GenomeInfoDb, GenomicAlignments, GenomicFeatures, GenomicRanges, gplots, graphics, grDevices, harmonicmeanp, heatmaply, htmltools, httr, IRanges, jsonlite, lattice, log4r, magrittr, MASS, Matrix, methods, NBPSeq, pander, parallel, qvalue, rmarkdown, rmdformats, Rsamtools, RSQLite, rtracklayer, S4Vectors, stats, stringr, SummarizedExperiment, survcomp, txdbmaker, utils, VennDiagram, vsn, yaml, zoo

Suggests BiocStyle, BiocManager, BSgenome, knitr, RMySQL, RUnit

Enhances TCC

Description Provides an interface to several normalization and statistical testing packages for RNA-Seq gene expression data. Additionally, it creates several diagnostic plots, performs meta-analysis by combinining the results of several statistical tests and reports the results in an interactive way.

License GPL (>= 3)

Encoding UTF-8

LazyData false

NeedsCompilation yes

URL http://www.fleming.gr

biocViews Software, GeneExpression, DifferentialExpression, WorkflowStep, Preprocessing, QualityControl, Normalization, ReportWriting, RNASeq, Transcription, Sequencing, Transcriptomics, Bayesian, Clustering, CellBiology, BiomedicalInformatics, FunctionalGenomics, SystemsBiology, ImmunoOncology, AlternativeSplicing, DifferentialSplicing, MultipleComparison, TimeCourse, DataImport, ATACSeq,

Epigenetics, Regression, ProprietaryPlatforms, GeneSetEnrichment, BatchEffect, ChIPSeq

VignetteBuilder knitr

BugReports https://github.com/pmoulos/metaseqR2/issues

Version 1.15.2

Date 2024-03-13

Collate 'annotation.R' 'argcheck.R' 'count.R' 'deseq.R' 'export.R' 'external.R' 'filter.R' 'json.R' 'main.R' 'meta.R' 'noiseq.R' 'norm.R' 'metaseqr2-data.R' 'metaseqR2-package.R' 'plot.R' 'query.R' 'reportdb.R' 'sim.R' 'stat.R' 'tracks.R' 'util.R' 'zzz.R'

git_url https://git.bioconductor.org/packages/metaseqR2

git_branch devel

git_last_commit 8378cbf

git_last_commit_date 2024-03-14

Repository Bioconductor 3.19

Date/Publication 2024-04-23

Author Panagiotis Moulos [aut, cre]

Maintainer Panagiotis Moulos <moulos@fleming.gr>

Contents

buildAnnotationDatabase . . . 3
buildCustomAnnotation . . . 5
combineBonferroni . . . 8
combineHarmonic . . . 9
combineMaxp . . . 10
combineMinp . . . 10
combineSimes . . . 11
combineWeight . . . 12
createSignalTracks . . . 13
diagplotAvgFtd . . . 14
diagplotBoxplot . . . 15
diagplotCor . . . 17
diagplotDeHeatmap . . . 18
diagplotEdaseq . . . 19
diagplotFiltered . . . 20
diagplotFtd . . . 21
diagplotMds . . . 22
diagplotNoiseq . . . 23
diagplotPairs . . . 25
diagplotRoc . . . 26
diagplotVenn . . . 27

diagplotVolcano . . . 28
downsampleCounts . . . 30
estimateAufcWeights . . . 30
estimateSimParams . . . 32
getAnnotation . . . 33
getDefaults . . . 34
getInstalledAnnotations . . . 35
getWeights . . . 36
hg19pvalues . . . 37
importCustomAnnotation . . . 37
libsizeListMm9 . . . 38
loadAnnotation . . . 39
makeSimDataSd . . . 40
makeSimDataTcc . . . 42
metaseqr2 . . . 42
metaseqrPlot . . . 59
metaTest . . . 61
mm9GeneCounts . . . 63
normalizeAbsseq . . . 63
normalizeDeseq . . . 64
normalizeDeseq2 . . . 65
normalizeDss . . . 66
normalizeEdaseq . . . 67
normalizeEdger . . . 69
normalizeNbpseq . . . 70
normalizeNoiseq . . . 71
read2count . . . 72
readTargets . . . 73
sampleListMm9 . . . 75
statAbsseq . . . 75
statDeseq . . . 76
statDeseq2 . . . 77
statDss . . . 79
statEdger . . . 80
statLimma . . . 81
statNbpseq . . . 82
statNoiseq . . . 83

Index **85**

`buildAnnotationDatabase`

Build a local annotation database for metaseqR2

Description

This function creates a local annotation database to be used with metaseqr2 so as to avoid long time on the fly annotation downloads and formatting.

Usage

```
buildAnnotationDatabase(organisms, sources,
db = file.path(system.file(package = "metaseqR2"),
    "annotation.sqlite"),
forceDownload = TRUE, rc = NULL)
```

Arguments

organisms	a list of organisms and versions for which to download and build annotations. Check the main metaseqr2 help page for details on supported organisms and the Details section below.
sources	a character vector of public sources from which to download and build annotations. Check the main metaseqr2 help page for details on supported annotation sources.
db	a valid path (accessible at least by the current user) where the annotation database will be set up. It defaults to system.file(package = "metaseqR2"), "annotation.sqlite") that is, the installation path of metaseqR2 package. See also Details.
forceDownload	by default, buildAnnotationDatabase will not download an existing annotation again (FALSE). Set to TRUE if you wish to update the annotation database for a particular version.
rc	fraction (0-1) of cores to use in a multicore system. It defaults to NULL (no parallelization). Sometimes used for building certain annotation types.

Details

Regarding the organisms argument, it is a list with specific format which instructs buildAnnotationDatabase on which organisms and versions to download from the respective sources. Such a list may have the format: organisms=list(hg19=75, mm9=67, mm10=96:97) This is explained as follows:

- A database comprising the human genome versions hg19 and the mouse genome versions mm9, mm10 will be constructed.
- If "ensembl" is in sources, version 75 is downloaded for hg19 and versions 67, 96, 97 for mm9, mm10.
- If "ucsc" or "refseq" are in sources, the latest versions are downloaded and marked by the download date. As UCSC and RefSeq versions are not accessible in the same way as Ensembl, this procedure cannot always be replicated.

organisms can also be a character vector with organism names/versions (e.g. organisms = c("mm10","hg19")), then the latest versions are downloaded in the case of Ensembl.

Regarding db, this controls the location of the installation database. If the default is used, then there is no need to provide the local database path to any function that uses the database (e.g. the main metaseqr2). Otherwise, the user will either have to provide this each time, or the annotation will have to be downloaded and used on-the-fly.

Value

The function does not return anything. Only the SQLite database is created or updated.

Author(s)

Panagiotis Moulos

Examples

```
# Build a test database with one genome
myDb <- file.path(tempdir(),"testann.sqlite")

organisms <- list(mm10=75)
sources <- "ensembl"

# If the example is not running in a multicore system, rc is ignored
#buildAnnotationDatabase(organisms,sources,db=myDb,rc=0.5)

# A more complete case, don't run as example
# Since we are using Ensembl, we can also ask for a version
#organisms <- list(
#    mm9=67,
#    mm10=96:97,
#    hg19=75,
#    hg38=96:97
#)
#sources <- c("ensembl", "refseq")

## Build on the default location (depending on package location, it may
## require root/sudo)
#buildAnnotationDatabase(organisms,sources)

## Build on an alternative location
#myDb <- file.path(path.expand("~"),"my_ann.sqlite")
#buildAnnotationDatabase(organisms,sources,db=myDb)
```

---

buildCustomAnnotation *Import custom annotation to the metaseqR2 annotation database from GTF file*

Description

This function imports a GTF file with some custom annotation to the metaseqR2 annotation database.

Usage

```
buildCustomAnnotation(gtfFile, metadata,
db = file.path(system.file(package = "metaseqR2"),
    "annotation.sqlite"), rewrite=TRUE)
```

Arguments

`gtfFile`	a GTF file containing the gene structure of the organism to be imported.
`metadata`	a list with additional information about the annotation to be imported. See Details.
`db`	a valid path (accessible at least by the current user) where the annotation database will be set up. It defaults to `system.file(package = "metaseqR2"), "annotation.sqlite")` that is, the installation path of metaseqR2 package. See also Details.
`rewrite`	if custom annotation found, rwrite? (default `FALSE`). Set to `TRUE` if you wish to update the annotation database for a particular custom annotation.

Details

Regarding the `metadata` argument, it is a list with specific format which instructs `buildCustomAnnotation` on importing the custom annotation. Such a list may has the following members:

- `organism` a name of the organism which is imported (e.g. `"my_mm9"`). This is the only mandatory member.
- `source` a name of the source for this custom annotation (e.g. `"my_mouse_db"`). If not given or `NULL`, the word `"inhouse"` is used.
- `version` a string denoting the version. If not given or `NULL`, current date is used.
- `chromInfo` it can be one of the following:
  - a tab-delimited file with two columns, the first being the chromosome/sequence names and the second being the chromosome/sequence lengths.
  - a BAM file to read the header from and obtain the required information
  - a `data.frame` with one column with chromosome lengths and chromosome names as `rownames`.

See the examples below for a `metadata` example.

Regarding `db`, this controls the location of the installation database. If the default is used, then there is no need to provide the local database path to any function that uses the database (e.g. the main `metaseqr2`). Otherwise, the user will either have to provide this each time, or the annotation will have to be downloaded and used on-the-fly.

Value

The function does not return anything. Only the SQLite database is created or updated.

Author(s)

Panagiotis Moulos

Examples

```
# Dummy database as example
customDir <- file.path(tempdir(),"test_custom")
dir.create(customDir)
```

```
myDb <- file.path(customDir,"testann.sqlite")
chromInfo <- data.frame(length=c(1000L,2000L,1500L),
    row.names=c("A","B","C"))

# Build with the metadata list filled (you can also provide a version)
if (.Platform$OS.type == "unix") {
    buildCustomAnnotation(
        gtfFile=file.path(system.file(package="metaseqR2"),
            "dummy.gtf"),
        metadata=list(
            organism="dummy",
            source="dummy_db",
            version=1,
            chromInfo=chromInfo
        ),
        db=myDb
    )

    # Try to retrieve some data
    myGenes <- loadAnnotation(genome="dummy",refdb="dummy_db",
        level="gene",type="gene",db=myDb)
    myGenes
}

## Real data!
## Setup a temporary directory to download files etc.
#customDir <- file.path(tempdir(),"test_custom")
#dir.create(customDir)

#myDb <- file.path(customDir,"testann.sqlite")

## Gene annotation dump from Ensembl
#download.file(paste0("ftp://ftp.ensembl.org/pub/release-98/gtf/",
#  "dasypus_novemcinctus/Dasypus_novemcinctus.Dasnov3.0.98.gtf.gz"),
#  file.path(customDir,"Dasypus_novemcinctus.Dasnov3.0.98.gtf.gz"))

## Chromosome information will be provided from the following BAM file
## available from Ensembl
#bamForInfo <- paste0("ftp://ftp.ensembl.org/pub/release-98/bamcov/",
#  "dasypus_novemcinctus/genebuild/Dasnov3.broad.Ascending_Colon_5.1.bam")

## Build with the metadata list filled (you can also provide a version)
#buildCustomAnnotation(
#  gtfFile=file.path(customDir,"Dasypus_novemcinctus.Dasnov3.0.98.gtf.gz"),
#  metadata=list(
#    organism="dasNov3_test",
#    source="ensembl_test",
#    chromInfo=bamForInfo
#  ),
#  db=myDb
#)

## Try to retrieve some data
```

```
#dasGenes <- loadAnnotation(genome="dasNov3_test",refdb="ensembl_test",
#  level="gene",type="gene",db=myDb)
#dasGenes
```

`combineBonferroni` *Combine p-values with Bonferroni's method*

Description

This function combines p-values from the various statistical tests supported by metaseqR2 using the Bonferroni's method (see reference in the main `metaseqr2` help page or in the vignette).

Usage

```
combineBonferroni(p, zerofix = NULL)
```

Arguments

`p`	a vector of p-values for each statistical tests).
`zerofix`	`NULL` (default) or a fixed numeric value between 0 and 1.

Details

The argument `zerofix` is used to correct for the case of a p-value which is equal to 0 as a result of internal numerical and approximation procedures. When `NULL`, random numbers greater than 0 and less than or equal to 0.5 are used to multiply the offending p-values with the lowest provided non-zero p-value, maintaining thus a virtual order of significance, avoiding having the same p-values for two tests and assuming that all zero p-values represent extreme statistical significance. When a numeric between 0 and 1, this number is used for the above multiplication instead.

Value

A vector of combined p-values.

Author(s)

Panagiotis Moulos

Examples

```
p <- matrix(runif(300),100,3)
pc <- combineBonferroni(p)
```

combineHarmonic *Combine p-values using weights*

Description

This function combines p-values from the various statistical tests supported by metaseqR using p-value weights.

Usage

```
combineHarmonic(p, w, zerofix = NULL)
```

Arguments

p	a p-value matrix (rows are genes, columns are statistical tests).
w	a weights vector, must sum to 1.
zerofix	NULL (default) or a fixed numeric value between 0 and 1.

Details

The argument zerofix is used to correct for the case of a p-value which is equal to 0 as a result of internal numerical and approximation procedures. When NULL, random numbers greater than 0 and less than or equal to 0.5 are used to multiply the offending p-values with the lowest provided non-zero p-value, maintaining thus a virtual order of significance, avoiding having the same p-values for two tests and assuming that all zero p-values represent extreme statistical significance. When a numeric between 0 and 1, this number is used for the above multiplication instead.

Value

A vector of combined p-values.

Author(s)

Panagiotis Moulos

Examples

```
p <- matrix(runif(300),100,3)
pc <- combineHarmonic(p,w=c(0.2,0.5,0.3))
```

combineMaxp — *Combine p-values using the maximum p-value*

Description

This function combines p-values from the various statistical tests supported by metaseqR by taking the maximum p-value.

Usage

```
combineMaxp(p)
```

Arguments

p	a p-value matrix (rows are genes, columns are statistical tests).

Value

A vector of combined p-values.

Author(s)

Panagiotis Moulos

Examples

```
p <- matrix(runif(300),100,3)
pc <- combineMaxp(p)
```

combineMinp — *Combine p-values using the minimum p-value*

Description

This function combines p-values from the various statistical tests supported by metaseqR by taking the minimum p-value.

Usage

```
combineMinp(p)
```

Arguments

p	a p-value matrix (rows are genes, columns are statistical tests).

Value

A vector of combined p-values.

Author(s)

Panagiotis Moulos

Examples

```
p <- matrix(runif(300),100,3)
pc <- combineMinp(p)
```

---

combineSimes *Combine p-values with Simes' method*

Description

This function combines p-values from the various statistical tests supported by metaseqR using the Simes' method (see reference in the main `metaseqr2` help page or in the vignette).

Usage

```
combineSimes(p, zerofix = NULL)
```

Arguments

`p`	a p-value matrix (rows are genes, columns are statistical tests).
`zerofix`	`NULL` (default) or a fixed numeric value between 0 and 1.

Details

The argument `zerofix` is used to correct for the case of a p-value which is equal to 0 as a result of internal numerical and approximation procedures. When `NULL`, random numbers greater than 0 and less than or equal to 0.5 are used to multiply the offending p-values with the lowest provided non-zero p-value, maintaining thus a virtual order of significance, avoiding having the same p-values for two tests and assuming that all zero p-values represent extreme statistical significance. When a numeric between 0 and 1, this number is used for the above multiplication instead.

Value

A vector of combined p-values.

Author(s)

Panagiotis Moulos

Examples

```
p <- matrix(runif(300),100,3)
pc <- combineSimes(p)
```

`combineWeight` *Combine p-values using weights*

Description

This function combines p-values from the various statistical tests supported by metaseqR using p-value weights.

Usage

```
combineWeight(p, w, zerofix = NULL)
```

Arguments

`p`	a p-value matrix (rows are genes, columns are statistical tests).
`w`	a weights vector, must sum to 1.
`zerofix`	`NULL` (default) or a fixed numeric value between 0 and 1.

Details

The argument `zerofix` is used to correct for the case of a p-value which is equal to 0 as a result of internal numerical and approximation procedures. When `NULL`, random numbers greater than 0 and less than or equal to 0.5 are used to multiply the offending p-values with the lowest provided non-zero p-value, maintaining thus a virtual order of significance, avoiding having the same p-values for two tests and assuming that all zero p-values represent extreme statistical significance. When a numeric between 0 and 1, this number is used for the above multiplication instead.

Value

A vector of combined p-values.

Author(s)

Panagiotis Moulos

Examples

```
p <- matrix(runif(300),100,3)
pc <- combineWeight(p,w=c(0.2,0.5,0.3))
```

`createSignalTracks` *Create bigWig signal tracks*

Description

This function creates bigWig files to be used for exploring RNA signal in genome browsers. When strands are separated, a UCSC genome browser trackhub is created to group tracks for the same sample. A link to the created data is returned.

Usage

```
createSignalTracks(targets, org, urlBase = NULL,
    stranded = FALSE, normTo = 1e+9, exportPath = ".",
    hubInfo = list(name = "MyHub", shortLabel = "My hub",
    longLabel = "My hub", email = "someone@example.com"),
    fasta = NULL, gtf = NULL, forceHub = FALSE,
    overwrite = FALSE, rc = NULL)
```

Arguments

`targets`	a tab-delimited file with the experimental description or the output of `readTargets`. See also the `sampleList` argument in the main `metaseqr2` pipeline.
`org`	See the `org` argument in the main `metaseqr2` pipeline.
`urlBase`	a valid URL which is prepended to the created bigWig files.
`stranded`	Separate + and - strands and create separate bigWig files.
`normTo`	the total sum of signal to be used as the normalization target. See also the `trackInfo` argument in the main `metaseqr2` pipeline.
`exportPath`	path to export tracks.
`hubInfo`	information regarding the track hub created when `stranded=TRUE`. See also the `trackInfo` argument in the main `metaseqr2` pipeline.
`overwrite`	overwrite tracks if they exist? Defaults to `FALSE`.
`fasta`	reference genome in FASTA format for the case of analyzing a custom, non-directly supported organism. It will be converted to the .2bit format and written along with a track hub.
`gtf`	a GTF file describing gene models in the case of analyzing a custom, non-directly supported organism. It will be converted to the .bigBed format and written along with a track hub.
`forceHub`	when `stranded=TRUE`, a UCSC Genome Browser trackhub is created, otherwise only tracklines describing individual tracks. If `TRUE`, a trackhub is always created.
`rc`	Fraction of cores to use.

Value

A string with the link(s) to the created tracks.

Author(s)

Panagiotis Moulos

Examples

```
dataPath <- system.file("extdata",package="metaseqR2")
targets <- data.frame(samplename=c("C","T"),
    filename=file.path(dataPath,c("C.bam","T.bam")),
    condition=c("Control","Treatment"),
    paired=c("single","single"),stranded=c("forward","forward"))
path <- tempdir()
write.table(targets,file=file.path(path,"targets.txt"),
    sep="\t",row.names=FALSE,quote=FALSE)
if (.Platform$OS.type == "unix")
    link <- createSignalTracks(file.path(path,"targets.txt"),"mm9")
```

---

diagplotAvgFtd — *Create average False (or True) Discovery curves*

Description

This function creates false (or true) discovery curves using a list containing several outputs from diagplotFtd.

Usage

```
diagplotAvgFtd(ftdrObj, output = "x11",
    path = NULL, draw = TRUE, ...)
```

Arguments

ftdrObj	a list with outputs from diagplotFtd.
output	one or more R plotting device to direct the plot result to. Supported mechanisms: "x11" (default), "png", "jpg", "bmp", "pdf" or "ps".
path	the path to create output files.
draw	boolean to determine whether to plot the curves or just return the calculated values (in cases where the user wants the output for later averaging for example). Defaults to TRUE (make plots).
...	further arguments to be passed to plot devices, such as parameter from par.

Value

A named list with two members: the first member (avgFtdr) contains a list with the means and the standard deviations of the averaged ftdrObj and are used to create the plot. The second member (path) contains the path to the created figure graphic.

Author(s)

Panagiotis Moulos

Examples

```
p11 <- 0.001*matrix(runif(300),100,3)
p12 <- matrix(runif(300),100,3)
p21 <- 0.001*matrix(runif(300),100,3)
p22 <- matrix(runif(300),100,3)
p31 <- 0.001*matrix(runif(300),100,3)
p32 <- matrix(runif(300),100,3)
p1 <- rbind(p11,p21)
p2 <- rbind(p12,p22)
p3 <- rbind(p31,p32)
rownames(p1) <- rownames(p2) <- rownames(p3) <-
    paste("gene",1:200,sep="_")
colnames(p1) <- colnames(p2) <- colnames(p3) <-
    paste("method",1:3,sep="_")
truth <- c(rep(1,40),rep(-1,40),rep(0,20),
    rep(1,10),rep(2,10),rep(0,80))
names(truth) <- rownames(p1)
ftdObj1 <- diagplotFtd(truth,p1,N=100,draw=FALSE)
ftdObj2 <- diagplotFtd(truth,p2,N=100,draw=FALSE)
ftdObj3 <- diagplotFtd(truth,p3,N=100,draw=FALSE)
ftdObj <- list(ftdObj1,ftdObj2,ftdObj3)
avgFtdObj <- diagplotAvgFtd(ftdObj)
```

---

diagplotBoxplot *Boxplots wrapper for the metaseqR2 package*

Description

A wrapper over the general boxplot function, suitable for matrices produced and processed with the metaseqr package. Intended for internal use but can be easily used as stand-alone. It can colors boxes based on group depending on the name argument.

Usage

```
diagplotBoxplot(mat, name = NULL, logIt = "auto",
    yLim = "default", isNorm = FALSE, output = "x11",
    path = NULL, altNames = NULL, ...)
```

Arguments

`mat`	the count data matrix.
`name`	the names of the samples plotted on the boxplot. See also Details.
`logIt`	whether to log transform the values of mat or not. It can be `TRUE`, `FALSE` or `"auto"` for auto-detection. Auto-detection log transforms by default so that the boxplots are smooth and visible.

yLim	custom y-axis limits. Leave the string "default" for default behavior.
isNorm	a logical indicating whether object contains raw or normalized data. It is not essential and it serves only plot annotation purposes.
output	one or more R plotting device to direct the plot result to. Supported mechanisms: "x11" (default), "png", "jpg", "bmp", "pdf", "ps" or "json". The latter is currently available for the creation of interactive volcano plots only when reporting the output, through the highcharts javascript library (JSON for boxplots not yet available).
path	the path to create output files.
altNames	an optional vector of names, e.g. HUGO gene symbols, alternative or complementary to the unique rownames of mat (which must exist!). It is used only in JSON output.
...	further arguments to be passed to plot devices, such as parameter from par.

Details

Regarding name, if NULL, the function check the column names of mat. If they are also NULL, sample names are autogenerated. If name="none", no sample names are plotted. If name is a list, it should be the sampleList argument provided to the manin metaseqr2 function. In that case, the boxes are colored per group.

Value

The filename of the boxplot produced if it's a file.

Author(s)

Panagiotis Moulos

Examples

```
# Non-normalized boxplot
require(DESeq2)
dataMatrix <- metaseqR2:::exampleCountData(2000)
sampleList <- list(A=c("A1","A2"),B=c("B1","B2","B3"))
diagplotBoxplot(dataMatrix,sampleList)

# Normalized boxplot
normArgs <- getDefaults("normalization","deseq2")
object <- normalizeDeseq2(dataMatrix,sampleList,normArgs)
diagplotBoxplot(object,sampleList)
```

diagplotCor *Summarized correlation plots*

Description

This function uses the read counts matrix to create heatmap or correlogram correlation plots.

Usage

```
diagplotCor(mat, type = c("heatmap", "correlogram"),
    output = "x11", path = NULL, ...)
```

Arguments

mat	the read counts matrix or data frame.
type	create heatmap of correlogram plots.
output	one or more R plotting device to direct the plot result to. Supported mechanisms: "x11" (default), "png", "jpg", "bmp", "pdf" or "ps".
path	the path to create output files.
...	further arguments to be passed to plot devices, such as parameter from par.

Value

The filename of the pairwise comparisons plot produced if it's a file.

Author(s)

Panagiotis Moulos

Examples

```
dataMatrix <- metaseqR2:::exampleCountData(2000)
diagplotCor(dataMatrix,type="heatmap")
diagplotCor(dataMatrix,type="correlogram")
```

`diagplotDeHeatmap` *Diagnostic heatmap of differentially expressed genes*

Description

This function plots a heatmap of the differentially expressed genes produced by the metaseqr workflow, useful for quality control, e.g. whether samples belonging to the same group cluster together.

Usage

```
diagplotDeHeatmap(x, scale = c("asis", "zscore"), con = NULL,
    output = "x11", path = NULL, ...)
```

Arguments

`x`	the data matrix to create a heatmap for.
`scale`	value scale in the heatmap. As provided (`scale="asis"`, default) or Z-scores (`scale="zscore"`)
`con`	an optional string depicting a name (e.g. the contrast name) to appear in the title of the volcano plot.
`output`	one or more R plotting device to direct the plot result to. Supported mechanisms: `"x11"` (default), `"png"`, `"jpg"`, `"bmp"`, `"pdf"`, `"ps"`.
`path`	the path to create output files.
`...`	further arguments to be passed to plot devices, such as parameter from `par`.

Value

The filenames of the plots produced in a named list with names the `whichPlot` argument. If `output="x11"`, no output filenames are produced.

Author(s)

Panagiotis Moulos

Examples

```
dataMatrix <- metaseqR2:::exampleCountData(2000)
sampleList <- list(A=c("A1","A2"),B=c("B1","B2","B3"))
contrast <- "A_vs_B"
M <- normalizeEdger(dataMatrix,sampleList)
p <- statEdger(M,sampleList,contrast)
diagplotDeHeatmap(dataMatrix[p[[1]]<0.05,])
```

diagplotEdaseq *Diagnostic plots based on the EDASeq package*

Description

A wrapper around the plotting functions availale in the EDASeq normalization Bioconductor package. For analytical explanation of each plot please see the vignette of the EDASeq package. It is best to use this function through the main plotting function `metaseqrPlot`.

Usage

```
diagplotEdaseq(x, sampleList, covar = NULL,
    isNorm = FALSE,
    whichPlot = c("meanvar", "meandiff", "gcbias", "lengthbias"),
    output = "x11", altNames = NULL, path = NULL, ...)
```

Arguments

`x`	the count data matrix.
`sampleList`	the list containing condition names and the samples under each condition.
`covar`	The covariate to plot counts against. Usually `"gc"` or `"length"`.
`isNorm`	a logical indicating whether object contains raw or normalized data. It is not essential and it serves only plot annotation purposes.
`whichPlot`	the EDASeq package plot to generate. See Details.
`output`	one or more R plotting device to direct the plot result to. Supported mechanisms: `"x11"` (default), `"png"`, `"jpg"`, `"bmp"`, `"pdf"` or `"ps"`.
`altNames`	optional names, alternative or complementary to the rownames of `x`. It is used only in JSON output.
`path`	the path to create output files.
`...`	further arguments to be passed to plot devices, such as parameter from `par`.

Details

Regarding `whichPlot`, it can be one or more of `"meanvar"`, `"meandiff"`, `"gcbias"` or `"lengthbias"`. Please refer to the documentation of the EDASeq package for details on the use of these plots. The `whichPlot="lengthbias"` case is not covered by EDASeq documentation, however it is similar to the GC-bias plot when the covariate is the gene length instead of the GC content.

Value

The filenames of the plot produced in a named list with names the which.plot argument. If `output="x11"`, no output filenames are produced.

Author(s)

Panagiotis Moulos

Examples

```
dataMatrix <- metaseqR2:::exampleCountData(2000)
sampleList <- list(A=c("A1","A2"),B=c("B1","B2","B3"))
diagplotEdaseq(dataMatrix,sampleList,whichPlot="meandiff")
```

---

diagplotFiltered *Diagnostic plot for filtered genes*

Description

This function plots a grid of four graphs depicting: in the first row, the numbers of filtered genes per chromosome in the first column and per biotype in the second column. In the second row, the percentages of filtered genes per chromosome related to the whole genome in the first columns and per biotype in the second column.

Usage

```
diagplotFiltered(x, y, output = "x11", path = NULL, ...)
```

Arguments

x	an annotation data frame like the ones produced by getAnnotation. x should be the filtered annotation according to metaseqR's filters.
y	an annotation data frame like the ones produced by getAnnotation. y should contain the total annotation without the application of any metaseqr filter.
output	one or more R plotting device to direct the plot result to. Supported mechanisms: "x11" (default), "png", "jpg", "bmp", "pdf" or "ps".
path	the path to create output files.
...	further arguments to be passed to plot devices, such as parameter from par.

Value

The filenames of the plots produced in a named list with names the which.plot argument. If output="x11", no output filenames are produced.

Author(s)

Panagiotis Moulos

Examples

```
data("mm9GeneData",package="metaseqR2")
y <- mm9GeneCounts[,c(1:6,8,7)]
x <- y[-sample(1:nrow(y),1000),]
diagplotFiltered(x,y)
```

diagplotFtd *Create False (or True) Positive (or Negative) curves*

Description

This function creates false (or true) discovery curves using a matrix of p-values (such a matrix can be derived for example from the result table of metaseqr2 by subsetting the table to get the p-values from several algorithms) given a ground truth vector for differential expression.

Usage

```
diagplotFtd(truth, p, type = "fpc", N = 2000,
    output = "x11", path = NULL, draw = TRUE, ...)
```

Arguments

truth	the ground truth differential expression vector. It should contain only zero and non-zero elements, with zero denoting non-differentially expressed genes and non-zero, differentially expressed genes. Such a vector can be obtained for example by using the makeSimDataSd function, which creates simulated RNA-Seq read counts based on real data. The elements of truth MUST be named (e.g. each gene's name).
p	a p-value matrix whose rows correspond to each element in the truth vector. If the matrix has a colnames attribute, a legend will be added to the plot using these names, else a set of column names will be auto-generated. p can also be a list or a data frame. The p-values MUST be named (e.g. each gene's name).
type	what to plot, can be "fpc" for False Positive Curves (default), "tpc" for True Positive Curves, "fnc" for False Negative Curves or "tnc" for True Negative Curves.
N	create the curves based on the top (or bottom) N ranked genes (default is 2000) to be used with type="fpc" or type="tpc".
output	one or more R plotting device to direct the plot result to. Supported mechanisms: "x11" (default), "png", "jpg", "bmp", "pdf" or "ps".
path	the path to create output files.
draw	boolean to determine whether to plot the curves or just return the calculated values (in cases where the user wants the output for later averaging for example). Defaults to TRUE (make plots).
...	further arguments to be passed to plot devices, such as parameter from par.

Value

A named list with two members: the first member (ftdr) contains the values used to create the plot. The second member (path) contains the path to the created figure graphic.

Author(s)

Panagiotis Moulos

Examples

```
p1 <- 0.001*matrix(runif(300),100,3)
p2 <- matrix(runif(300),100,3)
p <- rbind(p1,p2)
rownames(p) <- paste("gene",1:200,sep="_")
colnames(p) <- paste("method",1:3,sep="_")
truth <- c(rep(1,40),rep(-1,40),rep(0,20),
    rep(1,10),rep(2,10),rep(0,80))
names(truth) <- rownames(p)
ftdObj <- diagplotFtd(truth,p,N=100)
```

---

diagplotMds *Multi-Dimensinal Scale plots or RNA-Seq samples*

Description

Creates a Multi-Dimensional Scale plot for the given samples based on the count data matrix. MDS plots are very useful for quality control as you can easily see of samples of the same groups are clustered together based on the whole dataset.

Usage

```
diagplotMds(x, sampleList, method = "spearman",
    logIt = TRUE, output = "x11", path = NULL, ...)
```

Arguments

`x`	the count data matrix.
`sampleList`	the list containing condition names and the samples under each condition.
`method`	which correlation method to use. Same as the method parameter in `cor` function.
`logIt`	whether to log transform the values of x or not.
`output`	one or more R plotting device to direct the plot result to. Supported mechanisms: `"x11"` (default), `"png"`, `"jpg"`, `"bmp"`, `"pdf"`, `"ps"` or `"json"`. The latter is currently available for the creation of interactive volcano plots only when reporting the output, through the highcharts javascript library.
`path`	the path to create output files.
`...`	further arguments to be passed to plot devices, such as parameter from `par`.

Value

The filename of the MDS plot produced if it's a file.

Author(s)

Panagiotis Moulos

Examples

```
dataMatrix <- metaseqR2:::exampleCountData(5000)
sampleList <- list(A=c("A1","A2"),B=c("B1","B2","B3"))
diagplotMds(dataMatrix,sampleList)
```

---

diagplotNoiseq *Diagnostic plots based on the NOISeq package*

Description

A wrapper around the plotting functions availale in the NOISeq Bioconductor package. For analytical explanation of each plot please see the vignette of the NOISeq package. It is best to use this function through the main plotting function `metaseqrPlot`.

Usage

```
diagplotNoiseq(x, sampleList, covars,
    whichPlot = c("biodetection", "countsbio", "saturation",
    "rnacomp", "readnoise", "biodist"),
    output = "x11",
    biodistOpts = list(p = NULL, pcut = NULL, name = NULL),
    path = NULL, isNorm = FALSE, ...)
```

Arguments

`x`	the count data matrix.
`sampleList`	the list containing condition names and the samples under each condition.
`covars`	a list (whose annotation elements are ideally a subset of an annotation data frame produced by `getAnnotation`) with the following members: data (the data matrix), length (gene length), gc (the gene gc_content), chromosome (a data frame with chromosome name and co-ordinates), factors (a factor with the experimental condition names replicated by the number of samples in each experimental condition) and biotype (each gene's biotype as depicted in Ensembl-like annotations).
`whichPlot`	the NOISeq package plot to generate. See Details
`biodistOpts`	a list with the following members: p (a vector of p-values, e.g. the p-values of a contrast), pcut (a unique number depicting a p-value cutoff, required for the `"biodist"` case), name (a name for the `"biodist"` plot, e.g. the name of the contrast.
`output`	one or more R plotting device to direct the plot result to. Supported mechanisms: `"x11"` (default), `"png"`, `"jpg"`, `"bmp"`, `"pdf"` or `"ps"`.

path	the path to create output files.
isNorm	a logical indicating whether object contains raw or normalized data. It is not essential and it serves only plot annotation purposes.
...	further arguments to be passed to plot devices, such as parameter from par.

Details

Regarding whichPlot, It can be one or more of "biodetection", "countsbio", "saturation", "rnacomp", "readnoise" or "biodist". Please refer to the documentation of the NOISeq package for details on the use of these plots. The whichPlot="saturation" case is modified to be more informative by producing two kinds of plots.

Value

The filenames of the plots produced in a named list with names the whichPlot argument. If output="x11", no output filenames are produced.

Note

Please note that in case of "biodist" plots, the behavior of the function is unstable, mostly due to the very specific inputs this plotting function accepts in the NOISeq package. We have tried to predict unstable behavior and avoid exceptions through the use of tryCatch but it's still possible that you might run onto an error.

Author(s)

Panagiotis Moulos

Examples

```
dataMatrix <- metaseqR2:::exampleCountData(5000)
sampleList <- list(A=c("A1","A2"),B=c("B1","B2","B3"))
lengths <- round(1000*runif(nrow(dataMatrix)))
starts <- round(1000*runif(nrow(dataMatrix)))
ends <- starts + lengths
covars <- list(
    data=dataMatrix,
    length=lengths,
    gc=runif(nrow(dataMatrix)),
    chromosome=data.frame(
        chromosome=c(rep("chr1",nrow(dataMatrix)/2),
        rep("chr2",nrow(dataMatrix)/2)),
        start=starts,
        end=ends
    ),
    factors=data.frame(class=metaseqR2:::asClassVector(sampleList)),
    biotype=c(rep("protein_coding",nrow(dataMatrix)/2),rep("ncRNA",
        nrow(dataMatrix)/2))
)
p <- runif(nrow(dataMatrix))
diagplotNoiseq(dataMatrix,sampleList,covars=covars,
```

```
biodistOpts=list(p=p,pcut=0.1,name="A_vs_B"))
```

---

diagplotPairs *Massive X-Y, M-D correlation plots*

Description

This function uses the read counts matrix to create pairwise correlation plots. The upper diagonal of the final image contains simple scatterplots of each sample against each other (log2 scale) while the lower diagonal contains mean-difference plots for the same samples (log2 scale). This type of diagnostic plot may not be interpretable for more than 10 samples.

Usage

```
diagplotPairs(x, output = "x11", altNames = NULL,
    path = NULL, ...)
```

Arguments

x	the read counts matrix or data frame.
output	one or more R plotting device to direct the plot result to. Supported mechanisms: "x11" (default), "png", "jpg", "bmp", "pdf" or "ps".
altNames	optional names, alternative or complementary to the rownames of x. It is used only in JSON output.
path	the path to create output files.
...	further arguments to be passed to plot devices, such as parameter from par.

Value

The filename of the pairwise comparisons plot produced if it's a file.

Author(s)

Panagiotis Moulos

Examples

```
dataMatrix <- metaseqR2:::exampleCountData(2000)
diagplotPairs(dataMatrix)
```

diagplotRoc — *Create basic ROC curves*

Description

This function creates basic ROC curves using a matrix of p-values (such a matrix can be derived for example from the result table of `metaseqr2` by subsetting the table to get the p-values from several algorithms) given a ground truth vector for differential expression and a significance level.

Usage

```
diagplotRoc(truth, p, sig = 0.05, x = "fpr",
    y = "tpr", output = "x11", path = NULL,
    draw = TRUE, ...)
```

Arguments

`truth`	the ground truth differential expression vector. It should contain only zero and non-zero elements, with zero denoting non-differentially expressed genes and non-zero, differentially expressed genes. Such a vector can be obtained for example by using the `makeSimDataSd` function, which creates simulated RNA-Seq read counts based on real data.
`p`	a p-value matrix whose rows correspond to each element in the `truth` vector. If the matrix has a `colnames` attribute, a legend will be added to the plot using these names, else a set of column names will be auto-generated. `p` can also be a list or a data frame.
`sig`	a significance level (0 < `sig` <=1).
`x`	what to plot on x-axis, can be one of `"fpr"`, `"fnr"`, `"tpr"`, `"tnr"` for False Positive Rate, False Negative Rate, True Positive Rate and True Negative Rate respectively.
`y`	what to plot on y-axis, same as `x` above.
`output`	one or more R plotting device to direct the plot result to. Supported mechanisms: `"x11"` (default), `"png"`, `"jpg"`, `"bmp"`, `"pdf"` or `"ps"`.
`path`	the path to create output files.
`draw`	boolean to determine whether to plot the curves or just return the calculated values (in cases where the user wants the output for later averaging for example). Defaults to `TRUE` (make plots).
`...`	further arguments to be passed to plot devices, such as parameter from `par`.

Value

A named list with two members. The first member is a list containing the ROC statistics: `TP` (True Postives), `FP` (False Positives), `FN` (False Negatives), `TN` (True Negatives), `FPR` (False Positive Rate), `FNR` (False Negative Rate), `TPR` (True Positive Rate), `TNR` (True Negative Rate), `AUC` (Area Under the Curve). The second is the path to the created figure graphic.

Author(s)

Panagiotis Moulos

Examples

```
p1 <- 0.001*matrix(runif(300),100,3)
p2 <- matrix(runif(300),100,3)
p <- rbind(p1,p2)
rownames(p) <- paste("gene",1:200,sep="_")
colnames(p) <- paste("method",1:3,sep="_")
truth <- c(rep(1,40),rep(-1,40),rep(0,20),rep(1,10),
    rep(2,10),rep(0,80))
names(truth) <- rownames(p)
rocObj <- diagplotRoc(truth,p)
```

---

diagplotVenn *Venn diagrams when performing meta-analysis*

Description

This function uses the R package VennDiagram and plots an up to 5-way Venn diagram depicting the common and specific to each statistical algorithm genes, for each contrast. Mostly for internal use because of its main argument which is difficult to construct, but can be used independently if the user grasps the logic.

Usage

```
diagplotVenn(pmat, fcmat = NULL, pcut = 0.05,
    fcut = 0.5, direction = c("dereg", "up", "down"),
    nam = as.character(round(1000 * runif(1))),
    output = "x11", path = NULL, altNames = NULL, ...)
```

Arguments

`pmat`	a matrix with p-values corresponding to the application of each statistical algorithm. See also Details.
`fcmat`	an optional matrix with fold changes corresponding to the application of each statistical algorithm. See also Details.
`pcut`	if `fcmat` is supplied, an absolute fold change cutoff to be applied to `fcmat` to determine the differentially expressed genes for each algorithm.
`fcut`	a p-value cutoff for statistical significance. Defaults to `0.05`.
`direction`	if `fcmat` is supplied, a keyword to denote which genes to draw in the Venn diagrams with respect to their direction of regulation. See Details.
`nam`	a name to be appended to the output graphics file (if `"output"` is not `"x11"`).
`output`	one or more R plotting device to direct the plot result to. Supported mechanisms: `"x11"` (default), `"png"`, `"jpg"`, `"bmp"`, `"pdf"` or `"ps"`.

path	the path to create output files. If "path" is not NULL, a file with the intersections in the Venn diagrams will be produced and written in "path".
altNames	an optional named vector of names, e.g. HUGO gene symbols, alternative or complementary to the unique gene names which are the rownames of pmat. The names of the vector must be the rownames of pmat.
...	further arguments to be passed to plot devices, such as parameter from par.

Details

Regarding pmat, the p-value matrix must have the colnames attribute and the colnames should correspond to the name of the algorithm used to fill the specific column (e.g. if "statistics"=c("deseq","edger","nbpseq") then colnames(pmat) <- c("deseq","edger","nbpseq").

Regarding fcmat, the fold change matrix must have the colnames attribute and the colnames should correspond to the name of the algorithm used to fill the specific column (see the parameter pmat).

Regarding direction, it can be one of "dereg" for the total of regulated genes, where abs(fcmat[,n])>=fcut (default), "up" for the up-regulated genes where fcmat[,n]>=fcut or "down" for the up-regulated genes where fcmat[,n]<=-fcut.

Value

The filenames of the plots produced in a named list with names the which.plot argument. If output="x11", no output filenames are produced.

Author(s)

Panagiotis Moulos

Examples

```
require(VennDiagram)
p1 <- 0.01*matrix(runif(300),100,3)
p2 <- matrix(runif(300),100,3)
p <- rbind(p1,p2)
rownames(p) <- paste("gene",1:200,sep="_")
colnames(p) <- paste("method",1:3,sep="_")
vennContents <- diagplotVenn(p)
```

---

diagplotVolcano *(Interactive) volcano plots of differentially expressed genes*

Description

This function plots a volcano plot or returns a JSON string which is used to render aninteractive in case of HTML reporting.

Usage

```
diagplotVolcano(f, p, con = NULL, fcut = 1, pcut = 0.05,
    altNames = NULL, output = "x11", path = NULL, ...)
```

Arguments

f	the fold changes which are to be plotted on the x-axis.
p	the p-values whose -log10 transformation is going to be plotted on the y-axis.
con	an optional string depicting a name (e.g. the contrast name) to appear in the title of the volcano diagplot.
fcut	a fold change cutoff so as to draw two vertical lines indicating the cutoff threshold for biological significance.
pcut	a p-value cutoff so as to draw a horizontal line indicating the cutoff threshold for statistical significance.
altNames	an optional vector of names, e.g. HUGO gene symbols, alternative or complementary to the unique names of f or p (one of them must be named!). It is used only in JSON output.
output	one or more R plotting device to direct the plot result to. Supported mechanisms: "x11" (default), "png", "jpg", "bmp", "pdf", "ps" or "json". The latter is currently available for the creation of interactive volcano plots only when reporting the output, through the highcharts javascript library.
path	the path to create output files.
...	further arguments to be passed to plot devices, such as parameter from par.

Value

The filenames of the plots produced in a named list with names the which.plot argument. If output="x11", no output filenames are produced.

Author(s)

Panagiotis Moulos

Examples

```
dataMatrix <- metaseqR2:::exampleCountData(5000)
sampleList <- list(A=c("A1","A2"),B=c("B1","B2","B3"))
contrast <- "A_vs_B"
M <- normalizeEdger(dataMatrix,sampleList)
p <- statEdger(M,sampleList,contrast)
ma <- apply(M[,sampleList$A],1,mean)
mb <- apply(M[,sampleList$B],1,mean)
f <- log2(ifelse(mb==0,1,mb)/ifelse(ma==0,1,ma))
diagplotVolcano(f,p[[1]],con=contrast)
#j <- diagplotVolcano(f,p[[1]],con=contrast,output="json")
```

downsampleCounts *Downsample read counts*

Description

This function downsamples the library sizes of a read counts table to the lowest library size, according to the methdology used in (Soneson and Delorenzi, BMC Bioinformatics, 2013).

Usage

```
downsampleCounts(counts)
```

Arguments

counts	the read counts table which is subjected to downsampling.

Details

The downsampling process involves random sampling. For guaranteed reproducibility, be sure to use `set.seed` before downsampling. By default, when the metaseqR2 package is loaded, the seed is set to `42`.

Value

The downsampled counts matrix.

Author(s)

Panagiotis Moulos

Examples

```
dataMatrix <- metaseqR2:::exampleCountData(5000)
D <- downsampleCounts(dataMatrix)
```

estimateAufcWeights *Estimate AUFC weights*

Description

This function automatically estimates weights for the `"weight"` and `"dperm_weight"` options of metaseqR2 for combining p-values from multiple statistical tests. It creates simulated dataset based on real data and then performs statistical analysis with metaseqR2 several times in order to derive False Discovery Curves. Then, the average areas under the false discovery curves are used to construct weights for each algorithm, according to its performance when using simulated data.

Usage

```
estimateAufcWeights(counts, normalization,
    statistics, nsim = 10, N = 10000,
    samples = c(3, 3), ndeg = c(500, 500),
    top = 500, modelOrg = "mm9", fcBasis = 1.5,
    drawFpc = FALSE, rc = NULL,
    ...)
```

Arguments

counts	the real raw counts table from which the simulation parameters will be estimated. It must not be normalized and must contain only integer counts, without any other annotation elements and unique gene identifiers as the rownames attribute.
normalization	same as normalization in metaseqr2.
statistics	same as statistics in metaseqr2.
nsim	the number of simulations to perform to estimate the weights. It default to 10.
N	the number of genes to produce. See makeSimDataSd.
samples	a vector with 2 integers, which are the number of samples for each condition (two conditions currently supported).
ndeg	a vector with 2 integers, which are the number of differentially expressed genes to be produced. The first element is the number of up-regulated genes while the second is the number of down-regulated genes.
fcBasis	the minimum fold-change for deregulation.
top	the top top best ranked (according to p-value) to use, to calculate area under the false discovery curve.
modelOrg	the organism from which the data are derived. It must be one of metaseqr2 supported organisms.
drawFpc	draw the averaged false discovery curves? Default to FALSE.
rc	the fraction of the available cores to use in a multicore system.
...	Further arguments to be passed to estimateSimParams.

Details

The weight estimation process involves a lot of random sampling. For guaranteed reproducibility, be sure to use set.seed prior to any calculations. By default, when the metaseqR2 package is loaded, the seed is set to 42.

Value

A vector of weights to be used in metaseqr2 with the weights option.

Author(s)

Panagiotis Moulos

Examples

```
require(zoo)
data("mm9GeneData",package="metaseqR2")
weights <- estimateAufcWeights(
    counts=as.matrix(mm9GeneCounts[sample(nrow(mm9GeneCounts),1000),9:12]),
    normalization="edaseq",
    statistics=c("edger","limma"),
    nsim=1,N=100,ndeg=c(10,10),top=10,modelOrg=NULL,
    rc=0.01,libsizeGt=1e+5
)
```

`estimateSimParams` *Estimate negative binomial parameters from real data*

Description

This function reads a read counts table containing real RNA-Seq data (preferebly with more than 20 samples so as to get as much accurate as possible estimations) and calculates a population of count means and dispersion parameters which can be used to simulate an RNA-Seq dataset with synthetic genes by drawing from a negative binomial distribution. This function works in the same way as described in (Soneson and Delorenzi, BMC Bioinformatics, 2013) and (Robles et al., BMC Genomics, 2012).

Usage

```
estimateSimParams(realCounts, libsizeGt = 3e+6,
    rowmeansGt = 5,eps = 1e-11, rc = NULL, draw = FALSE)
```

Arguments

`realCounts`	a text tab-delimited file with real RNA-Seq data. See Details.
`libsizeGt`	a library size below which samples are excluded from parameter estimation (default: `3000000`).
`rowmeansGt`	a row means (mean counts over samples for each gene) below which genes are excluded from parameter estimation (default: 5).
`eps`	the tolerance for the convergence of `optimize` function. Defaults to 1e-11.
`rc`	in case of parallel optimization, the fraction of the available cores to use.
`draw`	boolean to determine whether to plot the estimated simulation parameters (mean and dispersion) or not. Defaults to `FALSE` (do not draw a mean-dispersion scatterplot).

Details

Regarding `realCounts`, the file should strictly contain a unique gene name (e.g. Ensembl accession) in the first column and all other columns should contain read counts for each gene. Each column must be named with a unique sample identifier. See examples in the ReCount database http://bowtie-bio.sourceforge.net/recount/.

Also, the parameter estimation involves a lot of random sampling. For guaranteed reproducibility, be sure to use `set.seed` prior to any calculations. By default, when the metaseqR2 package is loaded, the seed is set to `42`.

Value

A named list with two members: `muHat` which contains negative binomial mean estimates and `phiHat` which contains dispersion estimates.

Author(s)

Panagiotis Moulos

Examples

```
dataMatrix <- metaseqR2:::exampleCountData(2000)
parList <- estimateSimParams(dataMatrix,libsizeGt=3e+4)
```

---

getAnnotation *Annotation downloader*

Description

For Ensembl based annotations, this function connects to the EBI's Biomart service using the package biomaRt and downloads annotation elements (gene co-ordinates, exon co-ordinates, gene identifications, biotypes etc.) for each of the supported organisms. For UCSC/RefSeq annotations, it connects to the respective SQL databases if the package `RMySQL` is present, otherwise it downloads flat files and build a temporary SQLite database to make the necessaru build queries. See the help page of `metaseqr2` for a list of supported organisms.

Usage

```
getAnnotation(org, type, refdb = "ensembl", ver = NULL,
    rc = NULL)
```

Arguments

`org`	the organism for which to download annotation (one of the supported ones).
`type`	`"gene"`, `"exon"` or `"utr"`. Same as the `countType` in `metaseqr2`.
`refdb`	the online source to use to fetch annotation. It can be `"ensembl"` (default), `"ucsc"` or `"refseq"`. In the later two cases, an SQL connection is opened with the UCSC public databases.

ver	the version of the annotation to use.
rc	Fraction of cores to use. Same as the rc in buildAnnotationDatabase.

Value

A data frame with the canonical (not isoforms!) genes or exons of the requested organism. When type="genes", the data frame has the following columns: chromosome, start, end, gene_id, gc_content, strand, gene_name, biotype. When type="exon" the data frame has the following columns: chromosome, start, end, exon_id, gene_id, strand, gene_name, biotype. When type="utr" the data frame has the following columns: chromosome, start, end, transcript_id, gene_id, strand, gene_name, biotype. The gene_id and exon_id correspond to Ensembl, UCSC or RefSeq gene, transcript and exon accessions respectively. The gene_name corresponds to HUGO nomenclature gene names.

Note

The data frame that is returned contains only "canonical" chromosomes for each organism. It does not contain haplotypes or random locations and does not contain chromosome M.

Author(s)

Panagiotis Moulos

Examples

```
mm10Genes <- getAnnotation("mm10","gene")
```

---

getDefaults *Default parameters for several metaseqr functions*

Description

This function returns a list with the default settings for each filtering, statistical and normalization algorithm included in the metaseqR package. See the documentation of the main function and the documentation of each statistical and normalization method for details.

Usage

```
getDefaults(what, method = NULL)
```

Arguments

what	a keyword determining the procedure for which to fetch the default settings according to method parameter. It can be one of "normalization", "statistics", "geneFilter", "exonFilter" or "biotypeFilter".
method	the supported algorithm included in metaseqR for which to fetch the default settings. Se Details.

Details

When `what` is `"normalization"`, method is one of `"edaseq"`, `"deseq"`, `"edger"`, `"noiseq"` or `"nbpseq"`. When `what` is `"statistics"`, method is one of `"deseq"`, `"edger"`, `"noiseq"`, `"limma"`, or `"nbpseq"`. When `method` is `"biotypeFilter"`, `what` is the input organism (see the main `metaseqr2` help page for a list of supported organisms).

Value

A list with default setting that can be used directly in the call of metaseqr.

Author(s)

Panagiotis Moulos

Examples

```
normArgsEdaseq <- getDefaults("normalization","edaseq")
statArgsEdger <- getDefaults("statistics","edger")
```

---

getInstalledAnnotations

Load a metaseqR2 annotation element

Description

This function returns a data frame with information on locally installed, supported or custom, annotations.

Usage

```
getInstalledAnnotations(obj = NULL)
```

Arguments

`obj`	`NULL` or the path to a metaseqR2 SQLite annotation database. If `NULL`, the function will try to guess the location of the SQLite database.

Value

The function returns a `data.frame` object with the installed local annotations.

Author(s)

Panagiotis Moulos

Examples

```
db <- file.path(system.file(package="metaseqR2"),
    "annotation.sqlite")
if (file.exists(db))
    ig <- getInstalledAnnotations(obj=db)
```

---

getWeights *Get precalculated statistical test weights*

Description

This function returns pre-calculated weights for human, chimpanzee, mouse, fruitfly and arabidopsis based on the performance of simulated datasets estimated from real data from the ReCount database (http://bowtie-bio.sourceforge.net/recount/). Currently pre-calculated weights are available only when all six statistical tests are used and for normalization with EDASeq. For other combinations, use the estimateAufcWeights function.

Usage

```
getWeights(org = c("human", "chimpanzee", "mouse",
    "fruitfly", "arabidopsis", "rat"))
```

Arguments

org	"human", "chimpanzee", "mouse", "fruitfly", "arabidopsis" or "rat".

Value

A named vector of convex weights.

Author(s)

Panagiotis Moulos

Examples

```
wh <- getWeights("human")
```

hg19pvalues *p-values from human RNA-Seq data with two conditions, four samples*

Description

This data set contains p-values calculated with each of the supported statistical testing algorithms in metaseqR2 for 1000 genes. The purpose of this matrix is to demonstrate the p-value combination methods as well as be used for a playground for other such methods and with other metaseqR2 facilities.

Format

a `matrix` with p-values from metaseqR2 supported tests.

Author(s)

Panagiotis Moulos

Source

Giakountis et al. (https://doi.org/10.1016/j.celrep.2016.05.038)

importCustomAnnotation *Import a metaseqR2 custom annotation element*

Description

This function creates a local annotation database to be used with metaseqr2 so as to avoid long time on the fly annotation downloads and formatting.

Usage

```
importCustomAnnotation(gtfFile, metadata,
    level = c("gene", "transcript", "exon"),
    type = c("gene", "exon", "utr"))
```

Arguments

`gtfFile`	a GTF file containing the gene structure of the organism to be imported.
`metadata`	a list with additional information about the annotation to be imported. The same as in the `buildCustomAnnotation` man page.
`level`	same as the `transLevel` in `metaseqr2`.
`type`	same as the `countType` in `metaseqr2`.

Value

The function returns a GenomicRanges object with the requested annotation.

Author(s)

Panagiotis Moulos

Examples

```
# Dummy GTF as example
chromInfo <- data.frame(length=c(1000L,2000L,1500L),
    row.names=c("A","B","C"))

# Build with the metadata list filled (you can also provide a version)
myGenes <- importCustomAnnotation(
    gtfFile=file.path(system.file(package="metaseqR2"),"dummy.gtf"),
    metadata=list(
        organism="dummy",
        source="dummy_db",
        version=1,
        chromInfo=chromInfo
    ),
    level="gene",type="gene"
)

## Real data!
## Gene annotation dump from Ensembl
#download.file(paste0("ftp://ftp.ensembl.org/pub/release-98/gtf/",
#  "dasypus_novemcinctus/Dasypus_novemcinctus.Dasnov3.0.98.gtf.gz"),
#  file.path(tempdir(),"Dasypus_novemcinctus.Dasnov3.0.98.gtf.gz"))

## Build with the metadata list filled (you can also provide a version)
#dasGenes <- importCustomAnnotation(
#  gtfFile=file.path(tempdir(),"Dasypus_novemcinctus.Dasnov3.0.98.gtf.gz"),
#  metadata=list(
#    organism="dasNov3_test",
#    source="ensembl_test"
#  ),
#  level="gene",type="gene"
#)
```

---

libsizeListMm9 *Mouse RNA-Seq data with two conditions, four samples*

Description

The library size list for mm9GeneCounts. See the data set description.

Format

a named `list` with library sizes.

Author(s)

Panagiotis Moulos

Source

ENCODE (http://genome.ucsc.edu/encode/)

---

loadAnnotation *Load a metaseqR2 annotation element*

Description

This function creates loads an annotation element from the local annotation database to be used with metaseqr2. If the annotation is not found and the organism is supported, the annotation is created on the fly but not imported in the local database. Use `buildAnnotationDatabase` for this purpose.

Usage

```
loadAnnotation(genome, refdb,
    level = c("gene", "transcript", "exon"),
    type = c("gene", "exon", "utr"), version="auto",
    db = file.path(system.file(package = "metaseqR2"),
        "annotation.sqlite"), summarized = FALSE,
        asdf = FALSE, rc = NULL)
```

Arguments

`genome`	a `metaseqr2` supported organisms or a custom, imported by the user, name. See also the main `metaseqr2` man page.
`refdb`	a `metaseqr2` supported annotation source or a custom, imported by the user, name. See also the main `metaseqr2` man page.
`level`	same as the `transLevel` in `metaseqr2`.
`type`	same as the `countType` in `metaseqr2`.
`version`	same as the `version` in `metaseqr2`.
`db`	same as the `db` in `buildAnnotationDatabase`.
`summarized`	if `TRUE`, retrieve summarized, non-overlaping elements where appropriate (e.g. exons).
`asdf`	return the result as a `data.frame` (default `FALSE`).
`rc`	same as the `rc` in `buildAnnotationDatabase`.

Value

The function returns a GenomicRanges object with the requested annotation.

Author(s)

Panagiotis Moulos

Examples

```
db <- file.path(system.file(package="metaseqR2"),
    "annotation.sqlite")
if (file.exists(db))
    gr <- loadAnnotation(genome="hg19",refdb="ensembl",
        level="gene",type="gene",db=db)
```

---

makeSimDataSd — *Create simulated counts using the Soneson-Delorenzi method*

Description

This function creates simulated RNA-Seq gene expression datasets using the method presented in (Soneson and Delorenzi, BMC Bioinformatics, 2013). For the time being, it creates only simulated datasets with two conditions.

Usage

```
makeSimDataSd(N, param, samples = c(5, 5),
    ndeg = rep(round(0.1*N), 2), fcBasis = 1.5,
    libsizeRange = c(0.7, 1.4), libsizeMag = 1e+7,
    modelOrg = NULL, simLengthBias = FALSE)
```

Arguments

N	the number of genes to produce.
param	a named list with negative binomial parameter sets to sample from. The first member is the mean parameter to sample from (muHat) and the second the dispersion (phiHat). This list can be created with the estimateSimParams function.
samples	a vector with 2 integers, which are the number of samples for each condition (two conditions currently supported).
ndeg	a vector with 2 integers, which are the number of differentially expressed genes to be produced. The first element is the number of up-regulated genes while the second is the number of down-regulated genes.
fcBasis	the minimum fold-change for deregulation.

`libsizeRange` a vector with 2 numbers (generally small, see the default), as they are multiplied with `libsizeMag`. These numbers control the library sized of the synthetic data to be produced.

`libsizeMag` a (big) number to multiply the `libsizeRange` to produce library sizes.

`modelOrg` the organism from which the real data are derived from. It must be one of the supported organisms (see the main `metaseqr2` help page). It is used to sample real values for GC content.

`simLengthBias` a boolean to instruct the simulator to create genes whose read counts is proportional to their length. This is achieved by sorting in increasing order the mean parameter of the negative binomial distribution (and the dispersion according to the mean) which will cause an increasing gene count length with the sampling. The sampled lengths are also sorted so that in the final gene list, shorter genes have less counts as compared to the longer ones. The default is FALSE.

Details

The simulated data generation involves a lot of random sampling. For guaranteed reproducibility, be sure to use `set.seed` prior to any calculations. By default, when the metaseqR2 package is loaded, the seed is set to 42.

Value

A named list with two members. The first member (`simdata`) contains the synthetic dataset

Author(s)

Panagiotis Moulos

Examples

```
dataMatrix <- metaseqR2:::exampleCountData(2000)
## File "bottomly_read_counts.txt" from the ReCount database
#download.file(paste("http://bowtie-bio.sourceforge.net/recount/",
#    "countTables/bottomly_count_table.txt",sep=""),
#    destfile="~/bottomly_count_table.txt")
N <- 2000
#parList <- estimateSimParams("~/bottomly_read_counts.txt")
parList <- estimateSimParams(dataMatrix,libsizeGt=3e+4)
sim <- makeSimDataSd(N,parList)
synthData <- sim$simdata
trueDeg <- which(sim$truedeg!=0)
```

makeSimDataTcc — *Create simulated counts using TCC package*

Description

This function creates simulated RNA-Seq gene expression datasets using the `simulateReadCounts` function from the Bioconductor package TCC and it adds simulated annoation elements. For further information please consult the TCC package documentation.

Usage

```
makeSimDataTcc(...)
```

Arguments

`...`	parameters to the `simulateReadCounts` function.

Value

A list with the following members: `simdata` holding the simulated dataset complying with metaseqr2 requirements, and `simparam` holding the simulation parameters (see TCC documentation). Note that the produced data are based in an Arabidopsis dataset.

Author(s)

Panagiotis Moulos

Examples

```
if (require(TCC)) {
dd <- makeSimDataTcc(Ngene=1000,PDEG=0.2,
    DEG.assign=c(0.9,0.1),
    DEG.foldchange=c(5,5),replicates=c(3,3))
head(dd$simdata)
}
```

metaseqr2 — *The main metaseqr2 pipeline*

Description

This function is the main metaseqr2 workhorse and implements the main metaseqr2 workflow which performs data read, filtering, normalization and statistical selection, creates diagnostic plots and exports the results and a report if requested. The metaseqr2 function is responsible for assembling all the steps of the metaseqr2 pipeline which i) reads the input gene or exon read count table ii) performs prelimininary filtering of data by removing chrM and other non-essential information for a typical differential gene expression analysis as well as a preliminary expression filtering based on the exon counts, if an exon read count file is provided. iii) performs data normalization with one of currently widely used algorithms, including EDASeq (Risso et al., 2011), DESeq (Anders and Huber, 2010), edgeR (Robinson et al., 2010), NOISeq (Tarazona et al., 2012) or no normalization iv) performs a second stage of filtering based on the normalized gene expression according to several gene filters v) performs statistical testing with one or more of currently widely used algorithms, including DESeq (Anders and Huber, 2010), edgeR (Robinson et al., 2010), NOISeq (Tarazona et al., 2012), limma (Smyth et al., 2005) for RNA-Seq data vi) in the case of multiple statistical testing algorithms, performs meta-analysis using one of six vailable methods (see the meta.p argument) vii) exports the resulting differentially expressed gene list in text tab-delimited format viii) creates a set of diagnostic plots either available in the aforementioned packages or metaseqr2 specific ones and ix) creates a comprehensive HTML report which summarizes the run information, the results and the diagnostic plots. Certain diagnostic plots (e.g. the volcano plot) can be interactive with the use of the external Highcharts (http://www.highcharts.com) JavaScript library for interactive graphs. Although the inputs to the metaseqr2 workflow are many, in practice, setting only very few of them and accepting the defaults as the rest can result in quite comprehensible results for mainstream organisms like mouse, human, fly and rat.

Usage

```
metaseqr2(counts, sampleList, excludeList = NULL,
    fileType = c("auto", "sam", "bam", "bed"),
    path = NULL, contrast = NULL, libsizeList = NULL,
    embedCols = list(idCol = 4, gcCol = NA, nameCol = NA,
        btCol = NA),
    annotation = NULL, org = c("hg18", "hg19", "hg38", "mm9",
        "mm10", "rn5", "rn6", "dm3", "dm6", "danrer7",
        "pantro4", "susscr3", "tair10", "equcab2"),
    refdb = c("ensembl", "ucsc", "refseq"), version = "auto",
    transLevel = c("gene", "transcript", "exon"),
    countType = c("gene", "exon","utr"),
    utrOpts = list(frac = 1, minLength = 300, downstream = 50),
    exonFilters = list(minActiveExons = list(exonsPerGene = 5,
        minExons = 2, frac = 1/5)),
    geneFilters = list(length = list(length = 500),
        avgReads = list(averagePerBp = 100, quantile = 0.25),
        expression = list(median = TRUE, mean = FALSE,
            quantile = NA, known = NA, custom = NA),
        biotype = getDefaults("biotypeFilter", org[1]),
        presence = list(frac = 0.25, minCount = 10,
            perCondition = FALSE)),
    whenApplyFilter = c("postnorm", "prenorm"),
```

```
normalization = c("deseq", "deseq2", "edaseq", "edger",
    "noiseq", "nbpseq", "absseq", "dss", "each", "none"),
normArgs = NULL,
statistics = c("deseq", "deseq2", "edger", "noiseq",
    "limma", "nbpseq", "absseq", "dss"),
statArgs = NULL,
adjustMethod = sort(c(p.adjust.methods, "qvalue")),
metaP = if (length(statistics) > 1) c("simes",
    "bonferroni", "fisher", "dperm_min", "dperm_max",
    "dperm_weight", "fperm", "whitlock", "minp", "maxp",
    "weight", "pandora", "none") else "none",
weight = rep(1/length(statistics), length(statistics)),
nperm = 10000, pcut = NA, logOffset = 1, pOffset = NULL,
preset = NULL, qcPlots = c("mds", "biodetection",
    "countsbio", "saturation", "readnoise","filtered",
    "correl", "pairwise", "boxplot", "gcbias",
    "lengthbias", "meandiff", "meanvar", "rnacomp",
    "deheatmap", "volcano", "biodist", "mastat",
    "statvenn", "foldvenn", "deregulogram"),
figFormat = c("png", "jpg", "tiff", "bmp", "pdf", "ps"),
outList = FALSE, exportWhere = NA,
exportWhat = c("annotation", "p_value", "adj_p_value",
    "meta_p_value", "adj_meta_p_value", "fold_change",
    "stats", "counts","flags"),
exportScale = c("natural", "log2", "log10", "vst",
    "rpgm"),
exportValues = c("raw", "normalized"),
exportStats = c("mean", "median", "sd", "mad", "cv",
    "rcv"),
exportCountsTable = FALSE,
restrictCores = 0.6, report = TRUE, reportTop = 0.1,
reportTemplate = "default", saveGeneModel = TRUE,
verbose = TRUE, runLog = TRUE,
reportDb = c("dexie", "sqlite"),
localDb = file.path(system.file(package = "metaseqR2"),
    "annotation.sqlite"),
offlineReport = TRUE,
createTracks = FALSE, overwriteTracks = FALSE,
trackExportPath = file.path(exportWhere, "tracks"),
trackInfo = list(stranded = FALSE, normTo = 1e+9,
    urlBase = "http://www.trackserver.me",
    fasta = NULL, gtf = NULL,
    hubInfo = list(name = "MyHub", shortLabel = "My hub",
    longLabel = "My hub long",
    email = "someone@example.com")), .progressFun = NULL,
    .exportR2C = FALSE,...)
```

Arguments

`counts` a text tab-delimited file containing gene, exon or 3'UTR counts in one of the following formats: i) the first column contains unique gene or exon identifiers and the rest of the columns contain the read counts for each sample. ii) The first n columns should contain only **gene** annotation elements like chromosomal locations, gene accessions, exon accessions, GC content etc. and the rest columns should contain gene read counts, iii) `counts` can also be an `.RData` file with previous analysis elements (see Details) and iv) counts can be a list representing the gene model (see Details). Several restrictions apply im each of the four cases. See Details for analytical descriptions.

`sampleList` a list containing condition names and the samples under each condition or a small tab-delimited file with the experiment description. Not needed when restoring a previous analysis. See Details for analytical description.

`excludeList` a list of samples to exclude, in the same (list) format as `sampleList` above.

`path` an optional path where all the BED/BAM files are placed, to be prepended to the BAM/BED file names in the targets file. See Details for further information.

`fileType` the type of raw input files. It can be `"auto"` for auto-guessing, `"bed"` for BED files, `"sam"` for SAM files or `"bam"` for BAM files.

`contrast` a character vector of contrasts to be tested in the statistical testing step(s) of the pipeline. Each element of contrast should STRICTLY have the format "ConditionA_vs_ConditionB_vs_...". Special attention is needed as fold change calculations are based on this argument. If it is `NULL`, no statistical testing of fold change calculations are performed. See Details for further information.

`libsizeList` an optional named list where names represent samples (MUST be the same as the samples in `sample.list`) and members are the library sizes (the sequencing depth) for each sample. For example `libsize.list <- list(Sample_A1=32456913, Sample_A2=4346818)`.

`embedCols` a named list with column numbers to guide the case of embedded annotation. See Details for further information.

`annotation` It can be one of i) `NULL` (default) to use the existing annotation database or fetch on the fly, ii) `"embedded"` if the annotation elements are embedded in the read counts file (restrictions apply) or iii) a list with a path to a GTF file and certain required metadata. See Details for a full description.

`org` the supported organisms by `metaseqr2` or a user-named organism which has been imported to the database. See Details for more information.

`refdb` the reference annotation repository from which to retrieve annotation elements to use with metaseqr2. It can be one of `"ensembl"` (default), `"ucsc"` or `"refseq"` or a user based one (similar to the `org` argument).

`version` the version of the annotation to use. See Details.

`transLevel` perform differential expression analysis at which transcriptional unit, can be one of `"gene"` (default), `"transcript"` for reporting differential expression at the transcript level or `"exon"` for exon level.

`countType` the type of reads inside the counts file. It can be one of `"gene"`, `"exon"` or `"utr"` for Quant-Seq (Lexogen) protocol. This is a very important and mandatory parameter as it defines the course of the workflow.

utrOpts — a named list with members frac which is the fraction (0-1) of the 3' UTR region to count reads in, minLength the minimum acceptable 3'UTR length irrespective of frac and downstream the number of base pairs to flank the end of the 3' UTR of transcripts when analyzing Quant-Seq data.

exonFilters — a named list whose names are the names of the supported exon filters and its members the filter parameters. See section "Exon filters" below for details.

geneFilters — a named list whose names are the names of the supported gene filters and its members the filter parameters. See section "Gene filters" below for details.

whenApplyFilter — a character string determining when to apply the exon and/or gene filters, relative to normalization. It can be "prenorm" to apply apply the filters and exclude genes from further processing before normalization, or "postnorm" to apply the filters after normalization (default). See also Details.

normalization — the normalization algorithm to be applied on the count data. It can be one of "edaseq" for EDASeq normalization, "deseq" for the normalization algorithm in the DESq package (default), "edger" for the normalization algorithms present in the edgeR package "noiseq" for the normalization algorithms present in the NOISeq package "nbpseq" for the normalization algorithms present in the NBPSeq package or "none" to not normalize the data (highly unrecommended). Algorithm specific arguments can be passed through the normArgs argument).

normArgs — a named list whose names are the names of the normalization algorithm parameters and its members parameter values. See section "Normalization parameters" below for details. Leave NULL for the defaults of normalization.

statistics — one or more statistical analyses to be performed by the metaseqr2 pipeline. It can be one or more of "deseq" (default) to conduct statistical test(s) implemented in the DESeq package, "edger" to conduct statistical test(s) implemented in the edgeR package, "limma" to conduct the RNA-Seq version of statistical test(s) implemented in the limma package, "noiseq" to conduct statistical test(s) implemented in the NOISeq package, "nbpseq" to conduct statistical test(s) implemented in the NBPSeq package, "deseq2" to conduct statistical test(s) implemented in the DESeq2 package, "dss" to conduct statistical test(s) implemented in the DSS package and "absseq" to conduct statistical test(s) implemented in the ABSSeq package. In any case individual algorithm parameters are controlled by the contents of the statArgs list. Finally, it can be NA. In this case no testing is performed and only fold changes are provided if contrast is not NULL.

statArgs — a named list whose names are the names of the statistical algorithms used in the pipeline. Each member is another named list whose names are the algorithm parameters and its members are the parameter values. See section "Statistics parameters" below for details. Leave NULL for the defaults of statistics.

adjustMethod — the multiple testing p-value adjustment method. It can be one of p.adjust.methods or "qvalue" from the qvalue Bioconductor package. Defaults to "BH" for Benjamini-Hochberg correction.

metaP — the meta-analysis method to combine p-values from multiple statistical tests . It can be one of "simes" (default), "bonferroni", "minp", "maxp", "weight", "pandora", "dperm_min", "dperm_max", "dperm_weight", "fisher", "fperm", "whitlock" or "none". See Details for a full description.

`weight` a vector of weights with the same length as the `statistics` vector containing a weight for each statistical test. It should sum to 1. **Use with caution with the** `dperm_weight` **parameter! Theoretical background is not yet solid and only experience shows improved results!**

`nperm` the number of permutations performed to derive the meta p-value when `metaP="fperm"` or `metaP="dperm"`. It defaults to 10000.

`pcut` a p-value cutoff for exporting differentially genes, default is to export all the non-filtered genes.

`logOffset` an offset to be added to values during logarithmic transformations in order to avoid Infinity (default is `1`).

`pOffset` a value between `0` and `1` to multiply potential zero p-values with for the combination methods including weighting or `NULL` (default). See also Details.

`preset` an analysis strictness preset. `preset` can be one of `"all_basic"`, `"all_normal"`, `"all_full"`, `"medium_basic"`, `"medium_normal"`, `"medium_full"`, `"strict_basic"`, `"strict_normal"` or `"strict_full"`, each of which control the strictness of the analysis and the amount of data to be exported. For an explanation of the presets, see the section "Presets" below.

`qcPlots` a set of diagnostic plots to show/create. It can be one or more of `"mds"`, `"biodetection"`, `"rnacomp"`, `"countsbio"`, `"saturation"`, `"readnoise"`, `"filtered"`, `"boxplot"`, `"gcbias"`, `"lengthbias"`, `"meandiff"`, `"meanvar"`, `"deheatmap"`, `"volcano"`, `"mastat"`, `"biodist"`, `"statvenn"`, `"foldvenn"`. See also Details.

`figFormat` the format of the output diagnostic plots. It can be one or more of `"png"`, `"jpg"`, `"tiff"`, `"bmp"`, `"pdf"`, `"ps"`. The native format `"x11"` (for direct display) is not provided as an option as it may not render the proper display of some diagnostic plots in some devices.

`outList` a logical controlling whether to export a list with the results in the running environment.

`exportWhere` an output directory for the project results (report, lists, diagnostic plots etc.)

`exportWhat` the content of the final lists. It can be one or more of `"annotation"`, to bind the annoation elements for each gene, `"p_value"`, to bind the p-values of each method, `"adj_p_value"`, to bind the multiple testing adjusted p-values, `"meta_p_value"`, to bind the combined p-value from the meta-analysis, `"adj_meta_p_value"`, to bind the corrected combined p-value from the meta-analysis, `"fold_change"`, to bind the fold changes of each requested contrast, `"stats"`, to bind several statistics calclulated on raw and normalized counts (see the `exportStats` argument), `"counts"`, to bind the raw and normalized counts for each sample.

`exportScale` export values from one or more transformations applied to the data. It can be one or more of `"natural"`, `"log2"`, `"log10"`, `"vst"` (Variance Stabilizing Transormation, see the documentation of DESeq package) and `"rpgm"` which is ratio of mapped reads per gene model (either the gene length or the sum of exon lengths, depending on `countType` argument). Note that this is not RPKM as reads are already normalized for library size using one of the supported normalization methods. Also, `"rpgm"` might be misleading when `normalization` is other than `"deseq"`.

`exportValues` It can be one or more of `"raw"` to export raw values (counts etc.) and `"normalized"` to export normalized counts.

`exportStats` calculate and export several statistics on raw and normalized counts, condition-wise. It can be one or more of `"mean"`, `"median"`, `"sd"`, `"mad"`, `"cv"` for the Coefficient of Variation, `"rcv"` for a robust version of CV where the median and the MAD are used instead of the mean and the standard deviation.

`exportCountsTable` exports also the calculated read counts table when input is read from bam files and exports also the normalized count table in all cases. Defaults to `FALSE`.

`restrictCores` in case of parallel execution of several subfunctions, the fraction of the available cores to use. In some cases if all available cores are used (`restrictCores=1` and the system does not have sufficient RAM, the pipeline running machine might significantly slow down.

`report` a logical value controlling whether to produce a summary report or not. Defaults to `TRUE`.

`reportTop` a fraction of top statistically significant genes to append to the HTML report. This helps in keeping the size of the report as small as possible, as appending the total gene list might create a huge HTML file. Users can always retrieve the whole gene lists from the report links. Defaults to `0.1` (top 10 genes). Set to `NA` or `NULL` to append all the statistically significant genes to the HTML report.

`reportTemplate` an HTML template to use for the report. Do not change this unless you know what you are doing.

`saveGeneModel` in case of exon analysis, a list with exon counts for each gene will be saved to the file `exportWhere/data/gene_model.RData`. This file can be used as input to metaseqR for exon count based analysis, in order to avoid the time consuming step of assembling the counts for each gene from its exons

`verbose` print informative messages during execution? Defaults to `TRUE`.

`runLog` write a log file of the `metaseqr2` run using package log4r. Defaults to `TRUE`. The filename will be auto-generated.

`reportDb` database system to use for storing the report intereactive graphs. Can be `"sqlite"` (default) or `"dexie"`. See Details for further explanation on what should be used.

`localDb` the metaseqR2 annotaation database location. See also `link{buildAnnotationDatabase}`.

`offlineReport` `TRUE` (default) to download and include the required JavaScript libraries to properly view the report offline. Ignored if `report=FALSE`

`.`

`createTracks` option to create normalized bigWig files to display in a genome browser (e.g. UCSC). Defaults to `FALSE`.

`overwriteTracks` overwrite tracks if they already exist? Defaults to `FALSE`.

`trackExportPath` where to export the bigWig files, defaults to `file.path(exportWhere,"tracks")`.

`trackInfo` if `createTracks=TRUE`, a list with additional required information to create the tracks. See Details for further explanation.

`.progressFun` a function which updates a `Progress` object from shiny. This function must accept a `detail` argument. See http://shiny.rstudio.com/articles/progress.html

`.exportR2C`	export additional RData along with `saveGeneModel`.
`...`	further arguments that may be passed to plotting functions, related to `par`.

Details

When `counts` is a tab-delimited file, the following restrictions apply:

- In the case (i) the first cell of each row is a gene or exon accession and the rest are integers representing the counts for that accession. In that case, the `annotation` parameter should strictly be `NULL` or an external file in GTF format.
- In the case (ii) the `annotation` parameter can also be `"embedded"`. The ideal embedded annotation contains 8 columns, chromosome, gene or exon start, gene or exon end, gene or exon accession, GC-content (fraction or percentage), strand, HUGO gene symbol and gene biotype (e.g. "protein_coding" or "ncRNA"). When the `annotation` parameter is `"embedded"`, certain of these features are mandatory (co-ordinates and accessions). If they are not present, the pipeline will not run. If additional elements are not present (e.g. GC content or biotypes), certain features of metaseqr2 will not be available. For example, EDASeq normalization will not be performed based on a GC content covariate but based on gene length which is not what the authors of EDASeq suggest. If biotypes are not present, a lot of diagnostic plots will not be available. If the HUGO gene symbols are missing, the final annotation will contain only gene accessions and thus be less comprehensible. Counts can be a data frame satisfying the above conditions. It is a data frame by default when `read2count` is used.
- In the case (iii) the .RData file (output of `save` function contains static input elements (list containing the gene model (exon counts for each gene), gene and exon annotation to avoid re-(down)loading and/or gene counts depending on `countType`). This kind of input facilitates the re-analysis of the same experiment, using different filtering, normalization and statistical algorithms. This `.RData` file is produced when `saveGeneModel=TRUE`.
- In the case (iv) `counts` can be a list representing the gene model (exon/UTR counts for each gene). This `.RData` file can be generated by setting `saveGeneModel=TRUE` when performing data analysis for the first time.

Regarding `sampleList` it should have the format `sampleList <- list(ConditionA=c("Sample_A1", "Sample_A2","Sample_A3"), ConditionB=c("Sample_B1","Sample_B2"), ConditionC=c("Sample_C1","Sample_C2`
The names of the samples in list members MUST match the column names containing the read counts in the counts file. If they do not match, the pipeline will either crash or at best, ignore several of your samples. Alternative, `sampleList` can be a small tab-delimited file structured as follows: the first line of the external tab delimited file should contain column names (names are not important). The first column MUST contain UNIQUE sample names and the second column MUST contain the biological condition where each of the samples in the first column should belong to. If the `counts` argument is missing, the `sampleList` argument MUST be a targets text tab-delimited file which contains the sample names, the BAM/BED file names and the biological conditions/groups for each sample/file. The file should be text tab-delimited and structured as follows: the first line of the external tab delimited file should contain column names (names are not important). The first column MUST contain UNIQUE sample names. The second column MUST contain the raw BAM/BED files WITH their full path. Alternatively, the `path` argument should be provided. If `path` is not provided and if the files in the second column of the targets file do not contain a path to a directory, the current directory is assumed to be the BAM/BED file container. The third column MUST contain the biological condition where each of the samples in the first column should belong to.

Regarding contrast, a valid example based on the sampleList above is contrast <- c("ConditionA_vs_ConditionB", "ConditionA_vs_ConditionC", "ConditionA_vs_ConditionB_vs_ConditionC"). The first element of pairwise contrasts (e.g. "ConditionA" above) MUST be the control condition or any reference that ConditionB is checked against. metaseqr2 uses this convention to properly calculate fold changes.

Regarding embedCols, this list must contain four members, named idCol, gcCol, nameCol and btCol, which hold the position in the delimited file with embedded annotation, where unique gene ids, GC content, gene names and gene biotypes respectively are located. More specifically:

- idCol is an integer denoting the column number in the file (or data frame) provided with the counts argument, where the unique gene accessions are. Default to 4 which is the standard feature name column in a BED file.
- gcCol is an integer denoting the column number in the file (or data frame) provided with the counts argument, where each gene's GC content is given. If not provided, GC content normalization provided by EDASeq will not be available.
- nameCol is an integer denoting the column number in the file (or data frame) provided with the counts argument, where the HUGO gene symbols are given. If not provided, it will not be available when reporting results. In addition, the "known" gene filter will not be available for application.
- btCol is an integer denoting the column number in the file (or data frame) provided with the counts argument, where the gene biotypes are given. If not provided, the "biodetection", "countsbio", "saturation", "filtered" and "biodist" plots will not be available.

Regarding annotation instructs metaseqr2 where to find the annotation for the given counts file. It can be one of i) "download" (default) for automatic downloading of the annotation for the organism specified by the org parameter (using biomaRt), ii) "embedded" if the annotation elements are embedded in the read counts file or iv) a file specified by the user which should be as similar as possible to the "download" case, in terms of column structure.

Regarding org, it can be, for human genomes "hg18", "hg19" or "hg38", for mouse genomes "mm9", "mm10", for rat genomes "rn5" or "rn6", for drosophila genome "dm3" or "dm6", for zebrafish genome "danrer7", "danrer10" or "danrer11", for chimpanzee genome "pantro4", "pantro5", for pig genome "susscr3", "susscr11", for Arabidopsis thaliana genome "tair10" and for Equus caballus genome "equcab2". Finally, it can be "USER_NAMED_ORG" with a custom organism which has been imported to the annotation database by the user using a GTF file. For example org="mm10_p1".

Regarding version, this is an integer denoting the version of the annotation to use from the local annotation database or fetch on the fly. For Ensembl, it corresponds to Ensembl releases, while for UCSC/RefSeq, it is the date of creation (locally).

Regarding whenApplyFilter, in the case of whenApplyFilter="prenorm", a first normalization round is applied to a copy of the gene counts matrix in order to derive the proper normalized values that will constitute the several expression-based filtering cutoffs.

Regarding metaP, for the "fisher" and "fperm" methods, see the documentation of the R package MADAM. For the "whitlock" method, see the documentation of the survcomp Bioconductor package. With the "maxp" option, the final p-value is the maximum p-value out of those returned by each statistical test. This is equivalent to an "intersection" of the results derived from each algorithm so as to have a final list with the common genes returned by all statistical tests. Similarly, when meta.p="minp", is equivalent to a "union" of the results derived from each algorithm so as

to have a final list with all the genes returned by all statistical tests. The latter can be used as a very lose statistical threshold to aggregate results from all methods regardless of their False Positive Rate. With the "simes" option, the method proposed by Simes (Simes, R. J., 1986) is used. With the "dperm_min", "dperm.max", "dperm.weight" options, a permutation procedure is initialed, where nperm permutations are performed across the samples of the normalized counts matrix, producing nperm permuted instances of the initital dataset. Then, all the chosen statistical tests are re-executed for each permutation. The final p-value is the number of times that the p-value of the permuted datasets is smaller than the original dataset. The p-value of the original dataset is created based on the choice of one of dperm.min, dperm.max or dperm.weight options. In case of dperm.min, the intial p-value vector is consists of the minimum p-value resulted from the applied statistical tests for each gene. The maximum p-value is used with the dperm.max option. With the dperm.weight option, the weight weighting vector for each statistical test is used to weight each p-value according to the power of statistical tests (some might work better for a specific dataset). Be careful as the permutation procedure usually requires a lot of time. However, it should be the most accurate. This method will NOT work when there are no replicated samples across biological conditions. In that case, use meta.p="simes" instead. Finally, there are the "minp", "maxp" and "weight" options which correspond to the latter three methods but without permutations. Generally, permutations would be accurate to use when the experiment includes >5 samples per condition (or even better 7-10) which is rather rare in RNA-Seq experiments. Finally, "pandora" is the same as "weight" and is added to be in accordance with the main algorithm name.

Regarding pOffset, it is used to correct for the case of a p-value which is equal to 0 as a result of internal numerical and approximation procedures. When NULL, random numbers greater than 0 and less than or equal to 0.5 are used to multiply the offending p-values with the lowest provided non-zero p-value, maintaining thus a virtual order of significance, avoiding having the same p-values for two tests and assuming that all zero p-values represent extreme statistical significance. When a numeric between 0 and 1, this number is used for the above multiplication instead.

Regarding qcPlots The "mds" stands for Mutlti-Dimensional Scaling and it creates a PCA-like plot but using the MDS dimensionality reduction instead. It has been succesfully used for NGS data (e.g. see the package htSeqTools) and it shows how well samples from the same condition cluster together. For "biodetection", "countsbio", "saturation", "rnacomp", "readnoise", "biodist" see the vignette of NOISeq package. The "saturation" case has been rewritten in order to display more samples in a more simple way. In addition, the "readnoise" plots represent an older version or the RNA composition plot included in older versions of NOISeq. For "gcbias", "lengthbias", "meandiff", "meanvar" see the vignette of EDASeq package. "lenghtbias" is similar to "gcbias" but using the gene length instead of the GC content as covariate. The "boxplot" option draws boxplots of log2 transformed gene counts. The "filtered" option draws a 4-panel figure with the filtered genes per chromosome and per biotype, as absolute numbers and as fractions of the genome. See also the help page of diagplotFiltered. The "deheatmap" option performs hierarchical clustering and draws a heatmap of differentially expressed genes. In the context of diagnostic plots, it's useful to see if samples from the same groups cluster together after statistical testing. The "volcano" option draws a volcano plot for each contrast and if a report is requested, an interactive volcano plot is presented in the HTML report. The "venn" option will draw an up to 5-way Venn diagram depicting the common and specific to each statistical algorithm genes and for each contrast, when meta-analysis is performed. The "correl" option creates two correlation graphs: the first one is a correlation heatmap (a correlation matrix which depicts all the pairwise correlations between each pair of samples in the counts matrix is drawn as a clustered heatmap) and the second one is a correlogram plot, which summarizes the correlation matrix in the form of ellipses (for an explanation please see the vignette/documentation of the R package

corrplot. Set `qcPlots=NULL` if you don't want any diagnostic plots created.

Regarding `reportDb`, contrary with the first version of metaseqR, all graphs in the metaseqR2 report are interactive with the usage of the JavaScript libraries Highcharts, plotly (heatmaply) and jvenn.js. However, this adds a great burden regarding rendering the final HTML file and its content, a burden which becomes heavier by the fact the metaseqR2 report is rendered using `knitr` and `rmarkdown` instead of raw HTML (previously, `brew`). Therefore, the pre-calculated JSON objects representing the graphs are stored either in a report-specific IndexedDB (https://javascript.info/indexeddb) flavor called Dexie (https://dexie.org/) (default) or in an SQLite database and then queried using `sql.js` (https://github.com/kripken/sql.js/). Dexie is prefered because it is very efficient and can produce an independent report that does not need to be served through a web-server and can be viewed locally. Although Dexie is very efficient, some caution is required as `knitr` and `render` from `rmarkdown` are not very memory efficient when rendering larger HTML files. A large HTML file may be produced when analyzing a large dataset with a lot of contrasts that may result in a lot of tables. In such cases, if the report generation crashes with errors related to memory, try lowering the `reportTop` argument. `reportTop` does not affect the final lists of differentially expressed genes, only the report tables. The same must be applied also if the report takes too much time to load. If the report is to be served through a web server like Apache (e.g. when the report is provided by a facility to end users), `reportDb="sqlite"` may be preferred as the total report size will be smaller because of an SQLite database hosting all plots which are queried when required but from the SQLite database and not from the in-browser database (Dexie). **Note** that when using an SQLite database, you will **NOT** be able to view the report in any browser other than Microsoft Edge because of security policies regarding local file access. **Note** also that `sql.js` is a rather large JavaScript library (around 2.5MB).

Regarding `trackInfo`, it is a helper list to guide the bigWig track creation and has the following members:

- `stranded`, which can be `TRUE` or `FALSE` depending on whether you wish to create stranded tracks by separating + and - strand reads. In the case of stranded tracks, a UCSC Genome Brower trackhub is created. Individual tracks can be retrieved from the trackhub.
- `normTo`, which is a large integer, denoting the total sum of signal to be used as the normalization target. It defaults to `1e+9`. This means that if for a particular sample the sum of signal is 1.5e+9 (`sum(sapply(coverage(x),sum)) == 1.5e+9`) then this is linearly scaled to 1e+9.
- `urlBase`, which is a base url appended to the bigWig files produced (the base path of the `bigDataUrl` in UCSC Genome Browser track lines).
- `hubInfo`, a list with the track hub description in case of stranded tracks. Please see the track hub specifications at the UCSC Genome Browser site.
- `fasta`, reference genome in FASTA format for the case of analyzing a custom, non-directly supported organism. It will be converted to the .2bit format and written along with a track hub.
- `gtf`, a GTF file describing gene models in the case of analyzing a custom, non-directly supported organism. It will be converted to the .bigBed format and written along with a track hub. Essentially the same as `annotation$gtf`.

All files (bigWig files, track/trackhub info) are written in the `tracks` subdirectory of the main path where the report and the outputs are written.

Value

If `outList` is `TRUE`, a named list whose length is the same as the number of requested contrasts. Each list member is named according to the corresponding contrast and contains a data frame of differentially expressed genes for that contrast. The contents of the data frame are defined by the `exportWhat, exportScale, exportStats, exportValues` parameters. If `report` is `TRUE`, the output list contains two main elements. The first is described above (the analysis results) and the second contains the same results but in HTML formatted tables.

Exon filters

The exon filters are a set of filters which are applied after the gene models are assembled from the read counts of individual exons and before the gene expression is summarized from the exons belonging to each gene. These filters can be applied when the input read counts file contains exon reads. It is not applicable when the input file already contains gene counts. Such filters can be for example "accept genes where all the exons contain more than x reads" or "accept genes where there is read presence in at least m/n exons, n being the total exons of the gene". Such filters are NOT meant for detecting differential splicing as also the whole metaseqr2 pipeline, thus they should not be used in that context. The `exonFilters` argument is a named list of filters, where the names are the filter names and the members are the filter parameters (named lists with parameter name, parameter value). See the usage of the `metaseqr2` function for an example of how these lists are structured. The supported exon filters in the current version are: i) `minActiveExons` which implements a filter for demanding m out of n exons of a gene to have a certain read presence with parameters `exonsPerGene`, `minExons` and `frac`. The filter is described as follows: if a gene has up to `exonsPerGene` exons, then read presence is required in at least `minExons` of them, else read presence is required in a `frac` fraction of the total exons. With the default values, the filter instructs that if a gene has up to 5 exons, read presence is required in at least 2, else in at least 20 exons, in order to be accepted. More filters will be implemented in future versions and users are encouraged to propose exon filter ideas to the author by mail. See `metaseqr2` usage for the defaults. Set `exonFilters=NULL` to not apply any exon filtering.

Gene filters

The gene filters are a set of filters applied to gene expression as this is manifested through the read presence on each gene and are preferably applied after normalization. These filters can be applied both when the input file or data frame contains exon read counts and gene read counts. Such filter can be for example "accept all genes above a certain count threshold" or "accept all genes with expression above the median of the normalized counts distribution" or "accept all with length above a certain threshold in kb" or "exclude the 'pseudogene' biotype from further analysis". The supported gene filters in the current version, which have the same structure as the exon filters (named list of lists with filter names, parameter names and parameter arguments) are: i) `length` which implements a length filter where genes are accepted for further analysis if they are above `length` (its parameter) kb. ii) `avg.reads` which implements a filter where a gene is accepted for further analysis if it has more average reads than the `quantile` of the average count distribution per `averagePerBp` base pairs. In summary, the reads of each gene are averaged per `averagePerBp` based on each gene's length (in case of exons, input the "gene's length" is the sum of the lengths of exons) and the `quantile` quantile of the average counts distribution is calculated for each sample. Genes passing the filter should have an average read count larger than the maximum of the vector of the quantiles calculated above. iii) `expression` which implements a filter based on the

overall expression of a gene. The parameters of this filter are: median, where genes below the median of the overall count distribution are not accepted for further analysis (this filter has been used to distinguish between "expressed" and "not expressed" genes in several cases, e.g. (Mokry et al., NAR, 2011) with a logical as value, mean which is the same as median but using the mean, quantile which is the same as the previous two but using a specific quantile of the total counts distribution, known, where in this case, a set of known not-expressed genes in the system under investigation are used to estimate an expression cutoff. This can be quite useful, as the genes are filtered based on a "true biological" cutoff instead of a statistical cutoff. The value of this filter is a character vector of HUGO gene symbols (MUST be contained in the annotation, thus it's better to use annotation="download") whose counts are used to build a "null" expression distribution. The 90th quantile of this distribution is then the expression cutoff. This filter can be combined with any other filter. Be careful with gene names as they are case sensitive and must match exactly ("Pten" is different from "PTEN"!). iv) biotype where in this case, genes with a certain biotype (MUST be contained in the annotation, thus it's better to use annotation="download") are excluded from the analysis. This filter is a named list of logical, where names are the biotypes in each genome and values are TRUE or FALSE. If the biotype should be excluded, the value should be TRUE else FALSE. See the result of get.defaults("biotype.filter","hg19") for an example. Finally, in future versions there will be support for user-defined filters in the form of a function. v) presence where in this case, a gene is further considered for statistical testing if frac (x100 for a percentage value) have more than minCount reads across all samples (perCondition=FALSE) or across the samples of each condition (perCondition=TRUE).

Normalization parameters

The normalization parameters are passed again as a named list where the names of the members are the normalization parameter names and the values are the normalization parameter values. You should check the documentation of the packages EDASeq, DESeq, edgeR, NOISeq and NBPSeq for the parameter names and parameter values. There are a few exceptions in parameter names: in case of normalization="edaseq" the only parameter names are within.which and between.which, controlling the withing lane/sample and between lanes/samples normalization algorithm. In the case of normalization="nbpseq", there is one additional parameter called main.method which can take the calues "nbpseq" or "nbsmyth". These values correspond to the two different workflows available in the NBPSeq package. Please, consult the NBPSeq package documentation for further details. For the rest of the algorithms, the parameter names are the same as the names used in the respective packages. For examples, please use the getDefaults function.

Statistics parameters

The statistics parameters as passed to statistical algorithms in metaseqr2, exactly with the same way as the normalization parametes above. In this case, there is one more layer in list nesting. Thus, statArgs is a named list whose names are the names the algorithms used (see the statistics parameter). Each member is another named list,with parameters to be used for each statistical algorithm. Again, the names of the member lists are parameter names and the values of the member lists are parameter values. You should check the documentations of DESeq, edgeR, NOISeq, limma and NBPSeq for these parameters. There are a few exceptions in parameter names: In case of statistics="edger", apart from the rest of the edgeR statistical testing arguments, there is the argument mainMethod which can be either "classic" or "glm", again defining whether the binomial test or GLMs will be used for statistical testing. For examples, please use the getDefaults function. When statistics="nbpseq", apart from the rest arguments of the

NBPSeq functions `estimate.disp` and `estimate.dispersion`, there is the argument `mainMethod` which can be `"nbpseq"` or `"nbsmyth"`. This argument determines the parameters to be used by the `estimate.dispersion` function or by the `estimate.disp` function to estimate RNA-Seq count dispersions. The difference between the two is that they constitute different starting points for the two workflows in the package NBPSeq. The first worklfow (with `mainMethod="nbpseq"` and the `estimate.dispersion` function is NBPSeq package specific, while the second (with `mainMethod="nbsmyth"` and the `estimate.disp` function is similar to the workflow of the edgeR package. For additional information regarding the statistical testing in NBPSeq, please consult the documentation of the NBPSeq package.

Presets

The analysis presets are a set of keywords (only one can be used) that predefine some of the parameters of the metaseqr2 pipeline. For the time being they are quite simple and they control i) the strictness of filtering and statistical thresholding with three basic levels ("all", "medium", "strict") and ii) the data columns that are exported, again in three basic ways ("basic", "normal", "full") controlling the amount of data to be exported. These keywords can be combined with a dot in the middle (e.g. `"all.basic"` to define an analysis preset. When using analysis presets, the following argumentsof metaseqr2 are overriden: `exonFilters`, `geneFilters`, `pcut`, `exportWhat`, `exportScale`, `exportValues`, `exonStats`. If you want to explicitly control the above arguments, the `preset` argument should be set to `NULL` (default). Following is a synopsis of the different presets and the values of the arguments they moderate:

- `"all_basic"`: use all genes (do not filter) and export all genes and basic annotation and statistics elements. In this case, the above described arguments become:
  - `exonFilters=NULL`
  - `geneFilters=NULL`
  - `pcut=1`
  - `exportWhat=c("annotation","p_value", "adj_p_value","meta_p_value", "adj_meta_p_value","fold_`
  - `exportScale=c("natural","log2")`
  - `exportValues=c("normalized")`
  - `exportStats=c("mean")`
- `"all_normal"`: use all genes (do not filter) and export all genes and normal annotation and statistics elements. In this case, the above described arguments become:
  - `exonFilters=NULL`
  - `geneFilters=NULL`
  - `pcut=1`
  - `exportWhat=c("annotation","p_value", "adj_p_value","meta_p_value", "adj_meta_p_value","fold_` `"counts")`
  - `exportScale=c("natural","log2")`
  - `exportValues=c("normalized")`
  - `exportStats=c("mean","sd","cv")`
- `"all_full"`: use all genes (do not filter) and export all genes and all annotation and statistics elements. In this case, the above described arguments become:
  - `exonFilters=NULL`
  - `geneFilters=NULL`

  - pcut=1
  - exportWhat=c("annotation","p_value", "adj_p_value","meta_p_value", "adj_meta_p_value","fold_
  - exportScale=c("natural","log2","log10","vst")
  - exportValues=c("raw","normalized")
  - exportStats=c("mean","median","sd","mad", "cv","rcv")
- "medium_basic": apply a medium set of filters and and export statistically significant genes and basic annotation and statistics elements. In this case, the above above described arguments become:
  - exonFilters=list(minActiveExons= list(exonsPerGene=5,minExons=2,frac=1/5))
  - geneFilters=list(length=list(length=500), avgReads=list(averagePerBp=100,quantile=0.25), expression=list(median=TRUE,mean=FALSE,quantile=NA, known=NA,custom=NA), biotype=getDefaults("biotypeFilter",org[1]))
  - pcut=0.05
  - exportWhat=c("annotation","p_value", "adj_p_value","meta_p_value", "adj_meta_p_value","fold_c
  - exportScale=c("natural","log2")
  - exportValues=c("normalized")
  - exportStats=c("mean")
- "medium_normal": apply a medium set of filters and export statistically significant genes and normal annotation and statistics elements. In this case, the above described arguments become:
  - exonFilters=list(minActiveExons= list(exonsPerGene=5,minExons=2,frac=1/5))
  - geneFilters=list(length=list(length=500), avgReads=list(averagePerBp=100,quantile=0.25), expression=list(median=TRUE,mean=FALSE, quantile=NA,known=NA,custom=NA), biotype=getDefaults("biotypeFilter",org[1]))
  - pcut=0.05
  - export.what=c("annotation","p_value", "adj_p_value","meta_p_value", "adj_meta_p_value","fold_ "stats","counts")
  - exportScale=c("natural","log2")
  - exportValues=c("normalized")
  - exportStats=c("mean","sd","cv")
- "medium_full": apply a medium set of filters and export statistically significant genes and full annotation and statistics elements. In this case, the above described arguments become:
  - exonFilters=list(minActiveExons= list(exonsPerGene=5,minExons=2,frac=1/5))
  - geneFilters=list(length=list(length=500), avgReads=list(averagePerBp=100,quantile=0.25), expression=list(median=TRUE,mean=FALSE, quantile=NA,known=NA,custom=NA), biotype=getDefaults("biotypeFilter",org[1]))
  - pcut=0.05
  - exportWhat=c("annotation","p_value", "adj_p_value","meta_p_value", "adj_meta_p_value","fold_c "stats","counts")
  - exportScale=c("natural","log2","log10", "vst")
  - exportValues=c("raw","normalized")
  - exportStats=c("mean","median","sd","mad", "cv","rcv")
- "strict_basic": apply a strict set of filters and and export statistically significant genes and basic annotation and statistics elements. In this case, the above described arguments become:

  - exonFilters=list(minActiveExons= list(exonsPerGene=4,minExons=2,frac=1/4))
  - geneFilters=list(length=list(length=750), avgReads=list(averagePerBp=100,quantile=0.5), expression=list(median=TRUE,mean=FALSE, quantile=NA,known=NA,custom=NA), biotype=getDefaults("biotypeFilter",org[1]))
  - pcut=0.01
  - exportWhat=c("annotation","p_value", "adj_p_value","meta.p.value", "adj_meta_p_value","fold_
  - exportScale=c("natural","log2")
  - exportValues=c("normalized")
  - exportStats=c("mean")

- "strict_normal": apply a strict set of filters and export statistically significant genes and normal annotation and statistics elements. In this case, the above described arguments become:
  - exonFilters=list(minActiveExons= list(exonsPerGene=4,minExons=2,frac=1/4))
  - geneFilters=list(length=list(length=750), avgReads=list(averagePerBp=100,quantile=0.5), expression=list(median=TRUE,mean=FALSE, quantile=NA,known=NA,custom=NA), biotype=getDefaults("biotypeFilter",org[1]))
  - pcut=0.01
  - exportWhat=c("annotation","p_value", "adj_p_value","meta_p_value", "adj_meta_p_value","fold_ "stats","counts")
  - exportScale=c("natural","log2")
  - exportValues=c("normalized")
  - exportStats=c("mean","sd","cv")
- "strict_full": apply a strict set of filters and export statistically significant genes and full annotation and statistics elements. In this case, the above described arguments become:
  - exonFilters=list(minActiveExons= list(exonsPerGene=4,minExons=2,frac=1/4))
  - geneFilters=list(length=list(length=750), avgReads=list(averagePerBp=100,quantile=0.5), expression=list(median=TRUE,mean=FALSE, quantile=NA,known=NA,custom=NA), biotype=getDefaults("biotypeFilter",org[1]))
  - pcut=0.01
  - exportWhat=c("annotation","p_value", "adj_p_value","meta_p_value", "adj_meta_p_value","fold_ "stats","counts")
  - exportScale=c("natural","log2","log10", "vst")
  - exportValues=c("raw","normalized")
  - exportStats=c("mean","median","sd","mad", "cv","rcv")

Note

Currently only gene and exon annotation from Ensembl (http://www.ensembl.org), UCSC and RefSeq are supported. In addition, the user may choose to use own GTF file on the fly or import to the backend annotation database (see buildAnnotationDatabase). Thus, the unique gene ids in the counts files should correspond to valid Ensembl, UCSC or RefSeq gene or exon accessions for the organism of interest, or according to the user's GTF. If you are not sure about the source of your counts file or do not know how to produce it, it's better to start from the original BAM/BED files (metaseqr2 will use the read2count function to create a counts file). Keep in mind that in the

case of BED files, the performance will be significantly lower and the overall running time significanlty higher as the R functions which are used to read BED files to proper structures (GenomicRanges) and calculate the counts are quite slow. An alternative way is maybe the easyRNASeq package (Delhomme et al, 2012). The `read2count` function does not use this package but rather makes use of standard Bioconductor functions to handle NGS data. If you wish to work outside R, you can work with other popular read counters such as the HTSeq read counter (http://www-huber.embl.de/users/anders/HTSeq/doc/overview.html). Please also note that in the current version, the members of the `geneFilters` and `exonFilters` lists are not checked for validity so be careful to supply with correct names otherwise the pipeline will crash or at the best case scenario, will ignore the filters. Also note that when you are supplying metaseqr2 wtih an exon counts table, gene annotation is always downloaded so please be sure to have a working internet connection. In addition to the above, if you have a multiple core system, be very careful on how you are using the `restrictCores` argument and generally how many cores you are using with scripts purely written in R. The analysis with exon read data can very easily cause memory problems, so unless you have more than 64Gb of RAM available, consider setting restrict.cores to something like 0.2 when working with exon data. Finally, if you do not wish to download the same annotation again and again when performing multiple analyses, it is best to use the `buildAnnotationDatabase` function to download and store the resulting data frames in local SQLite database and then use these files with the `org`, `refdb` and `version` options.

Please note that the **meta-analysis** feature provided by metaseqr2 does not satisfy the strict definition of "meta-analysis", which is the combination of multiple similar datasets under the same statistical methodology. Instead it is the use of mulitple statistical tests applied to the same data. For the Simes method, please consult also "Simes, R. J. (1986). "An improved Bonferroni procedure for multiple tests of significance". Biometrika 73 (3): 751–754."

Also, if `weight="meta_perm"` ideally one would want to create the same set of indices for a given dataset so as to create reproducible p-values. To achieve this, use the `set.seed` function prior to any calculations. When metaseqR2 is loaded, the random seed is set to `42`.

Author(s)

Panagiotis Moulos

Examples

```
# An example pipeline with gene counts
data("mm9GeneData",package="metaseqR2")

result <- metaseqr2(
    counts=mm9GeneCounts,
    sampleList=sampleListMm9,
    contrast=c("adult_8_weeks_vs_e14.5"),
    libsizeList=libsizeListMm9,
    annotation="embedded",
    org="mm9",
    countType="gene",
    normalization="edger",
    statistics="edger",
    pcut=0.05,
    figFormat="png",
    qcPlots="mds",
```

```
    exportWhat=c("annotation","p_value","adj_p_value","fold_change"),
    exportScale="natural",
    exportValues="normalized",
    exportStats="mean",
    exportWhere=file.path(tempdir(),"test1"),
    restrictCores=0.01,
    geneFilters=list(
        length=list(
            length=500
        ),
        avgReads=list(
            averagePerBp=100,
            quantile=0.25
        ),
        expression=list(
            median=TRUE,
            mean=FALSE,
            quantile=NA,
            known=NA,
            custom=NA
        ),
        biotype=getDefaults("biotypeFilter","mm9")
    ),
    outList=TRUE
)
head(result$data[["adult_8_weeks_vs_e14.5"]])
```

---

metaseqrPlot *Diagnostic plots for the metaseqR2 package*

Description

This is the main function for producing sructured quality control and informative graphs base on the results of the various steps of the metaseqR package. The graphs produced span a variety of issues like good sample reproducibility (Multi-Dimensional Scaling plot, biotype detection, heatmaps. diagplotMetaseqr, apart from implementing certain package-specific plots, is a wrapper around several diagnostic plots present in other RNA-Seq analysis packages such as EDASeq and NOISeq.

Usage

```
metaseqrPlot(object, sampleList, annotation = NULL,
    contrastList = NULL, pList = NULL,
    thresholds = list(p = 0.05, f = 1),
    plotType = c("mds", "biodetection", "countsbio",
        "saturation", "readnoise", "rnacomp", "correl",
        "pairs", "boxplot", "gcbias", "lengthbias",
        "meandiff", "meanvar", "deheatmap", "volcano",
        "biodist", "filtered", "mastat", "deregulogram",
        "statvenn", "foldvenn"),
    isNorm = FALSE, output = "x11", path = NULL, ...)
```

Arguments

object	a matrix or a data frame containing count data derived before or after the normalization procedure, filtered or not by the metaseqR2's filters and/or p-value.
sampleList	the list containing condition names and the samples under each condition.
annotation	a data frame containing annotation elements for each row in object. See also Details.
contrastList	the vector of contrasts as defined in the main help page of metaseqr2.
pList	a list of p-values for each contrast as obtained from any of the stat* methods of the metaseqr package. See also Details.
thresholds	a list with the elements "p" and "f" which are the p-value and the fold change cutoff when diagplotType="volcano".
plotType	one or more of the diagnostic plots supported in metaseqR2 package. See also Details.
isNorm	a logical indicating whether object contains raw or normalized data. It is not essential and it serves only plot annotation purposes.
output	one or more R plotting device to direct the plot result to. See Details.
path	the path to create output files.
...	further arguments to be passed to plot devices, such as parameter from par.

Details

Regarding object, the object can be fed to any of the diagplotMetaseqr plotting systems but not every plot is meaningful. For example, it's meaningless to create a "biodist" plot for a count matrix before normalization or statistical testing.

Regarding annotation, usually, it is a subset of the annotation obtained by getAnnotation or a subset of possibly embedded annotation with the input counts table. This parameter is optional and required only when diagplotType is any of "biodetection", "countsbio", "saturation", "rnacomp", "readnoise", "biodist", "gcbias", "lengthbias" or "filtered".

Regarding contrastList, this parameter is optional and required only when diagplotType is any of "deheatmap", "volcano" or "biodist". It can also be a named structured list of contrasts as returned by the internal function metaseqR2:::makeContrastList.

Regarding diagplotType, many of these plots require the presence of additional package, something that is checked while running the main metaseqr2 function. The supported plots are "mds", "biodetection", "countsbio", "saturation", "rnacomp", "boxplot", "gcbias", "lengthbias", "meandiff", "meanvar", "deheatmap", "volcano", "biodist", "filtered", "readnoise", "venn", "correl", "pairwise". For a brief description of these plots please see the main metaseqr2 help page.

Regarding pList, this parameter is optional and required only when diagplotType is any of "deheatmap", "volcano" or "biodist".

Regarding output, supported mechanisms are: "png", "jpg", "bmp", "pdf", "ps" or "json". The latter is currently available for the creation of interactive volcano plots only when reporting the output, through the highcharts javascript library. The default plotting ("x11") is not supported due to instability in certain devices.

Value

A named list containing the file names of the produced plots. Each list member is names according to the selected plotting device and is also a named list, whose names are the plot types. The final contents are the file names in case the plots are written to a physical location (not meaningful for "x11").

Note

In order to make the best out of this function, you should generally provide the annotation argument as most and also the most informative plots depend on this. If you don't know what is inside your counts table or how many annotation elements you can provide by embedding it, it's always best to setup a local databse so as to use predefined annotations that work better with the functions of the whole package.

Author(s)

Panagiotis Moulos

Examples

```
dataMatrix <- metaseqR2:::exampleCountData(2000)
sampleList <- list(A=c("A1","A2"),B=c("B1","B2","B3"))
contrast <- "A_vs_B"
metaseqrPlot(dataMatrix,sampleList,plotType=c("mds","boxplot"))

normArgs <- getDefaults("normalization","deseq2")
object <- normalizeDeseq2(dataMatrix,sampleList,normArgs)
metaseqrPlot(object,sampleList,plotType="boxplot")

## More
#p <- statDeseq2(object,sampleList)
#metaseqrPlot(object,sampleList,contrastList=contrast,pList=p,
#    plotType="volcano")
```

---

metaTest *Meta-analysis using several RNA-Seq statistics*

Description

This function calculates the combined p-values when multiple statistical algorithms are applied to the input dataset. It is a helper and it requires very specific arguments so it should not be used individually

Usage

```
metaTest(cpList,
    metaP = c("simes", "bonferroni", "fisher", "harmonic",
    "dperm_min", "dperm_max", "dperm_weight", "fperm",
    "whitlock", "minp", "maxp", "weight", "pandora",
    "none"), counts, sampleList, statistics, statArgs,
    libsizeList, nperm = 10000,
    weight = rep(1/length(statistics), length(statistics)),
    pOffset = NULL, rc = NULL)
```

Arguments

cpList	a named list whose names are the contrasts requested from metaseqr2. Each member is a p-value matrix whose colnames are the names of the statistical tests applied to the data. See the main metaseqr2 help page.
metaP	the p-value combination method to use. See the main metaseqr2 help page.
counts	the normalized and possibly filtered read counts matrix. See the main metaseqr2 help page.
sampleList	the list containing condition names and the samples under each condition. See the main metaseqr2 help page.
statistics	the statistical algorithms used in metaseqr2. See the main metaseqr2 help page.
statArgs	the parameters for each statistical argument. See the main metaseqr2 help page.
libsizeList	a list with library sizes. See the main metaseqr2 and the stat* help pages.
nperm	the number of permutations (Monte Carlo simulations) to perform.
weight	a numeric vector of weights for each statistical algorithm.
pOffset	NULL (default) or a fixed numeric value between 0 and 1. See also the main metaseqr2 man page.
rc	the fraction of the available cores to use in a multicore system.

Details

Ideally one would want to create the same set of indices for a given dataset so as to create reproducible p-values. To achieve this, use the set.seed function prior to any calculations.

Value

A named list with combined p-values. The names are the contrasts and the list members are combined p-value vectors, one for each contrast.

Author(s)

Panagiotis Moulos

Examples

```
cpList <- list(a=matrix(runif(100),50,2))
metaP <- metaTest(cpList,"simes")
```

mm9GeneCounts *Mouse RNA-Seq data with two conditions, four samples*

Description

This data set contains RNA-Seq gene read counts for 3 chromosomes. The data were downloaded from the ENCODE public repository and are derived from the study of Mortazavi et al., 2008 (Mortazavi A, Williams BA, McCue K, Schaeffer L, Wold B. Mapping and quantifying mammalian transcriptomes by RNA-Seq. Nat Methods. 2008 Jul;5(7):621-8). In their experiment, the authors studied among others genes expression at two developmental stages of mouse liver cells. It has two conditions-developmental stages (e14.5, adult_8_weeks) and four samples (e14.5_1, e14.5_2, a8w_1, a8w_2). It also contains a predefined `sampleList` and `libsizeList` named `sampleListMm9` and `libsizeListMm9`.

Format

a `data.frame` with gene read counts and some embedded annotation, one row per gene.

Author(s)

Panagiotis Moulos

Source

ENCODE (http://genome.ucsc.edu/encode/)

normalizeAbsseq *Normalization based on the ABSSeq package*

Description

This function is a wrapper over ABSSeq normalization. It accepts a matrix of gene counts (e.g. produced by importing an externally generated table of counts to the main metaseqr2 pipeline).

Usage

```
normalizeAbsseq(geneCounts, sampleList,
    normArgs = NULL, output = c("matrix", "native"))
```

Arguments

`geneCounts`	a table where each row represents a gene and each column a sample. Each cell contains the read counts for each gene and sample. Such a table can be produced outside metaseqr2 and is imported during the basic metaseqr2 workflow.
`sampleList`	the list containing condition names and the samples under each condition.
`normArgs`	a list of DESeq normalization parameters. See the result of `getDefaults("normalization", "deseq")` for an example and how you can modify it.
`output`	the class of the output object. It can be `"matrix"` (default) for versatility with other tools or `"native"` for the ABSSeq native S4 object (ABSDataSet). In the latter case it should be handled with suitable ABSSeq methods.

Value

A matrix or a ABSDataSet with normalized counts.

Author(s)

Dionysios Fanidis

Examples

```
dataMatrix <- metaseqR2:::exampleCountData(2000)
sampleList <- list(A=c("A1","A2"),B=c("B1","B2","B3"))
diagplotBoxplot(dataMatrix,sampleList)

normDataMatrix <- normalizeAbsseq(dataMatrix,sampleList)
diagplotBoxplot(normDataMatrix,sampleList)
```

`normalizeDeseq` *Normalization based on the DESeq package*

Description

This function is a wrapper over DESeq normalization. It accepts a matrix of gene counts (e.g. produced by importing an externally generated table of counts to the main metaseqr2 pipeline).

Usage

```
normalizeDeseq(geneCounts, sampleList,
    normArgs = NULL, output = c("matrix", "native"))
```

Arguments

`geneCounts`	a table where each row represents a gene and each column a sample. Each cell contains the read counts for each gene and sample. Such a table can be produced outside metaseqr2 and is imported during the basic metaseqr2 workflow.
`sampleList`	the list containing condition names and the samples under each condition.
`normArgs`	a list of DESeq normalization parameters. See the result of `getDefaults("normalization", "deseq")` for an example and how you can modify it.
`output`	the class of the output object. It can be `"matrix"` (default) for versatility with other tools or `"native"` for the DESeq native S4 object (CountDataSet). In the latter case it should be handled with suitable DESeq methods.

Value

A matrix or a CountDataSet with normalized counts.

Author(s)

Panagiotis Moulos

Examples

```
dataMatrix <- metaseqR2:::exampleCountData(2000)
sampleList <- list(A=c("A1","A2"),B=c("B1","B2","B3"))
diagplotBoxplot(dataMatrix,sampleList)

normDataMatrix <- normalizeDeseq(dataMatrix,sampleList)
diagplotBoxplot(normDataMatrix,sampleList)
```

`normalizeDeseq2` *Normalization based on the DESeq2 package*

Description

This function is a wrapper over DESeq2 normalization. It accepts a matrix of gene counts (e.g. produced by importing an externally generated table of counts to the main metaseqr2 pipeline).

Usage

```
normalizeDeseq2(geneCounts, sampleList,
    normArgs = NULL, output = c("matrix", "native"))
```

Arguments

`geneCounts`	a table where each row represents a gene and each column a sample. Each cell contains the read counts for each gene and sample. Such a table can be produced outside metaseqr2 and is imported during the basic metaseqr2 workflow.
`sampleList`	the list containing condition names and the samples under each condition.
`normArgs`	a list of DESeq normalization parameters. See the result of `getDefaults("normalization", "deseq")` for an example and how you can modify it.
`output`	the class of the output object. It can be `"matrix"` (default) for versatility with other tools or `"native"` for the DESeq2 native S4 object (DESeqDataSet). In the latter case it should be handled with suitable DESeq2 methods.

Value

A matrix or a DESeqDataSet with normalized counts.

Author(s)

Dionysios Fanidis

Examples

```
dataMatrix <- metaseqR2:::exampleCountData(2000)
sampleList <- list(A=c("A1","A2"),B=c("B1","B2","B3"))
diagplotBoxplot(dataMatrix,sampleList)

normDataMatrix <- normalizeDeseq2(dataMatrix,sampleList)
diagplotBoxplot(normDataMatrix,sampleList)
```

`normalizeDss` *Normalization based on the DSS package*

Description

This function is a wrapper over ABSSeq normalization. It accepts a matrix of gene counts (e.g. produced by importing an externally generated table of counts to the main metaseqr2 pipeline).

Usage

```
normalizeDss(geneCounts, sampleList,
    normArgs = NULL, output = c("matrix", "native"))
```

Arguments

geneCounts	a table where each row represents a gene and each column a sample. Each cell contains the read counts for each gene and sample. Such a table can be produced outside metaseqr2 and is imported during the basic metaseqr2 workflow.
sampleList	the list containing condition names and the samples under each condition.
normArgs	a list of DESeq normalization parameters. See the result of `getDefaults("normalization", "deseq")` for an example and how you can modify it.
output	the class of the output object. It can be `"matrix"` (default) for versatility with other tools or `"native"` for the DSS native S4 object (SeqCountSet). In the latter case it should be handled with suitable ABSSeq methods.

Value

A matrix or a SeqCountSet with normalized counts.

Author(s)

Dionysios Fanidis

Examples

```
dataMatrix <- metaseqR2:::exampleCountData(2000)
sampleList <- list(A=c("A1","A2"),B=c("B1","B2","B3"))
diagplotBoxplot(dataMatrix,sampleList)

normDataMatrix <- normalizeDss(dataMatrix,sampleList)
diagplotBoxplot(normDataMatrix,sampleList)
```

---

normalizeEdaseq *Normalization based on the EDASeq package*

Description

This function is a wrapper over EDASeq normalization. It accepts a matrix of gene counts (e.g. produced by importing an externally generated table of counts to the main metaseqr2 pipeline).

Usage

```
normalizeEdaseq(geneCounts, sampleList,
    normArgs = NULL, geneData = NULL,
    output = c("matrix", "native"))
```

Arguments

`geneCounts`	a table where each row represents a gene and each column a sample. Each cell contains the read counts for each gene and sample. Such a table can be produced outside metaseqr2 and is imported during the basic metaseqr2 workflow.
`sampleList`	the list containing condition names and the samples under each condition.
`normArgs`	a list of EDASeq normalization parameters. See the result of `getDefaults("normalization", "edaseq")` for an example and how you can modify it.
`geneData`	an optional annotation data frame (such the ones produced by `getAnnotation`) which contains the GC content for each gene and from which the gene lengths can be inferred by chromosome coordinates.
`output`	the class of the output object. It can be `"matrix"` (default) for versatility with other tools or `"native"` for the EDASeq native S4 object (SeqExpressionSet). In the latter case it should be handled with suitable EDASeq methods.

Value

A matrix or a SeqExpressionSet with normalized counts.

Author(s)

Panagiotis Moulos

Examples

```
dataMatrix <- metaseqR2:::exampleCountData(2000)
sampleList <- list(A=c("A1","A2"),B=c("B1","B2","B3"))
diagplotBoxplot(dataMatrix,sampleList)

lengths <- round(1000*runif(nrow(dataMatrix)))
starts <- round(1000*runif(nrow(dataMatrix)))
ends <- starts + lengths
gc=runif(nrow(dataMatrix))
geneData <- data.frame(
    chromosome=c(rep("chr1",nrow(dataMatrix)/2),
        rep("chr2",nrow(dataMatrix)/2)),
    start=starts,end=ends,gene_id=rownames(dataMatrix),gc_content=gc,
    row.names=rownames(dataMatrix)
)
normDataMatrix <- normalizeEdaseq(dataMatrix,sampleList,
    geneData=geneData)
diagplotBoxplot(normDataMatrix,sampleList)
```

`normalizeEdger` *Normalization based on the edgeR package*

Description

This function is a wrapper over edgeR normalization. It accepts a matrix of gene counts (e.g. produced by importing an externally generated table of counts to the main metaseqr2 pipeline).

Usage

```
normalizeEdger(geneCounts, sampleList,
    normArgs = NULL, output = c("matrix", "native"))
```

Arguments

`geneCounts`	a table where each row represents a gene and each column a sample. Each cell contains the read counts for each gene and sample. Such a table can be produced outside metaseqr2 and is imported during the basic metaseqr2 workflow.
`sampleList`	the list containing condition names and the samples under each condition.
`normArgs`	a list of edgeR normalization parameters. See the result of `getDefaults("normalization", "edger")` for an example and how you can modify it.
`output`	the class of the output object. It can be `"matrix"` (default) for versatility with other tools or `"native"` for the edgeR native S4 object (DGEList). In the latter case it should be handled with suitable edgeR methods.

Value

A matrix or a DGEList with normalized counts.

Author(s)

Panagiotis Moulos

Examples

```
dataMatrix <- metaseqR2:::exampleCountData(2000)
sampleList <- list(A=c("A1","A2"),B=c("B1","B2","B3"))
diagplotBoxplot(dataMatrix,sampleList)

normDataMatrix <- normalizeEdger(dataMatrix,sampleList)
diagplotBoxplot(normDataMatrix,sampleList)
```

`normalizeNbpseq` *Normalization based on the NBPSeq package*

Description

This function is a wrapper over DESeq normalization. It accepts a matrix of gene counts (e.g. produced by importing an externally generated table of counts to the main metaseqr2 pipeline).

Usage

```
normalizeNbpseq(geneCounts, sampleList,
    normArgs = NULL, libsizeList = NULL,
    output = c("matrix", "native"))
```

Arguments

`geneCounts`	a table where each row represents a gene and each column a sample. Each cell contains the read counts for each gene and sample. Such a table can be produced outside metaseqr2 and is imported during the basic metaseqr2 workflow.
`sampleList`	the list containing condition names and the samples under each condition.
`normArgs`	a list of NBPSeq normalization parameters. See the result of `getDefaults("normalization", "nbpseq")` for an example and how you can modify it.
`libsizeList`	an optional named list where names represent samples (MUST be the same as the samples in `sampleList`) and members are the library sizes (the sequencing depth) for each sample. If not provided, the default is the column sums of the `geneCounts` matrix.
`output`	the class of the output object. It can be `"matrix"` (default) for versatility with other tools or `"native"` for the NBPSeq native S4 object (a specific list). In the latter case it should be handled with suitable NBPSeq methods.

Value

A matrix with normalized counts or a list with the normalized counts and other NBPSeq specific parameters.

Author(s)

Panagiotis Moulos

Examples

```
dataMatrix <- metaseqR2:::exampleCountData(2000)
sampleList <- list(A=c("A1","A2"),B=c("B1","B2","B3"))
diagplotBoxplot(dataMatrix,sampleList)

normDataMatrix <- normalizeNbpseq(dataMatrix,sampleList)
diagplotBoxplot(normDataMatrix,sampleList)
```

normalizeNoiseq *Normalization based on the NOISeq package*

Description

This function is a wrapper over NOISeq normalization. It accepts a matrix of gene counts (e.g. produced by importing an externally generated table of counts to the main metaseqr2 pipeline).

Usage

```
normalizeNoiseq(geneCounts, sampleList,
    normArgs = NULL, geneData = NULL, logOffset = 1,
    output = c("matrix", "native"))
```

Arguments

geneCounts	a table where each row represents a gene and each column a sample. Each cell contains the read counts for each gene and sample. Such a table can be produced outside metaseqr2 and is imported during the basic metaseqr2 workflow.
sampleList	the list containing condition names and the samples under each condition.
normArgs	a list of NOISeq normalization parameters. See the result of `getDefaults("normalization", "noiseq")` for an example and how you can modify it.
geneData	an optional annotation data frame (such the ones produced by `get.annotation` which contains the GC content for each gene and from which the gene lengths can be inferred by chromosome coordinates.
logOffset	an offset to use to avoid infinity in logarithmic data transformations.
output	the class of the output object. It can be `"matrix"` (default) for versatility with other tools or `"native"` for the NOISeq native S4 object (SeqExpressionSet). In the latter case it should be handled with suitable NOISeq methods.

Value

A matrix with normalized counts.

Author(s)

Panagiotis Moulos

Examples

```
dataMatrix <- metaseqR2:::exampleCountData(2000)
sampleList <- list(A=c("A1","A2"),B=c("B1","B2","B3"))
diagplotBoxplot(dataMatrix,sampleList)

lengths <- round(1000*runif(nrow(dataMatrix)))
starts <- round(1000*runif(nrow(dataMatrix)))
ends <- starts + lengths
```

```
gc=runif(nrow(dataMatrix))
geneData <- data.frame(
    chromosome=c(rep("chr1",nrow(dataMatrix)/2),
        rep("chr2",nrow(dataMatrix)/2)),
    start=starts,end=ends,gene_id=rownames(dataMatrix),gc_content=gc,
    biotype=rep("protein_coding",nrow(dataMatrix)),
    row.names=rownames(dataMatrix)
)
normDataMatrix <- normalizeNoiseq(dataMatrix,sampleList,normArgs=NULL,geneData)
diagplotBoxplot(normDataMatrix,sampleList)
```

`read2count` *SAM/BAM/BED file reader helper for the metaseqr2 pipeline*

Description

This function is a helper for the `metaseqr2` pipeline, for reading SAM/BAM or BED files when a read counts file is not available. It can also be used very easily in an autonomous manner.

Usage

```
read2count(targets, annotation, fileType = targets$type,
    transLevel = "gene", utrOpts = list(frac = 1,
    minLength = 300, downstream = 50), interFeature = FALSE,
    rc = NULL)
```

Arguments

`targets`	a named list, the output of `readTargets` or an existing file with targets. See also the main `metaseqr2` man page.
`annotation`	a `GenomicRanges` or `data.frame` with genomic coordinates to use for read counting. See also `getAnnotation`.
`fileType`	the type of raw input files. It can be `"bed"` for BED files or `"sam"`, `"bam"` for SAM/BAM files. See the same argument in the main `metaseqr2` function for the case of auto-guessing.
`transLevel`	see the `transLevel` argument in the main `metaseqr2` function.
`utrOpts`	a named list with members `frac` which is the fraction (0-1) of the 3' UTR region to count reads in, `minLength` the minimum acceptable 3'UTR length irrespective of `frac` and `downstream` the number of base pairs to flank the end of the 3' UTR of transcripts when analyzing Quant-Seq data.
`interFeature`	see the `inter.feature` argument in `summarizeOverlaps`.
`rc`	the fraction of the available cores to use in a multicore system.

Value

A data frame with counts for each sample, ready to be passed to the main `metaseqr2` pipeline.

Author(s)

Panagiotis Moulos

Examples

```
dataPath <- system.file("extdata",package="metaseqR2")
targets <- data.frame(samplename=c("C","T"),
    filename=file.path(dataPath,c("C.bam","T.bam")),
    condition=c("Control","Treatment"),
    paired=c("single","single"),stranded=c("forward","forward"))
path <- tempdir()
write.table(targets,file=file.path(path,"targets.txt"),
    sep="\t",row.names=FALSE,quote=FALSE)
geneData <- loadAnnotation("mm10","ensembl","gene")
myTargets <- readTargets(file.path(path,"targets.txt"))
if (.Platform$OS.type == "unix") {
    r2c <- read2count(targets=myTargets,
        fileType=myTargets$type,annotation=geneData)
    geneCounts <- r2c$counts
    libsizeList <- r2c$libsize
}
```

---

readTargets *Creates sample list and BAM/BED file list from file*

Description

Create the main sample list and determine the BAM/BED files for each sample from an external file.

Usage

```
readTargets(input, path = NULL)
```

Arguments

input	a tab-delimited file or a YAML file specifically structured. See Details.
path	an optional path where all the BED/BAM files are placed, to be prepended to the BAM/BED file names in the targets file.

Details

Regarding the input file, this can be a simple text tab-delimited file or a YAML file describing the data to be analyzed.

Regarding the tab-delimited version, its columns must be structured as follows: the first line of the external tab delimited file should contain column names (names are not important). The first column MUST contain UNIQUE sample names. The second column MUST contain the raw BAM/BED files WITH their full path. Alternatively, the `path` argument should be provided (see below). The

third column MUST contain the biological condition where each of the samples in the first column should belong to. There is an optional fourth column which should contain the keywords "single" for single-end reads, "paired" for paired-end reads or "mixed" for BAM files that contain both single- and paired-end reads (e.g. after a mapping procedure with two round of alignment). If this column is not provided, single-end reads will be assumed. There is an optional fifth column which stranded read assignment. It should contain the keywords "forward" for a forward (5'->3') strand library construction protocol, "reverse" for a reverse (3'->5') strand library construction protocol, or "no" for unstranded/unknown protocol. If this column is not provided, unstranded reads will be assumed.

Regarding the YAML version, the same instructions apply, but this time instead of columns, the data are provided as a YAML array under a keyword/top-level field representing the respective header in the tab-delimited version. Alternatively, the aforementioned structure can be nested under a root level named strictly either targets or metaseqR2_targets. The latter can be especially useful when incorporating the metaseqR2 pipeline in a wider pipeline including various analyses and described using a workflow language such as CWL.

Value

A named list with four members. The first member is a named list whose names are the conditions of the experiments and its members are the samples belonging to each condition. The second member is like the first, but this time the members are named vectors whose names are the sample names and the vector elements are full path to BAM/BED files. The third member is like the second, but instead of filenames it contains information about single- or paired-end reads (if available). The fourth member is like the second, but instead of filenames it contains information about the strandedness of the reads (if available). The fifth member is the guessed type of the input files (SAM/BAM or BED). It will be used if not given in the main read2count function.

Author(s)

Panagiotis Moulos

Examples

```
dataPath <- system.file("extdata",package="metaseqR2")
targets <- data.frame(samplename=c("C","T"),
    filename=file.path(dataPath,c("C.bam","T.bam")),
    condition=c("Control","Treatment"),
    paired=c("single","single"),stranded=c("forward","forward"))
path <- tempdir()

# Tab delimited case
write.table(targets,file=file.path(path,"targets.txt"),
    sep="\t",row.names=FALSE,quote=FALSE)
theList <- readTargets(file.path(path,"targets.txt"),path=path)
sampleList <- theList$samples
bamfileList <- theList$files

# YAML case
require(yaml)
write_yaml(as.list(targets),file.path(path,"targets.yml"))
```

```
theYList <- readTargets(file.path(path,"targets.yml"),path=path)
identical(theList,theYList) # TRUE

# YAML case with nested targets
write_yaml(list(targets=as.list(targets)),
    file.path(path,"targets2.yml"))
theYList2 <- readTargets(file.path(path,"targets2.yml"),path=path)
identical(theYList,theYList2) # TRUE
```

---

sampleListMm9 — *Mouse RNA-Seq data with two conditions, four samples*

Description

The sample list for `mm9GeneCounts`. See the data set description.

Format

a named `list` with condition and sample names.

Author(s)

Panagiotis Moulos

Source

ENCODE (http://genome.ucsc.edu/encode/)

---

statAbsseq — *Statistical testing with ABSSeq*

Description

This function is a wrapper over DESeq statistical testing. It accepts a matrix of normalized gene counts or an S4 object specific to each normalization algorithm supported by metaseqR2.

Usage

```
statAbsseq(object, sampleList, contrastList = NULL,
    statArgs = NULL)
```

Arguments

`object`	a matrix or an object specific to each normalization algorithm supported by metaseqR2, containing normalized counts. See also Details.
`sampleList`	the list containing condition names and the samples under each condition.
`contrastList`	vector of contrasts as defined in the main help page of `metaseqr2`. See also Details.
`statArgs`	a list of DESeq statistical algorithm parameters. See the result of `getDefaults("statistics", "deseq")` for an example and how you can modify it. It is not required when the input object is already a CountDataSet from DESeq normalization as the dispersions are already estimated.

Details

Regarding `object`, apart from `matrix` (also for NOISeq), the object can be a `SeqExpressionSet` (EDASeq), `CountDataSet` (DESeq), `DGEList` (edgeR), `DESeqDataSet` (DESeq2), `SeqCountSet` (DSS) or `ABSDataSet` (ABSSeq).

Regarding `contrastList` it can also be a named structured list of contrasts as returned by the internal function `metaseqR2:::makeContrastList`.

Value

A named list of p-values, whose names are the names of the contrasts.

Author(s)

Dionysios Fanidis

Examples

```
require(ABSSeq)
dataMatrix <- metaseqR2:::exampleCountData(2000)
sampleList <- list(A=c("A1","A2"),B=c("B1","B2","B3"))
contrast <- "A_vs_B"
normDataMatrix <- normalizeAbsseq(dataMatrix,sampleList)
p <- statAbsseq(normDataMatrix,sampleList,contrast)
```

---

statDeseq *Statistical testing with DESeq*

Description

This function is a wrapper over DESeq statistical testing. It accepts a matrix of normalized gene counts or an S4 object specific to each normalization algorithm supported by metaseqR2.

Usage

```
statDeseq(object, sampleList, contrastList = NULL,
    statArgs = NULL)
```

Arguments

`object`	a matrix or an object specific to each normalization algorithm supported by metaseqR2, containing normalized counts. See also Details.
`sampleList`	the list containing condition names and the samples under each condition.
`contrastList`	vector of contrasts as defined in the main help page of `metaseqr2`. See also Details.
`statArgs`	a list of DESeq statistical algorithm parameters. See the result of `getDefaults("statistics", "deseq")` for an example and how you can modify it. It is not required when the input object is already a CountDataSet from DESeq normalization as the dispersions are already estimated.

Details

Regarding `object`, apart from `matrix` (also for NOISeq), the object can be a `SeqExpressionSet` (EDASeq), `CountDataSet` (DESeq), `DGEList` (edgeR), `DESeqDataSet` (DESeq2), `SeqCountSet` (DSS) or `ABSDataSet` (ABSSeq).

Regarding `contrastList` it can also be a named structured list of contrasts as returned by the internal function `metaseqR2:::makeContrastList`.

Value

A named list of p-values, whose names are the names of the contrasts.

Author(s)

Panagiotis Moulos

Examples

```
dataMatrix <- metaseqR2:::exampleCountData(1000)
sampleList <- list(A=c("A1","A2"),B=c("B1","B2","B3"))
contrast <- "A_vs_B"
normDataMatrix <- metaseqR2:::newCountDataSet(dataMatrix,
    c("A","A","B","B","B"))
normDataMatrix <- normalizeDeseq(dataMatrix,sampleList)
p <- statDeseq(normDataMatrix,sampleList,contrast)
```

`statDeseq2` *Statistical testing with DESeq2*

Description

This function is a wrapper over DESeq statistical testing. It accepts a matrix of normalized gene counts or an S4 object specific to each normalization algorithm supported by metaseqR2.

Usage

```
statDeseq2(object, sampleList, contrastList = NULL,
    statArgs = NULL)
```

Arguments

`object`	a matrix or an object specific to each normalization algorithm supported by metaseqR2, containing normalized counts. See also Details.
`sampleList`	the list containing condition names and the samples under each condition.
`contrastList`	vector of contrasts as defined in the main help page of `metaseqr2`. See also Details.
`statArgs`	a list of DESeq statistical algorithm parameters. See the result of `getDefaults("statistics", "deseq")` for an example and how you can modify it. It is not required when the input object is already a CountDataSet from DESeq normalization as the dispersions are already estimated.

Details

Regarding `object`, apart from `matrix` (also for NOISeq), the object can be a `SeqExpressionSet` (EDASeq), `CountDataSet` (DESeq), `DGEList` (edgeR), `DESeqDataSet` (DESeq2), `SeqCountSet` (DSS) or `ABSDataSet` (ABSSeq).

Regarding `contrastList` it can also be a named structured list of contrasts as returned by the internal function `metaseqR2:::makeContrastList`.

Value

A named list of p-values, whose names are the names of the contrasts.

Author(s)

Dionysios Fanidis

Examples

```
require(DESeq2)
dataMatrix <- metaseqR2:::exampleCountData(1000)
sampleList <- list(A=c("A1","A2"),B=c("B1","B2","B3"))
contrast <- "A_vs_B"
normDataMatrix <- normalizeDeseq2(dataMatrix,sampleList)
p <- statDeseq2(normDataMatrix,sampleList,contrast)
```

statDss *Statistical testing with DSS*

Description

This function is a wrapper over DESeq statistical testing. It accepts a matrix of normalized gene counts or an S4 object specific to each normalization algorithm supported by metaseqR2.

Usage

```
statDss(object, sampleList, contrastList = NULL,
    statArgs = NULL)
```

Arguments

`object`	a matrix or an object specific to each normalization algorithm supported by metaseqR2, containing normalized counts. See also Details.
`sampleList`	the list containing condition names and the samples under each condition.
`contrastList`	vector of contrasts as defined in the main help page of `metaseqr2`. See also Details.
`statArgs`	a list of DESeq statistical algorithm parameters. See the result of `getDefaults("statistics", "deseq")` for an example and how you can modify it. It is not required when the input object is already a CountDataSet from DESeq normalization as the dispersions are already estimated.

Details

Regarding `object`, apart from `matrix` (also for NOISeq), the object can be a `SeqExpressionSet` (EDASeq), `CountDataSet` (DESeq), `DGEList` (edgeR), `DESeqDataSet` (DESeq2), `SeqCountSet` (DSS) or `ABSDataSet` (ABSSeq).

Regarding `contrastList` it can also be a named structured list of contrasts as returned by the internal function `metaseqR2:::makeContrastList`.

Value

A named list of p-values, whose names are the names of the contrasts.

Author(s)

Dionysios Fanidis

Examples

```
require(DSS)
dataMatrix <- metaseqR2:::exampleCountData(2000)
sampleList <- list(A=c("A1","A2"),B=c("B1","B2","B3"))
contrast <- "A_vs_B"
normDataMatrix <- normalizeDss(dataMatrix,sampleList)
p <- statDss(normDataMatrix,sampleList,contrast)
```

---

statEdger *Statistical testing with edgeR*

Description

This function is a wrapper over edgeR statistical testing. It accepts a matrix of normalized gene counts or an S4 object specific to each normalization algorithm supported by metaseqR2.

Usage

```
statEdger(object, sampleList, contrastList = NULL,
    statArgs = NULL)
```

Arguments

`object`	a matrix or an object specific to each normalization algorithm supported by metaseqR2, containing normalized counts. See also Details.
`sampleList`	the list containing condition names and the samples under each condition.
`contrastList`	vector of contrasts as defined in the main help page of `metaseqr2`. See also Details.
`statArgs`	a list of edgeR statistical algorithm parameters. See the result of `getDefaults("statistics", "edger")` for an example and how you can modify it.

Details

Regarding `object`, apart from `matrix` (also for NOISeq), the object can be a `SeqExpressionSet` (EDASeq), `CountDataSet` (DESeq), `DGEList` (edgeR), `DESeqDataSet` (DESeq2), `SeqCountSet` (DSS) or `ABSDataSet` (ABSSeq).

Regarding `contrastList` it can also be a named structured list of contrasts as returned by the internal function `metaseqR2:::makeContrastList`.

Value

A named list of p-values, whose names are the names of the contrasts.

Author(s)

Panagiotis Moulos

Examples

```
require(edgeR)
dataMatrix <- metaseqR2:::exampleCountData(2000)
sampleList <- list(A=c("A1","A2"),B=c("B1","B2","B3"))
contrast <- "A_vs_B"
normDataMatrix <- normalizeEdger(dataMatrix,sampleList)
p <- statEdger(normDataMatrix,sampleList,contrast)
```

---

statLimma *Statistical testing with limma*

Description

This function is a wrapper over limma statistical testing. It accepts a matrix of normalized gene counts or an S4 object specific to each normalization algorithm supported by metaseqR2.

Usage

```
statLimma(object, sampleList, contrastList = NULL,
    statArgs = NULL)
```

Arguments

`object`	a matrix or an object specific to each normalization algorithm supported by metaseqR2, containing normalized counts. See also Details.
`sampleList`	the list containing condition names and the samples under each condition.
`contrastList`	vector of contrasts as defined in the main help page of `metaseqr2`. See also Details.
`statArgs`	a list of edgeR statistical algorithm parameters. See the result of `getDefaults("statistics", "limma")` for an example and how you can modify it.

Details

Regarding `object`, apart from `matrix` (also for NOISeq), the object can be a `SeqExpressionSet` (EDASeq), `CountDataSet` (DESeq), `DGEList` (edgeR), `DESeqDataSet` (DESeq2), `SeqCountSet` (DSS) or `ABSDataSet` (ABSSeq).

Regarding `contrastList` it can also be a named structured list of contrasts as returned by the internal function `metaseqR2:::makeContrastList`.

Value

A named list of p-values, whose names are the names of the contrasts.

Author(s)

Panagiotis Moulos

Examples

```
require(limma)
dataMatrix <- metaseqR2:::exampleCountData(2000)
sampleList <- list(A=c("A1","A2"),B=c("B1","B2","B3"))
contrast <- "A_vs_B"
normDataMatrix <- normalizeEdger(dataMatrix,sampleList)
p <- statLimma(normDataMatrix,sampleList,contrast)
```

---

statNbpseq *Statistical testing with NBPSeq*

Description

This function is a wrapper over NBPSeq statistical testing. It accepts a matrix of normalized gene counts or an S4 object specific to each normalization algorithm supported by metaseqR2.

Usage

```
statNbpseq(object, sampleList, contrastList = NULL,
    statArgs = NULL, libsizeList = NULL)
```

Arguments

`object`	a matrix or an object specific to each normalization algorithm supported by metaseqR2, containing normalized counts. See also Details.
`sampleList`	the list containing condition names and the samples under each condition.
`contrastList`	vector of contrasts as defined in the main help page of `metaseqr2`. See also Details.
`statArgs`	a list of NBPSeq statistical algorithm parameters. See the result of `getDefaults("statistics", "nbpseq")` for an example and how you can modify it. It is not required when the input object is already a list from NBPSeq normalization as the dispersions are already estimated.
`libsizeList`	an optional named list where names represent samples (MUST be the same as the samples in `sampleList`) and members are the library sizes (the sequencing depth) for each sample. If not provided, the default is the column sums of the `object` matrix.

Details

Regarding `object`, apart from `matrix` (also for NOISeq), the object can be a `SeqExpressionSet` (EDASeq), `CountDataSet` (DESeq), `DGEList` (edgeR), `DESeqDataSet` (DESeq2), `SeqCountSet` (DSS) or `ABSDataSet` (ABSSeq).

Regarding `contrastList` it can also be a named structured list of contrasts as returned by the internal function `metaseqR2:::makeContrastList`.

Value

A named list of p-values, whose names are the names of the contrasts.

Note

There is currently a problem with the NBPSeq package and the workflow that is specific to the NBPSeq package. The problem has to do with function exporting as there are certain functions which are not recognized from the package internally. For this reason and until it is fixed, only the Smyth workflow will be available with the NBPSeq package.

Author(s)

Panagiotis Moulos

Examples

```
require(NBPSeq)
dataMatrix <- metaseqR2:::exampleCountData(2000)
sampleList <- list(A=c("A1","A2"),B=c("B1","B2","B3"))
contrast <- "A_vs_B"
normDataMatrix <- normalizeNbpseq(dataMatrix,sampleList)
p <- statNbpseq(normDataMatrix,sampleList,contrast)
```

---

statNoiseq *Statistical testing with NOISeq*

Description

This function is a wrapper over NOISeq statistical testing. It accepts a matrix of normalized gene counts or an S4 object specific to each normalization algorithm supported by metaseqR2.

Usage

```
statNoiseq(object, sampleList, contrastList = NULL,
    statArgs = NULL, geneData = NULL, logOffset = 1)
```

Arguments

`object`	a matrix or an object specific to each normalization algorithm supported by metaseqR2, containing normalized counts. See also Details.
`sampleList`	the list containing condition names and the samples under each condition.
`contrastList`	vector of contrasts as defined in the main help page of `metaseqr2`. See also Details.
`statArgs`	a list of edgeR statistical algorithm parameters. See the result of `getDefaults("statistics", "noiseq")` for an example and how you can modify it.

geneData — an optional annotation data frame (such the ones produced by `get.annotation` which contains the GC content for each gene and from which the gene lengths can be inferred by chromosome coordinates.

logOffset — a number to be added to each element of data matrix in order to avoid Infinity on log type data transformations.

Details

Regarding `object`, apart from `matrix` (also for NOISeq), the object can be a `SeqExpressionSet` (EDASeq), `CountDataSet` (DESeq), `DGEList` (edgeR), `DESeqDataSet` (DESeq2), `SeqCountSet` (DSS) or `ABSDataSet` (ABSSeq).

Regarding `contrastList` it can also be a named structured list of contrasts as returned by the internal function `metaseqR2:::makeContrastList`.

Value

A named list of NOISeq q-values, whose names are the names of the contrasts.

Author(s)

Panagiotis Moulos

Examples

```
dataMatrix <- metaseqR2:::exampleCountData(1000)
sampleList <- list(A=c("A1","A2"),B=c("B1","B2","B3"))
contrast <- "A_vs_B"
lengths <- round(1000*runif(nrow(dataMatrix)))
starts <- round(1000*runif(nrow(dataMatrix)))
ends <- starts + lengths
gc=runif(nrow(dataMatrix))
biotype=rep("protein_coding",nrow(dataMatrix))
geneData <- data.frame(
    chromosome=c(rep("chr1",nrow(dataMatrix)/2),
    rep("chr2",nrow(dataMatrix)/2)),
        start=starts,end=ends,gene_id=rownames(dataMatrix),
    gc_content=gc,biotype=biotype
)
normArgs <- metaseqR2:::getDefaults("normalization","noiseq")
normDataMatrix <- normalizeNoiseq(dataMatrix,sampleList,normArgs,
    geneData)
p <- statNoiseq(normDataMatrix,sampleList,contrast,
    geneData=geneData)
```

Index

∗ **datasets**
hg19pvalues, 37
libsizeListMm9, 38
mm9GeneCounts, 63
sampleListMm9, 75

buildAnnotationDatabase, 3, *34*, *39*, *57, 58*
buildCustomAnnotation, 5, *37*

combineBonferroni, 8
combineHarmonic, 9
combineMaxp, 10
combineMinp, 10
combineSimes, 11
combineWeight, 12
cor, *22*
createSignalTracks, 13

data.frame, *6*, *39*
diagplotAvgFtd, 14
diagplotBoxplot, 15
diagplotCor, 17
diagplotDeHeatmap, 18
diagplotEdaseq, 19
diagplotFiltered, 20, *51*
diagplotFtd, *14*, 21
diagplotMds, 22
diagplotNoiseq, 23
diagplotPairs, 25
diagplotRoc, 26
diagplotVenn, 27
diagplotVolcano, 28
downsampleCounts, 30

estimateAufcWeights, 30, *36*
estimateSimParams, *31*, 32, *40*

getAnnotation, *20*, *23*, 33, *60*, *72*
getDefaults, 34, *54*
getInstalledAnnotations, 35
getWeights, 36

hg19pvalues, 37

importCustomAnnotation, 37

libsizeListMm9, 38
loadAnnotation, 39

makeSimDataSd, *21*, *26*, *31*, 40
makeSimDataTcc, 42
metaseqr2, *4*, *8*, *11*, *13*, *21*, *26*, *31*, *33*, *35*, *37*, *39*, *41*, 42, *60*, *62*, *72*, *76–83*
metaseqrPlot, *19*, *23*, 59
metaTest, 61
mm9GeneCounts, 63

normalizeAbsseq, 63
normalizeDeseq, 64
normalizeDeseq2, 65
normalizeDss, 66
normalizeEdaseq, 67
normalizeEdger, 69
normalizeNbpseq, 70
normalizeNoiseq, 71

optimize, *32*

p.adjust.methods, *46*
par, *14*, *16–22*, *24–26*, *28, 29*, *49*, *60*

read2count, *57, 58*, 72, *74*
readTargets, *13*, *72*, 73
render, *52*

sampleListMm9, 75
save, *49*
statAbsseq, 75
statDeseq, 76
statDeseq2, 77
statDss, 79
statEdger, 80
statLimma, 81
statNbpseq, 82
statNoiseq, 83