Package 'rnaseqWrapper'

August 29, 2016

Type Package

Title Wrapper for several R packages and scripts to automate RNA-seq analysis

Version 1.0-1

Date 2014-07-22

Author Mark Peterson

Maintainer Mark Peterson <mark.phillip.peterson@gmail.com>

Description This package is designed to streamline several of the common steps for RNA-seq analysis, including differential expression and variant discovery.
For the development build, or to contribute changes to this package, please see our repository at https://bitbucket.org/petersmp/rnaseqwrapper/

License GPL

Depends ecodist, gplots, gtools

Suggests topGO,seqinr,DESeq

NeedsCompilation no

Repository CRAN

Date/Publication 2014-07-23 15:59:32

R topics documented:

`rnaseqWrapper-package` *A Wrapper for RNAseq analysis*

Description

This package wraps several aspects of RNAseq already implemented in R, including DE analysis and analysis of variants. For the development build, or to contribute changes to this package, please see our repository at https://bitbucket.org/petersmp/rnaseqwrapper/

Details

Package:	rnaseqWrapper
Type:	Package
Version:	1.0-1
Date:	2014-07-22
License:	GPL

Author(s)

Mark Peterson

Maintainer: Mark Peterson <mark.phillip.peterson@gmail.com>

References

This package wraps access to several existing functions. Look at each function to see what other pieces are used.

Examples

```
## See the examples in each function, for your interests
```

calcBasicDE *Calculate basics differential expression between pairs of columns*

Description

Calculates the pairwise difference between a series of colums.

This assumes either differences between individuals, or between a single column for each group (e.g. mean, median, or log there of).

This function does not calculate statistical significance, and it is intended only for quick, course-scale analysis and visualization, not for final publication.

Usage

```
calcBasicDE(data,
            colID = "_mean_FPKM_log2",
            whichIndex = "allPairwise",
            matchEnd = TRUE,
            appendName = "diff")
```

Arguments

data	data.frame or matrix with genes as rows and columns containing information, such as counts, FPKM, etc.
colID	Which columns should be used for DE analysis, takes a single column per group, so expects either a single column per treatment (e.g. mean), or will report individual differences. Can be a vector for several things either character (to be grepped) or numeric to indicate which cols to analyze. Accepts regexp arguments, so be careful.
whichIndex	Which colID (e.g. "Control") should the others be compared against. Can be numeric (which of the colID to use) or characther for grep against the names of the colID columns (can have regexp). whichIndex="allPairwise" (default) calculates all pairwise comparisons.
matchEnd	if colID is character, should only the end of the column name be matched. This avoids matching unintended columns.
appendName	What string should be added to the end of each output column? Output column names will be "firstColName_minus_secondColName_appendName"

Details

This function calculates the difference between pairs of columns, but does not report a statistical test.

It is a simple wrapper to find the raw numerical differences between several columns, and output sensibly named columns.

Value

Returns a data.frame with one column for each calculated difference.

Rows and names match the input data.frame, so can be appended to the original data.frame using either cbind or merge.

Author(s)

Mark Peterson

See Also

DESeqWrapper, calcCombVals

Examples

```
## Only run if DESeq is available
if(require(DESeq)){

## Create sample data
## Could be reads or FPKM from your input

exampleCounts <- counts(makeExampleCountDataSet())


testComb <- calcCombVals(exampleCounts,
                         groupID=c("A","B","*"),
                         colID=c("all"))
head(testComb)

basicTest <- calcBasicDE(testComb,"all_mean")

head(basicTest)

}
```

---

calcCombVals *Calculate group values from RNAseq data*

Description

This script uses grep to combine the data from all individuals in a group, for a specific data type(s), and calculates the summary statistic requested.

Usage

```
calcCombVals(data, groupID, colID = "FPKM", combineCols = TRUE,
matchEnd = TRUE, FUN = "mean", functionName = NULL)
```

Arguments

data	A data.frame or matrix containing the individual data (columns) for several genes (rows) from which to calculate the test statistic FUN
groupID	A character vector or list. If character, include the group IDs, to be used as patterns in grep. Make sure the names are unique enough to match only what you want them to. To include all groups for a data type, set groupID = "all","", or "*". If a list, give the column numbers for each group, named with your desired group output name (e.g. groupID = list(control = c(1,2,3), treated = c(4,5,6)))
colID	A character vector of the columns of interest (e.g. FPKM). To include all columns for each group, set colID = "all","", or "*". colID is ignored if groupID is a list.
combineCols	Logical, should the resulting columns be put into one data.frame (default, combineCols = TRUE), or left as enteries in a list.
matchEnd	Logical, should the colId pattern only match the end of the string.
FUN	Either be a function or a character that can be coerced to function using match.fun(FUN). This will be used to name the output columns if functionName is not set. If not a character, set functionName to assign column names for the output
functionName	A string to use in naming the output columns or list names. Defaults to as.character(FUN) if not set.

Details

This function uses grep to grab a series of columns (by group and by column type), and then runs a summary function (FUN) on each set.

This can effectivley calculate group statistics for visualization and course analysis, such as means, medians, etc.

Value

If combineCols = TRUE, returns a data.frame with one column for each set (groupID by colID).

If combineCols = FALSE, returns a list with one entry for each set (groupID by colID).

Author(s)

Mark Peterson

See Also

calcBasicDE

Examples

```
## Only run if DESeq is available
if(require(DESeq)){

## Create sample data
## Could be reads or FPKM from your input
exampleCounts <- counts(makeExampleCountDataSet())


testComb <- calcCombVals(exampleCounts,
                         groupID=c("A","B","*"),
                         colID=c("all"))
head(testComb)
}
```

`calcHornMatrix` *Calculate a Horn distance matrix*

Description

This function calculates the pairwise Horn distance between samples based on relative presence of observed variables.

Usage

```
calcHornMatrix(inputTable)
```

Arguments

`inputTable`	A matrix or data.frame of relative frequencies of whatever is being measured. Each row is a variable (e.g. gene) and each column is a sample (e.g. individual). The algorithm expects each column to sum to 1. That is, each entry should be the portion of observations (e.g. reads) representing the variable (gene) from the total observations for a sample (individual).

Value

Returns a matrix of pair-wise similarity scores for each column.

Author(s)

Mark Peterson

References

This was (heavily) modified from a script provided by a collaborator of Gina Lamendella's. I need to get more information

Examples

```
## Only run if DESeq is available
if(require(DESeq)){

## Create sample counts
## Thes could be the reads or FPKM from input data instead
require(DESeq)
exampleCounts <- counts(makeExampleCountDataSet())

testHorn <- calcHornMatrix(exampleCounts)
head(testHorn)


## Plot the results
distMat <-as.dist( (1-testHorn), diag=FALSE, upper=FALSE)

# scores to plot MDS (ecodist)
scores_ADNA <-nmds(distMat, mindim=2, maxdim=2)
scores_ADNA <- nmds.min(scores_ADNA)

# Set colors to match treatments
treatCol <- c("red","red","blue","blue","blue")

plot(scores_ADNA[,1:2],pch=19,col=treatCol)

}
```

`calcLogVal` *Calculate log value of RNAseq columns*

Description

Calculate the log value of a series of columns, allowing an offset to avoid returning "-Inf."

Usage

```
calcLogVal(data, colID = "FPKM", offset = 0.1,
           setBase = 2, matchEnd = TRUE)
```

Arguments

`data`	A matrix or data.frame containing gene (row) data for several individuals (columns).
`colID`	Either a numeric vector of columns to calculate the log of, or a character vector with patterns (e.g., FPKM) to match in the names of `data`. If a character vector, make sure the names are unique enough to match only what you want them to, as they will be called to grep.

offset	How much to add before taking log to avoid returning -Inf.
setBase	A positive number to pass to log to set the logarthimic base. Defaults to setBase = 2
matchEnd	Logical, should the colId pattern only match the end of the string.

Value

Returns a data.frame with the log value of the desired columns.

Rows match the input data, so can be appended using cbind or merge.

Author(s)

Mark Peterson

Examples

```
## Only run if DESeq is available
if(require(DESeq)){

## Create sample data
## Could be reads or FPKM from your input
exampleCounts <- counts(makeExampleCountDataSet())

## Only calculate for the "2" columns
## This could be reads, if you have multiple column types
testLog <- calcLogVal(exampleCounts,
                      colID="2",
                      matchEnd=TRUE)
head(testLog)

## Calculate log of all columns
testLog2 <- calcLogVal(exampleCounts,
                       colID="*",
                       setBase=2)
head(testLog2)

}
```

---

calculateThirdPosBias *Calculate the third position bias of polymorphisms*

Description

Determines which variant positions are in the third codon position, or estimates that based on frequency, and reports the proportion of variants in each gene that are in the third position.

These are presumed to be synonomous more often, but if codon information is known, users should run determineSynonymous instead.

Usage

```
calculateThirdPosBias(varTable,
                      seqIDcol = 1,
                      refPosCol = "Reference.Position",
                      readCutoffs = 0,
                      colprepend = "nVar_",
                      codonStartPos = NULL)
```

Arguments

varTable	A data.frame with rows for each position in each gene with a variant present. Columns give various information for each included individual. This is expecting the format from readVariantFiles, which should be easy to emulate
seqIDcol	Which column is the sequence ID in? Can be numeric or character.
refPosCol	Which column is the referencece position in? Can be numeric or character.
readCutoffs	How many variable positions need to be present to calculate bias. Set to 1 (or 0 or NULL) to include all. Without a reference, small numbers will be almost meaningless.
colprepend	What name should the output columns be prepended with
codonStartPos	If "cds" assumes all start at position 1 (default). If NULL, will treat the frame as unknown and assign the most frequent position as the putative third position. In the future, can be a vector giving which position each gene starts at; currently not handled.

Value

Returns a matrix with genes as rows and with the number of variants in each position and proportion of variants in the third position as columns.

Author(s)

Mark Peterson

See Also

readVariantFiles, determineSynonymous

Examples

```
## Load example data
data(varScanExample)

calculateThirdPosBias(varScanExample,
                      refPosCol="Position",
```

```
                    codonStartPos="cds")
```

DESeqWrapper *A wrapper for DESeq*

Description

This function provides a useful wrapper for the DESeq package, automating several of the manual steps to provide a basic DE analysis.

Users should use this as a guide, not necessarily as a final analysis.

The functions `plotDE` and `plotDispEsts` are only called from within this function.

Usage

```
DESeqWrapper(countData,
             conditions,
             whichGeneNames = 0,
             outNamePrefix = "DESeqOutputs",
             comps = "allPairwise",
             conds = NULL,
             colorSet = NULL,
             makePDFs = TRUE,
             writeScaled = FALSE,
             writeDE = TRUE,
             pCut = 0.05,
             dispMethod = "pooled",
             dispSharingMode = "maximum")
```

Arguments

`countData`	A data.frame or matrix with raw count data for each gene (row) and sample (column).
`conditions`	Character vector with the groups to be analyzed.
`whichGeneNames`	Numeric or character, which column has the gene names. Defaults to 0, which is rownames (preferred).
`outNamePrefix`	A string to prepend to written files; can included path specification.
`comps`	character vector of which contrasts to run in the format 1vs2 with numbers matching order of `conditions` above
`conds`	A character vector of the conditions for each column, matching the order of the columns in `countData`. If "NULL" (default) the function will use grep on conditions against column names to automatically generate the groups
`colorSet`	Vector of colors, specified in any valid R format, to use as labels for the conditions. If "NULL" (default), default colors will be used.

makePDFs	Logical, should figures be output as pdfs. If false, no figures will be generated.
writeScaled	Logical, should the scaled countData be written as a .txt file.
writeDE	Logical, should the DE results be written as .txt files.
pCut	What (FDR-corrected) q-value should be used as a cut-off.
dispMethod	Which method should be used for estimating dispersion by DESeq? See estimateDispersions for details and available options.
dispSharingMode	Which sharing mode should be used for estimating dispersion by DESeq? See estimateDispersions for details and available options.

Value

Writes txt and pdf files as run, and returns a list with:

deOutputs	A list with one data.frame for each contrast analyzed
normalizedReads	The normalized read data
isSignificant	A data.frame telling whether each gene is sig for each contrast

Author(s)

Mark Peterson

Examples

```
## Only run if DESeq is available
if(require(DESeq)){

## Create an example count data set
exampleCounts <- counts(makeExampleCountDataSet())[1:500,]

## Note, from your data, this might look like:
# exampleCounts <- myInputData[,grep("READS",names(myInputData))]
# row.names(exampleCounts) <- myInputData$geneNameColumn

## Note, outputs save to disk are turned off
## Set each to TRUE to save to your working directory
test <- DESeqWrapper(exampleCounts,
                     conditions=c("A","B"),
                     writeScaled=FALSE,
                     writeDE=FALSE,
                     makePDFs=FALSE)

## Look at the outputs
head(test$deOutputs$AvsB)
head(test$normalizedReads)
head(test$isSignificant)

}
```

`determineSynonymous` *Determine whether or not variants are synonomous*

Description

From a table of variants, determine whether each is synonymous or non-synonomous, assuming all are in the coding region.

Usage

```
determineSynonymous(varTable,
                    seqIDcol = 1,
                    refPosCol = "Reference.Position",
                    refAlleleCol = "Reference",
                    varAlleleCol = "Allele",
                    readCutoffs = 1,
                    colprepend = "snvs_",
                    codonStartPos = "cds",
                    referenceSeqs)
```

Arguments

`varTable`	A data.frame with rows for each position in each gene with a variant present. Columns give various information for each included individual. This is expecting the format from `readVariantFiles`, which should be easy to emulate
`seqIDcol`	Which column is the sequence ID in? Can be numeric or character.
`refPosCol`	Which column is the referencece position in? Can be numeric or character.
`refAlleleCol`	Which column has the reference allele? Can be numeric or character.
`varAlleleCol`	Which column has the variable alleles? Can be numeric or character.
`readCutoffs`	How many variable positions need to be present to calculate bias. Set to 1 (or 0 or NULL) to include all. Without a reference, small numbers will be almost meaningless.
`colprepend`	What name should the output columns be prepended with.
`codonStartPos`	If "cds" assumes all start at position 1 (default). In the future, can be a vector giving which position each gene starts at; currently not handled.
`referenceSeqs`	List of FASTA sequences, with names being gene names as listed in `seqIDcol` and containing the sequences. This is the format produced by `read.fasta`, but can be emulated.

Value

Returns a list with:

`variantInfo`	A matrix of info for each variant, including the reference and variant values for allele and amino acid, and whether or not is synonomous
`geneInfo`	A matrix of info for each gene, with number and proportion for both synonymous and non-synonomous variants for the gene.

Author(s)

Mark Peterson

See Also

`read.fasta`, `nSynNonSites`, `kaksFromVariants`

Examples

```
## Load needed data
data(varScanExample)
data(fastaExample)

determineSynonymous(varTable=varScanExample,
                    refPosCol=2,
                    refAlleleCol="Ref",
                    varAlleleCol="Var",
                    referenceSeqs=fastaExample)
```

`heatmap.mark` *Enhanced Heat Map, further modified*

Description

This heatmap adds some functional control to the extensions provided by `heatmap.2` to the standard R `heatmap` function. Namely, this function adds the ability to suppress the label of the color key, and modifies the defaults for `scale`, `trace`, `col`, and `density.info` to match the more common usage in RNAseq analysis. In addition, it allows the suppression of the hardcoded layouts, using `plotNew = FALSE` to allow combining multiple heatmaps in a single figure, though caution is warranted in arranging your own layout.

Usage and details below are borrowed from that function; for more complete examples, see those help pages.

Usage

```
heatmap.mark (x,

            # dendrogram control
            Rowv = TRUE,
            Colv=if(symm)"Rowv" else TRUE,
            distfun = dist,
            hclustfun = hclust,
            dendrogram = c("both","row","column","none"),
            symm = FALSE,

            # data scaling
            scale = c("row","none", "column"),
            na.rm=TRUE,

            # image plot
            revC = identical(Colv, "Rowv"),
            add.expr,

            # mapping data to colors
            breaks,
            symbreaks=min(x < 0, na.rm=TRUE) || scale!="none",

            # colors
            col="rnaSeqColors",

            # block sepration
            colsep,
            rowsep,
            sepcolor="white",
            sepwidth=c(0.05,0.05),

            # cell labeling
            cellnote,
            notecex=1.0,
            notecol="cyan",
            na.color=par("bg"),

            # level trace
            trace=c("none","column","row","both"),
            tracecol="cyan",
            hline=median(breaks),
            vline=median(breaks),
            linecol=tracecol,

            # Row/Column Labeling
            margins = c(5, 5),
            ColSideColors,
```

```
RowSideColors,
cexRow = 0.2 + 1/log10(nr),
cexCol = 0.2 + 1/log10(nc),
labRow = NULL,
labCol = NULL,
srtRow = NULL,
srtCol = NULL,
adjRow = c(0,NA),
adjCol = c(NA,0),
offsetRow = 0.5,
offsetCol = 0.5,

# color key + density info
key = TRUE,
keysize = 1.5,
density.info=c("none","histogram","density"),
denscol=tracecol,
symkey = min(x < 0, na.rm=TRUE) || symbreaks,
densadj = 0.25,

# plot labels
main = NULL,
xlab = NULL,
ylab = NULL,

# plot layout
lmat = NULL,
lhei = NULL,
lwid = NULL,

# extras for this function
scaleLabel = NULL,
plotNew = TRUE,
...
)
```

Arguments

x	numeric matrix of the values to be plotted.
Rowv	determines if and how the *row* dendrogram should be reordered. By default, it is TRUE, which implies dendrogram is computed and reordered based on row means. If NULL or FALSE, then no dendrogram is computed and no reordering is done. If a dendrogram, then it is used "as-is", ie without any reordering. If a vector of integers, then dendrogram is computed and reordered based on the order of the vector.
Colv	determines if and how the *column* dendrogram should be reordered. Has the options as the Rowv argument above and *additionally* when x is a square matrix,

	Colv = "Rowv" means that columns should be treated identically to the rows.
distfun	function used to compute the distance (dissimilarity) between both rows and columns. Defaults to dist.
hclustfun	function used to compute the hierarchical clustering when Rowv or Colv are not dendrograms. Defaults to hclust.
dendrogram	character string indicating whether to draw 'none', 'row', 'column' or 'both' dendrograms. Defaults to 'both'. However, if Rowv (or Colv) is FALSE or NULL and dendrogram is 'both', then a warning is issued and Rowv (or Colv) arguments are honoured.
symm	logical indicating if x should be treated **symm**etrically; can only be true when x is a square matrix.
scale	character indicating if the values should be centered and scaled in either the row direction or the column direction, or none. The default is "row" if symm false, and "none" otherwise.
na.rm	logical indicating whether NA's should be removed.
revC	logical indicating if the column order should be reversed for plotting, such that e.g., for the symmetric case, the symmetry axis is as usual.
add.expr	expression that will be evaluated after the call to image. Can be used to add components to the plot.
breaks	(optional) Either a numeric vector indicating the splitting points for binning x into colors, or a integer number of break points to be used, in which case the break points will be spaced equally between min(x) and max(x).
symbreaks	Boolean indicating whether breaks should be made symmetric about 0. Defaults to TRUE if the data includes negative values, and to FALSE otherwise.
col	colors used for the image. Defaults to heat colors (heat.colors).
colsep, rowsep, sepcolor	(optional) vector of integers indicating which columns or rows should be separated from the preceding columns or rows by a narrow space of color sepcolor.
sepwidth	(optional) Vector of length 2 giving the width (colsep) or height (rowsep) the separator box drawn by colsep and rowsep as a function of the width (colsep) or height (rowsep) of a cell. Defaults to c(0.05, 0.05)
cellnote	(optional) matrix of character strings which will be placed within each color cell, e.g. p-value symbols.
notecex	(optional) numeric scaling factor for cellnote items.
notecol	(optional) character string specifying the color for cellnote text. Defaults to "green".
na.color	Color to use for missing value (NA). Defaults to the plot background color.
trace	character string indicating whether a solid "trace" line should be drawn across 'row's or down 'column's, 'both' or 'none'. The distance of the line from the center of each color-cell is proportional to the size of the measurement. Defaults to 'none'.
tracecol	character string giving the color for "trace" line. Defaults to "cyan".

`hline, vline, linecol`
: Vector of values within cells where a horizontal or vertical dotted line should be drawn. The color of the line is controlled by `linecol`. Horizontal lines are only plotted if `trace` is 'row' or 'both'. Vertical lines are only drawn if `trace` 'column' or 'both'. `hline` and `vline` default to the median of the breaks, `linecol` defaults to the value of `tracecol`.

`margins`
: numeric vector of length 2 containing the margins (see `par(mar= *)`) for column and row names, respectively.

`ColSideColors`
: (optional) character vector of length `ncol(x)` containing the color names for a horizontal side bar that may be used to annotate the columns of `x`.

`RowSideColors`
: (optional) character vector of length `nrow(x)` containing the color names for a vertical side bar that may be used to annotate the rows of `x`.

`cexRow, cexCol`
: positive numbers, used as `cex.axis` in for the row or column axis labeling. The defaults currently only use number of rows or columns, respectively.

`labRow, labCol`
: character vectors with row and column labels to use; these default to `rownames(x)` or `colnames(x)`, respectively.

`srtRow, srtCol`
: angle of row/column labels, in degrees from horizontal

`adjRow, adjCol`
: 2-element vector giving the (left-right, top-bottom) justification of row/column labels (relative to the text orientation).

`offsetRow, offsetCol`
: Number of character-width spaces to place between row/column labels and the edge of the plotting region.

`key`
: logical indicating whether a color-key should be shown.

`keysize`
: numeric value indicating the size of the key

`density.info`
: character string indicating whether to superimpose a 'histogram', a 'density' plot, or no plot ('none') on the color-key.

`denscol`
: character string giving the color for the density display specified by `density.info`, defaults to the same value as `tracecol`.

`symkey`
: Boolean indicating whether the color key should be made symmetric about 0. Defaults to `TRUE` if the data includes negative values, and to `FALSE` otherwise.

`densadj`
: Numeric scaling value for tuning the kernel width when a density plot is drawn on the color key. (See the `adjust` parameter for the `density` function for details.) Defaults to 0.25.

`main, xlab, ylab`
: main, x- and y-axis titles; defaults to none.

`lmat, lhei, lwid`
: visual layout: position matrix, column height, column width. See below for details

`scaleLabel`
: What label should be used for the colorkey? "NULL" suppresses the label

`plotNew`
: Logical. Should this heatmap be drawn on a new plot. If FALSE, you need to provide your own layout that will encompass all plots you intend to put in the figure. Refer to the argument information for `lmat`, `lhei`, and `lwid` as well as the details and examples below for more information on your options for this.

`...`
: additional arguments passed on to `image`

Details

If either Rowv or Colv are dendrograms they are honored (and not reordered). Otherwise, dendrograms are computed as dd <- as.dendrogram(hclustfun(distfun(X))) where X is either x or t(x).

If either is a vector (of "weights") then the appropriate dendrogram is reordered according to the supplied values subject to the constraints imposed by the dendrogram, by reorder(dd, Rowv), in the row case.

If either is missing, as by default, then the ordering of the corresponding dendrogram is by the mean value of the rows/columns, i.e., in the case of rows, Rowv <- rowMeans(x, na.rm=na.rm).

If either is NULL, *no reordering* will be done for the corresponding side.

If scale="row" the rows are scaled to have mean zero and standard deviation one. There is some empirical evidence from genomic plotting that this is useful.

The default colors range from red to white (heat.colors) and are not pretty. Consider using enhancements such as the **RColorBrewer** package, http://cran.r-project.org/src/contrib/PACKAGES.html#RColorBrewer to select better colors.

By default four components will be displayed in the plot. At the top left is the color key, top right is the column dendogram, bottom left is the row dendogram, bottom right is the image plot. When RowSideColor or ColSideColor are provided, an additional row or column is inserted in the appropriate location. This layout can be overriden by specifiying appropriate values for lmat, lwid, and lhei. lmat controls the relative postition of each element, while lwid controls the column width, and lhei controls the row height. See the help page for layout for details on how to use these arguments.

If plotNew = FALSE, then heatmap.mark will not reset the current layout before plotting. Thus, if this operates on a brand new plot, each of the four elements (described above) will be plotted as a separate plot. Instead, before running the first plot you intend to include, using layout or a similar function to specify the order in which plots should be placed. See the usage examples below for an example.

Value

Invisibly, a list with components

rowInd	row index permutation vector as returned by order.dendrogram.
colInd	column index permutation vector.
call	the matched call
rowMeans, rowSDs	mean and standard deviation of each row: only present if scale="row"
colMeans, colSDs	mean and standard deviation of each column: only present if scale="column"
carpet	reordered and scaled 'x' values used generate the main 'carpet'
rowDendrogram	row dendrogram, if present
colDendrogram	column dendrogram, if present
breaks	values used for color break points

col	colors used
vline	center-line value used for column trace, present only if trace="both" or trace="column"
hline	center-line value used for row trace, present only if trace="both" or trace="row"
colorTable	A three-column data frame providing the lower and upper bound and color for each bin

Note

The original rows and columns are reordered *in any case* to match the dendrogram, e.g., the rows by order.dendrogram(Rowv) where Rowv is the (possibly reorder()ed) row dendrogram.

heatmap.2() uses layout and draws the image in the lower right corner of a 2x2 layout. Consequentially, it can **not** be used in a multi column/row layout, i.e., when par(mfrow= *) or (mfcol= *) has been called.

heatmap.mark() allows this behavior to be over-ridden using plotNew = FALSE, though the user is cautioned that arranging the output manually may take substantial effort.

Author(s)

Mark Peterson, making small revisions to the fantastic code of Andy Liaw, original; and R. Gentleman, M. Maechler, W. Huber, G. Warnes, revisions.

See Also

image, hclust,heatmap.2

Examples

```
## Below are examples of the changes made from heatmap.2()
##  for more complete examples of all this code can do
##  see ?heatmap.2

#####################################
## Read in and prepare data to plot ##
#####################################

## Find where the data is stored (or use your own)
pathToData <- try(system.file("",package="rnaseqWrapper",mustWork=TRUE))

if(class(pathToData) != "try-error"){
## Make sure the data were found before proceeding

## Read in the data
## Note, the files here are compressed,
##  but yours do not need to be
countData <- mergeCountFiles(paste(pathToData,"/data/",sep=""),".genes.results.txt.gz")

## limit to count data for 50 rows
## note that these are not, necessarily DE genes
toPlot <- countData[51:100,grep(".expected_count",names(countData))]
```

```
## Trim the names to make the plots a bit nicer:
names(toPlot) <- gsub(".expected_count","", names(toPlot))

#################
## Simple plot ##
#################

heatmap.mark(as.matrix(toPlot),cexCol = 0.75,labRow = FALSE)


########################################
## More complex, add labels and legend ##
########################################

myLabelColors <- rep(c("red","blue"),each = dim(toPlot)[2]/2)

heatmap.mark(as.matrix(toPlot),
             cexCol = 0.75,labRow = FALSE,
             scaleLabel = "",
             ColSideColors = myLabelColors)

par(xpd=TRUE) ## To allow legend on top of other stuff
legend(x="topleft",inset=c(-.02,.08),
       bty="n", cex=.8,
       legend= c("Female","Male"),
       fill=unique(myLabelColors),
       title="Sex")
par(xpd=FALSE) ## To reset




#########################
## With multiple panels ##
#########################

## Set your own layout
## Note, that each heatmap plots 4 objects when no color labels are included
## So the offset for each additional one needs to b 4 + the options
## If you use row or column labels, additional plots are drawn
## In addition, you will likely want to play with the widths and
## heights of each element.

baseLayout <- matrix(c(4,3,2,1), nrow = 2, byrow = TRUE)

layout(cbind(baseLayout,baseLayout + 4),
       widths = c(1,2,1,2), heights = c(1,2), respect = FALSE)

heatmap.mark(as.matrix(toPlot),
             cexCol = 0.75,labRow = FALSE,
             scaleLabel = "",
             plotNew = FALSE)
```

```
heatmap.mark(as.matrix(toPlot),
             cexCol = 0.75,labRow = FALSE,
             scaleLabel = "",
             plotNew = FALSE)



}
```

---

kaksFromVariants *Calculate Ka/Ks ratios from a table of variants*

Description

From a table of variants, determine the ka/ks ratio and the number of synonomous/non-synonomous sites.

Usage

```
kaksFromVariants(varTable,
                    seqIDcol = 1,
                    refPosCol = "Reference.Position",
                    refAlleleCol = "Reference",
                    varAlleleCol = "Allele",
                    readCutoffs = 1,
                    codonStartPos = "cds",
                    referenceSeqs)
```

Arguments

varTable	A data.frame with rows for each position in each gene with a variant present. Columns give various information for each included individual. This is expecting the format from readVariantFiles, which should be easy to emulate. Small changes to the names make it ideal for VarScan formats as well.
seqIDcol	Which column is the sequence ID in? Can be numeric or character.
refPosCol	Which column is the referencece position in? Can be numeric or character.
refAlleleCol	Which column has the reference allele? Can be numeric or character.
varAlleleCol	Which column has the variable alleles? Can be numeric or character.
readCutoffs	How many variable positions need to be present to calculate bias. Set to 1 (or 0 or NULL) to include all. Without a reference, small numbers will be almost meaningless.
codonStartPos	If "cds" assumes all start at position 1 (default). In the future, can be a vector giving which position each gene starts at; currently not handled.

referenceSeqs	List of FASTA sequences, with names being gene names as listed in seqIDcol and containing the sequences. This is the format produced by read.fasta, but can be emulated.

Value

Returns a matrix of info for each gene with:

ka	rate of non-synonomous subsitutions
ks	rate of synonomous subsitutions
kaks	the ka/ks ratio; Note that ka/ks will be NA for genes with no scored variants and for any gene for which ka or ks are returned as negative
nSynSites	the number of sites deemed synonomous (includes half of the two-fold synonomous sites)
nNonSynSites	the number of sites deemed non-synonomous (includes half of the two-fold synonomous sites)

Author(s)

Mark Peterson

See Also

nSynNonSites, determineSynonymous, kaks, read.fasta

Examples

```
## Load needed data
data(varScanExample)
data(fastaExample)

kaksFromVariants(varTable=varScanExample,
                 refPosCol=2,
                 refAlleleCol="Ref",
                 varAlleleCol="Var",
                 referenceSeqs=fastaExample)
```

---

mergeCountFiles *Merge multiple expression count data files for RNAseq*

Description

Reads in the count data files from each sample of an RNAseq experiment and then combines the files into a single data.frame, useful for several downstream applications.

Usage

```
mergeCountFiles(fileDir,
                fileID = "*.genes.results$",
                fileSep = "\t",
                seqIDcol = 1,
                colsToKeep = c("expected_count","FPKM"),
                idCols = NULL,
                minMatchToMerge = 0.5)
```

Arguments

`fileDir`	The path to the directory containing all of the count data files.
`fileID`	Character to use to limit which files are imported; regular expressions allowed. This `fileID` is also removed from the file names when naming the output columns.
`fileSep`	The column delimiter used in the file (e.g. "," or "\t")
`seqIDcol`	Which column has the gene name or other identifier? Can be numeric or character.
`colsToKeep`	Which columns of info should be kept for the output? Can be a vector of either numeric or character.
`idCols`	Which columns of general site information should be kept? This limits these to one single column, rather than one for each sample. Note, that row.names of the output will already match `seqIDcol`. This can be a vector of either numeric or character, but must match the format of `seqIDcol` *if* the rows of your input data are not all in the same order.
`minMatchToMerge`	If your data are not in the same order, what portion of gene names must be in common to proceed with merge? This is a protection step to avoid accidentally merging datasets from different references.

Details

Reads in the count data files from `fileDir` and merges by gene. It checks to see if the information is all in the same order, and issues a warning if not because it may suggest data are from different reference files.

Value

Returns a data.frame with a row for each gene, and columns (named from the file names), for each sample for each data type kept

Author(s)

Mark Peterson

Examples

```
## Find where the data is stored (or use your own)
pathToData <- try(system.file("",package="rnaseqWrapper",mustWork=TRUE))

if(class(pathToData) != "try-error"){
## Make sure the data were found before proceeding

## Read in the data
## Note, the files here are compressed,
##  but yours do not need to be
testCountData <- mergeCountFiles(paste(pathToData,"/data/",sep=""),".genes.results.txt.gz")

## Display the contents
head(testCountData)

}

## Not run:
    ## On your data, it will look more like:
    mergedCountData <- mergeCountFiles("/path/to/countData")
    head(mergedCountData)

## End(Not run)
```

`nSynNonSites` *Calculate the number of (non) synonomous sites*

Description

From gene sequences, determine the number of synonomous/non-synonomous sites.

Usage

```
nSynNonSites(geneNames,
             codonStartPos = "cds",
             referenceSeqs)
```

Arguments

`geneNames`	A character vector or list (each with character element) giving the gene names to be analyzed.
`codonStartPos`	If "cds" assumes all start at position 1 (default). In the future, can be a vector giving which position each gene starts at; currently not handled.
`referenceSeqs`	List of FASTA sequences, with names being gene names as listed in `seqIDcol` and containing the sequences. This is the format produced by `read.fasta`, but can be emulated.

Value

Returns a matrix of info for each gene with:

nSynSites	the number of sites deemed synonomous (includes half of the two-fold synonomous sites)
nNonSynSites	the number of sites deemed non-synonomous (includes half of the two-fold synonomous sites)

Author(s)

Mark Peterson

See Also

determineSynonymous, kaksFromVariants, kaks, read.fasta

Examples

```
## Load needed data
data(fastaExample)

nSynNonSites(names(fastaExample),
             referenceSeqs=fastaExample)
```

---

parseVarScan *Parse a VarScan output*

Description

Separate the pool and sample call information into usable columns.

Usage

```
parseVarScan(file, sampleNames = NULL, ignoreIndels = TRUE)
```

Arguments

file	Either a character vector of length 1 giving the name of a tab-separated VarScan file to read in, or a data.frame or matrix containing the VarScan output table.
sampleNames	A character vector giving the names to assign to the split sample calls.
ignoreIndels	Logical: should positions with indels be omitted from the returned file (default: TRUE)

Details

This assumes that the headers have not been modified

Value

Returns a data.frame with rows for each row (ommiting the indels if ignoreIndels == TRUE), but with each portion of the calls separated.

Author(s)

Mark Peterson

References

Relies on the VarScan output documented here: http://varscan.sourceforge.net/using-varscan.html

Examples

```
## Get example data
## This could be just read in using read.table for your data
data(varScanExample)

parseTest <- parseVarScan(varScanExample,
                          sampleNames=LETTERS[1:10])

head(parseTest)
```

---

readVariantFiles *Read in variant files for RNAseq*

Description

Reads in the variant files from each sample of an RNAseq experiment and then combines the files into a single data.frame, useful for several downstream applications.

Usage

```
readVariantFiles(fileDir,
                 sepSymbol = "_",
                 fileID = "*_variants.txt",
                 firstColName = "SEQ_ID",
                 fileSep = "\t",
                 idCols = 5,
                 refPosCol = "Reference.Position",
```

```
colToSort = "Coverage",
removeDups = TRUE,
returnMerged = TRUE,
returnSing = FALSE,
limitGenes = NULL,
omitRefMatches = TRUE,
refAlleleCol = "Reference$",
varAlleleCol = "Allele")
```

Arguments

fileDir	The path to the directory containing all of the variant files.
sepSymbol	The symbol that separates the sample names from other info in the file name. Used to pull names for columns in the combined file. Set to "" if the full file name should be used.
fileID	character to use to limit which files are imported; regular expressions allowed
firstColName	What should the first column be renamed to. Set to NULL or "" to leave the column as is. Intended to stanardize and to match the column names in other parts of the analysis pipeline.
fileSep	The column delimiter used in the file (e.g. "," or "\t")
idCols	How many columns of position information are there? Avoids including duplicated information in the combined ouput.
refPosCol	Which column has the reference position? Can be numeric or character
colToSort	Which column should be used to keep one line per position, if `removeDups == TRUE`? Can be numeric or character.
removeDups	Logical, should duplicates at a position be removed? This is necessary to avoid massive over merging
returnMerged	Logical, should the merged variants be returned?
returnSing	Logical, should each of the separate variant files be returned?
limitGenes	A character vector listing the genes to include. This can be useful if your variant files include genes that you are not interested in analyzing (e.g. things without a blast hit).
omitRefMatches	Logical, should 'variants' which match the reference be excluded? This is useful if your variant file includes rows for reads aligning to the reference allele, which may be accidentally set as the main 'variant' in this function. Defaults to TRUE.
refAlleleCol	Which column has the reference allele? Can be numeric or character.
varAlleleCol	Which column has the variable alleles? Can be numeric or character.

Details

Reads in the variant files from `fileDir` and merges by gene and position.

Value

Output is based on returnMerged & returnSing returns:

If returnMerged: a data.frame with the merged variants

If returnSing: a list of the singVariants (cleaned if removeDups=TRUE)

If both TRUE: a list with both of the above

Author(s)

Mark Peterson

Examples

```
## Not run:

mergedVariants <- readVariantFiles (
      fileDir="path/to/variant/directory",
      fileID = "*_variants.txt",
      firstColName = "SEQ_ID",
      idCols = 4,
      refPosCol = "Region"
      )


## End(Not run)
```

---

rnaseqWrapperCountData

Expression data for use in examples for the rnaseqWrapper package

---

Description

Example expression data for 2654 genes, each of six files is data for a different sample. (read in with `mergeCountFiles`).

Usage

```
data(male_1.genes.results)
```

Format

Each file is a tab-delimited output from RSEM.

Source

These are from a currently unpublished analysis, and were randomly chosen.

See Also

mergeCountFiles

Examples

```
data(male_1.genes.results)

## Will normally read in with mergeCountData()
```

---

rnaseqWrapperData *Data for use in examples for the rnaseqWrapper package*

Description

A sample of 86 rows (from three genes) of a VarScan output, and the corrresponding fasta sequence information (read in with `read.fasta`).

Usage

```
data(varScanExample)
data(fastaExample)
```

Source

These are from a currently unpublished analysis, and were randomly chosen.

Examples

```
data(varScanExample)
head(varScanExample)

data(fastaExample)
fastaExample
```

---

runGOAnalysis *Run basic GO analysis*

Description

This function wraps the topGO-package an provides a streamlined appraoch to GO analysis.

Sensible defaults are included, though may not be sufficient for all uses.

Usage

```
runGOAnalysis(sigGenes,
              expGenes,
              goAnno,
              pValThresh = 1,
              plotGO = FALSE,
              ontology = "BP",
              algorithm = "weight",
              statistic = "fisher",
              description = NULL)
```

Arguments

sigGenes	A character vector with the names of significant genes.
expGenes	A character vector with the names of all reference genes, for example, those expressed in the tissu of interest.
goAnno	A (named) list of of character vectors with GO identifiers for each gene, such as returned by readMappings
pValThresh	Numeric, what p-value (not corrected, see value below) threshold should be used to determine what should be returned. Defaults to 1 to return all GO terms analyzed to allow the user to perform multiple-testing corrections as desired.
plotGO	Logical - Should a plot be generated? If TRUE, plots to the currently active device.
ontology	Which ontology should be analyzed by runTest? Can be any of "BP", "MF", or "CC", for Biological Process, Molecular Function, or Cellular Component, respectively.
algorithm	Which algorithm should be used by runTest? Available values can be found with whichAlgorithms; defaults to "weight".
statistic	Which statistic should be used by runTest? Available values can be found with whichTests; defaults to "fisher".
description	A string to use in describing the go data set. Not currently used because the GOData object is not returned.

Value

Returns a data.frame with a row for each significant GO term. Note that it returns p-values, rather than adjusted p-values. The authors of topGO appear to feel strongly about this, so I have deferred to their choice.

I do agree with them that the GO graph is inherently non-independent making most methods for correction overly-conservative.

In addition, the default method ("weight") has a built in correction.

Author(s)

Mark Peterson

Examples

```
## Only run if topGO is available
if(require(topGO)){

## Load the sample data from topGO
data(GOdata)

## Recreate the GO annotation
## NB: you will likely do this with readMappings()
goAnno <- inverseList(genesInTerm(GOdata,usedGO(GOdata)))

testOut <- runGOAnalysis(sigGenes(GOdata),
                         allGenes(GOdata),
                         goAnno,
                         algorithm = "classic",
                         pValThresh = 0.05)

head(testOut)

}
```

Index