Package: ribiosExpression (via r-universe)

May 16, 2026

Type Package

Title Data Structures and Utilities for Gene Expression Analysis

Version 1.3.5

Description Provides data structures and utility functions for gene expression analysis. It includes the DesignContrast class for representing study designs and contrasts used in differential expression analysis, functions for importing and exporting expression data in GCT/CLS formats, tools for probeset summarization and filtering, and interfaces to limma-based differential gene expression workflows. The package works with Biobase ExpressionSet objects and integrates with the limma framework.

Depends R (>= 4.4.0)

Imports methods, stats, utils, Biobase, ribiosIO (>= 1.0-18), ribiosUtils (>= 1.0-5), ribiosPlot, limma, ribiosArg, Matrix, ribiosAnnotation (>= 1.0-8), ComplexHeatmap, circlize, openxlsx

Suggests testthat, grid, BiocStyle, knitr, rmarkdown

License GPL-3

Encoding UTF-8

Additional_repositories https://bedapub.r-universe.dev

URL https://github.com/bedapub/ribiosExpression

BugReports https://github.com/bedapub/ribiosExpression/issues

biocViews GeneExpression, DifferentialExpression, Microarray, DataImport, Visualization

VignetteBuilder knitr

RoxygenNote 7.3.3

Collate 'AllClasses.R' 'AllGenerics.R' 'AllMethods.R' 'DesignContrast.R' 'annotate.R' 'assertFullRank.R' 'filter.R' 'fixDesignMatrixColnames.R' 'import.R' 'io_gct_cls.R' 'io_gmt.R' 'io_tab.R' 'limmaDgeTable.R' 'mergeEset.R'

'obsolete.R' 'readAnnotationFile.R' 'readFKtable.R'
'removeAllZeroVar.R' 'ribiosExpression-package.R'
'ribiosExpressionSet.R' 'sniffFeatureType.R' 'splitPCA.R'
'summarizeProbesets.R' 'summarizeSamples.R' 'transformations.R'
'truncateDgeTable.R' 'writeVarMetadata.R'

Remotes github::bedapub/ribiosIO, github::bedapub/ribiosPlot, github::bedapub/ribiosArg, github::bedapub/ribiosAnnotation

Config/pak/sysreqs libicu-dev libpng-dev libssl-dev perl libsasl2-dev

Repository https://bedapub.r-universe.dev

Date/Publication 2026-02-18 15:22:02 UTC

RemoteUrl https://github.com/bedapub/ribiosExpression

RemoteRef HEAD

RemoteSha 596d283f5a09e5934b4ff783f0df0d86e36977ee

Contents

`annotate` *Annotate eSet or probesets*

Description

The function annotates an object of `eSet`, or a vector of characters representing probesets.

Usage

```
annotate(object, target, check.target, ...)
```

Arguments

object	An object of eSet, or a character vector of probesets
target	Chip type to be annotated
check.target	Logical, with FALSE as default. If set to TRUE, before looking up the annotations, it first check whether target is one of the valid chip types supported by GTI, and stops if it is not the case.
...	Currently not implemented

Details

Once successfully annotated, the annotation slot of the eSet object is set to the value of target.

Value

An eSet, or a data.frame containing annotation information of the probesets.

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

Examples

```
data(ribios.ExpressionSet)
myset <- ribios.ExpressionSet[100:105,]

## eSet
## Not run:
annotate(myset, "HG_U95AV2")
annotate(myset, "HG_U_95AV2", check.target=TRUE)

## End(Not run)

## characters
## Not run:
annotate(featureNames(myset), "HG_U95AV2")

## End(Not run)
```

---

assertFullRank *Assert whether a matrix is of full rank numerically*

Description

Assert whether a matrix is of full rank numerically

Usage

```
assertFullRank(matrix)
```

Arguments

matrix	Numeric matrix

Value

If not full rank, the function stops. Otherwise, an invisible TRUE is returned

Examples

```
myMat <- matrix(c(1,1,1,0,1,1), ncol=2, byrow=FALSE)
assertFullRank(myMat)
```

---

contrastAnnotation *Extract the contrast annotation data.frame from an object*

Description

Extract the contrast annotation data.frame from an object

Usage

```
contrastAnnotation(object)

## S4 method for signature 'DesignContrast'
contrastAnnotation(object)

## S4 replacement method for signature 'DesignContrast'
contrastAnnotation(object) <- value
```

Arguments

object	An object, see supported methods below
value	A contrast annotation data.frame

Value

A data.frame annotating contrasts

Methods (by class)

- contrastAnnotation(DesignContrast): Return the contrast annotation data.frame from a DesignContrast object
- contrastAnnotation(DesignContrast) <- value: Assign a contrast annotation data.frame to a contrastContrast object

contrastAnnotation<- *Assign contrast annotation to an object*

Description

Assign contrast annotation to an object

Usage

```
contrastAnnotation(object) <- value
```

Arguments

object	An object, see supported methods below
value	Contrast annotation data.frame

Value

The modified object

contrastMatrix *Extract the contrast matrix from an object*

Description

Extract the contrast matrix from an object

Usage

```
contrastMatrix(object)

## S4 method for signature 'DesignContrast'
contrastMatrix(object)

## S4 replacement method for signature 'DesignContrast'
contrastMatrix(object) <- value

## S4 method for signature 'MArrayLM'
contrastMatrix(object)
```

Arguments

object	An object, see supported methods below
value	A contrast matrix

Value

A numeric contrast matrix

Methods (by class)

- `contrastMatrix(DesignContrast)`: Return the contrast matrix from a DesignContrast object
- `contrastMatrix(DesignContrast) <- value`: Assign a contrast matrix to a `contrastContrast` object
- `contrastMatrix(MArrayLM)`: Extract contrast matrix from an object of MArrayLM

`contrastMatrix<-` *Assign contrast matrix to an object*

Description

Assign contrast matrix to an object

Usage

```
contrastMatrix(object) <- value
```

Arguments

`object`	An object, see supported methods below
`value`	Contrast matrix

Value

The modified object

`contrastNames` *Extract contrastNames from an object*

Description

Extract contrastNames from an object

Usage

```
contrastNames(object)

## S4 method for signature 'DesignContrast'
contrastNames(object)

## S4 method for signature 'MArrayLM'
contrastNames(object)
```

Arguments

object	An object, see supported methods below

Value

A character vector of contrast names

Methods (by class)

- contrastNames(DesignContrast): Return contrast names, i.e., column names of the contrast matrix
- contrastNames(MArrayLM): Extract contrast names from an object of MArrayLM

---

contrastSampleIndices *Return indices of samples involved in the given contrast of two or more coefficients*

Description

Return indices of samples involved in the given contrast of two or more coefficients

Usage

```
contrastSampleIndices(object, contrast)

## S4 method for signature 'DesignContrast,character'
contrastSampleIndices(object, contrast)

## S4 method for signature 'DesignContrast,numeric'
contrastSampleIndices(object, contrast)
```

Arguments

object	A DesignContrast object
contrast	Either a contrast name or a integer indicating the index of the contrast

Value

An integer vector, indices of samples that are involved, sorted by the ascending order of the coefficients of the contrast

Methods (by class)

- contrastSampleIndices(object = DesignContrast, contrast = character): Use character string to specify the contrast
- contrastSampleIndices(object = DesignContrast, contrast = numeric): Use integer indices to specify the contrast

Examples

```
## one-way ANOVA
myDesCon <- parseDesignContrast(sampleGroups="As,Be,As,Be,As,Be",
   groupLevels="Be,As", dispLevels="Beryllium,Arsenic", contrasts="As-Be")
contrastSampleIndices(myDesCon, 1L)
myInterDesCon <- DesignContrast(
    designMatrix=matrix(c(rep(1,6), rep(0,2), rep(1,2), rep(0,2),
           rep(0,4), rep(1,2)), nrow=6, byrow=FALSE),
    contrastMatrix=matrix(c(0,1,0, 0,0,1, 0,-1,1), byrow=FALSE, nrow=3),
    groups=factor(rep(c("As", "Be", "Cd"), each=2)),
    dispLevels=c("Arsenic", "Beryllium", "Cadmium"))
cont1Ind <- contrastSampleIndices(myInterDesCon, 1L)
cont2Ind <- contrastSampleIndices(myInterDesCon, 2L)
cont3Ind <- contrastSampleIndices(myInterDesCon, 3L)
stopifnot(identical(cont1Ind, 1:4))
stopifnot(identical(cont2Ind, c(1:2, 5:6)))
stopifnot(identical(cont3Ind, c(3:6)))
```

`dataFrameTwoVecs` *Build a data.frame from two vectors of potential different lengths*

Description

Build a data.frame from two vectors of potential different lengths

Usage

```
dataFrameTwoVecs(vec1, vec2, col.names = c("Vec1", "Vec2"))
```

Arguments

`vec1`	A vector
`vec2`	Another vector
`col.names`	A character vector of length 2 giving column names of the output `data.frame`. The shorter vector of the two are extended to the same length by appending empty strings.

Value

A `data.frame` of two columns. The row count matches the longer vector

Examples

```
dataFrameTwoVecs(LETTERS[1:5], letters[2:9])
```

design2group *Infer groups from a design matrix*

Description

Infer groups from a design matrix

Usage

```
design2group(designMatrix)
```

Arguments

designMatrix	A design matrix

Value

A factor vector giving the groups inferred from the design matrix

A naive logic is used: samples of the same design vectors are of the same group.

The inference is known to fail when control variables, such as age or RIN numbers, vary between samples of the same group.

Examples

```
myDesign <- model.matrix(~gl(3,3))
design2group(myDesign)
```

DesignContrast *Contrast a DesignContrast object*

Description

Contrast a DesignContrast object

Usage

```
DesignContrast(
  designMatrix,
  contrastMatrix = NULL,
  groups = NULL,
  displevels = NULL,
  contrastAnnotation = NULL
)
```

Arguments

designMatrix	A design matrix
contrastMatrix	A contrast matrix. If null, no comparison can be done.
groups	A factor vector of the same length as the number of columns of the design matrix. If missing, design2group is used to infer groups.
dispLevels	A character vector of the same length as the number of levels encoded by group, indicating how different groups should be labelled. If missing, levels of group are used.
contrastAnnotation	A data.frame or NULL, annotating contrasts

Value

A DesignContrast object

Examples

```
myFac <- gl(3,3, labels=c("baseline", "treat1", "treat2"))
myDesign <- model.matrix(~myFac)
colnames(myDesign) <- c("baseline", "treat1", "treat2")
myContrast <- limma::makeContrasts(contrasts=c("treat1", "treat2"), levels=myDesign)
DesignContrast(myDesign, myContrast, groups=myFac)
DesignContrast(myDesign, myContrast, groups=myFac, dispLevels=c("C", "T1", "T2"))
```

DesignContrast-class *Study design and contrast information*

Description

The DesignContrast class represents key information in a designed experiment

Usage

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

Arguments

object	An object of DesignContrast.

Functions

- show(DesignContrast): The show method

Slots

`design` A numeric matrix. The number of rows equals the sample size. The columns corresponds to the variables of design

`contrasts` A numeric matrix. The number of rows equals the number of columns in the design matrix. The columns corresponds to the comparisons one wishes to make.

`groups` A factor vector, giving sample groups. The length equals the number of samples.

`dispLevels` A character vector, used for displaying sample groups. The length equals the number of levels of the `groups` factor.

`contrastAnnotation` A data.frame, used to annotate the contrasts.

Objects from the Class

Objects can be created by calls of the function `DesignContrast`. However, the users should not directly call this function. Instead, `parseDesignContrast` should be called.

`designMatrix` *Extract the design matrix from an object*

Description

Extract the design matrix from an object

Usage

```
designMatrix(object)

## S4 method for signature 'DesignContrast'
designMatrix(object)

## S4 replacement method for signature 'DesignContrast'
designMatrix(object) <- value

## S4 method for signature 'MArrayLM'
designMatrix(object)
```

Arguments

`object`	An object, see supported methods below
`value`	A design matrix

Value

A numeric design matrix

Methods (by class)

- `designMatrix(DesignContrast)`: Return the design matrix from a DesignContrast object
- `designMatrix(DesignContrast) <- value`: Assign a design matrix to a `DesignContrast` object
- `designMatrix(MArrayLM)`: Extract design matrix from an object of MArrayLM

`designMatrix<-` *Assign design matrix to an object*

Description

Assign design matrix to an object

Usage

```
designMatrix(object) <- value
```

Arguments

`object`	An object, see supported methods below
`value`	Design matrix

Value

The modified object

`designVariables` *Extract design variable names from an object*

Description

Extract design variable names from an object

Usage

```
designVariables(object)

## S4 method for signature 'DesignContrast'
designVariables(object)
```

Arguments

`object`	An object, see supported methods below

Value

A character vector of variable names

Methods (by class)

- `designVariables(DesignContrast)`: Return the names of variables (column names) in the design matrix of a DesignContrast object

---

`dispGroups` *Extract displayed group labels from an object*

Description

Extract displayed group labels from an object

Usage

```
dispGroups(object)

## S4 method for signature 'DesignContrast'
dispGroups(object)
```

Arguments

`object`	An object, see supported methods below

Value

A factor of display group labels

Methods (by class)

- `dispGroups(DesignContrast)`: Return the sample groups from a DesignContrast object , using display labels

`eSetToLongTable` *Transform eSet to long data.frame*

Description

Transform eSet to long data.frame

Usage

```
eSetToLongTable(
  x,
  exprsFun = function(eset) Biobase::exprs(eset),
  includeOtherAssayData = FALSE
)
```

Arguments

`x`	An `eSet` object
`exprsFun`	A function to extract expression values, by default `exprs`
`includeOtherAssayData`	Logical, whether other elements in the `assayData` environment (if present) should be returned.

Details

The function extracts exprs (and other values in the `assayData` environment), and return it in a long data.frame format with phenotypic data

Value

A `data.frame` in long format.

Examples

```
data(ribios.ExpressionSet, package="ribiosExpression")
exprsLongTbl <- eSetToLongTable(ribios.ExpressionSet)
seLongTbl <- eSetToLongTable(ribios.ExpressionSet,
   exprsFun=function(eset) Biobase::assayData(eset)$se.exprs)
```

exprsToLong *Transform an expression matrix to long table*

Description

Transform an expression matrix to long table

Usage

```
exprsToLong(x, ...)

## S4 method for signature 'matrix'
exprsToLong(
  x,
  idvar = "illID",
  timevar = "hybridID",
  valuevar = "value",
  ids = rownames(x),
  valueType = "raw"
)

## S4 method for signature 'eSet'
exprsToLong(x)
```

Arguments

x	A matrix or an ExpressionSet object
...	Other parameters
idvar	Variable name of the feature identifier, passed to reshape
timevar	The time variable, passed to reshape
valuevar	The value variable
ids	Feature identifiers
valueType	Character string, value type

Value

A data.frame

Methods (by class)

- exprsToLong(matrix): The method for matrix as input
- exprsToLong(eSet): The method for eSet as input

fixDesignMatrixColnames *Fix design matrix colnames so that they are legal variable names*

Description

Fix design matrix colnames so that they are legal variable names

Usage

```
fixDesignMatrixColnames(
  designMatrix,
  interceptChar = "_",
  removeContrastNames = TRUE
)
```

Arguments

designMatrix	A design matrix, produced by `model.matrix`
interceptChar	Character string, the value the interaction symbol (:) should be replaced with
removeContrastNames	Logical, whether the contrast variable name should be removed.

Value

The matrix with fixed colum names.

Examples

```
myFac1 <- gl(6,2, labels=sprintf("Fac1_%d", 1:6))
myFac2 <- gl(2,6, labels=c("Ctrl", "Dis"))
myVar <- rnorm(12)
myDesign <- model.matrix(~myFac1 * myFac2 + myVar)
head(myDesign)
head(fixDesignMatrixColnames(myDesign))
```

fixEmptyColumnName *Detect if any column has an empty string as name and fix*

Description

Detect if any column has an empty string as name and fix

Usage

```
fixEmptyColumnName(df, prefix = "X")
```

Arguments

df	A data.frame
prefix	A character string, the prefix to be used if an column's name is empty.

Details

If any column has an empty string as name, its replaced by the prefix appended by an index starting from 1

Value

A data.frame with fixed column names.

Examples

```
testDf <- data.frame("Col1"=LETTERS[1:3], "Col2"=letters[2:4])
colnames(testDf) <- c("", "")
testDf
fixEmptyColumnName(testDf)
fixEmptyColumnName(testDf, prefix="fData")
```

---

formatGmt *Make strings in the GMT format*

Description

Resulting string(s) can be exported into GMT file by writeLines

Usage

```
formatGmt(title, comment, genes)

## S4 method for signature 'character,character,character'
formatGmt(title, comment, genes)

## S4 method for signature 'character,missing,character'
formatGmt(title, genes)

## S4 method for signature 'character,character,list'
formatGmt(title, comment, genes)

## S4 method for signature 'character,missing,list'
formatGmt(title, genes)
```

Arguments

`title`	Character, title(s) of gene set(s)
`comment`	Character, comment(s) of gene set(s). Can be of the same length as the `title`, or be of length one: in the latter case, it will be replicated in gene set. This option can also be left out: the comment field of the GMT file will be left blank.
`genes`	A character vector of gene names, or a list of such vectors. In the former case, one GMT line is produced; otherwise multiple lines are returned. In the latter case, the length of the list must match the length of `title`.

Value

One or more lines of GMT file

Methods (by class)

- `formatGmt(title = character, comment = character, genes = character)`: title, comment, and genes are one character string
- `formatGmt(title = character, comment = missing, genes = character)`: title and genes are both one character string, comments are missing
- `formatGmt(title = character, comment = character, genes = list)`: title and comments are both vectors of character strings, genes are a list of the same length
- `formatGmt(title = character, comment = missing, genes = list)`: title is vectors of character strings, comments are missing, genes are a list of the same length as the title

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

Examples

```
formatGmt(title="GeneSet0", comment="My geneset", genes=c("MAPT", "MAPK", "AKT1"))
formatGmt(title="GeneSet0", genes=c("MAPT", "MAPK", "AKT1"))

formatGmt(title=c("GeneSet0", "GeneSet1"),
          comment=c("My geneset 0", "My geneset 1"),
          genes=list(c("MAPT", "MAPK", "AKT1"), c("EGFR", "CDC42")))
formatGmt(title=c("GeneSet0", "GeneSet1"),
          comment="My genesets",
          genes=list(c("MAPT", "MAPK", "AKT1"), c("EGFR", "CDC42")))
formatGmt(title=c("GeneSet0", "GeneSet1"),
          genes=list(c("MAPT", "MAPK", "AKT1"), c("EGFR", "CDC42")))
```

groups *Extract sample groups from an object*

Description

Extract sample groups from an object

Usage

```
groups(object)

## S4 method for signature 'DesignContrast'
groups(object)
```

Arguments

object	An object, see supported methods below

Value

A factor of sample groups

Methods (by class)

- groups(DesignContrast): Return the raw sample groups from a DesignContrast object

grp2gmt *Convert GRP files into GMT-formatted strings*

Description

GRP files are used by Connectivity Map on-line tool, which stores the information of a rank-ordered list of probesets. They are simply one-column text files, each line containing one probeset. grpFiles2gmt convert GRP files into GMT-formatted strings, which can be written in GMT files to be used by GSEA and other tools.

Usage

```
grp2gmt(txt, chiptype, name)

grpFiles2gmt(..., chiptype, n = -1L)
```

Arguments

txt	A vector of character strings, each containing one probeset
chiptype	Chip type, normally character representing the microarray chip type. If the option is missing, or is of value NA or NULL, no annotation is done.
name	Character, name of the gene set (the first field of the GMT file)
...	GRP file names
n	Integer, number of lines to be read; -1 indicates all lines should be read

Details

The function grp2gmt, called by grpFiles2gmt internally, annotates probesets when chiptype is supported by GTI, and transform them into the GMT format.

If chiptype is provided, the annotate function is called to fetch probeset annotations from the databank.

Value

A vector of character strings, each containing one line of a GMT file. They can be written to a file with the writeLines function.

Note

It is user's responsibility to check that all GRP files do exist and are readable.

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

References

See https://www.broadinstitute.org/connectivity-map-cmap for the use of GRP files in the Connectivity Map web tool.

Examples

```
up.file <- system.file("extdata/tags_up.grp", package="ribiosExpression")
down.file <- system.file("extdata/tags_down.grp", package="ribiosExpression")
grp2gmt(readLines(up.file, n=-1))
grpFiles2gmt(c(up.file, down.file), n=3)

## Not run:
grp2gmt(readLines(up.file, n=-1), chiptype="HG_U95AV2")
grpFiles2gmt(c(up.file, down.file), n=-1L, chiptype="HG_U95AV2")

## End(Not run)
```

`isInputDesignConsistent` *Test whether the input design matrix is consistent with the sample names*

Description

Test whether the input design matrix is consistent with the sample names

Usage

```
isInputDesignConsistent(descon, sampleNames)
```

Arguments

`descon`	A DesignContrast object
`sampleNames`	A vector of string characters, specifying sample names If the sample names in DesignContrast are identical with the given sample names, an invisible `TRUE` is returned. If the two sets are identical, however the order of sample names do not match, a warning message is raised, and an invisible `FALSE` is returned If the two sets have differences, the mismatching sample names are printed for diagnosis.

Value

A invisible logical value. `TRUE` if and only if the sample names match perfectly.

`keepMaxStatProbe` *Filter multiple probesets matching to the same gene by keeping the one with the maximum statistic (by default the variance).*

Description

The function filters features (commonly probesets) in an `ExpressionSet` object. It does not affect genes with only one feature present, or genes without an valid annotation (see details below). For genes with multiple probesets, the function calculates the statistic of each probeset across all samples and filter probesets by only keeping the one with the maximum of variance. Thereby an `ExpressionSet` returned by the function has only one probeset matching each gene.

Usage

```
keepMaxStatProbe(
  eset,
  probe.index.name,
  keepNAprobes = TRUE,
  stat = function(x) mean(x, na.rm = TRUE),
  ...
)
```

Arguments

eset	An ExpressionSet
probe.index.name	The column name of the fData(eset) data matrix, used as the index of gene to determine which features are matched to the same gene.
keepNAprobes	Logical, determines whether genes without an valid index name should kept or left out. See details below.
stat	Function or character, a function (or the name referring to it) which takes a vector of numerical values, and returns one value as the statistic, e.g. sd for standard deviations.
...	Parameters passed to the stat function. One of the most frequent used option might be na.rm=TRUE, see details and examples.

Details

Names of probesets are determined by the featureNames(eset) function.

The column of probe.index.name in the fData(eset) data.frame determines the index of genes, for example the Entrez GeneID, to which probesets are matched. Those genes without a valid index, whose index is either an empty string or NA, can be set to be left out by keepNAprobes=FALSE. If the option is set as TRUE, then these genes are kept in the returning object.

The stat function should only return one statistic, most favorably not NA, by taking a vector of numerical values. Most statistics can be calculated in a robust way by setting na.rm=TRUE. This option should be always used whenver possible. Otherwise when there is one or more missing value of a probeset, its statistic will probably be NA and this will lead to discard the probeset. Even worse, when all probesets matching to a gene have NAs, the gene will be totally filtered out, which is usually not desired. Therefore, set na.rm=TRUE through the ... option (see examples below) whenever possible.

Value

An filtered ExpressionSet.

Note

Note that when the statistics of two or more probesets tie (having the same value), the probeset chosed could be random (the probeset with its name ranked first when multiple names are converted into a factor vector).

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

Examples

```
library("Biobase")

example.mat <- matrix(c(1,1,3,4, 2,2,3,3, 4,5,6,7, 7,8,9,10), ncol=4, byrow=TRUE)
example.eset <- new("ExpressionSet", exprs=example.mat)

featureNames(example.eset) <- c("1a","1b","2","3")
fData(example.eset)$geneid <- c(1,1,2,3)

## keep probesets with the maximal variance
example.sd <- keepMaxStatProbe(example.eset, probe.index.name="geneid", stat=sd)
featureNames(example.sd)

## keep probesets with the maximal Median Absolute Deviation (MAD)
example.mad <- keepMaxStatProbe(example.eset, probe.index.name="geneid", stat=mad)
featureNames(example.mad)

## keep probesets with the maximal mean value
example.mean <- keepMaxStatProbe(example.eset,
probe.index.name="geneid", stat=mean)
featureNames(example.mean)

## note that NA value may cause problems, it is a good practice to make
## the stat function _resist_ to NA
na.eset <- example.eset
exprs(na.eset)[1,1] <- NA

## Not run:
## prone to error
na.mean <- keepMaxStatProbe(na.eset,
probe.index.name="geneid",stat=mean)
featureNames(na.mean)
## better
na.mean.narm <- keepMaxStatProbe(na.eset,
probe.index.name="geneid",na.rm=TRUE)
featureNames(na.mean.narm)

## End(Not run)
```

---

limmaDgeTable *Return dgeTable from a marrayLM object*

Description

Return dgeTable from a marrayLM object

Usage

```
limmaDgeTable(marrayLM, contrast = NULL, confint = TRUE)
```

Arguments

marrayLM	An object returned by 'lmFit' and 'eBayes'
contrast	NULL, or a character string indicating the contrast of interest
confint	Logical, whether confidence intervals should be returned

Value

A data.frame

---

limmaTopTable2dgeTable

Transform limma::topTable results to a DGEtable

Description

Transform limma::topTable results to a DGEtable

Usage

```
limmaTopTable2dgeTable(limmaTopTable)
```

Arguments

limmaTopTable	topTable returned by limma::topTable

Value

A data.frame known as DGEtable which has controlled column names

Examples

```
example.sd <- 0.3*sqrt(4/rchisq(100,df=4))
example.y <- matrix(rnorm(100*6,sd=example.sd),100,6)
example.y[1:2,4:6] <- example.y[1:2,4:6] + 2
rownames(example.y) <- paste("Gene",1:100)
example.design <- cbind(Grp1=1,Grp2vs1=c(0,0,0,1,1,1))
example.fit <- limma::lmFit(example.y,example.design)
example.fit <- limma::eBayes(example.fit)
example.tt <- limma::topTable(example.fit, coef=2)
example.dt <- limmaTopTable2dgeTable(example.tt)
head(example.dt)
```

matrixToLongTable *Transform a matrix to long table*

Description

Transform a matrix to long table

Usage

```
matrixToLongTable(x, valueLabel = "value", rowLabel = "row", colLabel = "col")
```

Arguments

x	A matrix
valueLabel	Character string, the label of the value
rowLabel	Character string, the name of the column holding the row names
colLabel	Character string, the name of the column holding the column names

Value

A data.frame

Examples

```
myMatrix <- matrix(rnorm(24), nrow=4, dimnames=list(LETTERS[1:4], letters[1:6]))
matrixToLongTable(myMatrix)
```

mergeEset *Merge two eSets by column binding*

Description

Merge two eSets by column binding

Usage

```
mergeEset(eset1, eset2, by.x, by.y, normalization = "quantile")
```

Arguments

eset1	An eSet object
eset2	Another eSet object
by.x	Column index of feature annotation of eset1
by.y	COlumn index of feature annotation of eset2
normalization	NULL or character string, which will be passed to normalizeBetweenArrays.

Value

A new eSet object

---

nContrast *Extract the number of contrasts from an object*

Description

Extract the number of contrasts from an object

Usage

```
nContrast(object)

## S4 method for signature 'DesignContrast'
nContrast(object)
```

Arguments

object	An object, see supported methods below

Value

An integer, number of contrasts

Methods (by class)

- nContrast(DesignContrast): Return the number of contrasts in a DesignContrast object

---

parseContrastStr *Parse contrast from strings*

Description

Parse contrast from strings

Usage

```
parseContrastStr(contrastStr)
```

Arguments

contrastStr	A vector of character strings

Value

A contrast matrix

`parseDesignContrast` *Parse study design and asked questions encoded in design and contrast matrices or in one-way ANOVA designs*

Description

Parse study design and asked questions encoded in design and contrast matrices or in one-way ANOVA designs

Usage

```
parseDesignContrast(
  designFile = NULL,
  contrastFile = NULL,
  sampleGroups = NULL,
  groupLevels = NULL,
  dispLevels = NULL,
  contrasts = NULL,
  expSampleNames = NULL
)
```

Arguments

`designFile`	A plain tab-delimited file with headers encoding the design matrix, or NULL
`contrastFile`	A plain tab-delimited file with headers encoding the contrast matrix, or NULL
`sampleGroups`	A character string concatenated by commas (e.g. A,B,C), or a plain text file containing one string per line (e.g. A*newline*B*newline*C), encoding sample group memberships.
`groupLevels`	Similar format as 'sampleGroups', encoding levels (e.g. order) of the sampleGroups
`dispLevels`	Similar format as 'sampleGroups', encoding the display of the groupLevels. Must match 'groupLevels'
`contrasts`	Similar format as 'sampleGroups', encoding contrasts in case of one-way ANOVA designs
`expSampleNames`	A vector of character strings giving the expected sample names (e.g. those in the input matrix)

Value

A S4-object 'DesignContrast'

Examples

```
## one-way ANOVA
parseDesignContrast(sampleGroups="As,Be,As,Be,As,Be",groupLevels="Be,As",
    dispLevels="Beryllium,Arsenic", contrasts="As-Be")
## design/contrast matrix
designFile <- system.file("extdata/example-designMatrix.txt",
    package="ribiosExpression")
contrastFile <- system.file("extdata/example-contrastMatrix.txt",
    package="ribiosExpression")
# minimal information
parseDesignContrast(designFile=designFile, contrastFile=contrastFile)
# with extra information about sample groups
parseDesignContrast(designFile=designFile, contrastFile=contrastFile,
    sampleGroups="As,Be,As,Be,As,Be",
    groupLevels="Be,As", dispLevels="Beryllium,Arsenic")
```

`parseDesignContrastFile` *Parse design and contrast from files*

Description

Parse design and contrast from files

Usage

```
parseDesignContrastFile(
  designFile,
  contrastFile,
  groupsStr = NULL,
  levelStr = NULL,
  dispLevelStr
)
```

Arguments

`designFile`	A tab-delimited file encoding the design matrix
`contrastFile`	A tab-delimited file encoding the contrast matrix
`groupsStr`	A vector of character strings, giving sample groups
`levelStr`	A vector of level strings
`dispLevelStr`	A vector of strings to be used as display labels, if exist

Value

A DesignContrast object

parseDesignContrastStr *Parse design and contrast from strings*

Description

Parse design and contrast from strings

Usage

```
parseDesignContrastStr(groupsStr, levelStr, dispLevelStr, contrastStr)
```

Arguments

groupsStr	A factor vector indicating sample groups
levelStr	Level strings
dispLevelStr	Display level strings
contrastStr	A vector of character strings indicating contrasts

Value

A DesignContrast object

plot.DesignContrast *Plot a DesignContrast object with two heatmaps*

Description

The function plots a ComplexHeatmap containing two matrices, one of the design matrix and the other of the contrast matrix.

Usage

```
## S3 method for class 'DesignContrast'
plot(
  x,
  y = NULL,
  title = NULL,
  clusterDesign = FALSE,
  clusterSamples = FALSE,
  clusterContrasts = FALSE,
  designRange = NULL,
  contrastRange = NULL,
  designParams = NULL,
  contrastParams = NULL,
  ...
)
```

Arguments

x	A DesignContrast object
y	NULL, ignored
title	Title of the object, used in the title of the heatmaps
clusterDesign	Logical, cluster rows of the design matrix
clusterSamples	Logical, cluster columns of the design matrix
clusterContrasts	Logical, cluster rows of the contrast matrix (notice that the contrast matrix required by limma is the transposed contrast matrix)
designRange	NULL or a vector of length 2, giving range of the design matrix to be visualized. If NULL, the whole range is used.
contrastRange	NULL or a vector of length 2, giving range of the contrast matrix to be visualized. If NULL, a symmetric range of the largest absolute value and its negate is used.
designParams	Other parameters passed to Heatmap for the design matrix
contrastParams	Other parameters passed to Heatmap for the contrast matrix
...	Ignored

Value

Heatmap object

Examples

```
myFac <- gl(3,3, labels=c("baseline", "treat1", "treat2"))
myDesign <- model.matrix(~myFac)
colnames(myDesign) <- c("baseline", "treat1", "treat2")
myContrast <- limma::makeContrasts(contrasts=c("treat1", "treat2"), levels=myDesign)
res1 <- DesignContrast(myDesign, myContrast, groups=myFac)
res2 <- DesignContrast(myDesign, myContrast, groups=myFac, dispLevels=c("C", "T1", "T2"))
plot(res1, title="DesCon 1")
plot(res2, title="DesCon 1 (identical)")
plot(res2, title="DesCon 1 (identical)", designRange=c(-2,2),
    contrastRange=c(-2,1),
    designParams=list(row_names_gp=grid::gpar(fontsize=8)),
    contrastParams=list(column_names_gp=grid::gpar(fontsize=12, color="red")))
```

---

readAnnotationFile *Read in an annotation file in the tsv-format, with or without row names*

Description

Read in an annotation file in the tsv-format, with or without row names

Usage

```
readAnnotationFile(file, outputKeyName = "FeatureName", ...)
```

Arguments

file	A tab-delimited file without quotes, the first column must contain identifiers (key names). In case the first column has no column name and it contains row names, they will be used as feature names.
outputKeyName	The key name used in the output data.frame. It does not need to exist in the column names of the input data.frame. In this situation, the first column will be used as output keys. If it does exist in the column names of the input data.frame, the content of that column must be identical with the first column of the annotation file.
...	Other parameters passed to readMatrix, which are further passed to read.table.

Value

A data.frame containing the annotation, with the first column named as outputKeyName that contains feature identifiers as character strings. In case the input table contains the column with the same name, the content in that column must match the row names, otherwise an error is reported.

This function is called by readFeatureAnnotationFile and readSampleAnnotationFile. Normal users are unlikely to use it.

Examples

```
f1 <- system.file("extdata",
  "featureAnnotation/featureAnnotationFile-withRowNames.txt",
  package="ribiosExpression")
f2 <- system.file("extdata",
  "featureAnnotation/featureAnnotationFile-withoutRowNames.txt",
  package="ribiosExpression")
# 'FeatureName' does not exist in the column names of f1
f1Read <- readAnnotationFile(f1, outputKeyName="FeatureName")
# 'GeneID' exists in the colum names of f2, and it is the first column.
f2Read <- readAnnotationFile(f2, outputKeyName="GeneID")
head(f1Read)
head(f2Read)
```

---

readEset *Read eSet object from plain files*

Description

Read eSet object from plain files

Usage

```
readEset(
  exprs.file,
  fData.file,
  pData.file,
  exprs.file.format = c("gct", "tsv"),
  sep = "\t",
  header = TRUE,
  ...
)
```

Arguments

exprs.file	Character, file name where exprs data is written to
fData.file	Character, optional, file name where fData data is written to
pData.file	Character, optional, file name where pData data is written to
exprs.file.format	Character, write exprs data in either gct or tsv format
sep	Character, separator
header	Logical, whether a head line is present
...	Passed to readFKtable

Details

The function can read in eSet object saved by writeEset by parsing three plain text files: exprs.file, fData.file, and pData.file.

Currently both tsv and gct formats are supported for expression file.

See writeEset for limitations of these functions.

Value

An ExpressionSet object.

See Also

writeEset, readFKtable.

Examples

```
data(sample.ExpressionSet, package="Biobase")
fData(sample.ExpressionSet) <- data.frame(
 ProbeID=featureNames(sample.ExpressionSet),
 row.names=featureNames(sample.ExpressionSet))
exprs.file <- tempfile()
fData.file <- tempfile()
pData.file <- tempfile()
writeEset(sample.ExpressionSet, exprs.file, fData.file, pData.file,
```

```
exprs.file.format="gct")
testRead1 <- readEset(exprs.file, fData.file, pData.file, exprs.file.format="gct")
writeEset(sample.ExpressionSet, exprs.file, fData.file, pData.file,
exprs.file.format="tsv")
testRead2 <- readEset(exprs.file, fData.file, pData.file, exprs.file.format="tsv")
```

`readExprsMatrix` *Read an expression matrix into an ExpressionSet object*

Description

The function reads in an expression matrix into an ExpressionSet object. The expression matrix should be saved in the file format supported by the `read_exprs_matrix` function: currently supported formats include tab-delimited file and gct files.

Usage

```
readExprsMatrix(x)
```

Arguments

x	A file containing an expression matrix

Details

The function is a wrapper of the `read_exprs_matrix` function in the `ribiosIO` package. The difference is it returns a valid `ExpressionSet` object instead of a primitive matrix.

Value

An `ExpressionSet` object holding the expression matrix. Both pData and fData are empty except for the feature/sample names recorded in the expression matrix.

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

See Also

`read_exprs_matrix` in the `ribiosIO` package.

Examples

```
idir <- system.file("extdata", package="ribiosExpression")
myeset <- readExprsMatrix(file.path(idir, "sample_eset_exprs.txt"))
myeset2 <- readExprsMatrix(file.path(idir, "test.gct"))
```

readFeatureAnnotationFile *Read in feature annotation file in the tsv-format, with or without row names*

Description

Read in feature annotation file in the tsv-format, with or without row names

Usage

```
readFeatureAnnotationFile(file, ...)
```

Arguments

file	A tab-delimited file without quotes, the first column must contain feauture identifiers. In case the first column has no column name and it contains row names, they will be used as feature names.
...	Other parameters passed to readMatrix, which are further passed to read.table.

Value

A data.frame containing feature annotation, with the first column named as 'FeatureName' that contains feature identifiers as character strings. In case the input table contains the column 'FeatureName', the content in that column must match the row names, otherwise an error is reported.

Examples

```
f1 <- system.file("extdata",
  "featureAnnotation/featureAnnotationFile-withRowNames.txt",
  package="ribiosExpression")
f2 <- system.file("extdata",
  "featureAnnotation/featureAnnotationFile-withoutRowNames.txt",
  package="ribiosExpression")
f1Read <- readFeatureAnnotationFile(f1)
f2Read <- readFeatureAnnotationFile(f2)
head(f1Read)
head(f2Read)
```

`readFKtable` *Read table with Foreign Keys*

Description

The concept `Foreign Keys` comes from relational databse systems. These keys can be used to cross-reference tables. Say we have two `data.frames`, one contains gene annotations and the other contains protein annotations. A column named `mRNArefseqID` may be the foreign key that can be used to specify relationships between gene and proteins.

Usage

```
readFKtable(file, fk, strict.order = FALSE, ...)
```

Arguments

`file`	A table file.
`fk`	Characters, foreign keys.
`strict.order`	Logical, whether the foreign keys must have the same order as they appear in the file.
`...`	Other parameters passed to the `read.table` function.

Details

The `readFKtable` reads a table from file, and checks if it contains provided foreign keys: either as row.names or in the first column.

Value

A `data.frame` if the FK-matching was successful, otherwise the function will print an error message and stop.

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

Examples

```
test.file <- tempfile()
fk.teams <- c("HSV", "FCB", "BVB")
## FK in row names
test.mat <- matrix(rnorm(9), nrow=3, dimnames=list(fk.teams, NULL))
write.table(test.mat, test.file)
readFKtable(test.file, fk=fk.teams)

## or: FK can be in the first column
test.df <- data.frame(team=fk.teams, pts=c(15,14,15),plc=c("H", "G", "H"))
write.table(test.df, test.file)
```

```
readFKtable(test.file, fk=fk.teams)

## try strict.order=TRUE
test.df <- data.frame(pts=c(15,14,13), plc=c("H", "G", "H"), row.names=rev(fk.teams))
write.table(test.df, test.file)
readFKtable(test.file, fk=fk.teams, strict.order=FALSE)
## Not run: readFKtable(test.file, fk=fk.teams, strict.order=TRUE)
```

`readGct` *Import ExpressionSet into gct/cls files the C version, about 5x faster than the R implementation for the ALL dataset*

Description

As complementary functions to `writeGctCls`, `readGctCls` reads a pair of gct and cls files (with same base names) into an `ExpressionSet` object.

Usage

```
readGct(gct.file)

readGctCls(file.base, gct.file, cls.file, add.fData.file, add.pData.file)
```

Arguments

`gct.file`	The name of the gct file (only valid when file.base is missing).
`file.base`	The full file name of gct/cls files without suffixe, if not in the current diretory, must contain the path (dirname) as well . For instance if it is set as `~/my/dir/input`, then the function seeks the file `~/my/dir/input.gct` and `~/my/dir/input.cls`.
`cls.file`	The name of the cls file (only valid when file.base is missing).
`add.fData.file`	Optional, file of additional feature data, see details.
`add.pData.file`	Optional, file of additional phenotype (sample) data, see details.

Details

The `readGctCls` function calls internally the `readGct` and `read_cls` functions to read in two formats respeectively. `readGct` returns a barely annotated `ExpressionSet` object, and `read_cls` returns a vector of levels encoding sample groups.

Since gct/cls contains only one property of features and samples each (Description in the gct file as well as sample groups/levels in the cls file), `readGctCls` allows users to provide additional fData/pData files. They should be tab-delimited files, with first column machting exactly the names of features or samples. They must be within the path specified by the `path` option, namely in the same directory of gls/cls files.sample

See example below.

Value

A ExpressionSet object

An ExpressionSet object. The Description column in the gct file is encoded in the desc column in the featureData of the resulting object. The sample groups in the cls file is encoded in the cls column in the phenoData.

Functions

- readGct(): readGct uses the C implementation of reading in a gct file

Note

The readGct function is a wrapper of the read_gct_matrix function in the ribiosIO package, which makes up the GCT matrix into an ExpressionSet object.

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

See Also

writeGctCls. See read_gct_matrix for underlying C code to import GCT files.

Examples

```
idir <- system.file("extdata", package="ribiosExpression")

sample.eset <- readGctCls(file.base=file.path(idir, "test"))

ext.eset <- readGctCls(file.base=file.path(idir, "test"),
add.fData.file=file.path(idir, "test.add.fData.txt"),
add.pData.file=file.path(idir, "test.add.pData.txt"))

stopifnot(identical(exprs(sample.eset), exprs(ext.eset)))

## try to compare pData(sample.eset) with pData(ext.eset), and similarly
## fData(sample.eset) with fData(ext.eset)
```

---

readSampleAnnotationFile

Read in sample annotation file in the tsv-format, with or without row names

---

Description

Read in sample annotation file in the tsv-format, with or without row names

Usage

```
readSampleAnnotationFile(file, ...)
```

Arguments

file	A tab-delimited file without quotes, the first column must contain sample identifiers. In case the first column has no column name and it contains row names, they will be used as sample names.
...	Other parameters passed to readMatrix, which are further passed to read.table.

Value

A data.frame containing sample annotation, with the first column named as 'ExperimentName' that contains sample identifiers as character strings. In case the input table contains the column 'ExperimentName', the content in that column must match the row names, otherwise an error is reported.

Examples

```
f1 <- system.file("extdata",
  "sampleAnnotation/sampleAnnotationFile-withRowNames.txt",
  package="ribiosExpression")
f2 <- system.file("extdata",
  "sampleAnnotation/sampleAnnotationFile-withoutRowNames.txt",
  package="ribiosExpression")
f1Read <- readSampleAnnotationFile(f1)
f2Read <- readSampleAnnotationFile(f2)
head(f1Read)
head(f2Read)
```

---

reannotate *Transform an eSet object of Bioc-annotation into of GTI-annotation*

Description

The function is used to transform an eSet object, which is annotated by Bioconductor annotation packages, into an object with annotation information from GTI.

Usage

```
reannotate(object, check.target, ...)
```

Arguments

object	An eSet object, with the annotation slot set as one of the valid annotations recognized by Bioconductor, for instance hgu95av2.
check.target	Logical, with FALSE as default. When set to TRUE, beforing fetching database for annotations, the function first checks whether the chip type is supported by GTI. If it is not the case, the function will print error message and stop.
...	Currently not implemented

Details

The translation between Bioconductor annotation package names and GTI chip types is performed by the bioc2gti function in the ribiosAnnotation package.

Once the re-annotation succeeds, the annotation slot of the eSet object will be overwritten by the corresponding chip type name in GTI.

Value

An eSet object with feature annotations updated by GTI, and the annotation slot is changed to the chip type in GTI.

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

See Also

annotate to annotate an eSet object without prior information of bioc-annotation, or if that information is not saved in the annotation slot.

Examples

```
data(ribios.ExpressionSet)
print(ribios.ExpressionSet)

## Not run:
gti.eSet <- reannotate(ribios.ExpressionSet)
gti.eSet <- reannotate(ribios.ExpressionSet, check.target=FALSE)
print(gti.eSet)

## End(Not run)
```

`removeAllZeroVar` *Remove all-zero variables from design matrix and the corresponding contrast matrix*

Description

Remove all-zero variables from design matrix and the corresponding contrast matrix

Usage

```
removeAllZeroVar(obj, contrasts)

## S3 method for class 'matrix'
removeAllZeroVar(obj, contrasts)

## S3 method for class 'DesignContrast'
removeAllZeroVar(obj, contrasts = NULL)
```

Arguments

`obj`	Either a design matrix, rows are samples, columns are independent variables, and values are coefficients. Or a `DesignContrast` object that contains both design and contrast matrices. The design matrix may contain columns of all zeros, the independent variable corresponding to which can not be estimated.
`contrasts`	Either `NULL`, in case `designMatrix` is a `DesignContrast` object, or a contrast matrix matching the design matrix, i.e. the number of rows must equal the number of column of the design matrix. Rows are independent variables, columns are names of the contrasts to be estimated, the values are operations applied to the coefficients.

Value

Either a list of two matrices (`design` and `contrasts`), or a `DesignContrast` object, depending on the input parameter type. The design matrix and contrast matrix have an attribute each, `notEstCoefs` and `notEstContrasts`, that keep track of filtered variables and contrasts.

Methods (by class)

- `removeAllZeroVar(matrix)`: S3 function for matrix as input
- `removeAllZeroVar(DesignContrast)`: S3 function for matrix as input

Examples

```
myTestDesign <- matrix(c(1,1,1,1, 1,1,0,0,0,0,1,1,0,0,0,0),
  byrow=FALSE, nrow=4L,
  dimnames=list(sprintf("S%d", 1:4), c("Baseline", "Trt1", "Trt2", "Trt3")))
myTestContrast <- matrix(c(0,1,0,0, 0,0,1,0, 0,0,0,1), nrow=4L, byrow=FALSE,
```

```
  dimnames=list(colnames(myTestDesign), c("Trt1", "Trt2", "Trt3")))
removeAllZeroVar(myTestDesign, myTestContrast)
removeAllZeroVar(DesignContrast(myTestDesign, myTestContrast))
```

`removeColRank` *Return rank of the matrix and the ranks of resulting matrices when each column is removed*

Description

Return rank of the matrix and the ranks of resulting matrices when each column is removed

Usage

```
removeColRank(matrix)
```

Arguments

`matrix`	A numeric matrix

Value

A `data.frame` with `n+1` rows, where `n` is the column count of the input matrix

Examples

```
myMat <- matrix(c(1,1,1, 0,1,1, 0,0,1, 1,0,0), ncol=4, byrow=FALSE)
removeColRank(myMat)
```

`ribios.ExpressionSet` *An ExpressionSet for case demonstrations*

Description

This object is adapted from the `sample.ExpressionSet` object, with feature annotations from GTI (Data stand: December 2011). It is used in case studies where functionalities of the `ribiosExpression` package are demonstrated.

Format

An `ExpressionSet` object.

References

Jitao David Zhang <jitao_david.zhang@roche.com>

Examples

```
data(ribios.ExpressionSet)
tbl <- eSetToLongTable(ribios.ExpressionSet)
```

ribiosExpressionSet *ribiosExpressionSet: An example of ExpressionSet with artificial expression data*

Description

ribiosExpressionSet: An example of ExpressionSet with artificial expression data

Format

An `ExpressionSet` object.

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

Examples

```
data(ribiosExpressionSet)
```

rowscale.ExpressionSet *Perform row-wise scaling to an ExpressionSet object*

Description

Perform row-wise scaling to an ExpressionSet object

Usage

```
## S3 method for class 'ExpressionSet'
rowscale(x, center = TRUE, scale = TRUE)
```

Arguments

`x`	An ExpressionSet object.
`center`	Logical, whether the mean values of rows should be set to zero.
`scale`	Logical, whether the standard deviations of rows should be normalised to one.

Value

An ExpressionSet object with row-scaled expression values.

`sniffFeatureType` *Sniff the feature type of an object that implements the featureNames method*

Description

Sniff the feature type of an object that implements the featureNames method

Usage

```
sniffFeatureType(object, majority = 0.5)
```

Arguments

`object`	Any object that `featureNames` can be applied to, for instance an `ExpressionSet`, an `EdgeObject`, etc.
`majority`	A numeric value, used for majority voting, passed to `guessFeatureType`.

Value

A character string indicating likely feature type.

See Also

`guessFeatureType`

`splitPCA` *Split an eSet object and run PCA on each split, return PCA scores as one data.frame*

Description

Split an eSet object and run PCA on each split, return PCA scores as one data.frame

Usage

```
splitPCA(eset, factor, func = function(e) exprs(e), ...)
```

Arguments

`eset`	An eSet object
`factor`	One or more factor vectors, used to split the eSet object
`func`	Function to retrieve values from split sub-eset objects
`...`	Passed to `pcaScores`

Value

A data.frame of PCA scores combined from all splits.

Examples

```
data(ribios.ExpressionSet, package="ribiosExpression")
fac1 <- gl(2,13)
pcaScore1 <- splitPCA(ribios.ExpressionSet, fac1)
```

---

summarizeProbesets *Summarize probesets belonging to the same gene*

Description

The summarizeRows function summarizes (collapses) rows of a numeric matrix by calculating summarizing statistics of rows that belong to the same factor level.

Usage

```
summarizeProbesets(
  eset,
  index.name,
  fun = mean,
  keep.nonindex = FALSE,
  keep.featureNames = FALSE,
  ...
)
```

Arguments

eset	An ExpressionSet object
index.name	Charcter, one column name in the eset, indicating the index of probesets, probably the column holding the GeneID.
fun	Function or character, the function used to summarize probes, mean by default. Other possibilities include median.
keep.nonindex	Logical, whether probesets without valid indices should be kept or not.
keep.featureNames	Logical, whether the featureNames of the input object should be kept whenever possible. When multiple probesets are summarized into one value representing, for example, one gene (by GeneID), one arbitrary probeset is used to name the value when this option is set to TRUE. Otherwise the GeneID will be used as the name of the value. In case no summary was possible, for instance the index is NA, old feature names are kept any way.
...	Futher parameters passed to the function

Details

summarizeRows is called internally by summarizeProbesets to collapse probesets that belong to one index (e.g. GeneID).

The action of this function is univariate: namely the fun is applied to all probesets on each sample independently. For example, if fun is mean, the average value of mutliple probesets is taken for each sample. With this function, there is no way to distinguish probesets on their expression profiles (for instance: find the probeset with the maximum average signal).

Value

An ExpressionSet, with probesets summarized by indices specified.

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

See Also

summarizeRows in the ribiosUtils package.

Examples

```
data(ribios.ExpressionSet, package="ribiosExpression")
ribios.mean <- summarizeProbesets(ribios.ExpressionSet,
index.name="GeneID", fun=mean)
ribios.mean

data(ribios.ExpressionSet, package="ribiosExpression")
ribios.mean.keepFeatureNames <- summarizeProbesets(ribios.ExpressionSet,
index.name="GeneID", fun=mean, keep.featureNames=TRUE)
ribios.mean

ribios.inval.mean <- summarizeProbesets(ribios.ExpressionSet,
index.name="GeneID", fun=mean, keep.nonindex=TRUE)

## the underlying method
ribios.meanMat <- ribiosUtils::summarizeRows(exprs(ribios.ExpressionSet),
fData(ribios.ExpressionSet)$GeneID, mean)
stopifnot(identical(exprs(ribios.mean), ribios.meanMat))

## keep old featureNames
ribios.inval.mean.old <- summarizeProbesets(ribios.ExpressionSet,
index.name="GeneID", fun=mean, keep.nonindex=TRUE, keep.featureNames=TRUE)
```

summarizeSamples *Summarize samples by applying the function to sample subsets*

Description

The function takes an eSet object and a factor of the same length as the object, and summarizes samples of the same factor level by applying the function.

Usage

```
summarizeSamples(
  eset,
  indSamples = eset$SAMPLEID,
  removeInvarCols = TRUE,
  fun = sum,
  ...
)

poolReplicates(eset, indSamples = eset$SAMPLEID, removeInvarCols = TRUE)

avgReplicates(eset, indSamples = eset$SAMPLEID, removeInvarCols = TRUE)

medianReplicates(eset, indSamples = eset$SAMPLEID, removeInvarCols = TRUE)
```

Arguments

eset	An eSet object.
indSamples	A factor of the same length as the sample number of the object
removeInvarCols	Logical, whether invariant columns of the resulting eSet pheno data should be removed or not. This is useful in case there are technical replicates.
fun	The function to be applied to summarize samples
...	Other parameters passed to the function

Details

poolReplicates and avgReplicates are two specific form of the more generic summarizeSamples function: they take sum and average of replicates given by the factor, respectively.

From version 1.1-7, the function summarizes not only exprs, but also all other objects in assayData

Value

A eSet object.

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

See Also

The function calls summarizeColumns internally.

Also see summarizeProbesets.

Examples

```
data(ribios.ExpressionSet, package="ribiosExpression")
index <- factor(c(gl(12,2), 13, 14))

(ss.eset1 <- summarizeSamples(ribios.ExpressionSet, index))

(ss.eset2 <- summarizeSamples(ribios.ExpressionSet, index, fun=mean,
na.rm=TRUE))
## equivalently
(ss.eset2 <- poolReplicates(ribios.ExpressionSet, index))

(ss.eset3 <- avgReplicates(ribios.ExpressionSet, index))
```

---

truncateDgeTable *Truncate dgeTable into tables of positively and negatively differentially expressed genes according to the pre-defined criteria*

Description

Truncate dgeTable into tables of positively and negatively differentially expressed genes according to the pre-defined criteria

Usage

```
truncateDgeTable(dgeTable)
```

Arguments

dgeTable	dgeTable A DGEtable defined in ribiosExpression. Notice that the column names returned by limma::topTable are remapped (see limmaTopTable2dgeTable).

Value

A list of two elements: 'pos' and 'neg'. Each contains a dgeTable of positively/negatively regulated genes

writeCls *Export ExpressionSet as Gct/Cls files*

Description

Gct/Cls file formats are required by the Gene Set Enrichment Analysis (GSEA) tool. Functions `writeGct` and `writeCls` exports file of two formats respectively, and `writeGctCls` calls the two function internally to write two files.

Usage

```
writeCls(eset, file = stdout(), sample.group.col = "group")

writeGctCls(
  eset,
  file.base,
  feat.name,
  feat.desc,
  sample.group.col,
  write.add.fData.file = TRUE,
  write.add.pData.file = TRUE
)
```

Arguments

`eset`	An object of the `eSet` class, for example an `ExpressionSet` object
`file`	Name of the Gct/Cls file. If left missing, the file is printed on the standard output.
`sample.group.col`	Integer, character or a factor vector of the same length as the sample number, indicating classes (groups) of samples. See details.
`file.base`	For writeGctCls, the base name of the two files: the suffix (.gct and .cls) will be appended
`feat.name`	Integer or character, indicating which column of the featureData should be used as feature descriptions. If the value is missing, the Description column of the Gct file will be left blank
`feat.desc`	Integer or character, indicating which column of the featureData should be used as feature names; if missing, results of the `featureNames` function will be used as identifiers in the Gct file. See details.
`write.add.fData.file`	Logical, whether additional featureData should be written into a file named `${file.base}.add.fData.txt`
`write.add.pData.file`	Logical, whether additional phenoData should be written into a file named `${file.base}.add.pData.tx`

Details

The `feat.name` option specifies what identifiers should be used for features (probesets). When the value is missing, `featureNames` is called to provide feature identifiers.

In contrast, the `sample.group.col` cannot be missing: since cls files encode groups (classes) of samples, and if `sample.group.col` was missing, it is usually impossible to get class information from `sampleNames`.

Internally `writeCls` calls `dfFactor` function to determine factor of samples. Therefore `sample.group.col` is to a certain degree generic: it can be a character string or integer index of the `pData(eset)` data matrix, or a factor vector of the same length as `ncol(eset)`.

Value

Functions are used for their side effects.

Functions

- `writeCls()`: writeCls

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

References

https://www.gsea-msigdb.org/gsea/doc/GSEAUserGuideTEXT.htm

See Also

See `dfFactor` for possible values of the `sample.group.col` option.

See `readGctCls` for importing functions.

Examples

```
data(sample.ExpressionSet, package="Biobase")
writeGct(sample.ExpressionSet[1:5, 1:4], file=stdout())
writeCls(sample.ExpressionSet, file=stdout(), sample.group.col="type")

tmpfile <- tempfile()
writeGctCls(sample.ExpressionSet, file.base=tmpfile, sample.group.col="type")
readLines(paste(tmpfile, ".cls",sep=""))
unlink(c(paste(tmpfile, ".cls", sep=""), paste(tmpfile, ".gct", sep="")))
```

writeEset *Export an ExpressionSet object as tab-delimited (or gct) files*

Description

Export an ExpressionSet object as tab-delimited (or gct) files

Usage

```
writeEset(
  eset,
  exprs.file,
  fData.file,
  pData.file,
  exprs.file.format = c("gct", "tsv"),
  feat.name = NULL,
  feat.desc = NULL
)
```

Arguments

eset	The eSet object to be exported
exprs.file	Character, file name where exprs data is written to
fData.file	Character, optional, file name where fData data is written to
pData.file	Character, optional, file name where pData data is written to
exprs.file.format	Character, write exprs data in either gct or tsv format
feat.name	Character, feature names or a column in fData to get feature names. If NULL, feature names of the eSet object will be used. Note that if not NULL, row names of both exprs and fData will be overwritten by the provided feat.name.
feat.desc	Character, feature descriptions or a column in fData to get feature descriptions. If NULL, the column in the gct file will be empty. Only used if exprs.file.format is gct.

readEset and writeEset provide a lightweighted mechanism to import/export essential information from/to plain text files. They can use up to three tab-delimited files to store information of an eSet (oftenly used is its subclass, ExpressionSet) object: a file holding the expression matrix as returned by the exprs function (exprs.file), a file containing feature annotations as returned by the fData function (fData.file), and finally a file containing sample annotations, as returned by pData (pData.file).

Value

NULL, only side effect is used

Note

One limitation of `readEset` and `writeEset` functions is that they only support the export/import of exactly **one** expression matrix from one `ExpressionSet` object. Although an `ExpressionSet` can hold more than one matrices other than the one known as `exprs`, they are currently not handled by `writeEset` or `readEset`. If such an `ExprssionSet` object is first written in plain files, and then read back as an `ExpressionSet`, matrices other than the one accessible by `exprs` will be discarded.

Similarly, other pieces of information saved in an `ExpressionSet`, e.g. experimental data, are lost as well after a cycle of exporting and subsequent importing. If keeping these information is important for you, other functions should be considered instead of `readEset` and `writeEset`, for instance to save an image in a binary file with the `save` function.

Yet another limitation is that factor information is lost. This hits especially the phenoData where factor information, such as sample groupping and orders of levels, may be important.

See Also

`readEset`

Examples

```
data(sample.ExpressionSet, package="Biobase")
exprs.file <- tempfile()
fData.file <- tempfile()
pData.file <- tempfile()
writeEset(sample.ExpressionSet, exprs.file, fData.file, pData.file,
exprs.file.format="gct")
writeEset(sample.ExpressionSet, exprs.file, fData.file, pData.file,
exprs.file.format="tsv")
```

---

writeGct *Export matrix or eSet that can be coerced as one into gct/cls files*

Description

Export matrix or eSet that can be coerced as one into gct/cls files

Usage

```
writeGct(obj, file, feat.name, feat.desc)

## S4 method for signature 'matrix'
writeGct(obj, file, feat.name, feat.desc)

## S4 method for signature 'eSet'
writeGct(obj, file, feat.name, feat.desc)
```

Arguments

obj	The input object, see methods below for supported data types
file	The output file
feat.name	Specifying feature names
feat.desc	Specifying feature descriptions

Value

Used for its side effect of writing files; returns invisibly.

Methods (by class)

- writeGct(matrix): Method for matrix as input, feta.name and feat.desc are passed to write_gct.
- writeGct(eSet): Use eSet as input. feat.name and feat.desc are variable (column) names in fData.

---

writeSampleGroups *Write sample groups and group levels into plain text files*

Description

Write sample groups and group levels into plain text files

Usage

```
writeSampleGroups(sampleGroups, sampleGroups.file, sampleGroupLevels.file)
```

Arguments

sampleGroups	Factor, encoding sample groups.
sampleGroups.file	Character, file name where the information of sample groups is written to.
sampleGroupLevels.file	Character, file name where the information of sample group levels is written to.

Details

The function is used to export sample group and group level information for differential gene expression analysis.

Value

Used for its side effect of writing files. Returns invisibly.

Examples

```
writeSampleGroups(gl(3,4), stdout(), stdout())
```

writeVarMetadata *writexlxs: write AnnotatedDataFrame to a xlsx file*

Description

writexlxs: write AnnotatedDataFrame to a xlsx file

Usage

```
writeVarMetadata(x, path = tempfile(fileext = ".xlsx"), overwrite = TRUE)

## Default S3 method:
writeVarMetadata(x, path = tempfile(fileext = ".xlsx"), overwrite = TRUE)
```

Arguments

x	An ExpressionSet object or an AnnotatedDataFrame.
path	The xlsx file name to be written to.
overwrite	Logical, whether the file should be overwritten if it exists.

Value

An invisible TRUE in case the file is successfully created, else FALSE.

Note

We also tried the writexl package but the comments are not well supported by writexl, therefore we stay with openxlsx

Examples

```
data("ribiosExpressionSet", package="ribiosExpression")
outfile <- tempfile()
writeVarMetadata(ribiosExpressionSet, path=outfile)
writeVarMetadata(phenoData(ribiosExpressionSet), path=outfile)
```

Index