Package ‘metaseqR’

April 10, 2015

Type Package

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

Author Panagiotis Moulos <moulos@fleming.gr>

Maintainer Panagiotis Moulos <moulos@fleming.gr>

Depends R (>= 2.13.0), EDASeq, DESeq, limma, qvalue

Imports edgeR, NOISeq, baySeq, NBPSeq, biomaRt, utils, gplots, corrplot, vsn, brew, rjson, log4r

Suggests BiocGenerics, GenomicRanges, rtracklayer, Rsamtools, MADAM, survcomp, VennDiagram, knitr, zoo, RUnit, BiocInstaller, BSgenome, RSQLite

Enhances parallel, TCC, RMySQL

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

LazyLoad yes

LazyData yes

URL http://www.fleming.gr

biocViews Software, GeneExpression, DifferentialExpression, WorkflowStep, Preprocessing, QualityControl, Normalization, ReportWriting, RNASeq

VignetteBuilder knitr

Version 1.4.13

Date 2015-01-09

Collate 'zzz.R' 'metaseqr.argcheck.R' 'metaseqr-data.R' 'metaseqr.export.R' 'metaseqr.filter.R' 'metaseqr.main.R' 'metaseqr.meta.R' 'metaseqr.norm.R' 'metaseqR-package.R' 'metaseqr.plot.R' 'metaseqr.query.R' 'metaseqr.sim.R' 'metaseqr.stat.R' 'metaseqr.util.R'

R topics documented:

metaseqR-package *The metaseqR Package*

Description

An R package for the analysis and result reporting of RNA-Seq gene expression data, using multiple statistical algorithms.

Details

Package:	metaseqR
Type:	Package
Version:	0.99.1
Date:	2014-03-11
Depends:	R (>= 2.13.0), EDASeq, DESeq, limma, NOISeq, baySeq
Encoding:	UTF-8
License:	GPL (>= 3)
LazyLoad:	yes
URL:	http://www.fleming.gr

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.

Author(s)

Panagiotis Moulos <moulos@fleming.gr>

as.class.vector *Create a class vector*

Description

Creates a class vector from a sample list. Internal to the `stat.*` functions. Mostly internal use.

Usage

```
as.class.vector(sample.list)
```

Arguments

`sample.list` the list containing condition names and the samples under each condition.

Value

A vector of condition names.

Author(s)

Panagiotis Moulos

Examples

```
sample.list <- list(A=c("A1","A2"),B=c("B1","B2","B3"))
clv <- as.class.vector(sample.list)
```

---

build.export *Results export builder*

Description

This function help build the output files of the metaseqr pipeline based on several elements produced during the pipeline execution. It is intended for internal use and not available to the users.

Usage

```
    build.export(gene.data, raw.gene.counts,
        norm.gene.counts, flags, sample.list, cnt,
        statistics, raw.list, norm.list,
        p.mat = matrix(NA, nrow(gene.data), length(statistics)),
        adj.p.mat = matrix(NA, nrow(gene.data), length(statistics)),
        sum.p = rep(NA, nrow(gene.data)),
        adj.sum.p = rep(NA, nrow(gene.data)),
        export.what = c("annotation", "p.value", "adj.p.value", "meta.p.value",
                "adj.meta.p.value", "fold.change", "stats", "counts","flags"),
        export.scale = c("natural", "log2", "log10", "vst"),
        export.values = c("raw", "normalized"),
        export.stats = c("mean", "median", "sd", "mad", "cv", "rcv"),
        log.offset = 1, report = TRUE)
```

Arguments

gene.data	an annotation data frame (such the ones produced by get.annotation).
raw.gene.counts	a matrix of filtering flags (0,1), created by the filtering functions.
norm.gene.counts	a matrix of normalized gene counts.
flags	a matrix of normalized gene counts.
sample.list	see the documentation of metaseqr.
cnt	the statistical contrast for which the export builder is currently running.

`statistics`	the statistical tests used (see the documentation of `metaseqr`).
`raw.list`	a list of transformed un-normalized counts, see the documentation of `make.transformation`.
`norm.list`	a list of transformed normalized counts, see the documentation of `make.transformation`.
`p.mat`	a matrix of p-values, see the documentation of `metaseqr`.
`adj.p.mat`	a matrix of adjusted p-values, see the documentation of `metaseqr`.
`sum.p`	a vector of combined p-values, see the documentation of `metaseqr`.
`adj.sum.p`	a vector of adjusted combined p-values, see the documentation of `metaseqr`.
`export.what`	see the documentation of `metaseqr`.
`export.scale`	see the documentation of `metaseqr`.
`export.values`	see the documentation of `metaseqr`.
`export.stats`	see the documentation of `metaseqr`.
`log.offset`	see the documentation of `metaseqr`.
`report`	see the documentation of `metaseqr`.

Value

A list with three members: a data frame to be exported in a text file, a long string with the result in a html formatted table (if `report=TRUE`) and the column names of the output data frame.

Author(s)

Panagiotis Moulos

Examples

```
## Not run:
# Not yet available

## End(Not run)
```

`calc.f1score` *Calculate the F1-score*

Description

This function calculates the F1 score (2*(precision*recall/precision+racall) or 2*TP/(2*TP+FP+FN) given a matrix of p-values (one for each statistical test used) and a vector of ground truth (DE or non-DE). This function serves as a method evaluation helper.

Usage

```
calc.f1score(truth, p, sig = 0.05)
```

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 `make.sim.data.sd` function, which creates simulated RNA-Seq read counts based on real data. It MUST be named with gene names, the same as in `p`.
`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. In any case, each row (or element) MUST be named with gene names (the same as in `truth`).
`sig`	a significance level (0 < `sig` <=1).

Value

A named list with two members. The first member is a data frame with the numbers used to calculate the TP/(FP+FN) ratio and the second member is the ratio TP/(FP+FN) for each statistical test.

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)
f1 <- calc.f1score(truth,p)
```

`calc.otr` *Calculate the ratio TP/(FP+FN)*

Description

This function calculates the ratio of True Positives to the sum of False Positives and False Negatives given a matrix of p-values (one for each statistical test used) and a vector of ground truth (DE or non-DE). This function serves as a method evaluation helper.

Usage

```
calc.otr(truth, p, sig = 0.05)
```

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 make.sim.data.sd function, which creates simulated RNA-Seq read counts based on real data. It MUST be named with gene names, the same as in p.
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. In any case, each row (or element) MUST be named with gene names (the same as in truth).
sig	a significance level (0 < sig <=1).

Value

A named list with two members. The first member is a data frame with the numbers used to calculate the TP/(FP+FN) ratio and the second member is the ratio TP/(FP+FN) for each statistical test.

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)
otr <- calc.otr(truth,p)
```

---

cddat *Old functions from NOISeq*

Description

Old functions from NOISeq to create the "readnoise" plots. Internal use only.

Usage

```
cddat(input)
```

Arguments

`input`	input to cddat.

Value

a list with data to plot.

Note

Adopted from an older version of NOISeq package (author: Sonia Tarazona).

Author(s)

Panagiotis Moulos

---

cdplot *Old functions from NOISeq*

Description

Old functions from NOISeq to create the `"readnoise"` plots. Internal use only.

Usage

```
cdplot(dat, samples = NULL, ...)
```

Arguments

`dat`	the returned list from `cddat`.
`samples`	the samples to plot.
`...`	further arguments passed to e.g. `par`.

Value

Nothing, it created the old RNA composition plot.

Note

Adopted from an older version of NOISeq package (author: Sonia Tarazona)

Author(s)

Panagiotis Moulos

check.contrast.format *Contrast validator*

Description

Checks if the contrast vector follows the specified format. Internal use only.

Usage

```
check.contrast.format(cnt, sample.list)
```

Arguments

`cnt`	contrasts vector.
`sample.list`	the input sample list.

Author(s)

Panagiotis Moulos

Examples

```
sample.list <- list(A=c("A1","A2"),B=c("B1","B2","B3"))
cnt <- c("A_vs_B") # Will work
#cnt <- c("A_vs_C") ## Will throw error!
check.contrast.format(cnt,sample.list)
```

check.file.args *File argument validator*

Description

Checks if a file exists for specific arguments requiring a file input. Internal use only.

Usage

```
check.file.args(arg.name, arg.value)
```

Arguments

`arg.name`	argument name to display in a possible error.
`arg.value`	the filename to check.

Author(s)

Panagiotis Moulos

Examples

```
# OK
check.file.args("file",system.file("metaseqr_report.html",
    package="metaseqR"))
## Error!
#check.file.args("file",system.file("metaseqr_report.htm",
#    package="metaseqR"))
```

---

check.graphics.file *Check graphics file*

Description

Graphics file checker. Internal use only.

Usage

```
check.graphics.file(o)
```

Arguments

o	the plotting device, see main metaseqr function

Author(s)

Panagiotis Moulos

---

check.graphics.type *Check plotting device*

Description

Plotting device checker. Internal use only.

Usage

```
check.graphics.type(o)
```

Arguments

o	the plotting device, see main metaseqr function

Author(s)

Panagiotis Moulos

check.libsize — *Library size validator*

Description

Checks the names of the supplied library sizes. Internal use only.

Usage

```
check.libsize(libsize.list, sample.list)
```

Arguments

`libsize.list`	the samples-names library size list.
`sample.list`	the input sample list.

Author(s)

Panagiotis Moulos

Examples

```
sample.list <- list(A=c("A1","A2"),B=c("B1","B2","B3"))
libsize.list.1 <- list(A1=1e+6,A2=1.1e+6,B1=1.2e+6,
B2=1.3e+6,B3=1.5e+6)
libsize.list.2 <- list(A1=1e+6,A2=1.1e+6,B1=1.2e+6,
B2=1.3e+6)
check.libsize(libsize.list.1,sample.list) # Will work
#check.libsize(libsize.list.2,sample.list) # Will throw error!
```

check.main.args — *Main argument validator*

Description

Checks if the arguments passed to `metaseqr` are valid and throws a warning about the invalid ones (which are ignored anyway because of the `...` in `metaseqr`. However, for this reason this function is useful as some important parameter faults might go unnoticed in the beginning and cause a failure afterwards.

Usage

```
check.main.args(main.args)
```

Arguments

main.args — a list of parameters with which metaseqr is called (essentially, the output of match.call.

Author(s)

Panagiotis Moulos

---

check.num.args — *Numeric argument validator*

Description

Checks if one or more given numeric argument(s) satisfy several rules concerning numeric arguments, e.g. proper bounds or proper format (e.g. it must be a number and not a character). Mostly for internal use.

Usage

```
check.num.args(arg.name, arg.value, arg.type, arg.bounds,
    direction)
```

Arguments

arg.name	the name of the argument that is checked (for display purposes).
arg.value	the value(s) of the argument to be checked.
arg.type	either the string "numeric" to denote generic double-like R numerics or "integer" for integer values.
arg.bounds	a numeric or a vector with 2 elements, restraining arg.value to be within the bounds defined by the input vector or e.g. larger (smaller) than the numeric value. See examples.
direction	a string denoting to which direction the arg.value should be compared with arg.bounds. For example, "both" should be given with a two element vector against which, arg.value will be checked to see whether it is smaller than the low boundary or larger than the higher boundary. In that case, the function will throw an error. The direction parameter can be one of: "both" (described above), "botheq" (as above, but the arg.val is also checked for equality - closed intervals), "gt" or "gte" (check whether arg.val is smaller or smaller than or equal to the first value of arg.bounds), "lt" or "lte" (check whether arg.val is larger or larger than or equal to the first value of arg.bounds).

Author(s)

Panagiotis Moulos

Examples

```
pcut <- 1.2 # A probability cannot be larger than 1! It will throw an error!
#check.num.args("pcut",pcut,"numeric",c(0,1),"botheq")
pcut <- 0.05 # Pass
check.num.args("pcut",pcut,"numeric",c(0,1),"botheq")
gc.col <- 3.4 # A column in a file cannot be real! It will throw an error!
#check.num.args("gc.col",gc.col,"integer",0,"gt")
gc.col <- 5L # Pass
check.num.args("gc.col",gc.col,"integer",0,"gt")
```

---

check.packages *Required packages validator*

Description

Checks if all the any required packages, not attached during installation or loading, are present according to metaseqR input options. Internal use only.

Usage

```
check.packages(m, p)
```

Arguments

m	meta-analysis method.
p	QC plot types.

Author(s)

Panagiotis Moulos

Examples

```
check.packages(c("simes","whitlock"),
    c("gcbias","correl"))
```

check.parallel — *Parallel run validator*

Description

Checks existence of multiple cores and loads multicore package.

Usage

```
check.parallel(rc)
```

Arguments

rc	fraction of available cores to use.

Author(s)

Panagiotis Moulos

Examples

```
multic <- check.parallel(0.8)
```

check.text.args — *Text argument validator*

Description

Checks if one or more given textual argument(s) is/are member(s) of a list of correct arguments. It's a more package-specific function similar to `match.arg`. Mostly for internal use.

Usage

```
check.text.args(arg.name, arg.value, arg.list,
    multiarg=FALSE)
```

Arguments

arg.name	the name of the argument that is checked (for display purposes).
arg.value	the value(s) of the argument to be checked.
arg.list	a vector of valid argument values for `arg.value` to be matched against.
multiarg	a logical scalar indicating whether `arg.name` accepts multiple arguments or not. In that case, all of the values in `arg.value` are checked against `arg.list`.

Author(s)

Panagiotis Moulos

Examples

```
# OK
check.text.args("count.type","gene",c("gene","exon"),
    multiarg=FALSE)
## Error!
#check.text.args("statistics","ebseq",c("deseq","edger",
#    "noiseq","bayseq","limma"), multiarg=TRUE)
```

---

combine.bonferroni *Combine p-values with Bonferroni's method*

Description

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

Usage

```
combine.bonferroni(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 <- combine.bonferroni(p)
```

combine.maxp *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

```
combine.maxp(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 <- combine.maxp(p)
```

combine.minp *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

```
combine.minp(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 <- combine.minp(p)
```

---

combine.simes *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 `metaseqr` help page or in the vignette).

Usage

```
combine.simes(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 <- combine.simes(p)
```

combine.weight *Combine p-values using weights*

Description

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

Usage

```
combine.weight(p, w)
```

Arguments

p	a p-value matrix (rows are genes, columns are statistical tests).
w	a weights vector, must sum to 1.

Value

A vector of combined p-values.

Author(s)

Panagiotis Moulos

Examples

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

construct.gene.model *Assemble a gene model based on exon counts*

Description

This function assembles gene models (single genes, not isoforms) based on the input exon read counts file and a gene annotation data frame, either from an external file provided by the user, or with the get.annotation function. The gene.data argument should have a specific format and for this reason it's better to use one of the two aforementioned ways to supply it. This function is intended mostly for internal use but can be used if the requirements are met.

Usage

```
construct.gene.model(exon.counts, sample.list, gene.data,
    multic = FALSE)
```

Arguments

- `exon.counts` the exon counts data frame produced by reading the exon read counts file.
- `sample.list` the list containing condition names and the samples under each condition.
- `gene.data` an annotation data frame from the same organism as `exon.counts` (such the ones produced by `get.annotation`).
- `multic` a logical value indicating the presence of multiple cores. Defaults to `FALSE`. Do not change it if you are not sure whether package multicore has been loaded or not.

Value

A named list where names represent samples. Each list member is a also a named list where names correspond to gene ids and members are named vectors. Each vector is named according to the exons corresponding to each gene and contains the read counts for each exon. This structure is used for exon filtering and assembling final gene counts in the metaseqr pipeline.

Author(s)

Panagiotis Moulos

Examples

```
# Takes some time to run...
data("hg19.exon.data",package="metaseqR")
gene.data <- get.annotation("hg19","gene","ensembl")
reduced.gene.data <- reduce.gene.data(hg19.exon.counts,
    gene.data)
multic <- check.parallel(0.4)
gene.model <- construct.gene.model(hg19.exon.counts,
    sample.list.hg19,gene.data,multic)
```

`diagplot.avg.ftd` *Create average False (or True) Discovery curves*

Description

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

Usage

```
diagplot.avg.ftd(ftdr.obj, output = "x11",
    path = NULL, draw = TRUE, ...)
```

Arguments

`ftdr.obj` a list with outputs from `diagplot.ftd`.

`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 (`avg.ftdr`) contains a list with the means and the standard deviations of the averaged `ftdr.obj` 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)
ftd.obj.1 <- diagplot.ftd(truth,p1,N=100,draw=FALSE)
ftd.obj.2 <- diagplot.ftd(truth,p2,N=100,draw=FALSE)
ftd.obj.3 <- diagplot.ftd(truth,p3,N=100,draw=FALSE)
ftd.obj <- list(ftd.obj.1,ftd.obj.2,ftd.obj.3)
avg.ftd.obj <- diagplot.avg.ftd(ftd.obj)
```

diagplot.boxplot *Boxplots wrapper for the metaseqR 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

```
diagplot.boxplot(mat, name = NULL, log.it = "auto",
    y.lim = "default", is.norm = FALSE, output = "x11",
    path = NULL, ...)
```

Arguments

`mat`	the count data matrix.
`name`	the names of the samples plotted on the boxdiagplot. 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 sample.list argument provided to the manin metaseqr function. In that case, the boxes are colored per group.
`log.it`	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.
`y.lim`	custom y-axis limits. Leave the string `"default"` for default behavior.
`is.norm`	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.
`...`	further arguments to be passed to plot devices, such as parameter from `par`.

Value

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

Author(s)

Panagiotis Moulos

Examples

```
# Non-normalized boxplot
require(DESeq)
data.matrix <- counts(makeExampleCountDataSet())
sample.list <- list(A=c("A1","A2"),B=c("B1","B2","B3"))
diagplot.boxplot(data.matrix,sample.list)

# Normalized boxplot
norm.args <- get.defaults("normalization","deseq")
object <- normalize.deseq(data.matrix,sample.list,norm.args)
diagplot.boxplot(object,sample.list)
```

---

diagplot.cor *Summarized correlation plots*

Description

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

Usage

```
diagplot.cor(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

```
require(DESeq)
data.matrix <- counts(makeExampleCountDataSet())
diagplot.cor(data.matrix,type="heatmap")
diagplot.cor(data.matrix,type="correlogram")
```

`diagplot.de.heatmap` *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

```
diagplot.de.heatmap(x, con = NULL, output = "x11",
    path = NULL, ...)
```

Arguments

`x`	the data matrix to create a heatmap for.
`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 `which.plot` argument. If `output="x11"`, no output filenames are produced.

Author(s)

Panagiotis Moulos

Examples

```
require(DESeq)
data.matrix <- counts(makeExampleCountDataSet())
sample.list <- list(A=c("A1","A2"),B=c("B1","B2","B3"))
contrast <- "A_vs_B"
M <- normalize.edger(data.matrix,sample.list)
p <- stat.edger(M,sample.list,contrast)
diagplot.de.heatmap(data.matrix[p[[1]]<0.05,])
```

diagplot.edaseq *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 diagplot.metaseqr.

Usage

```
diagplot.edaseq(x, sample.list, covar = NULL,
    is.norm = FALSE,
    which.plot = c("meanvar", "meandiff", "gcbias", "lengthbias"),
    output = "x11", path = NULL, ...)
```

Arguments

- `x` the count data matrix.
- `sample.list` the list containing condition names and the samples under each condition.
- `covar` The covariate to plot counts against. Usually "gc" or "length".
- `is.norm` a logical indicating whether object contains raw or normalized data. It is not essential and it serves only plot annotation purposes.
- `which.plot` the EDASeq package plot to generate. 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 which.plot="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.
- `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 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

```
require(DESeq)
data.matrix <- counts(makeExampleCountDataSet())
sample.list <- list(A=c("A1","A2"),B=c("B1","B2","B3"))
diagplot.edaseq(data.matrix,sample.list,which.plot="meandiff")
```

---

diagplot.filtered *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

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

Arguments

`x`	an annotation data frame like the ones produced by `get.annotation`. `x` should be the filtered annotation according to metaseqR's filters.
`y`	an annotation data frame like the ones produced by `get.annotation`. `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

```
y <- get.annotation("mm9","gene")
x <- y[-sample(1:nrow(y),10000),]
diagplot.filtered(x,y)
```

diagplot.ftd *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 metaseqr by subsetting the table to get the p-values from several algorithms) given a ground truth vector for differential expression.

Usage

```
diagplot.ftd(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 make.sim.data.sd 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)
ftd.obj <- diagplot.ftd(truth,p,N=100)
```

---

diagplot.mds *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

```
diagplot.mds(x, sample.list, method = "spearman",
    log.it = TRUE, output = "x11", path = NULL, ...)
```

Arguments

`x`	the count data matrix.
`sample.list`	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.
`log.it`	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

```
require(DESeq)
data.matrix <- counts(makeExampleCountDataSet())
sample.list <- list(A=c("A1","A2"),B=c("B1","B2","B3"))
diagplot.mds(data.matrix,sample.list)
```

---

diagplot.metaseqr *Diagnostic plots for the metaseqR 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. diagplot.metaseqr, 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

```
diagplot.metaseqr(object, sample.list, annotation = NULL,
    contrast.list = NULL, p.list = NULL,
    thresholds = list(p = 0.05, f = 1),
    diagplot.type = c("mds", "biodetection", "countsbio", "saturation",
    "readnoise", "rnacomp", "correl", "pairs", "boxplot", "gcbias",
    "lengthbias", "meandiff", "meanvar", "deheatmap", "volcano",
    "biodist", "filtered", "venn"),
    is.norm = 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 metaseqR's filters and/or p-value. The object can be fed to any of the `diagplot.metaseqr` 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.
`sample.list`	the list containing condition names and the samples under each condition.
`annotation`	a data frame containing annotation elements for each row in object. Usually, a subset of the annotation obtained by `get.annotation` or a subset of possibly embedded annotation with the input counts table. This parameter is optional and required only when diagplot.type is any of `"biodetection"`, `"countsbio"`, `"saturation"`, `"rnacomp"`, `"readnoise"`, `"biodist"`, `"gcbias"`, `"lengthbias"` or `"filtered"`.

contrast.list	a named structured list of contrasts as returned by make.contrast.list or just the vector of contrasts as defined in the main help page of metaseqr. This parameter is optional and required only when diagplot.type is any of "deheatmap", "volcano" or "biodist".
p.list	a list of p-values for each contrast as obtained from any of the stat.* methods of the metaseqr package. This parameter is optional and required only when diagplot.type is any of "deheatmap", "volcano" or "biodist".
thresholds	a list with the elements "p" and "f" which are the p-value and the fold change cutoff when diagplot.type="volcano".
diagplot.type	one or more of the diagnostic plots supported in metaseqR package. Many of these plots require the presence of additional package, something that is checked while running the main metaseqr 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 metaseqr help page.
is.norm	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: "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.
path	the path to create output files.
...	further arguments to be passed to plot devices, such as parameter from par.

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 set the annotation parameter of the main metaseqr function to "download" to use predefined annotations that work better with the functions of the whole package.

Author(s)

Panagiotis Moulos

Examples

```
require(DESeq)
data.matrix <- counts(makeExampleCountDataSet())
sample.list <- list(A=c("A1","A2"),B=c("B1","B2","B3"))
contrast <- "A_vs_B"
diagplot.metaseqr(data.matrix,sample.list,diagplot.type=c("mds","boxplot"))

norm.args <- get.defaults("normalization","deseq")
object <- normalize.deseq(data.matrix,sample.list,norm.args)
diagplot.metaseqr(object,sample.list,diagplot.type="boxplot")

p <- stat.deseq(object,sample.list)
diagplot.metaseqr(object,sample.list,contrast.list=contrast,p.list=p,
        diagplot.type="volcano")
```

`diagplot.noiseq` *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 `diagplot.metaseqr`.

Usage

```
diagplot.noiseq(x, sample.list, covars,
    which.plot = c("biodetection", "countsbio", "saturation", "rnacomp",
            "biodist"),
    output = "x11",
    biodist.opts = list(p = NULL, pcut = NULL, name = NULL),
    path = NULL, is.norm = FALSE, ...)
```

Arguments

`x`	the count data matrix.
`sample.list`	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 `get.annotation`) 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).

`which.plot` the NOISeq package plot to generate. It can be one or more of "biodetection", "countsbio", "saturation", "rnacomp", "readnoise" or "biodist". Please refer to the documentation of the EDASeq package for details on the use of these plots. The which.plot="saturation" case is modified to be more informative by producing two kinds of plots. See diagplot.noiseq.saturation.

`biodist.opts` 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.

`is.norm` 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.

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.

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

```
require(DESeq)
data.matrix <- counts(makeExampleCountDataSet())
sample.list <- list(A=c("A1","A2"),B=c("B1","B2","B3"))
lengths <- round(1000*runif(nrow(data.matrix)))
starts <- round(1000*runif(nrow(data.matrix)))
ends <- starts + lengths
covars <- list(
    data=data.matrix,
    length=lengths,
    gc=runif(nrow(data.matrix)),
    chromosome=data.frame(
        chromosome=c(rep("chr1",nrow(data.matrix)/2),
            rep("chr2",nrow(data.matrix)/2)),
        start=starts,
```

```
        end=ends
    ),
    factors=data.frame(class=as.class.vector(sample.list)),
    biotype=c(rep("protein_coding",nrow(data.matrix)/2),rep("ncRNA",
            nrow(data.matrix)/2))
)
p <- runif(nrow(data.matrix))
diagplot.noiseq(data.matrix,sample.list,covars=covars,
        biodist.opts=list(p=p,pcut=0.1,name="A_vs_B"))
```

---

diagplot.noiseq.saturation *Simpler implementation of saturation plots inspired from NOISeq package*

Description

Helper function for diagplot.noiseq to plot feature detection saturation as presented in the NOISeq package vignette. It has two main outputs: a set of figures, one for each input sample depicting the saturation for each biotype and one single multiplot which depicts the saturation of all samples for each biotype. It expands the saturation plots of NOISeq by allowing more samples to be examined in a simpler way. Don't use this function directly. Use either diagplot.metaseqr or diagplot.noiseq.

Usage

```
diagplot.noiseq.saturation(x, o, tb, path = NULL)
```

Arguments

x	the count data matrix.
o	one or more R plotting device to direct the plot result to. Supported mechanisms: "x11" (default), "png", "jpg", "bmp", "pdf" or "ps".
tb	the vector of biotypes, one for each row of x.
path	the path to create output files.

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

`diagplot.pairs` *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

```
diagplot.pairs(x, output = "x11", 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"`.
`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

```
require(DESeq)
data.matrix <- counts(makeExampleCountDataSet())
diagplot.pairs(data.matrix)
```

diagplot.roc *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 metaseqr 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

```
diagplot.roc(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 make.sim.data.sd 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)
roc.obj <- diagplot.roc(truth,p)
```

---

diagplot.venn *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

```
diagplot.venn(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, alt.names = NULL, ...)
```

Arguments

pmat	a matrix with p-values corresponding to the application of each statistical algorithm. 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")`).
fcmat	an optional matrix with fold changes corresponding to the application of each statistical algorithm. 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`).
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. 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.

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".

alt.names
: 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.

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

```
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="_")
venn.contents <- diagplot.venn(p)
```

---

diagplot.volcano *(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

```
diagplot.volcano(f, p, con = NULL, fcut = 1, pcut = 0.05,
    alt.names = 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.
`alt.names`	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

```
require(DESeq)
data.matrix <- counts(makeExampleCountDataSet())
sample.list <- list(A=c("A1","A2"),B=c("B1","B2","B3"))
contrast <- "A_vs_B"
M <- normalize.edger(data.matrix,sample.list)
p <- stat.edger(M,sample.list,contrast)
ma <- apply(M[,sample.list$A],1,mean)
mb <- apply(M[,sample.list$B],1,mean)
f <- log2(ifelse(mb==0,1,mb)/ifelse(ma==0,1,ma))
diagplot.volcano(f,p[[1]],con=contrast)
j <- diagplot.volcano(f,p[[1]],con=contrast,output="json")
```

disp *Message displayer*

Description

Displays a message during execution of the several functions. Internal use.

Usage

```
disp(...)
```

Arguments

`...`	a vector of elements that compose the display message.

Author(s)

Panagiotis Moulos

Examples

```
i <- 1
disp("Now running iteration ",i,"...")
```

downsample.counts *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

```
downsample.counts(counts,seed=42)
```

Arguments

`counts`	the read counts table which is subjected to downsampling.
`seed`	random seed for reproducible downsampling.

Value

The downsampled counts matrix.

Author(s)

Panagiotis Moulos

Examples

```
# Dowload locally the file "bottomly_count_table.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")
M <- as.matrix(read.delim("~/bottomly_count_table.txt",row.names=1))
D <- downsample.counts(M)
```

---

estimate.aufc.weights *Estimate AUFC weights*

Description

This function automatically estimates weights for the "weight" and "dperm.weight" options of metaseqR for combining p-values from multiple statistical tests. It creates simulated dataset based on real data and then performs statistical analysis with metaseqR 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

```
estimate.aufc.weights(counts, normalization,
    statistics, nsim = 10, N = 10000,
    samples = c(3, 3), ndeg = c(500, 500),
    top = 500, model.org = "mm9", fc.basis=1.5,
    seed = NULL, draw.fpc = FALSE, multic = FALSE,
    ...)
```

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 metaseqr.
statistics	same as statistics in metaseqr.
nsim	the number of simulations to perform to estimate the weights. It default to 10.
N	the number of genes to produce. See make.sim.data.sd.

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.
fc.basis	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.
model.org	the organism from which the data are derived. It must be one of metaseqr supported organisms.
seed	a list of seed for reproducible simulations. Defaults to NULL.
draw.fpc	draw the averaged false discovery curves? Default to FALSE.
multic	whether to run in parallel (if package parallel is present or not.
...	Further arguments to be passed to estimate.sim.params.

Value

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

Author(s)

Panagiotis Moulos

Examples

```
data("mm9.gene.data",package="metaseqR")
multic <- check.parallel(0.8)
weights <- estimate.aufc.weights(
    counts=as.matrix(mm9.gene.counts[,9:12]),
    normalization="edaseq",
    statistics=c("deseq","edger"),
    nsim=3,N=100,ndeg=c(10,10),top=10,model.org="mm9",
    seed=10,multic=multic,libsize.gt=1e+5
)
```

---

estimate.sim.params *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

```
estimate.sim.params(real.counts, libsize.gt = 3e+6,
    rowmeans.gt = 5,eps = 1e-11,
    restrict.cores = 0.8, seed = 42, draw = FALSE)
```

Arguments

`real.counts`	a text tab-delimited file with real RNA-Seq data. 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/`.
`libsize.gt`	a library size below which samples are excluded from parameter estimation (default: 3000000).
`rowmeans.gt`	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.
`restrict.cores`	in case of parallel optimization, the fraction of the available cores to use.
`seed`	a seed to use with random number generation for reproducibility.
`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).

Value

A named list with two members: `mu.hat` which contains negative binomial mean estimates and `phi.hat` which contains dispersion estimates.

Author(s)

Panagiotis Moulos

Examples

```
# Dowload locally the 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")
# Estimate simulation parameters
par.list <- estimate.sim.params("~/bottomly_count_table.txt")
```

filter.exons *Filter gene expression based on exon counts*

Description

This function performs the gene expression filtering based on exon read counts and a set of exon filter rules. For more details see the main help pages of `metaseqr`.

Usage

```
filter.exons(the.counts, gene.data, sample.list,
    exon.filters, restrict.cores = 0.8)
```

Arguments

`the.counts`	a named list created with the `construct.gene.model` function. See its help page for details.
`gene.data`	an annotation data frame usually obtained with `get.annotation` containing the unique gene accession identifiers.
`sample.list`	the list containing condition names and the samples under each condition.
`exon.filters`	a named list with exon filters and their parameters. See the main help page of `metaseqr` for details.
`restrict.cores`	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 (`restrict.cores=1` and the system does not have sufficient RAM, the running machine might significantly slow down.

Value

a named list with two members. The first member (`result` is a named list whose names are the exon filter names and its members are the filtered rownames of `gene.data`. The second member is a matrix of binary flags (0 for non-filtered, 1 for filtered) for each gene. The rownames of the flag matrix correspond to gene ids.

Author(s)

Panagiotis Moulos

Examples

```
data("hg19.exon.data",package="metaseqR")
exon.counts <- hg19.exon.counts
gene.data <- get.annotation("hg19","gene")
sample.list <- sample.list.hg19
exon.filters <- get.defaults("exon.filter")
the.counts <- construct.gene.model(exon.counts,sample.list,
```

```
    gene.data)
filter.results <- filter.exons(the.counts,gene.data,
    sample.list,exon.filters)
```

---

filter.genes *Filter gene expression based on gene counts*

Description

This function performs the gene expression filtering based on gene read counts and a set of gene filter rules. For more details see the main help pages of `metaseqr`.

Usage

```
filter.genes(gene.counts, gene.data, gene.filters)
```

Arguments

`gene.counts`	a matrix of gene counts, preferably after the normalization procedure.
`gene.data`	an annotation data frame usually obtained with `get.annotation` containing the unique gene accession identifiers.
`gene.filters`	a named list with gene filters and their parameters. See the main help page of `metaseqr` for details.

Value

a named list with three members. The first member (`result` is a named list whose names are the gene filter names and its members are the filtered rownames of `gene.data`. The second member (`cutoff` is a named list whose names are the gene filter names and its members are the cutoff values corresponding to each filter. The third member is a matrix of binary flags (0 for non-filtered, 1 for filtered) for each gene. The rownames of the flag matrix correspond to gene ids.

Author(s)

Panagiotis Moulos

Examples

```
data("mm9.gene.data",package="metaseqR")
gene.counts <- mm9.gene.counts
sample.list <- sample.list.mm9
gene.counts <- normalize.edger(as.matrix(gene.counts[,9:12]),
    sample.list)
gene.data <- get.annotation("mm9","gene")
gene.filters <- get.defaults("gene.filter","mm9")
filter.results <- filter.genes(gene.counts,gene.data,
    gene.filters)
```

filter.high *Filtering helper*

Description

High score filtering function. Internal use.

Usage

```
filter.high(x, f)
```

Arguments

x	a data numeric matrix.
f	a threshold.

Author(s)

Panagiotis Moulos

Examples

```
data("mm9.gene.data",package="metaseqR")
counts <- as.matrix(mm9.gene.counts[,9:12])
f <- filter.low(counts,median(counts))
```

filter.low *Filtering helper*

Description

Low score filtering function. Internal use.

Usage

```
filter.low(x, f)
```

Arguments

x	a data numeric matrix.
f	a threshold.

Author(s)

Panagiotis Moulos

Examples

```
data("mm9.gene.data",package="metaseqR")
counts <- as.matrix(mm9.gene.counts[,9:12])
f <- filter.low(counts,median(counts))
```

`get.annotation` *Annotation downloader*

Description

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. See the help page of `metaseqr` for a list of supported organisms. The function downloads annotation for an organism genes or exons.

Usage

```
get.annotation(org, type, refdb="ensembl",
    multic=FALSE)
```

Arguments

`org`	the organism for which to download annotation.
`type`	either `"gene"` or `"exon"`.
`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.
`multic`	a logical value indicating the presence of multiple cores. Defaults to `FALSE`. Do not change it if you are not sure whether package parallel has been loaded or not. It is used in the case of `type="exon"` to process the return value of the query to the UCSC Genome Browser database.

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. The gene_id and exon_id correspond to Ensembl gene 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

```
hg19.genes <- get.annotation("hg19","gene","ensembl")
mm9.exons <- get.annotation("mm9","exon","ucsc")
```

---

get.arg *Argument getter*

Description

Get argument(s) from a list of arguments, e.g. normalization arguments.

Usage

```
get.arg(arg.list, arg.name)
```

Arguments

arg.list	the initial list of a method's (e.g. normalization) arguments. Can be created with the get.defaults function.
arg.name	the argument name inside the argument list to fetch its value.

Value

The argument sub-list.

Author(s)

Panagiotis Moulos

Examples

```
norm.list <- get.defaults("normalization","egder")
a <- get.arg(norm.list,c("main.method","logratioTrim"))
```

`get.biotypes` *Biotype converter*

Description

Returns biotypes as character vector. Internal use.

Usage

```
get.biotypes(a)
```

Arguments

a	the annotation data frame (output of `get.annotation`).

Value

A character vector of biotypes.

Author(s)

Panagiotis Moulos

Examples

```
hg18.genes <- get.annotation("hg18","gene")
hg18.bt <- get.biotypes(hg18.genes)
```

`get.bs.organism` *Return a proper formatted BSgenome organism name*

Description

Returns a properly formatted BSgenome package name according to metaseqR's supported organism. Internal use.

Usage

```
get.bs.organism(org)
```

Arguments

org	one of metaseqR supported organisms.

Value

A proper BSgenome package name.

Author(s)

Panagiotis Moulos

Examples

```
bs.name <- get.bs.organism("hg18")
```

---

get.dataset *Annotation downloader helper*

Description

Returns a dataset (gene or exon) identifier for each organism recognized by the Biomart service for Ensembl. Internal use.

Usage

```
get.dataset(org)
```

Arguments

org	the organism for which to return the identifier.

Value

A string with the dataset identifier.

Author(s)

Panagiotis Moulos

Examples

```
dm3.id <- get.dataset("dm3")
```

get.defaults *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

```
get.defaults(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", "gene.filter", "exon.filter" or "biotype.filter".
method	the supported algorithm included in metaseqR for which to fetch the default settings. 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", "bayseq", "limma" or "nbpseq". When method is "biotype.filter", what is the input organism (see the main metaseqr 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

```
norm.args.edaseq <- get.defaults("normalization","edaseq")
stat.args.edger <- get.defaults("statistics","edger")
```

get.ensembl.annotation *Ensembl annotation downloader*

Description

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. See the help page of `metaseqr` for a list of supported organisms. The function downloads annotation for an organism genes or exons.

Usage

```
get.ensembl.annotation(org, type)
```

Arguments

`org`	the organism for which to download annotation.
`type`	either `"gene"` or `"exon"`.

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. The gene_id and exon_id correspond to Ensembl gene 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

```
hg19.genes <- get.ensembl.annotation("hg19","gene")
mm9.exons <- get.ensembl.annotation("mm9","exon")
```

get.exon.attributes — *Annotation downloader helper*

Description

Returns a vector of genomic annotation attributes which are used by the biomaRt package in order to fetch the exon annotation for each organism. It has no parameters. Internal use.

Usage

```
get.exon.attributes(org)
```

Arguments

`org`	one of the supported organisms.

Value

A character vector of Ensembl exon attributes.

Author(s)

Panagiotis Moulos

Examples

```
exon.attr <- get.exon.attributes("mm9")
```

get.gc.content — *Return a named vector of GC-content for each genomic region*

Description

Returns a named numeric vector (names are the genomic region names, e.g. genes) given a data frame which can be converted to a GRanges object (e.g. it has at least chromosome, start, end fields). This function works best when the input annotation data frame has been retrieved using one of the SQL queries generated from `get.ucsc.query`, used in `get.ucsc.annotation`.

Usage

```
get.gc.content(ann, org)
```

Arguments

`ann`	a data frame which can be converted to a GRanges object, that means it has at least the chromosome, start, end fields. Preferably, the output of `link{get.ucsc.annotation}`.
`org`	one of metaseqR supported organisms.

Value

A named numeric vector.

Author(s)

Panagiotis Moulos

Examples

```
ann <- get.ucsc.annotation("mm9","gene","ucsc")
gc <- get.gc.content(ann,"mm9")
```

---

get.gene.attributes *Annotation downloader helper*

Description

Returns a vector of genomic annotation attributes which are used by the biomaRt package in order to fetch the gene annotation for each organism. It has no parameters. Internal use.

Usage

```
get.gene.attributes(org)
```

Arguments

org	one of the supported organisms.

Value

A character vector of Ensembl gene attributes.

Author(s)

Panagiotis Moulos

Examples

```
gene.attr <- get.gene.attributes("mm9")
```

`get.host` *Annotation downloader helper*

Description

Returns the appropriate Ensembl host address to get different versions of annotation from. Internal use.

Usage

```
get.host(org)
```

Arguments

`org`	the organism for which to return the host address.

Value

A string with the host address.

Author(s)

Panagiotis Moulos

Examples

```
mm9.hist <- get.host("mm9")
```

`get.preset.opts` *Return several analysis options given an analysis preset*

Description

This is a helper function which returns a set of metaseqr pipeline options, grouped together according to a preset keyword. It is intended mostly for internal use.

Usage

```
get.preset.opts(preset, org)
```

Arguments

`preset`	preset can be one of `"all.basic"`, `"all.normal"`, `"all.full"`, `"medium.basic"`, `"medium.normal"`,
`org`	one of the supported organisms. See `metaseqr` main help page. `"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 main `metaseqr` help page.

Value

A named list with names `exon.filters`, `gene.filters`, `pcut`, `export.what`, `export.scale`, `export.values` and `export.stats`, each of which correspond to an element of the metaseqr pipeline.

Author(s)

Panagiotis Moulos

Examples

```
strict.preset <- get.preset.opts("strict.basic","mm9")
```

---

get.strict.biofilter *Group together a more strict biotype filter*

Description

Returns a list with TRUE/FALSE according to the biotypes that are going to be filtered in a more strict way than the defaults. This is a helper function for the analysis presets of metaseqR. Internal use only.

Usage

```
get.strict.biofilter(org)
```

Arguments

`org`	one of the supported organisms.

Value

A list of booleans, one for each biotype.

Author(s)

Panagiotis Moulos

Examples

```
sf <- get.strict.biofilter("hg18")
```

`get.ucsc.annotation` *UCSC/RefSeq annotation downloader*

Description

This function connects to the UCSC Genome Browser public database and downloads annotation elements (gene co-ordinates, exon co-ordinates, gene identifications etc.) for each of the supported organisms, but using UCSC instead of Ensembl. See the help page of `metaseqr` for a list of supported organisms. The function downloads annotation for an organism genes or exons.

Usage

```
get.ucsc.annotation(org, type, refdb="ucsc",
    multic=FALSE)
```

Arguments

`org`	the organism for which to download annotation.
`type`	either `"gene"` or `"exon"`.
`refdb`	either `"ucsc"` or `"refseq"`.
`multic`	a logical value indicating the presence of multiple cores. Defaults to `FALSE`. Do not change it if you are not sure whether package parallel has been loaded or not. It is used in the case of `type="exon"` to process the return value of the query to the UCSC Genome Browser database.

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. The gene_id and exon_id correspond to UCSC or RefSeq gene 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. Note also that as the UCSC databases do not contain biotype classifications like Ensembl. These will be returned as `NA` and as a result, some quality control plots will not be available.

Author(s)

Panagiotis Moulos

Examples

```
hg19.genes <- get.ucsc.annotation("hg19","gene","ucsc")
mm9.exons <- get.ucsc.annotation("mm9","exon","refseq")
```

---

get.ucsc.credentials *Return host, username and password for UCSC Genome Browser database*

Description

Returns a character vector with a hostname, username and password to connect to the UCSC Genome Browser database to retrieve annotation. Internal use mostly.

Usage

```
get.ucsc.credentials()
```

Value

A named character vector.

Author(s)

Panagiotis Moulos

Examples

```
db.creds <- get.ucsc.credentials()
```

---

get.ucsc.dbl *Download annotation from UCSC servers, according to organism and source*

Description

Directly downloads UCSC and RefSeq annotation files from UCSC servers to be used with metaseqR. This functionality is used when the package RMySQL is not available for some reason, e.g. Windows machines. It created an SQLite database where the same queries can be used.

Usage

```
get.ucsc.dbl(org, type, refdb="ucsc")
```

Arguments

org	one of metaseqR supported organisms.
type	either "gene" or "exon".
refdb	one of "ucsc" or "refseq" to use the UCSC or RefSeq annotation sources respectively.

Value

An SQLite database.

Author(s)

Panagiotis Moulos

Examples

```
db.file <- get.ucsc.dbl("hg18","gene","ucsc")
```

---

get.ucsc.organism *Return a proper formatted organism alias*

Description

Returns the proper UCSC Genome Browser database organism alias based on what is given to metaseqR. Internal use.

Usage

```
get.ucsc.organism(org)
```

Arguments

org	one of the metaseqr supported organism.

Value

A proper organism alias.

Author(s)

Panagiotis Moulos

Examples

```
org <- get.ucsc.organism("danrer7")
```

get.ucsc.query — *Return queries for the UCSC Genome Browser database, according to organism and source*

Description

Returns an SQL query to be used with a connection to the UCSC Genome Browser database and fetch metaseqR supported organism annotations. This query is constructed based on the data source and data type to be returned.

Usage

```
get.ucsc.query(org, type, refdb="ucsc")
```

Arguments

`org`	one of metaseqR supported organisms.
`type`	either `"gene"` or `"exon"`.
`refdb`	one of `"ucsc"` or `"refseq"` to use the UCSC or RefSeq annotation sources respectively.

Value

A valid SQL query.

Author(s)

Panagiotis Moulos

Examples

```
db.query <- get.ucsc.query("hg18","gene","ucsc")
```

get.ucsc.tabledef — *Get SQLite UCSC table defintions, according to organism and source*

Description

Creates a list of UCSC Genome Browser database tables and their SQLite definitions with the purpose of creating a temporary SQLite database to be used used with metaseqR. This functionality is used when the package RMySQL is not available for some reason, e.g. Windows machines.

Usage

```
get.ucsc.tabledef(org, type, refdb="ucsc", what="queries")
```

Arguments

org	one of metaseqR supported organisms.
type	either "gene" or "exon".
refdb	one of "ucsc" or "refseq" to use the UCSC or RefSeq annotation sources respectively.
what	either "queries" for SQLite table definitions or "fields" for only a vector of table field names.

Value

A list with SQLite table definitions.

Author(s)

Panagiotis Moulos

Examples

```
db.tabledefs <- get.ucsc.tabledef("hg18","gene","ucsc")
```

get.ucsc.tbl.tpl *Create SQLite UCSC table template defintions*

Description

Returns an SQLIte table template defintion, according to UCSC Genome Browser database table schemas. This functionality is used when the package RMySQL is not available for some reason, e.g. Windows machines. Internal use only.

Usage

```
get.ucsc.tbl.tpl(tab, what="queries")
```

Arguments

tab	name of UCSC database table.
what	"queries" for SQLite table definitions or "fields" for table column names.

Value

An SQLite table definition.

Author(s)

Panagiotis Moulos

Examples

```
db.table.tmpl <- get.ucsc.tbl.tpl("knownCanonical")
```

`get.valid.chrs` *Annotation downloader helper*

Description

Returns a vector of chromosomes to maintain after annotation download. Internal use.

Usage

```
get.valid.chrs(org)
```

Arguments

`org`	the organism for which to return the chromosomes.

Value

A character vector of chromosomes.

Author(s)

Panagiotis Moulos

Examples

```
hg18.chr <- get.valid.chrs("hg18")
```

get.weights — *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 estimate.aufc.weights function.

Usage

```
get.weights(org = c("human", "chimpanzee", "mouse",
    "fruitfly", "arabidopsis"))
```

Arguments

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

Value

A named vector of convex weights.

Author(s)

Panagiotis Moulos

Examples

```
wh <- get.weights("human")
```

graphics.close — *Close plotting device*

Description

Wrapper function to close a plotting device. Internal use only.

Usage

```
graphics.close(o)
```

Arguments

o	the plotting device, see main metaseqr function

Author(s)

Panagiotis Moulos

Examples

```
graphics.close("pdf")
```

`graphics.open` *Open plotting device*

Description

Wrapper function to open a plotting device. Internal use only.

Usage

```
graphics.open(o, f, ...)
```

Arguments

`o`	the plotting device, see main metaseqr function
`f`	a filename, if the plotting device requires it (e.g. `"pdf"`)
`...`	further arguments to be passed to plot devices, such as parameter from `par`.

Author(s)

Panagiotis Moulos

Examples

```
graphics.open("pdf","test.pdf",width=12,height=12)
```

hg19.exon.counts — *Human RNA-Seq data with three conditions, three samples*

Description

This data set contains RNA-Seq exon read counts for 3 chromosomes. The data are from an experiment studying the effect of a long non-coding RNA related to the ASCL2 gene in WNT signaling and intestinal cancer. It has two conditions (CON, DOX) and four samples (CON_BR1, CON_BR2, DOX_BR1, DOX_BR2). It also contains a predefined `sample.list` and `libsize.list` named `sample.list.hg18` and `libsize.list.hg18`.

Format

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

Author(s)

Panagiotis Moulos

Source

GEO (http://www.ncbi.nlm.nih.gov/geo/)

libsize.list.hg19 — *Human RNA-Seq data with three conditions, three samples*

Description

The library size list for `hg19.exon.counts`. See the data set description.

Format

a named `list` with library sizes.

Author(s)

Panagiotis Moulos

Source

GEO (http://www.ncbi.nlm.nih.gov/geo/)

libsize.list.mm9 — *Mouse RNA-Seq data with two conditions, four samples*

Description

The library size list for `mm9.gene.counts`. See the data set description.

Format

a named `list` with library sizes.

Author(s)

Panagiotis Moulos

Source

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

load.bs.genome — *Loads (or downloads) the required BSGenome package*

Description

Retrieves the required BSgenome package when the annotation source is `"ucsc"` or `"refseq"`. These packages are required in order to estimate the GC-content of the retrieved genes from UCSC or RefSeq.

Usage

```
load.bs.genome(org)
```

Arguments

`org`	one of `metaseqr` supported organisms.

Value

A proper BSgenome package name.

Author(s)

Panagiotis Moulos

Examples

```
bs.obj <- load.bs.genome("mm9")
```

log2disp *Display value transformation*

Description

Logarithmic transformation for display purposes. Internal use only.

Usage

```
log2disp(mat, base = 2)
```

Arguments

mat	input data matrix
base	logarithmic base, 2 or 10

Author(s)

Panagiotis Moulos

make.contrast.list *Create contrast lists from contrast vectors*

Description

Returns a list, properly structured to be used within the `stat.*` functions of the metaseqr package. See the main documentation for the structure of this list and the example below. This function is mostly for internal use, as the `stat.*` functions can be supplied directly with the contrasts vector which is one of the main `metaseqr` arguments.

Usage

```
make.contrast.list(contrast, sample.list)
```

Arguments

contrast	a vector of contrasts in the form "ConditionA_vs_ConditionB" or "ConditionA_vs_ConditionB_vs_ConditionC_vs_...". In case of Control vs Treatment designs, the Control condition should ALWAYS be the first.
sample.list	the list of samples in the experiment. See also the main help page of `metaseqr`.

Value

A named list whose names are the contrasts and its members are named vectors, where the names are the sample names and the actual vector members are the condition names. See the example.

Author(s)

Panagiotis Moulos

Examples

```
sample.list <- list(Control=c("C1","C2"),TreatmentA=c("TA1","TA2"),TreatmentB=c("TB1","TB2"))
contrast <- c("Control_vs_TreatmentA","Control_vs_TreatmentA_vs_TreatmentB")
cl <- make.contrast.list(contrast,sample.list)
```

---

make.export.list *Intitialize output list*

Description

Initializes metaseqr R output. Internal use only.

Usage

```
make.export.list(con)
```

Arguments

con	The contrasts.

Value

An empty named list.

Author(s)

Panagiotis Moulos

---

make.fold.change *Calculates fold changes*

Description

Returns a matrix of fold changes based on the requested contrast, the list of all samples and the data matrix which is produced by the metaseqr workflow. For details on the `contrast`, `sample.list` and `log.offset` parameters, see the main usage page of metaseqr. This function is intended mostly for internal use but can also be used independently.

Usage

```
make.fold.change(contrast, sample.list, data.matrix,
    log.offset = 1)
```

Arguments

`contrast`	the vector of requested statistical comparison contrasts.
`sample.list`	the list containing condition names and the samples under each condition.
`data.matrix`	a matrix of gene expression data whose column names are the same as the sample names included in the sample list.
`log.offset`	a number to be added to each element of data matrix in order to avoid Infinity on log type data transformations.

Value

A matrix of fold change ratios, treatment to control, as these are parsed from contrast.

Author(s)

Panagiotis Moulos

Examples

```
data.matrix <- round(1000*matrix(runif(400),100,4))
rownames(data.matrix) <- paste("gene_",1:100,sep="")
colnames(data.matrix) <- c("C1","C2","T1","T2")
fc <- make.fold.change("Control_vs_Treatment",list(Control=c("C1","C2"),
    Treatment=c("T1","T2")),data.matrix)
```

---

make.grid — *Optimize rectangular grid plots*

Description

Returns a vector for an optimized m x m plot grid to be used with e.g. `par(mfrow)`. m x m is as close as possible to the input n. Of course, there will be empty grid positions if n < m x m.

Usage

```
make.grid(n)
```

Arguments

`n`	An integer, denoting the total number of plots to be created.

Value

A 2-element vector with the dimensions of the grid.

Author(s)

Panagiotis Moulos

Examples

```
g1 <- make.grid(16) # Returns c(4,4)
g2 <- make.grid(11) # Returns c(4,3)
```

---

make.highcharts.points *Interactive volcano plot helper*

Description

Creates a list which contains the data series of a scatterplot, to be used for serialization with highcharts JavaScript plotting. framework. Internal use only.

Usage

```
make.highcharts.points(x, y, a)
```

Arguments

x	The x coordinates (should be a named vector!).
y	The y coordinates.
a	Alternative names for each point.

Value

A list that is later serialized to JSON.

Author(s)

Panagiotis Moulos

make.html.body — *HTML report helper*

Description

Returns a character vector with an html formatted table. Essentially, it collapses the input rows to a single character and puts a <tbody></tbody> tag set around. It is meant to be applied to the output of `make.html.rows`. Internal use.

Usage

```
make.html.body(mat)
```

Arguments

mat	the character vector produced by `make.html.rows`.

Value

A character vector with the body of mat formatted in html.

Author(s)

Panagiotis Moulos

Examples

```
data.matrix <- round(1000*matrix(runif(400),100,4))
rownames(data.matrix) <- paste("gene_",1:100,sep="")
colnames(data.matrix) <- c("C1","C2","T1","T2")
the.cells <- make.html.cells(data.matrix)
the.header <- make.html.header(the.cells[1,])
the.rows <- make.html.rows(the.cells)
the.body <- make.html.body(the.rows)
```

make.html.cells — *HTML report helper*

Description

Returns a character matrix with html formatted table cells. Essentially, it converts the input data to text and places them in a <td></td> tag set. Internal use.

Usage

```
make.html.cells(mat, type = "numeric", digits = 3)
```

Arguments

mat	the data matrix (numeric or character)
type	the type of data in the matrix ("numeric" or "character")
digits	the number of digits on the right of the decimal points to pass to formatC. It has meaning when type="numeric".

Value

A character matrix with html formatted cells.

Author(s)

Panagiotis Moulos

Examples

```
data.matrix <- round(1000*matrix(runif(400),100,4))
rownames(data.matrix) <- paste("gene_",1:100,sep="")
colnames(data.matrix) <- c("C1","C2","T1","T2")
the.cells <- make.html.cells(data.matrix)
```

---

make.html.header *HTML report helper*

Description

Returns a character vector with an html formatted table head row. Essentially, it collapses the input row to a single character and puts a <th></th> tag set around. It is meant to be applied to the output of make.html.cells. Internal use.

Usage

```
make.html.header(h)
```

Arguments

h	the colnames of a matrix or data frame, usually as output of make.html.cells function.

Value

A character vector with html formatted header of a matrix.

Author(s)

Panagiotis Moulos

Examples

```
data.matrix <- round(1000*matrix(runif(400),100,4))
rownames(data.matrix) <- paste("gene_",1:100,sep="")
colnames(data.matrix) <- c("C1","C2","T1","T2")
the.cells <- make.html.cells(data.matrix)
the.header <- make.html.header(the.cells[1,])
```

---

make.html.rows *HTML report helper*

Description

Returns a character vector with html formatted rows. Essentially, it collapses every row of a matrix to a single character and puts a <tr></tr> tag set around. It is meant to be applied to the output of `make.html.cells`. Internal use.

Usage

```
make.html.rows(mat)
```

Arguments

`mat`	the data matrix, usually the output of `make.html.cells` function.

Value

A character vector with html formatted rows of a matrix.

Author(s)

Panagiotis Moulos

Examples

```
data.matrix <- round(1000*matrix(runif(400),100,4))
rownames(data.matrix) <- paste("gene_",1:100,sep="")
colnames(data.matrix) <- c("C1","C2","T1","T2")
the.cells <- make.html.cells(data.matrix)
the.rows <- make.html.rows(the.cells)
```

make.html.table *HTML report helper*

Description

Returns a character vector with a fully html formatted table. Essentially, it binds the outputs of make.html.cells, make.html.rows, make.html.header and make.html.body to the final table and optionally assigns an id attribute. The above functions are meant to format a data table so as it can be rendered by external tools such as DataTables.js during a report creation. It is meant for internal use.

Usage

```
make.html.table(b, h = NULL, id = NULL)
```

Arguments

b	the table body as produced by make.html.body.
h	the table header as produced by make.html.header.
id	the table id attribute.

Value

A fully formatted html table.

Author(s)

Panagiotis Moulos

Examples

```
data.matrix <- round(1000*matrix(runif(400),100,4))
rownames(data.matrix) <- paste("gene_",1:100,sep="")
colnames(data.matrix) <- c("C1","C2","T1","T2")
the.cells <- make.html.cells(data.matrix)
the.header <- make.html.header(the.cells[1,])
the.rows <- make.html.rows(the.cells)
the.body <- make.html.body(the.rows)
the.table <- make.html.table(the.body,the.header,id="my_table")
```

make.matrix *Results output build helper*

Description

Returns a list of matrices based on the export scales that have been chosen from the main function and a subset of samples based on the sample names provided in the `sample.list` argument of the main `metaseqr` function. Internal use.

Usage

```
make.matrix(samples, data.list, export.scale = "natural")
```

Arguments

`samples`	a set of samples from the dataset under processing. They should match sample names from `sample.list`. See also the main help page of `metaseqr`.
`data.list`	a list containing natural or transformed data, typically an output from `make.transformation`.
`export.scale`	the output transformations used as input also to `make.transformation`.

Value

A named list whose names are the elements in `export.scale`. Each list member is the respective sample subest data matrix.

Author(s)

Panagiotis Moulos

Examples

```
data.matrix <- round(1000*matrix(runif(400),100,4))
rownames(data.matrix) <- paste("gene_",1:100,sep="")
colnames(data.matrix) <- c("C1","C2","T1","T2")
tr <- make.transformation(data.matrix,c("log2","vst"))
mm <- make.matrix(c("C1","T1"),tr,"log2")
head(tr$vst)
```

`make.path.struct` *Project path constructor helper*

Description

Helper for `make.project.path`. Internal use only.

Usage

```
make.path.struct(main.path)
```

Arguments

`main.path`	The desired project path.

Value

A named list whose names are the conditions of the experiments and its members are the samples belonging to each condition.

Author(s)

Panagiotis Moulos

`make.permutation` *Create counts matrix permutations*

Description

This function creates a permuted read counts matrix based on the `contrast` argument (to define new virtual contrasts of the same number) and on the `sample.list` to derive the number of samples for each virtual condition.It is a helper for the `meta.perm` function.

Usage

```
make.permutation(counts, sample.list, contrast,
    repl = FALSE)
```

Arguments

`counts`	the gene read counts matrix.
`sample.list`	the list containing condition names and the samples under each condition.
`contrast`	the contrasts vector. See the main `metaseqr` help page.
`repl`	the same as the replace argument in `sample` function.

Value

A list with three members: the matrix of permuted per sample read counts, the virtual sample list and the virtual contrast to be used with the `stat.*` functions.

Author(s)

Panagiotis Moulos

Examples

```
data("mm9.gene.data",package="metaseqR")
per <- make.permutation(mm9.gene.counts,sample.list.mm9,
    "e14.5_vs_adult_8_weeks")
```

`make.project.path` *Project path constructor*

Description

Create the main metaseqr project path. Internal use only.

Usage

```
make.project.path(path, f = NULL)
```

Arguments

`path`	The desired project path. Can be NULL for auto-generation.
`f`	The input counts table file.

Value

A list with project path elements.

Author(s)

Panagiotis Moulos

make.report.messages *Initializer of report messages*

Description

Initializes metaseqR report tmeplate messages output. Internal use only.

Usage

```
make.report.messages(lang)
```

Arguments

lang	The language of the report. For now, only english ("en") is supported.

Value

An named list with messages for each input option.

Author(s)

Panagiotis Moulos

make.sample.list *Creates sample list from file*

Description

Create the main sample list from an external file.

Usage

```
make.sample.list(input)
```

Arguments

input	a 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.

Value

A named list whose names are the conditions of the experiments and its members are the samples belonging to each condition.

Author(s)

Panagiotis Moulos

Examples

```
targets <- data.frame(sample=c("C1","C2","T1","T2"),
    condition=c("Control","Control","Treatment","Treatment"))
write.table(targets,file="targets.txt",sep="\t",row.names=FALSE,
    quote="")
sample.list <- make.sample.list("targets.txt")
```

---

make.sim.data.sd *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

```
make.sim.data.sd(N, param, samples = c(5, 5),
    ndeg = rep(round(0.1*N), 2), fc.basis = 1.5,
libsize.range = c(0.7, 1.4), libsize.mag = 1e+7,
    model.org = NULL, sim.length.bias = FALSE,
    seed = NULL)
```

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 (mu.hat) and the second the dispersion (phi.hat). This list can be created with the estimate.sim.params 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.
fc.basis	the minimum fold-change for deregulation.
libsize.range	a vector with 2 numbers (generally small, see the default), as they are multiplied with libsize.mag. These numbers control the library sized of the synthetic data to be produced.

`libsize.mag` a (big) number to multiply the `libsize.range` to produce library sizes.

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

`sim.length.bias` 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.

`seed` a seed to use with random number generation for reproducibility.

Value

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

Author(s)

Panagiotis Moulos

Examples

```
# 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 <- 10000
par.list <- estimate.sim.params("~/bottomly_read_counts.txt")
sim <- make.sim.data.sd(N,par.list)
synth.data <- sim$simdata
true.deg <- which(sim$truedeg!=0)
```

---

make.sim.data.tcc *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

```
make.sim.data.tcc(...)
```

Arguments

...	parameters to the simulateReadCounts function.

Value

A list with the following members: simdata holding the simulated dataset complying with metaseqr 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

```
dd <- make.sim.data.tcc(Ngene=10000,PDEG=0.2,
        DEG.assign=c(0.9,0.1),
        DEG.foldchange=c(5,5),replicates=c(3,3))
head(dd$simdata)
```

---

make.stat *Calculates several statistices on read counts*

Description

Returns a matrix of statistics calculated for a set of given samples. Internal use.

Usage

```
make.stat(samples, data.list, stat, export.scale)
```

Arguments

samples	a set of samples from the dataset under processing. They should match sample names from sample.list. See also the main help page of metaseqr.
data.list	a list containing natural or transformed data, typically an output from make.transformation.
stat	the statistics to calculate. Can be one or more of "mean", "median", "sd", "mad", "cv", "rcv". See also the main help page of metaseqr.
export.scale	the output transformations used as input also to make.transformation.

Value

A matrix of statistics calculated based on the input sample names. The different data transformnations are appended columnwise.

Author(s)

Panagiotis Moulos

Examples

```
data.matrix <- round(1000*matrix(runif(400),100,4))
rownames(data.matrix) <- paste("gene_",1:100,sep="")
colnames(data.matrix) <- c("C1","C2","T1","T2")
tr <- make.transformation(data.matrix,c("log2","vst"))
st <- make.stat(c("C1","C2"),tr,c("mean","sd"),c("log2","vst"))
```

---

make.transformation *Calculates several transformation of counts*

Description

Returns a list of transformed (normalized) counts, based on the input count matrix data.matrix. The data transformations are passed from the `export.scale` parameter and the output list is named accordingly. This function is intended mostly for internal use but can also be used independently.

Usage

```
make.transformation(data.matrix, export.scale,
    log.offset = 1)
```

Arguments

`data.matrix`	the raw or normalized counts matrix. Each column represents one input sample.
`export.scale`	a character vector containing one of the supported data transformations (`"natural"`, `"log2"`, `"log10"`,`"vst"`). See also the main help page of metaseqr.
`log.offset`	a number to be added to each element of data.matrix in order to avoid Infinity on log type data transformations.

Value

A named list whose names are the elements in export.scale. Each list member is the respective transformed data matrix.

Author(s)

Panagiotis Moulos

Examples

```
data.matrix <- round(1000*matrix(runif(400),100,4))
rownames(data.matrix) <- paste("gene_",1:100,sep="")
colnames(data.matrix) <- c("C1","C2","T1","T2")
tr <- make.transformation(data.matrix,c("log2","vst"))
head(tr$vst)
```

`make.venn.areas` *Helper for Venn diagrams*

Description

This function creates a list with names the arguments of the Venn diagram construction functions of the R package VennDiagram and list members the internal encoding (uppercase letters A to E and combinations among then) used to encode the pairwise comparisons to create the intersections needed for the Venn diagrams. Internal use mostly.

Usage

```
make.venn.areas(n)
```

Arguments

`n`	the number of the sets used for the Venn diagram.

Value

A named list, see descritpion.

Author(s)

Panagiotis Moulos

Examples

```
sets <- c("apple","pear","banana")
pairs <- make.venn.pairs(sets)
areas <- make.venn.areas(length(sets))
```

make.venn.colorscheme *Helper for Venn diagrams*

Description

This function returns a list of colorschemes accroding to the number of sets. Internal use.

Usage

```
make.venn.colorscheme(n)
```

Arguments

n	the number of the sets used for the Venn diagram.

Value

A list with colors for fill and font.

Author(s)

Panagiotis Moulos

Examples

```
sets <- c("apple","pear","banana")
cs <- make.venn.colorscheme(length(sets))
```

make.venn.counts *Helper for Venn diagrams*

Description

This function creates a list with names the arguments of the Venn diagram construction functions of the R package VennDiagram and list members are initially NULL. They are filled by the diagplot.venn function. Internal use mostly.

Usage

```
make.venn.counts(n)
```

Arguments

n	the number of the sets used for the Venn diagram.

Value

A named list, see descritpion.

Author(s)

Panagiotis Moulos

Examples

```
sets <- c("apple","pear","banana")
counts <- make.venn.counts(length(sets))
```

---

make.venn.pairs *Helper for Venn diagrams*

Description

This function creates a list of pairwise comparisons to be performed in order to create an up to 5-way Venn diagram using the R package VennDiagram. Internal use mostly.

Usage

```
make.venn.pairs(algs)
```

Arguments

algs	a vector with the names of the sets (up to length 5, if larger, it will be truncated with a warning).

Value

A list with as many pairs as the comparisons to be made for the construction of the Venn diagram. The pairs are encoded with the uppercase letters A through E, each one corresponding to order of the input sets.

Author(s)

Panagiotis Moulos

Examples

```
sets <- c("apple","pear","banana")
pairs <- make.venn.pairs(sets)
```

meta.perm *Permutation tests for meta-analysis*

Description

This function performs permutation tests in order to derive a meta p-value by combining several of the statistical algorithms of metaseqr. This is probably the most accurate way of combining multiple statistical algorithms for RNA-Seq data, as this issue is different from the classic interpretation of the term "meta-analysis" which implies the application of the same statistical test on different datasets treating the same subject/experiment. For other methods, see also the main `metaseqr` help page. You should keep in mind that the permutation procedure can take a long time, even when executed in parallel.

Usage

```
meta.perm(contrast, counts, sample.list, statistics,
    stat.args, libsize.list, nperm = 10000,
    weight = rep(1/ncol(counts), ncol(counts)),
    select = c("min", "max", "weight"), replace = "auto",
    reprod=TRUE, multic = FALSE)
```

Arguments

`contrast`	the contrasts to be tested by each statistical algorithm. See the main `metaseqr` help page.
`counts`	a normalized read counts table, one row for each gene, one column for each sample.
`sample.list`	the list containing condition names and the samples under each condition. See the main `metaseqr` help page.
`statistics`	the statistical algorithms used in metaseqr. See the main `metaseqr` help page.
`stat.args`	the parameters for each statistical algorithm. See the main `metaseqr` help page.
`libsize.list`	a list with library sizes. See the main `metaseqr` 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.
`select`	how to select the initial vector of p-values. It can be `"min"` to select the minimum p-value for each gene (more conservative), `"max"` to select the maximum p-value for each gene (less conservative), `"weight"` to apply the weights to the p-value vector for each gene and derive a weighted p-value.
`replace`	same as the `replace` argument in the `sample` function. Implies bootstraping or simple resampling without replacement. It can also be `"auto"`, to determine bootstraping or not with the following rule: if `ncol(counts)<=6 replace=FALSE else replace=TRUE`. This protects from the case of having zero variability across resampled conditions. In such cases, most statistical tests would crash.

reprod — create reproducible permutations. Ideally one would want to create the same set of indices for a given dataset so as to create reproducible p-values. If reprod=TRUE, a fixed seed is used by meta.perm for all the datasets analyzed with metaseqr. If reprod=FALSE, then the p-values will not be reproducible, although statistical significance is not expected to change for a large number of resambling. Finally, reprod can be a numeric vector of seeds with the same length as nperm so that the user can supply his/her own seeds.

multic — use multiple cores to execute the premutations. This is an external parameter and implies the existence of multicore package in the execution environment. See the main metaseqr help page.

Value

A vector of meta p-values

Author(s)

Panagiotis Moulos

Examples

```
# Not yet available
```

---

meta.test — *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

```
meta.test(cp.list,
    meta.p = c("simes", "bonferroni", "fisher", "dperm.min",
    "dperm.max", "dperm.weight", "fperm", "whitlock",
    "minp", "maxp", "weight", "none"), counts, sample.list,
    statistics, stat.args, libsize.list, nperm = 10000,
    weight = rep(1/length(statistics), length(statistics)),
    reprod=TRUE, multic = FALSE)
```

Arguments

cp.list	a named list whose names are the contrasts requested from metaseqr. Each member is a p-value matrix whose colnames are the names of the statistical tests applied to the data. See the main metaseqr help page.
meta.p	the p-value combination method to use. See the main metaseqr help page.
counts	the normalized and possibly filtered read counts matrix. See the main metaseqr help page.
sample.list	the list containing condition names and the samples under each condition. See the main metaseqr help page.
statistics	the statistical algorithms used in metaseqr. See the main metaseqr help page.
stat.args	the parameters for each statistical argument. See the main metaseqr help page.
libsize.list	a list with library sizes. See the main metaseqr 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.
reprod	create reproducible permutations when meta.p="dperm.min", meta.p="dperm.max" or meta.p="dperm.weight". Ideally one would want to create the same set of indices for a given dataset so as to create reproducible p-values. If reprod=TRUE, a fixed seed is used by meta.perm for all the datasets analyzed with metaseqr. If reprod=FALSE, then the p-values will not be reproducible, although statistical significance is not expected to change for a large number of resambling. Finally, reprod can be a numeric vector of seeds with the same length as nperm so that the user can supply his/her own seeds.
multic	use multiple cores to execute the premutations. This is an external parameter and implies the existence of multicore package in the execution environment. See the main metaseqr help page.

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

```
# Not yet available
```

meta.worker — *Permutation tests helper*

Description

This function performs the statistical test for each permutation. Internal use only.

Usage

```
meta.worker(x,co,sl,cnt,s,r,sa,ll,
    el,w)
```

Arguments

x	a virtual list with the random seed and the permutation index.
co	the counts matrix.
sl	the sample list.
cnt	the contrast name.
s	the statistical algorithms.
sa	the parameters for each statistical algorithm.
ll	a list with library sizes.
r	same as the `replace` argument in the `sample` function.
el	min, max or weight.
w	a numeric vector of weights for each statistical algorithm

Value

A matrix of p-values.

Author(s)

Panagiotis Moulos

Examples

```
# Not yet available
```

metaseqr *The main metaseqr pipeline*

Description

This function is the main metaseqr workhorse and implements the main metaseqr workflow which performs data read, filtering, normalization and statistical selection, creates diagnostic plots and exports the results and a report if requested. The metaseqr function is responsible for assembling all the steps of the metaseqr 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, baySeq (Hardcastle et al., 2012) vi) in the case of multiple statistical testing algorithms, performs meta-analysis using one of five available 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 metaseqr 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 metaseqr 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

```
metaseqr(counts, sample.list,
    file.type = c("auto", "sam", "bam", "bed"),
    path = NULL, contrast = NULL, libsize.list = NULL,
    id.col = 4, gc.col = NA, name.col = NA, bt.col = NA,
    annotation = c("download", "embedded"),
    org = c("hg18", "hg19", "hg38", "mm9", "mm10", "rn5", "dm3",
        "danrer7", "pantro4", "susscr3", "tair10", "custom"),
    refdb = c("ensembl", "ucsc", "refseq"),
    count.type = c("gene", "exon"),
    exon.filters = list(min.active.exons = list(exons.per.gene = 5,
            min.exons = 2, frac = 1/5)),
    gene.filters = list(length = list(length = 500),
            avg.reads = list(average.per.bp = 100, quantile = 0.25),
            expression = list(median = TRUE, mean = FALSE, quantile = NA,
                    known = NA, custom = NA),
            biotype = get.defaults("biotype.filter", org[1])),
    when.apply.filter = c("postnorm", "prenorm"),
```

```
normalization = c("edaseq", "deseq", "edger", "noiseq", "nbpseq",
        "each", "none"),
norm.args = NULL,
statistics = c("deseq", "edger", "noiseq", "bayseq", "limma",
        "nbpseq"),
stat.args = NULL,
adjust.method = sort(c(p.adjust.methods, "qvalue")),
meta.p = if (length(statistics) > 1) c("simes", "bonferroni", "fisher",
        "dperm.min", "dperm.max", "dperm.weight", "fperm", "whitlock",
        "minp", "maxp", "weight", "none") else "none",
weight = rep(1/length(statistics), length(statistics)),
nperm = 10000, reprod=TRUE, pcut = NA, log.offset = 1,
preset = NULL,
qc.plots = c("mds", "biodetection", "countsbio", "saturation",
        "readnoise", "filtered", "correl", "pairwise", "boxplot",
        "gcbias", "lengthbias", "meandiff", "meanvar", "rnacomp",
        "deheatmap", "volcano", "biodist"),
fig.format = c("png", "jpg", "tiff", "bmp", "pdf", "ps"),
out.list = FALSE, export.where = NA,
export.what = c("annotation", "p.value", "adj.p.value",
        "meta.p.value", "adj.meta.p.value", "fold.change",
        "stats", "counts","flags"),
export.scale = c("natural", "log2", "log10", "vst"),
export.values = c("raw", "normalized"),
export.stats = c("mean", "median", "sd", "mad", "cv",
        "rcv"),
export.counts.table = FALSE,
restrict.cores = 0.6, report = TRUE, report.top = 0.1,
report.template = "default", save.gene.model = TRUE,
verbose = TRUE, run.log = TRUE, ...)
```

Arguments

`counts` a text tab-delimited file containing gene or exon 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. Thus 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 `"download"` or an external file in proper format. ii) The first n columns should contain gene or exon annotation elements like chromosomal locations, gene accessions, exon accessions, GC content etc. In that case, 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 metaseqr 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. Generally, it's best to set the `annotation` parameter to `"download"` to ensure the most comprehensible results. Counts can be a data frame satisfying the above conditions. It is a data frame by default when `read2count` is used. counts can also be an .RData file (output of `save` function which contains static input elements (list containing the gene model (exon counts for each gene constructed by the `construct.gene.model` function, gene and exon annotation to avoid re-downloading and/or gene counts depending on `count.type`). This kind of input facilitates the re-analysis of the same experiment, using different filtering, normalization and statistical algorithms. Finally, counts can be a list representing the gene model (exon counts for each gene) constructed by the `construct.gene.model` function (provided for backwards compatibility). This .RData file can be generated by setting `save.gene.model=TRUE` when performing data analysis for the first time.

`sample.list` a list containing condition names and the samples under each condition. It should have the format `sample.list <- 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, `sample.list` 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. In this case, the function `make.sample.list` is used. If the `counts` argument is missing, the `sample.list` 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 (see below). The third column MUST contain the biological condition where each of the samples in the first column should belong to.

`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. If not given 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.

`file.type` 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 metaseqr pipeline. Each element of contrast should STRICTLY have the format "ConditionA_vs_ConditionB_vs_...". A valid example based on the `sample.list` 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. metaseqr uses this convention to properly calculate fold changes. If it's NULL, a contrast between the first two members of the sample.list will be auto-generated.

libsize.list
: 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).

id.col
: an integer denoting the column number in the file (or data frame) provided with the counts argument, where the unique gene or exon accessions are. Default to 4 which is the standard feature name column in a BED file.

gc.col
: 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.

name.col
: 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.

bt.col
: 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.

annotation
: instructs metaseqr 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.

org
: the supported organisms by metaseqr. These can be, for human genomes "hg18", "hg19" or "hg38", for mouse genomes "mm9", "mm10", for rat genome "rn5", for drosophila genome "dm3", for zebrafish genome "danrer7", for chimpanzee genome "pantro4", for pig genome "susscr3" and for Arabidopsis thaliana genome "tair10". Finally, "custom" will instruct metaseqR to completely ignore the org argument and depend solely on annotation file provided by the user.

refdb
: the reference annotation repository from which to retrieve annotation elements to use with metaseqr. It can be one of "ensembl" (default), "ucsc" or "refseq".

count.type
: the type of reads inside the counts file. It can be one of "gene" or "exon". This is a very important and mandatory parameter as it defines the course of the workflow.

exon.filters
: 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.

gene.filters
: 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.

`when.apply.filter`
: 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). In the case of `when.apply.filter="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.

`normalization`
: the normalization algorithm to be applied on the count data. It can be one of `"edaseq"` (default) for EDASeq normalization, `"deseq"` for the normalization algorithm (individual options specified by the `norm.args` argument) in the DESq package, `"edger"` for the normalization algorithms present in the edgeR package (specified by the `norm.args` argument), `"noiseq"` for the normalization algorithms present in the NOISeq package (specified by the `norm.args` argument), `"nbpseq"` for the normalization algorithms present in the NBPSeq package (specified by the `norm.args` argument) or `"none"` to not normalize the data (highly unrecommended). It can also be `"each"` where in this case, the normalization applied will be specific to each statistical test used (i.e. the normalization method bundled with each package and used in its examples and documentation). The last choice is for future use!

`norm.args`
: 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`. If `normalization="each"`, it must be a named list of lists, where each sub-list contains normalization parameters specific to each statistical test to be used. The last choice is for future use!

`statistics`
: one or more statistical analyses to be performed by the metaseqr 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, `"bayseq"` to conduct statistical test(s) implemented in the baySeq package and `"nbpseq"` to conduct statistical test(s) implemented in the NBPSeq package. In any case individual algorithm parameters are controlled by the contents of the `stat.args` list.

`stat.args`
: 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`.

`adjust.method`
: 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.

`meta.p`
: the meta-analysis method to combine p-values from multiple statistical tests **(experimental! see also the second note below, regarding meta-analysis)**. It can be one of `"simes"` (default), `"bonferroni"`, `"minp"`, `"maxp"`, `"weight"`, `"dperm.min"`, `"dperm.max"`, `"dperm.weight"`, `"fisher"`, `"fperm"`, `"whitlock"` or`"none"`. 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.

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 meta.p="fperm" or meta.p="dperm". It defaults to 10000.

reprod
: create reproducible permutations when meta.p="dperm.min", meta.p="dperm.max" or meta.p="dperm.weight". Ideally one would want to create the same set of indices for a given dataset so as to create reproducible p-values. If reprod=TRUE, a fixed seed is used by meta.perm for all the datasets analyzed with metaseqr. If reprod=FALSE, then the p-values will not be reproducible, although statistical significance is not expected to change for a large number of resambling. Finally, reprod can be a numeric vector of seeds with the same length as nperm so that the user can supply his/her own seeds.

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

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

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.

qc.plots
: 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", "biodist", "venn". 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. See the help page of diagplot.noiseq.saturation. 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 diagplot.filtered. 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 qc.plots=NULL if you don't want any diagnostic plots created.

fig.format
: 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.

out.list
: a logical controlling whether to export a list with the results in the running environment.

export.where
: an output directory for the project results (report, lists, diagnostic plots etc.)

export.what
: 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 export.stats argument), "counts", to bind the raw and normalized counts for each sample.

export.scale
: 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).

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

export.stats
: calculate and export several statistics on raw and normalized counts, conditionwise. 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.

export.counts.table
: 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.

restrict.cores
: 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 (restrict.cores=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.

report.top
: 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.

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

save.gene.model
: in case of exon analysis, a list with exon counts for each gene will be saved to the file export.where/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.

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

...
: further arguments that may be passed to plotting functions, related to par.

Value

If out.list 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 `export.what`, `export.scale`, `export.stats`, `export.values` 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 metaseqr pipeline, thus they should not be used in that context. The `exon.filters` 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 `metaseqr` function for an example of how these lists are structured. The supported exon filters in the current version are: i) `min.active.exons` which implements a filter for demanding m out of n exons of a gene to have a certain read presence with parameters `exons.per.gene`, `min.exons` and `frac`. The filter is described as follows: if a gene has up to `exons.per.gene` exons, then read presence is required in at least `min.exons` 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 `metaseqr` usage for the defaults. Set `exon.filters=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 `average.per.bp` base pairs. In summary, the reads of each gene are averaged per `average.per.bp` 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.

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 get.defaults function.

Statistics parameters

The statistics parameters as passed to statistical algorithms in metaseqr, exactly with the same way as the normalization parametes above. In this case, there is one more layer in list nesting. Thus, stat.args 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, baySeq, 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 main.method 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 get.defaults function. When statistics="nbpseq", apart from the rest arguments of the NBPSeq functions estimate.disp and estimate.dispersion, there is the argument main.method 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 main.method="nbpseq" and the estimate.dispersion function is NBPSeq package specific, while the second (with main.method="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. **Additinally, please note that 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 (thus** `stat.args$main.method="nbpseq"` **will not be available)!**

Presets

The analysis presets are a set of keywords (only one can be used) that predefine some of the parameters of the metaseqr 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 metaseqr are overriden: `exon.filters`, `gene.filters`, `pcut`, `export.what`, `export.scale`, `export.values`, `exon.stats`. 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:
  - `exon.filters=NULL`
  - `gene.filters=NULL`
  - `pcut=1`
  - `export.what=c("annotation","p.value","adj.p.value","meta.p.value", "adj.meta.p.value","fold.`
  - `export.scale=c("natural","log2")`
  - `export.values=c("normalized")`
  - `export.stats=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:
  - `exon.filters=NULL`
  - `gene.filters=NULL`
  - `pcut=1`
  - `export.what=c("annotation","p.value","adj.p.value","meta.p.value", "adj.meta.p.value","fold.`
  - `export.scale=c("natural","log2")`
  - `export.values=c("normalized")`
  - `export.stats=c("mean","sd","cv")`

  In this case, the above described arguments become:
  - `exon.filters=NULL`
  - `gene.filters=NULL`
  - `pcut=1`
  - `export.what=c("annotation","p.value","adj.p.value","meta.p.value", "adj.meta.p.value","fold.`
  - `export.scale=c("natural","log2","log10","vst")`

- export.values=c("raw","normalized")
- export.stats=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 described arguments become:
  - exon.filters=list(min.active.exons=list(exons.per.gene=5,min.exons=2,frac=1/5))
  - gene.filters=list(length=list(length=500), avg.reads=list(average.per.bp=100,quantile=0.25), expression=list(median=TRUE,mean=FALSE,quantile=NA,known=NA,custom=NA), biotype=get.defaults("biotype.filter",org[1]))
  - pcut=0.05
  - export.what=c("annotation","p.value","adj.p.value","meta.p.value", "adj.meta.p.value","fold.
  - export.scale=c("natural","log2")
  - export.values=c("normalized")
  - export.stats=c("mean")
- "medium.normal": apply a medium set of filters and and export statistically significant genes and normal annotation and statistics elements. In this case, the above described arguments become:
  - exon.filters=list(min.active.exons=list(exons.per.gene=5,min.exons=2,frac=1/5))
  - gene.filters=list(length=list(length=500), avg.reads=list(average.per.bp=100,quantile=0.25), expression=list(median=TRUE,mean=FALSE,quantile=NA,known=NA,custom=NA), biotype=get.defaults("biotype.filter",org[1]))
  - pcut=0.05
  - export.what=c("annotation","p.value","adj.p.value","meta.p.value", "adj.meta.p.value","fold.
  - export.scale=c("natural","log2")
  - export.values=c("normalized")
  - export.stats=c("mean","sd","cv")

  and statistics elements. In this case, the above described arguments become:
  - exon.filters=list(min.active.exons=list(exons.per.gene=5,min.exons=2,frac=1/5))
  - gene.filters=list(length=list(length=500), avg.reads=list(average.per.bp=100,quantile=0.25), expression=list(median=TRUE,mean=FALSE,quantile=NA,known=NA,custom=NA), biotype=get.defaults("biotype.filter",org[1]))
  - pcut=0.05
  - export.what=c("annotation","p.value","adj.p.value","meta.p.value", "adj.meta.p.value","fold.
  - export.scale=c("natural","log2","log10","vst")
  - export.values=c("raw","normalized")
  - export.stats=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:
  - exon.filters=list(min.active.exons=list(exons.per.gene=4,min.exons=2,frac=1/4))
  - gene.filters=list(length=list(length=750), avg.reads=list(average.per.bp=100,quantile=0.5), expression=list(median=TRUE,mean=FALSE,quantile=NA,known=NA,custom=NA), biotype=get.defaults("biotype.filter",org[1]))
  - pcut=0.01

- export.what=c("annotation","p.value","adj.p.value","meta.p.value", "adj.meta.p.value","fold.
- export.scale=c("natural","log2")
- export.values=c("normalized")
- export.stats=c("mean")

• "strict.normal": apply a strict set of filters and and export statistically significant genes and normal annotation and statistics elements. In this case, the above described arguments become:

- exon.filters=list(min.active.exons=list(exons.per.gene=4,min.exons=2,frac=1/4))
- gene.filters=list(length=list(length=750), avg.reads=list(average.per.bp=100,quantile=0.5), expression=list(median=TRUE,mean=FALSE,quantile=NA,known=NA,custom=NA), biotype=get.defaults("biotype.filter",org[1]))
- pcut=0.01
- export.what=c("annotation","p.value","adj.p.value","meta.p.value", "adj.meta.p.value","fold.
- export.scale=c("natural","log2")
- export.values=c("normalized")
- export.stats=c("mean","sd","cv")

and statistics elements. In this case, the above described arguments become:

- exon.filters=list(min.active.exons=list(exons.per.gene=4,min.exons=2,frac=1/4))
- gene.filters=list(length=list(length=750), avg.reads=list(average.per.bp=100,quantile=0.5), expression=list(median=TRUE,mean=FALSE,quantile=NA,known=NA,custom=NA), biotype=get.defaults("biotype.filter",org[1]))
- pcut=0.01
- export.what=c("annotation","p.value","adj.p.value","meta.p.value", "adj.meta.p.value","fold.
- export.scale=c("natural","log2","log10","vst")
- export.values=c("raw","normalized")
- export.stats=c("mean","median","sd","mad","cv","rcv")

Note

Please note that currently only gene and exon annotation from Ensembl (http://www.ensembl.org) are supported. Thus, the unique gene or exon ids in the counts files should correspond to valid Ensembl gene or exon accessions for the organism of interest. 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 (metaseqr 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 gene.filters and exon.filters 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 metaseqr 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 restrict.cores 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 get.annotation function to download and store the resulting data frames in local files and then use these files with the annotation option.

Please note that the **meta-analysis** feature provided by metaseqr is currently experimental and 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 so the results at this point are not guaranteed and should be interpreted appropriately. We are working on a more solid methodology for combining multiple statistical tests based on multiple testing correction and Monte Carlo methods. 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."

Author(s)

Panagiotis Moulos

Examples

```
# An example pipeline with exon counts
data("hg19.exon.data",package="metaseqR")
metaseqr(
 counts=hg19.exon.counts,
 sample.list=list(normal="normal",paracancerous="paracancerous",cancerous="cancerous"),
 contrast=c("normal_vs_paracancerous","normal_vs_cancerous",
        "normal_vs_paracancerous_vs_cancerous"),
 libsize.list=libsize.list.hg19,
 id.col=4,
 annotation="download",
 org="hg19",
 count.type="exon",
 normalization="edaseq",
 statistics="deseq",
 pcut=0.05,
 qc.plots=c("mds", "biodetection", "countsbio", "saturation", "rnacomp",
        "boxplot", "gcbias", "lengthbias", "meandiff", "readnoise","meanvar",
        "readnoise", "deheatmap", "volcano", "biodist", "filtered"),
 fig.format=c("png","pdf"),
 export.what=c("annotation","p.value","adj.p.value","fold.change","stats",
        "counts"),
 export.scale=c("natural","log2","log10","vst"),
 export.values=c("raw","normalized"),
 export.stats=c("mean","median","sd","mad","cv","rcv"),
 restrict.cores=0.8,
 gene.filters=list(
     length=list(
         length=500
     ),
```

```
        avg.reads=list(
            average.per.bp=100,
            quantile=0.25
        ),
        expression=list(
            median=TRUE,
            mean=FALSE
        ),
        biotype=get.defaults("biotype.filter","hg18")
 )
)

# An example pipeline with gene counts
data("mm9.gene.data",package="metaseqR")
result <- metaseqr(
 counts=mm9.gene.counts,
 sample.list=list(e14.5=c("e14.5_1","e14.5_2"), adult_8_weeks=c("a8w_1","a8w_2")),
 contrast=c("e14.5_vs_adult_8_weeks"),
 libsize.list=libsize.list.mm9,
 annotation="download",
 org="mm9",
 count.type="gene",
 normalization="edger",
 statistics=c("deseq","edger","noiseq"),
 meta.p="fisher",
 pcut=0.05,
 fig.format=c("png","pdf"),
 export.what=c("annotation","p.value","meta.p.value","adj.meta.p.value",
        "fold.change"),
 export.scale=c("natural","log2"),
 export.values="normalized",
 export.stats=c("mean","sd","cv"),
 export.where=getwd(),
 restrict.cores=0.8,
 gene.filters=list(
     length=list(
         length=500
     ),
     avg.reads=list(
            average.per.bp=100,
            quantile=0.25
     ),
     expression=list(
            median=TRUE,
            mean=FALSE,
            quantile=NA,
            known=NA,
            custom=NA
     ),
     biotype=get.defaults("biotype.filter","mm9")
 ),
 out.list=TRUE
)
```

```
head(result$data[["e14.5_vs_adult_8_weeks"]])
```

---

mlfo *MLE dispersion estimate*

Description

MLE function used to estimate negative binomial dispersions from real RNA-Seq data, as in (Soneson and Delorenzi, BMC Bioinformatics, 2013) and (Robles et al., BMC Genomics, 2012). Internal use.

Usage

```
mlfo(phi, y)
```

Arguments

phi	the parameter to be optimized.
y	count samples used to perform the optimization.

Value

The objective function value.

Author(s)

Panagiotis Moulos

Examples

```
# Not yet available
```

`mm9.gene.counts` *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 `sample.list` and `libsize.list` named `sample.list.mm9` and `libsize.list.mm9`.

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/)

`nat2log` *General value transformation*

Description

Logarithmic transformation. Internal use only.

Usage

```
nat2log(x, base = 2, off = 1)
```

Arguments

`x`	input data matrix
`base`	logarithmic base, 2 or 10
`off`	offset to avoid Infinity

Author(s)

Panagiotis Moulos

normalize.deseq *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 metaseqr pipeline).

Usage

```
normalize.deseq(gene.counts, sample.list,
    norm.args = NULL, output = c("matrix", "native"))
```

Arguments

gene.counts	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 metaseqr and is imported during the basic metaseqr workflow.
sample.list	the list containing condition names and the samples under each condition.
norm.args	a list of DESeq normalization parameters. See the result of `get.defaults("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

```
require(DESeq)
data.matrix <- counts(makeExampleCountDataSet())
sample.list <- list(A=c("A1","A2"),B=c("B1","B2","B3"))
diagplot.boxplot(data.matrix,sample.list)

norm.data.matrix <- normalize.deseq(data.matrix,sample.list)
diagplot.boxplot(norm.data.matrix,sample.list)
```

normalize.edaseq *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 metaseqr pipeline).

Usage

```
normalize.edaseq(gene.counts, sample.list,
    norm.args = NULL, gene.data = NULL,
    output = c("matrix", "native"))
```

Arguments

- `gene.counts` 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 metaseqr and is imported during the basic metaseqr workflow.
- `sample.list` the list containing condition names and the samples under each condition.
- `norm.args` a list of EDASeq normalization parameters. See the result of `get.defaults("normalization", "edaseq")` for an example and how you can modify it.
- `gene.data` 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.
- `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

```
require(DESeq)
data.matrix <- counts(makeExampleCountDataSet())
sample.list <- list(A=c("A1","A2"),B=c("B1","B2","B3"))
diagplot.boxplot(data.matrix,sample.list)

lengths <- round(1000*runif(nrow(data.matrix)))
starts <- round(1000*runif(nrow(data.matrix)))
```

```
ends <- starts + lengths
gc=runif(nrow(data.matrix))
gene.data <- data.frame(
    chromosome=c(rep("chr1",nrow(data.matrix)/2),
        rep("chr2",nrow(data.matrix)/2)),
    start=starts,end=ends,gene_id=rownames(data.matrix),gc_content=gc
)
norm.data.matrix <- normalize.edaseq(data.matrix,sample.list,
    gene.data=gene.data)
diagplot.boxplot(norm.data.matrix,sample.list)
```

`normalize.edger` *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 metaseqr pipeline).

Usage

```
normalize.edger(gene.counts, sample.list,
    norm.args = NULL, output = c("matrix", "native"))
```

Arguments

`gene.counts`	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 metaseqr and is imported during the basic metaseqr workflow.
`sample.list`	the list containing condition names and the samples under each condition.
`norm.args`	a list of edgeR normalization parameters. See the result of `get.defaults("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

```
require(DESeq)
data.matrix <- counts(makeExampleCountDataSet())
sample.list <- list(A=c("A1","A2"),B=c("B1","B2","B3"))
diagplot.boxplot(data.matrix,sample.list)

norm.data.matrix <- normalize.edger(data.matrix,sample.list)
diagplot.boxplot(norm.data.matrix,sample.list)
```

---

normalize.nbpseq *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 metaseqr pipeline).

Usage

```
normalize.nbpseq(gene.counts, sample.list,
    norm.args = NULL, libsize.list = NULL,
    output = c("matrix", "native"))
```

Arguments

`gene.counts`	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 metaseqr and is imported during the basic metaseqr workflow.
`sample.list`	the list containing condition names and the samples under each condition.
`norm.args`	a list of NBPSeq normalization parameters. See the result of `get.defaults("normalization", "nbpseq")` for an example and how you can modify it.
`libsize.list`	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. If not provided, the default is the column sums of the `gene.counts` 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

```
require(DESeq)
data.matrix <- counts(makeExampleCountDataSet())
sample.list <- list(A=c("A1","A2"),B=c("B1","B2","B3"))
diagplot.boxplot(data.matrix,sample.list)

norm.data.matrix <- normalize.nbpseq(data.matrix,sample.list)
diagplot.boxplot(norm.data.matrix,sample.list)
```

---

normalize.noiseq *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 metaseqr pipeline).

Usage

```
normalize.noiseq(gene.counts, sample.list,
    norm.args = NULL, gene.data = NULL, log.offset = 1,
    output = c("matrix", "native"))
```

Arguments

`gene.counts`	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 metaseqr and is imported during the basic metaseqr workflow.
`sample.list`	the list containing condition names and the samples under each condition.
`norm.args`	a list of NOISeq normalization parameters. See the result of `get.defaults("normalization", "noiseq")` for an example and how you can modify it.
`gene.data`	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.
`log.offset`	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

```
require(DESeq)
data.matrix <- counts(makeExampleCountDataSet())
sample.list <- list(A=c("A1","A2"),B=c("B1","B2","B3"))
diagplot.boxplot(data.matrix,sample.list)

lengths <- round(1000*runif(nrow(data.matrix)))
starts <- round(1000*runif(nrow(data.matrix)))
ends <- starts + lengths
gc=runif(nrow(data.matrix))
gene.data <- data.frame(
    chromosome=c(rep("chr1",nrow(data.matrix)/2),
        rep("chr2",nrow(data.matrix)/2)),
    start=starts,end=ends,gene_id=rownames(data.matrix),gc_content=gc
)
norm.data.matrix <- normalize.noiseq(data.matrix,sample.list,gene.data)
diagplot.boxplot(norm.data.matrix,sample.list)
```

---

read.targets — *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

```
read.targets(input, path = NULL)
```

Arguments

`input` a 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. 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 or "paired" for paired-end reads. If this column is not provided, single-end reads will be assumed. There is an optional fifth column which should contain the keywords "yes" for standed RNA-Seq reads or "no" for unstranded/unknown. If this column is not provided, unstranded reads will be assumed.

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.

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

```
targets <- data.frame(sample=c("C1","C2","T1","T2"),
    filename=c("C1_raw.bam","C2_raw.bam","T1_raw.bam","T2_raw.bam"),
    condition=c("Control","Control","Treatment","Treatment"))
path <- "/home/chakotay/bam"
write.table(targets,file="~/targets.txt",sep="\t",row.names=FALSE,
    quote=FALSE)
the.list <- read.targets("~/targets.txt",path=path)
sample.list <- the.list$samples
bamfile.list <- the.list$files
```

---

read2count — *SAM/BAM/BED file reader helper for the metaseqr pipeline*

Description

This function is a helper for the metaseqr pipeline, for reading SAM/BAM or BED files when a read counts file is not available.

Usage

```
read2count(targets, annotation, file.type = targets$type,
    has.all.fields = FALSE, multic = FALSE)
```

Arguments

targets	a named list, the output of read.targets.
annotation	see the annotation argument in the main metaseqr function. The "annotation" parameter here is the result of the same parameter in the main function. See also get.annotation.
file.type	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 metaseqr function for the case of auto-guessing.
has.all.fields	a logical variable indicating if all annotation fields used by metaseqr are available (that is apart from the main chromosome, start, end, unique id and strand columns, if also present are the gene name and biotype columns). The default is FALSE.
multic	a logical value indicating the presence of multiple cores. Defaults to FALSE. Do not change it if you are not sure whether package parallel has been loaded or not.

Value

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

Author(s)

Panagiotis Moulos

Examples

```
## Not run:
my.targets <- read.targets("my_mm9_study_bam_files.txt")
sample.list <- my.targets$samples
file.list <- my.targets$files
gene.data <- get.annotation("mm9","gene")
r2c <- read2count(files.list=file.list,
    file.type=my.targets$type,annotation=gene.data)
gene.counts <- r2c$counts
libsize.list <- r2s$libsize

## End(Not run)
```

---

reduce.exons *Merges exons to create a unique set of exons for each gene*

Description

This function uses the "reduce" function of IRanges to construct virtual unique exons for each gene, so as to avoid inflating the read counts for each gene because of multiple possible transcripts. If the user wants transcripts instead of genes, they should be supplied to the original annotation table.

Usage

```
reduce.exons(gr, multic = FALSE)
```

Arguments

gr	a GRanges object created from the supplied annotation (see also the read2count and get.annotation functions.
multic	a logical value indicating the presence of multiple cores. Defaults to FALSE. Do not change it if you are not sure whether package parallel has been loaded or not.

Value

A GRanges object with virtual merged exons for each gene/transcript.

Author(s)

Panagiotis Moulos

Examples

```
require(GenomicRanges)
multic <- check.parallel(0.8)
ann <- get.annotation("mm9","exon")
gr <- makeGRangesFromDataFrame(
    df=ann,
    keep.extra.columns=TRUE,
    seqnames.field="chromosome"
)
re <- reduce.exons(gr,multic=multic)
```

---

reduce.gene.data   *Reduce the gene annotation in case of not all chromosomes present in counts*

Description

This function reduces the gene annotation in case of exon reads and when the data to be analyzed do not contain all the standard chromosomes of the genome under investigation. This can greatly reduce processing time in these cases.

Usage

```
reduce.gene.data(exon.data, gene.data)
```

Arguments

exon.data the exon annotation already reduced to the size of the input exon counts table.

gene.data an annotation data frame from the same organism as `exon.counts` (such the ones produced by `get.annotation`).

Value

The `gene.data` annotation, reduced to have the same chromosomes as in `exon.data`, or the original `gene.data` if `exon.data` do contain the standard chromosomes.

Author(s)

Panagiotis Moulos

Examples

```
data("hg19.exon.data",package="metaseqR")
gene.data <- get.annotation("hg19","gene","ensembl")
reduced.gene.data <- reduce.gene.data(hg19.exon.counts,
    gene.data)
```

`sample.list.hg19` *Human RNA-Seq data with three conditions, three samples*

Description

The sample list for `hg19.exon.counts`. See the data set description.

Format

a named `list` with condition and sample names.

Author(s)

Panagiotis Moulos

Source

GEO (http://www.ncbi.nlm.nih.gov/geo/)

`sample.list.mm9` *Mouse RNA-Seq data with two conditions, four samples*

Description

The sample list for `mm9.gene.counts`. 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/)

`set.arg` *Argument setter*

Description

Set argument(s) to a list of arguments, e.g. normalization arguments.

Usage

```
set.arg(arg.list, arg.name, arg.value = NULL)
```

Arguments

`arg.list`	the initial list of a method's (e.g. normalization) arguments. Can be created with the `get.defaults` function.
`arg.name`	a named list with names the new arguments to be set, and mebers the values to be set or a vector of argument names. In this case, `arg.value` must be supplied.
`arg.value`	when `arg.name` is a vector of argument names, the values corresponding to these arguments.

Value

the `arg.list` with the changed `arg.value` for `arg.name`.

Author(s)

Panagiotis Moulos

Examples

```
norm.list <- get.defaults("normalization","egder")
set.arg(norm.list,list(main.method="glm",logratioTrim=0.4))
```

---

stat.bayseq *Statistical testing with baySeq*

Description

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

Usage

```
stat.bayseq(object, sample.list, contrast.list = NULL,
    stat.args = NULL, libsize.list = NULL)
```

Arguments

`object`	a matrix or an object specific to each normalization algorithm supported by metaseqR, containing normalized counts. Apart from matrix (also for NOISeq), the object can be a SeqExpressionSet (EDASeq), CountDataSet (DESeq) or DGEList (edgeR).
`sample.list`	the list containing condition names and the samples under each condition.
`contrast.list`	a named structured list of contrasts as returned by `make.contrast.list` or just the vector of contrasts as defined in the main help page of `metaseqr`.
`stat.args`	a list of edgeR statistical algorithm parameters. See the result of `get.defaults("statistics", "bayseq")` for an example and how you can modify it.
`libsize.list`	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. If not provided, they will be estimated from baySeq.

Value

A named list of the value 1-likelihood that a gene is differentially expressed, whose names are the names of the contrasts.

Author(s)

Panagiotis Moulos

Examples

```
require(DESeq)
data.matrix <- counts(makeExampleCountDataSet())
sample.list <- list(A=c("A1","A2"),B=c("B1","B2","B3"))
contrast <- "A_vs_B"
norm.data.matrix <- normalize.edaseq(data.matrix,sample.list,gene.data)
p <- stat.bayseq(norm.data.matrix,sample.list,contrast)
```

`stat.deseq` *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 metaseqR.

Usage

```
stat.deseq(object, sample.list, contrast.list = NULL,
    stat.args = NULL)
```

Arguments

`object`	a matrix or an object specific to each normalization algorithm supported by metaseqR, containing normalized counts. Apart from matrix (also for NOISeq), the object can be a SeqExpressionSet (EDASeq), CountDataSet (DESeq) or DGEList (edgeR).
`sample.list`	the list containing condition names and the samples under each condition.
`contrast.list`	a named structured list of contrasts as returned by `make.contrast.list` or just the vector of contrasts as defined in the main help page of `metaseqr`.
`stat.args`	a list of DESeq statistical algorithm parameters. See the result of `get.defaults("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.

Value

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

Author(s)

Panagiotis Moulos

Examples

```
require(DESeq)
data.matrix <- counts(makeExampleCountDataSet())
sample.list <- list(A=c("A1","A2"),B=c("B1","B2","B3"))
contrast <- "A_vs_B"
norm.data.matrix <- normalize.deseq(data.matrix,sample.list)
p <- stat.deseq(norm.data.matrix,sample.list,contrast)
```

---

stat.edger *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 metaseqR.

Usage

```
stat.edger(object, sample.list, contrast.list = NULL,
    stat.args = NULL)
```

Arguments

object	a matrix or an object specific to each normalization algorithm supported by metaseqR, containing normalized counts. Apart from matrix (also for NOISeq), the object can be a SeqExpressionSet (EDASeq), CountDataSet (DESeq) or DGEList (edgeR).
sample.list	the list containing condition names and the samples under each condition.
contrast.list	a named structured list of contrasts as returned by make.contrast.list or just the vector of contrasts as defined in the main help page of metaseqr.
stat.args	a list of edgeR statistical algorithm parameters. See the result of get.defaults("statistics", "edger") for an example and how you can modify it.

Value

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

Author(s)

Panagiotis Moulos

Examples

```
require(DESeq)
data.matrix <- counts(makeExampleCountDataSet())
sample.list <- list(A=c("A1","A2"),B=c("B1","B2","B3"))
contrast <- "A_vs_B"
norm.data.matrix <- normalize.edger(data.matrix,sample.list)
p <- stat.edger(norm.data.matrix,sample.list,contrast)
```

`stat.limma` *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 metaseqR.

Usage

```
stat.limma(object, sample.list, contrast.list = NULL,
    stat.args = NULL)
```

Arguments

`object`	a matrix or an object specific to each normalization algorithm supported by metaseqR, containing normalized counts. Apart from matrix (also for NOISeq), the object can be a SeqExpressionSet (EDASeq), CountDataSet (DESeq) or DGEList (edgeR).
`sample.list`	the list containing condition names and the samples under each condition.
`contrast.list`	a named structured list of contrasts as returned by `make.contrast.list` or just the vector of contrasts as defined in the main help page of `metaseqr`.
`stat.args`	a list of edgeR statistical algorithm parameters. See the result of `get.defaults("statistics", "limma")` for an example and how you can modify it.

Value

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

Author(s)

Panagiotis Moulos

Examples

```
require(DESeq)
data.matrix <- counts(makeExampleCountDataSet())
sample.list <- list(A=c("A1","A2"),B=c("B1","B2","B3"))
contrast <- "A_vs_B"
norm.data.matrix <- normalize.edger(data.matrix,sample.list)
p <- stat.limma(norm.data.matrix,sample.list,contrast)
```

---

stat.nbpseq *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 metaseqR.

Usage

```
stat.nbpseq(object, sample.list, contrast.list = NULL,
    stat.args = NULL, libsize.list = NULL)
```

Arguments

`object`	a matrix or an object specific to each normalization algorithm supported by metaseqR, containing normalized counts. Apart from matrix (also for NOISeq), the object can be a SeqExpressionSet (EDASeq), CountDataSet (DESeq), DGEList (edgeR) or list (NBPSeq).
`sample.list`	the list containing condition names and the samples under each condition.
`contrast.list`	a named structured list of contrasts as returned by `make.contrast.list` or just the vector of contrasts as defined in the main help page of `metaseqr`.
`stat.args`	a list of NBPSeq statistical algorithm parameters. See the result of `get.defaults("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.
`libsize.list`	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. If not provided, the default is the column sums of the `gene.counts` matrix.

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(DESeq)
data.matrix <- counts(makeExampleCountDataSet())
sample.list <- list(A=c("A1","A2"),B=c("B1","B2","B3"))
contrast <- "A_vs_B"
norm.data.matrix <- normalize.nbpseq(data.matrix,sample.list)
p <- stat.nbpseq(norm.data.matrix,sample.list,contrast)
```

`stat.noiseq` *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 metaseqR.

Usage

```
stat.noiseq(object, sample.list, contrast.list = NULL,
    stat.args = NULL, gene.data = NULL, log.offset = 1)
```

Arguments

`object`	a matrix or an object specific to each normalization algorithm supported by metaseqR, containing normalized counts. Apart from matrix (also for NOISeq), the object can be a SeqExpressionSet (EDASeq), CountDataSet (DESeq) or DGEList (edgeR).
`sample.list`	the list containing condition names and the samples under each condition.
`contrast.list`	a named structured list of contrasts as returned by `make.contrast.list` or just the vector of contrasts as defined in the main help page of `metaseqr`.
`stat.args`	a list of edgeR statistical algorithm parameters. See the result of `get.defaults("statistics", "noiseq")` for an example and how you can modify it.

gene.data 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.

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

Value

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

Author(s)

Panagiotis Moulos

Examples

```
require(DESeq)
data.matrix <- counts(makeExampleCountDataSet())
sample.list <- list(A=c("A1","A2"),B=c("B1","B2","B3"))
contrast <- "A_vs_B"
lengths <- round(1000*runif(nrow(data.matrix)))
starts <- round(1000*runif(nrow(data.matrix)))
ends <- starts + lengths
gc=runif(nrow(data.matrix))
gene.data <- data.frame(
    chromosome=c(rep("chr1",nrow(data.matrix)/2),
        rep("chr2",nrow(data.matrix)/2)),
    start=starts,end=ends,gene_id=rownames(data.matrix),gc_content=gc
)
norm.data.matrix <- normalize.noiseq(data.matrix,sample.list,gene.data)
p <- stat.noiseq(norm.data.matrix,sample.list,contrast,
    gene.data=gene.data)
```

---

validate.alg.args *Validate normalization and statistical algorithm arguments*

Description

This function checks and validates the arguments passed by the user to the normalization and statistics algorithms supported by metaseqR. As these are given into lists and passed to the algorithms, the list members must be checked for `NULL`, valid names etc. This function performs these checks and ignores any invalid arguments.

Usage

```
validate.alg.args(normalization, statistics,
    norm.args, stat.args)
```

Arguments

`normalization`	a keyword determining the normalization strategy to be performed by metaseqR. See `metaseqr` main help page for details.
`statistics`	the statistical tests to be performed by metaseqR. See `metaseqr` main help page for details.
`norm.args`	the user input list of normalization arguments. See `metaseqr` main help page for details.
`stat.args`	the user input list of statistical test arguments. See `metaseqr` main help page for details.

Value

A list with two members (`norm.args`, `stat.args`) with valid arguments to be used as user input for the algorithms supported by metaseqR.

Author(s)

Panagiotis Moulos

Examples

```
normalization <- "edaseq"
statistics <- "edger"
norm.args <- get.defaults("normalization","edaseq")
stat.args <- get.defaults("statistics","deseq")
# Will return as is
val <- validate.alg.args(normalization,statistics,norm.args,stat.args)
val$norm.args
val$stat.args
# but...
stat.args <- c(stat.args,my.irrelevant.arg=999)
val <- validate.alg.args(normalization,statistics,norm.args,stat.args)
# irrelevant argument will be removed
val$norm.args
val$stat.args
```

`validate.list.args` *Validate list parameters for several metaseqR functions*

Description

This function validates the arguments passed by the user to the normalization, statistics and filtering algorithms supported by metaseqR. As these are given into lists and passed to the algorithms, the list member names must be valid algorithm arguments for the pipeline not to crash. This function performs these checks and ignores any invalid arguments.

Usage

```
validate.list.args(what, method = NULL, arg.list)
```

Arguments

`what`	what a keyword determining the procedure for which to validate arguments. It can be one of `"normalization"`, `"statistics"`, `"gene.filter"`, `"exon.filter"` or `"biotype.filter"`.
`method`	the normalization/statistics/filtering algorithm included in metaseqR for which to validate user input. 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"`, `"bayseq"`, `"limma"` or `"nbpseq"`. When `method` is `"biotype.filter"`, `what` is the input organism (see the main `metaseqr` help page for a list of supported organisms).
`arg.list`	the user input list of arguments.

Value

A list with valid arguments to be used as user input in the algorithms supported by metaseqR.

Author(s)

Panagiotis Moulos

Examples

```
norm.args.edger <- list(method="TMM",refColumn=NULL,
    logratioTrim=0.3,sumTrim=0.05,doWeighting=TRUE,
    Bcutoff=-1e10,p=0.75)
# Bcutoff does not exist, will throw a warning and ignore it.
norm.args.edger <- validate.list.args("normalization",
    "edger",norm.args.edger)
```

`wapply` *List apply helper*

Description

A wrapper around normal and parallel apply (`mclapply` or multicore package) to avoid excessive coding for control of single or parallel code execution. Internal use.

Usage

```
wapply(m, ...)
```

Arguments

m	a logical indicating whether to execute in parallel or not.
...	the rest arguments to lapply (or mclapply)

Author(s)

Panagiotis Moulos

Examples

```
multic <- check.parallel(0.8)
# Test meaningful only in machines where parallel computation
# supported
if (multic) {
    system.time(r<-wapply(TRUE,1:10,function(x) runif(1e+6)))
    system.time(r<-wapply(FALSE,1:10,function(x) runif(1e+6)))
}
```

---

wp.adjust *Multiple testing correction helper*

Description

A wrapper around the p.adjust function to include also the qvalue adjustment procedure from the qvalue package. Internal use.

Usage

```
wp.adjust(p, m)
```

Arguments

p	a vector of p-values.
m	the adjustment method. See the help of p.adjust.

Author(s)

Panagiotis Moulos

Index