arrayMagic

June 3, 2004

R topics documented:

arrayData-class *Class arrayData, a simple container for raw data and related information*

Description

A simple class to store raw data, annotation information for spots and hybridisations, as well as weights.

Creating Objects

```
 new('arrayData',
intensities = ..., # optional; object of class array
weights = ..., # optional; object of class matrix
spotAttr = ..., # optional; object of class data.frame
hybAttrList = ..., # optional; list of two objects of class data.frame
)
```

Slots

`intensities`**:** Object of class `array`; three-dimensional; dim = nrOfSpots x nrOfChannels x nrOfHybridisations; second dimension must contain "green" and "red" and possibly "greenBackground" and "redBackground"; default `NULL`

`weights`**:** Object of class `matrix`; dim = nrOfSpots x nrOfHybridisations; range = [0,1]; default: `NULL`.

`spotAttr`**:** Object of class `data.frame`; dim = nrOfSpots x nrOfSpotCharacteristics; default `NULL`

`hybAttrList`**:** list of two objects named "green" and "red" of class `data.frame`; the dimension of each `data.frame` is given by nrOfHybridisations x nrOfHybridisationCharacteristics ; default `NULL`

Methods

intensities<- (arrayData): Set the `intensities` of `arrayData`.

getIntensities (arrayData): Returns the `intensities` object.

weights<- (arrayData): Set the `weights` of `arrayData`.

getWeights (arrayData): Returns the `weights` object.

spotAttr<- (arrayData): Set the `spotAttr` of `arrayData`.

getSpotAttr (arrayData): Returns the `spotAttr` object.

getHybAttr (arrayData): Returns the "intersection" of the "red" and "green" `data.frame` of `hybAttrList` or if one is `NULL` the other one. The "intersection" are those columns which match in name and content or `NULL`.

getHybAttrGreen (arrayData): Returns the "green" list element of `hybAttrList`, i.e. a `data.frame`.

getHybAttrRed (arrayData): Returns the "red" list element of `hybAttrList`, i.e. a `data.frame`.

hybAttrList<- (arrayData): Set the `hybAttrList` of `arrayData`.

getHybAttrList (arrayData): Returns the `hybAttrList` object.

show (arrayData): Renders information about the `arrayData` object on standard out.

[(arrayData,i,j): A subset operator, where i corresponds to the spots and j to the hybridisations.

cbind (...): Concatenates `arrayData` objects. Spots/rows are assumed to match; possibly you have to subset and reorder the objects beforehand cf. `cbind.arrayData`.

Author(s)

Andreas Buness <a.buness@dkfz.de>

See Also

`exprSetRG-class`

Examples

```
intensities <- array(data=runif(120),dim=c(20,2,3))
dimnames(intensities) <- list(NULL, c("green","red"), NULL)
spotAttr <- data.frame(Name=I(rep(c("x","y","k","l","z"),4)),
                       Index=c(1:20))
arrayDataObject <- new("arrayData", intensities=intensities, weights=intensities[,1,],
                       spotAttr=spotAttr, hybAttrList=NULL)
print(arrayDataObject)
hybs <- c(1,3)
spots <- c(1:10, 14)
aD <- arrayDataObject[spots,hybs]
print(aD)
stopifnot( all( getIntensities(arrayDataObject)[spots, , hybs] == getIntensities(aD) ) )
stopifnot( all( getWeights(arrayDataObject)[spots, hybs] == getWeights(aD) ) )
stopifnot( all( getSpotAttr(arrayDataObject)[spots, ] == getSpotAttr(aD) ) )

hybAttr <- data.frame(Name=I(c("hx","hy","hz")),
                      Index=c(1:3))
arrayDataObject <- new("arrayData", intensities=intensities, weights=intensities[,1,],
                       spotAttr=spotAttr, hybAttrList=list(green=hybAttr,red=hybAttr))
hybAttrGreen <- data.frame(Name=I(c("hx","hy","hz")),
                      Index=c(4:6))
aDObject <- new("arrayData", intensities=intensities, weights=intensities[,1,],
                       spotAttr=spotAttr, hybAttrList=list(green=hybAttrGreen,red=hybAttr))
print(arrayDataObject)
cbind(aDObject,aDObject)
hybs <- c(1)
spots <- c(1)
aD <- arrayDataObject[spots,hybs]
print(aD)
stopifnot( all( getIntensities(arrayDataObject)[spots, , hybs] == getIntensities(aD) ) )
stopifnot( all( getWeights(arrayDataObject)[spots, hybs] == getWeights(aD) ) )
stopifnot( all( getSpotAttr(arrayDataObject)[spots, ] == getSpotAttr(aD) ) )
stopifnot( all( getHybAttr(arrayDataObject)[spots, ] == getHybAttr(aD) ) )
stopifnot( all( getHybAttrRed(arrayDataObject)[spots, ] == getHybAttrRed(aD) ) )
stopifnot( all( getHybAttrGreen(arrayDataObject)[spots, ] == getHybAttrGreen(aD) ) )

stopifnot( all( getHybAttrRed(aDObject) == hybAttr) )
stopifnot( all( getHybAttrGreen(aDObject) == hybAttrGreen) )
stopifnot( all( getHybAttr(aDObject) == data.frame(Name=I(c("hx","hy","hz"))) ) )
weights(aDObject) <- intensities[,2,]
intensities(aDObject) <- intensities
spotAttr(aDObject) <- spotAttr
hybAttrList(aDObject) <- list(green=hybAttr,red=hybAttrGreen)
aD <- new("arrayData")
stopifnot( class(aD) == "arrayData" )
```

---

cbind.arrayData *Cbind for objects of class arrayData*

Description

Cbind for objects of class arrayData, cf. arrayData-class, you may simply use cbind instead of cbind.arrayData .

Usage

```
cbind.arrayData(...)
```

Arguments

...	arrayData objects

Details

cf. arrayData-class

Value

an object of class arrayData

Author(s)

Andreas Buness <a.buness@dkfz.de>

See Also

cbind arrayData-class arrayData

Examples

`cbind.exprSetRG` *Cbind for objects of class exprSetRG*

Description

Cbind for objects of class exprSetRG, cf. `exprSetRG-class`, you may simply use cbind instead of cbind.exprSetRG.

Usage

```
cbind.exprSetRG(...)
```

Arguments

`...`	exprSetRG objects

Details

cf. `exprSetRG-class`

Value

an object of class `exprSetRG`

Author(s)

Andreas Buness <a.buness@dkfz.de>

See Also

`cbind` `exprSetRG-class` `exprSetRG`

Examples

`colCors` *Correlation coefficients between the corresponding columns (rows) of two matrices that have the same size.*

Description

Correlation coefficients between the corresponding columns (rows) of two matrices that have the same size.

Usage

```
rowCors(x,y)
colCors(x,y)
```

Arguments

x	Matrix
y	Matrix, same size as x

Details

The implementation is naive.

Value

A vector with the correlation coefficients for each row (`rowWiseSds`), or column (`rowWiseSds`)

See Also

colSums

Examples

```
x  = matrix(runif(1e6), ncol=100)
y  = matrix(runif(1e6), ncol=100)

commands = c(
  "c1 <<- colCors(x,y)",
  "c2 <<- sapply(1:ncol(x), function(i) cor(x[,i], y[,i]))",
  "c3 <<- rowCors(x,y)",
  "c4 <<- sapply(1:nrow(x), function(i) cor(x[i,], y[i,]))")

times = sapply(commands, function(text) system.time(eval(parse(text=text)))[1])
print(t(times))

stopifnot(all(abs(c1-c2) < 1e-3))
stopifnot(all(abs(c3-c4) < 1e-3))
```

---

colorramp *colorramp*

Description

Arguments

palette

Value

Author(s)

Examples

`detectReplicas` *detectReplicas*

Description

Usage

```
detectReplicas(arrayDescription, spotIdentifier = "ID", identifiersToBeSkipped = "Blank")
```

Arguments

`arrayDescription`	an object of class `data.frame` which contains column named as `spotIdentifier`; required; default missing
`spotIdentifier`	character string; required; default: "ID"
`identifiersToBeSkipped`	vector of character strings; required; default: "Blank"; identifiers to be ignored

Value

A list which contains `nrOfReplicas` and `spotReplicas`. `nrOfReplicas`: one integer characterizing the number of spot replicates given for each identifier if existing or otherwise `NA`. `spotReplicas`: a list of the length of the unique identifiers where each element contains a vector of indexes corresponding to the given identifier (i.e. the name of the list element) otherwise `NA`

Author(s)

Andreas Buness <a.buness@dkfz.de>

Examples

```
aD <- data.frame(ID=c("z", "x", "x", "x", "y", "z", "z", "y", "y"))
re <- detectReplicas(aD, identifiersToBeSkipped = c("Blank", "Control1", "Control2"))
stopifnot(re[["nrOfReplicas"]] == 3 )

aD <- data.frame(ID=c("Blank", "Control1", "Blank", "Control2"))
re <- detectReplicas(aD, identifiersToBeSkipped = c("Blank", "Control1", "Control2"))
stopifnot(is.na(re[["nrOfReplicas"]]))
```

exprSetRG-class *Class exprSetRG for two colour DNA microarray data (extension of exprSet)*

Description

This is a class representation for two colour DNA Microarray Data. The class is based on the class `exprSet` of the `Biobase` package. The red and green channels are stored in a single `exprSet` object. The information on the corresponding red-green pairs is stored separately. Several class methods offer a convienent way to access and set data.

Creating Objects

```
 new('exprSetRG',
channels = ..., # object of class matrix with columns "green" and "red"
exprs = ..., # object of class matrix
se.exprs = ..., # object of class matrix
phenoData= ..., # object of class phenoData
notes = ..., # object of class character
annotation = ..., # object of class character
)
```

Slots

`indGreen`: Object of class "vector"; indexes of the green channel

`indRed`: Object of class "vector"; indexes of the red channel

`exprs`: Object of class "matrix"; the observed expression levels. This is a matrix with columns representing the red and green channels, the pairs given in `channels` define slides and rows representing genes.

`se.exprs`: Object of class "matrix"; this is a matrix of the same dimensions as `exprs` which may for example contain standard error estimates for the corresponding expression levels.

`phenoData`: Object of class "phenoData" This is an instance of class `phenoData` containing channel annotations. The columns of the pData slot of this entity represent variables and the rows represent channels.

`notes`: Object of class "character" Vector of explanatory text; default: ""

`annotation`: Object of class "character"; default: ""

Extends

Class `"exprSet"`, directly.

Methods

show (exprSetRG): renders information about the exprSetRG in a concise way on stdout, cf. exprSet

getExprSetLogRatio (exprSetRG): Returns an `exprSet` object of the difference of the expression levels, i.e. the green channel green minus the red channel. The `se.exprs` slot contains the sum of the se.exprs of both channels. The `phenoData` slot it the result of a call `phenoDataSlide` on the object `exprSetRG`.

getExprSetGreenMinusRed same as `getExprSetLogRatio`

getExprSetGreen (exprSetRG): Returns an `exprSet` object of the expression levels of the green channel

getExprSetRed (exprSetRG): Returns an `exprSet` object of the expression levels of the red channel

phenoDataSlide (exprSetRG): Returns an `phenoData` object of the slides which contains those annotation information (variables), which are the same for both channels and not `NA`, e.g. the slide number, and all other information is presented channel-wise, i.e. prefixed with "greenSpecific_" and "redSpecific_". Do not use varLabels-names for subsetting.

pDataSlide (exprSetRG): Returns an `pData` object, i.e. `pData(phenoDataSlide(exprSetRG))`; cf. `phenoDataSlide`. Do not use varLabels-names for subsetting.

phenoDataGreen (exprSetRG): Returns an `phenoData` object of the annotation information given for the green channel. Do not use varLabels-names for subsetting.

pDataGreen (exprSetRG): Returns the `pData` object given by `phenoDataGreen`. Do not use varLabels-names for subsetting.

phenoDataRed (exprSetRG): Returns an `phenoData` object of the annotation information given for the red channel. Do not use varLabels-names for subsetting.

pDataRed (exprSetRG): Returns the `pData` object given by `phenoDataRed`. Do not use varLabels-names for subsetting.

slideSubset (exprSetRG,i,j): Subsetting; i corresponds to the rows and j corresponds to the slides. j is given by indexes or logicals related to the order of the channel pairs; cf. the constructor slot `channels`

getIndGreen (exprSetRG): An accessor function for slot `indGreen`. The matching pairs of indGreen and indRed define slides.

getIndRed (exprSetRG): An accessor function for slot `indRed`. The matching pairs of indGreen and indRed define slides.

[(exprSetRG,i,j,type): A subset operator. Ensures that both the data and the annotation information (`phenoData`) are subseted properly. This may mix up the pairing of the channels, i.e. the validity of an exprSetRG, only if you use type == "invalidExprSetRG".

cbind (...): Concatenates `exprSetRG` objects. Genes/rows are assumed to match; cf. `cbind.exprSetRG`

as.exprSet (exprSetRG): Class cast, returns an object of `exprSet`, the channel information is discarded.

Author(s)

Andreas Buness <a.buness@dkfz.de>

See Also

`exprSet-class`, `arrayData-class`

Examples

```
indGreen=1:3
indRed=4:6
channels <- matrix( c(indGreen,indRed), nrow=length(indGreen), byrow=FALSE )
```

```
colnames(channels) <- c("green","red")
eSA <- new("exprSetRG", exprs=matrix(1:60, ncol=6, nrow=10), phenoData=
        new("phenoData", pData=data.frame(matrix(0,nrow=6,ncol=1)),
            varLabels=list(rep("varLabel1",1))), channels=channels)
stopifnot( all(pDataSlide(eSA) ==pData(eSA)[1:3,,drop=FALSE]) )
eSAGreen <- getExprSetGreen(eSA)
eSARed <- getExprSetRed(eSA)
eSALogRatio <- getExprSetLogRatio(eSA)
eSALogRatio2 <- getExprSetGreenMinusRed(eSA)
stopifnot( identical( eSALogRatio, eSALogRatio2 ) )
stopifnot( identical( exprs(eSALogRatio), exprs(eSAGreen)-exprs(eSARed)) )
eSAPart <- eSA[,c(1,3,1,4,6,4)]
eSAInvalid <- eSA[,c(1,3,1,5,6,4),type="invalidExprSetRG"]
eSAPart2 <- slideSubset(eSA,j=c(1,3,1))

eSAeSA <- cbind(eSA, eSA)
eSAeSAPart2 <- cbind(eSA, eSAPart2)
stopifnot( class(as.exprSet(eSA)) == "exprSet" )
```

fdc *FDC (false discovery count)*

Description

Estimate the FDC (false discovery count) through permutations

Usage

```
fdc(x, fac,
  teststatfun = "rowFtests",
  nrperm      = 100,
  nrgenesel   = c(10, 20, 40, 60, 80, 100, 200),
  ...)
```

Arguments

`x`	Matrix.
`fac`	Factor, with `length(fac)=ncol(x)`.
`teststatfun`	Character. Name of a function that takes arguments `x` and `fac`, and returns a list with component `statistic`. See for example `rowFtests`.
`nrperm`	Numeric. Number of permutations.
`nrgenesel`	Numeric. A vector with the 'number of genes' for which the FDC is to be calculated.
`...`	Further arguments passed to codeteststatfun.

Value

A list with elements `stat`: the test statistics; `mpstat`: median permuted test statistics; `fdc`: estimated false discovery counts; `thresh`: the threshholds associated with `nrgenesel`; `nrgenesel`

Author(s)

Wolfgang Huber <w.huber@dkfz.de>

See Also

`rowFtests`

Examples

```
## data matrix: 2000 genes, 16 samples
x  <- matrix(runif(2000*16), ncol=16)
## 8 blue and 8 red samples
fac <- factor(c(rep("blue", 8), rep("red", 8)))
## implant differential signal into the first 50 genes
x[1:50, fac=="blue"] <- x[1:50, fac=="blue"] + 1

res <- fdc(x, fac)
plot(res$nrgenesel, res$fdc, pch=16, col="blue",
    xlab="Number of genes selected",
    ylab="Expected number of false discoveries")
abline(a=0, b=1, col="red", lwd=2)

qqplot(res$stat, res$mpstat, pch=".")
abline(a=0, b=1, col="red", lwd=2)
```

`interweave` *interweave*

Description

The first object is "interweaved" with the second object and the resulting object with the following object and continues as long as the last object is reached. Note: `AsIs` class objects are coerced to a vector before interweaving.

Usage

```
interweave(...)
```

Arguments

`...`	vectors, matrices, one- and two-dimensional arrays; an `AsIs` class object is coerced to a vector

Value

interweaved object

Author(s)

Andreas Buness <a.buness@dkfz.de>

Examples

```
x <- seq(1,100,2)
y <- seq(2,100,2)
z <- interweave(x,y)
stopifnot( all.equal(z,1:100) )
```

`normalise` *Normalisation of Microarray Data*

Description

An object of class `exprSetRG` is generated which contains the result of the normalisation.

Usage

```
normalise(arrayDataObject, subtractBackground = FALSE, method = "vsn", subGroups = NULL, cha
```

Arguments

`arrayDataObject`	object of class `arrayData`; required; ; default missing. `arrayDataObject` must contain the raw data, i.e. a three dimensional array (spot x channel x hybridisation), cf. `getIntensities` and must contain information on the hybridisations, cf. `getHybAttr`. If argument `subGroups` is specified it must also contain information on the spots, cf. `getSpotAttr`. Note: Weights are only used by loess-type normalisations.
`subtractBackground`	logical; default: `FALSE`
`method`	character string; required; default: "`vsn`"; possible values: "`none`", "`vsn`", "`quantile`", "`loess`", "`loessScale`", "`loessQuantile`". Note: "`quantile`" and "`loess*`" data are transformed to the (natural) logarithmic scale. Note: Weights are only used by loess-type normalisations. Note: "`loessScale`" and "`loessQuantile`" refer to a loess normalisation followed by a between slide normalisation, cf. the function `normalizeBetweenArrays` of the `limma` package.
`subGroups`	`NULL` or character string; required; default: `NULL`; `subGroups` allows to define subgroups on each hybridisation which are normalised separately like a print-tip normalisation. The list must contain a column name refering to the `data.frame` of `getSpotAttr(arrayDataObject)` like "Block" in case of GenePix data. The column itself must contain integer values. Note: In case of `method == "vsn"`, `vsn` is called with the argument `strata`, which is different from a separate normalisation of each subgroup.
`channelsSeparately`	logical; required; default: `FALSE`; If `channelsSeparately` ist set to `TRUE` each channel is normalised separately. Only meaningful for single channel normalisation methods like "vsn" "quantile" normalisation but not for ratio based normalisation like "loess".

`hybridisationGroups`
list of vectors of indexes or character string ”slideBySlide”; optional; default: missing. Each group of hybridisations is normalised separately. If missing all hybridisations are taken as one group. Only meaningful normalisation methods like ”vsn” and ”quantile”. The indexes must refer to the third dimension of `getIntensities(arrayDataObject)` and have to contain all hybridisations.

`spotIdentifier`
character string; optional; default missing. `spotIdentifier` specifies the column of `getSpotAttr(arrayDataObject)` which must contain non-unique spot or gene identifiers. The identifiers are used as names for the resulting `exprSetRG`-object; cf. function `geneNames`

`verbose` logical; required; default: TRUE

Value

object of class `exprSetRG`

Author(s)

Andreas Buness <a.buness@dkfz.de>

See Also

`processArrayData`, `exprSetRG-class`,

Examples

```
intensities <- array(data=runif(720),dim=c(120,2,3))
dimnames(intensities) <- list(NULL, c("green","red"), NULL)
hybAttr <- data.frame(Name=I(c("hx","hy","hz")), Index=c(1:3))
 arrayDataObject <- new("arrayData",
                      intensities=intensities,
                      hybAttrList=list(red=hybAttr,green=hybAttr)
                     )
exprSetRGObject <- normalise(arrayDataObject = arrayDataObject,
                       subtractBackground = FALSE,
                       method = "none",
                       verbose = TRUE
                       )
nRed <- exprs(getExprSetRed(exprSetRGObject))
nGreen <- exprs(getExprSetGreen(exprSetRGObject))
stopifnot( all.equal.numeric( nRed, intensities[,"red",] ) )
stopifnot( all.equal.numeric( nGreen, intensities[,"green",] ) )
```

`plot.imageMatrix` *Visualisation of a matrix*

Description

Visualisation of a data matrix, e.g. a matrix of distance or similarity scores, or any numeric matrix

Usage

```
plot.imageMatrix(x, labelMatrix, zlim, separateZScale=FALSE, zScale=TRUE, colourRamp, revers
```

Arguments

`x`	data matrix, class `matrix`; required; default missing
`labelMatrix`	matrix of labels of class `array` of dim(x); optional; default missing
`zlim`	numeric vector; defines the z-range of the plot; default missing
`zScale`	logical; adds a scale to the plot; required; default TRUE
`separateZScale`	logical; extra plot for the scale; required; default FALSE
`colourRamp`	vector of colours; optional; default missing
`reverseYaxis`	logical; required; default `TRUE`
`labels`	vector of names corresponding to a quadratic `x`; optional; default missing
`xLabels`	vector of names corresponding to the columns of `x` overrides `labels`; optional; default missing
`yLabels`	vector of names corresponding to the rows of `x` overrides `labels`; optional; default missing
`width`	graphics window width; required; default: 8
`height`	graphics window height; required; default: 7
`labelMatrixTextScaling`	numeric; required; default: 0.7
`plotOutput`	character string specifying either ”standard”, ”screen” or ”twoScreens” or ”pdf”; required; default: ”standard”
`fileName`	character string specifying the file path and file name; optional; default missing
`...`	arguments are passed to `image`(), for example `xlab`, `ylab` (x- and y-axis label) and `main` (plot title)

Details

if no labels are supplied the dimnames or alternatively a numbering is used instead

Author(s)

Andreas Buness <a.buness@dkfz.de>

Examples

```
plot.imageMatrix(x=matrix(c(3,4,4,3),nrow=2, ncol=2),labels=c("one","two"))
ma <- matrix(c(0.3,0.01,0.7,0.1,0.5,0.3,1,0.5,01), nrow=3,ncol=3)
class(ma) <- c("imageMatrix", "matrix")
plot(ma, labelMatrix=ma,labels=c("one","two", "three"), zlim=c(0,1))
ma <- matrix(c(0.3, 0.01, 0.7, 0.1, 0.5, 0.3, 1, 0.5, 1, 0, 1, 0), nrow = 4, ncol = 3, byrow=TRUE )
class(ma) <- c("imageMatrix", "matrix")
plot(ma, labelMatrix = ma, xLabels = c("one", "two", "three"), zlim=c(0,1))
plot(ma, reverseYaxis=FALSE, labelMatrix = ma, xLabels = c("one", "two", "three"), zlim=c(0,1))
```

`plotDistributions` *Visualise Distributions*

Description

Boxplot like visualisation of distributions, only the boxes, i.e. the median and roughly the second and third quartile are plotted (cf. `boxplot.stats`). The plots may help to identify shortcomings of the raw data or normalised data. The argument `quantiles` can be used to visualize two or three arbitrary quantiles with boxes.

Usage

```
plotDistributions(dataMatrix, transFunc, quantiles, main, labels, xlab, ylab, colourVector,
```

Arguments

`dataMatrix`	data matrix, where columns represent distributions, e.g. raw array data or normalised data; required; default missing
`transFunc`	unary function; optional; default missing. Data transformation function, e.g. log.
`quantiles`	missing by default; a vector of two or three increasing quantiles used to determine the boxes to be drawn
`main`	plot title, type character string
`labels`	vector of names, may substitute column names of `dataMatrix`
`xlab`	label for x axis
`ylab`	label for y axis
`colourVector`	vector of colours
`width`	graphics window width
`height`	graphics window height
`fileName`	optional; default: ”plotDistributionsOutput.pdf”
`savePath`	optional; default: missing
`plotOutput`	character string specifying either ”standard”, ”screen” or ”pdf”; default: ”standard”

Details

Default of `transFunc` is no transformation, i.e. identity. If `labels` are supplied at first the column names or secondly a numbering are used instead.By default the `colourVector` is defined as alternating darkred and darkgreen.

Value

Author(s)

Andreas Buness <a.buness@dkfz.de>

See Also

`boxplot.stats`

Examples

```
plotDistributions(cbind(rnorm(100),rnorm(100)),
       main="Random Gaussians", labels = c("N1","N2"), ylab="scale")
plotDistributions(as.matrix(1:100), quantiles=c(0.25,0.85),
       main = "Random Gaussians", labels = c("N1"), ylab = "scale")
```

`processArrayData` *Automated processing of two colour DNA microarray data*

Description

Automated processing of image analyis result files and related annotation information.

Usage

```
processArrayData(spotIdentifier = "Name", verbose = TRUE, loadPath = ".", slideDescriptionF
```

Arguments

`spotIdentifier`	character string; required; default "Name". `spotIdentifier` specifies the column in the image analysis result files which contain possibly non-unique spot or gene identifiers.
`verbose`	logical; required; default: TRUE
`loadPath`	character string; required; default: ".". The path is used to load the `slideDescriptionFile` and data files; note: "." refers to the working directory.
`slideDescriptionFile`	character string; required; default "slideDescription.txt". The first line of the file must contain all column names, in particular the column named `fileNameColum` and possibly additionally a column named `slideNameColumn`
`deleteBlanks`	logical; required; default: TRUE. Any blank character is removed from the text which is read from the `slideDescriptionFile`

`fileNameColumn`
: character string; required; default: ”fileName”. `fileNameColumn` specifies the column which contains all image analysis result files.

`slideNameColumn`
: character string; optional; default missing. If `slideNameColumn` is missing the value is set to `fileNameColumn`.

`channelColumn`
: optional; cf. `readIntensities`; default: `NULL`

`type`
: character string to characterize the file type like ”GenePix” or ”generic”; note e.g. ”generic” requires the arguments `dataColumns` and `spotAnnoColumns`; default: ”GenePix”; cf. `readIntensities` for details

`dataColumns`
: required for `type` ”generic”; cf. `readIntensities`; default: `NULL`

`spotAnnoColumns`
: required for `type` ”generic”; cf. `readIntensities`; default: `NULL`

`skip`
: optional; cf. `readIntensities`; default: `NULL`

`normalisationMethod`
: character string; required; default: ”vsn”; cf. `normalise`

`subtractBackground`
: logical; required; default: `FALSE`

`spotsRemovedBeforeNormalisation`
: vector of character strings; required; default: `NULL`. All spots which match the string(s) will be excluded already before normalisation and will not be present in the data objects at all.

`spotsRemovedAfterNormalisation`
: vector of character strings; required; default: `NULL`; cf. `spotsRemovedBeforeNormalisation`.

`subGroups`
: `NULL` or character string; required; cf. `normalise` default: `NULL`

`channelsSeparately`
: logical; required; default `FALSE`

`hybridisationGroups`
: list of numeric vectors; cf. `normalise`; required; default: NULL

`savePath`
: character string; required; default: ”.”. The list of results objects is stored in the directory `savePath`. If the path does not exist a directory is created; note: ”.” refers to the working directory.

`objectsFileName`
: character string; optional; default missing. `objectsFileName` specifies the name of the file which contains the ”resultList”, i.e. the return value of the function.

`plotOutput`
: character string; required; default: ”screen”; Possible values: ”screen” or ”pdf”.

Value

A list of objects, i.e. an ”exprSetRGObject” ”arrayDataObject” with corresponding class types `exprSetRG-class` and `arrayData-class`. Side-effects: The result list ”resultList” is stored as file `objectsFileName` if in the directory `savePath` if the argument is supplied and the `slideDescriptionFile` is stored in the directory `savePath`.

Author(s)

Andreas Buness <a.buness@dkfz.de>

See Also

readpDataSlides, readIntensities, processArrayDataObject, normalise, exprSetRG-class, arrayData-class

Examples

```
        LOADPATH <- file.path(.path.package("arrayMagic"), "extdata")
        SAVEPATH <- tempdir()
         SLIDEDESCRIPTIONFILE <- "slideDescription"

         resultList <- processArrayData(
                              loadPath=LOADPATH,
                               savePath=SAVEPATH,
                              slideDescriptionFile=SLIDEDESCRIPTIONFILE
                   )
        writeToFile(arrayDataObject=resultList$arrayDataObject,
                    exprSetRGObject=resultList$exprSetRGObject,
                    fileName="normalisedData.txt",
                    savePath=SAVEPATH)

        summarizedResult <- slideMerge(exprSetRGObject=resultList$exprSetRGObject, slideMergeColumn="re

         qPL <- qualityParameters(arrayDataObject=resultList$arrayDataObject,
                                  exprSetRGObject=resultList$exprSetRGObject)

         visualiseQualityParameters(qualityParameters=qPL$qualityParameters,
                                    savePath=tempdir())

         qualityDiagnostics(
                     arrayDataObject=resultList$arrayDataObject,
                     exprSetRGObject=resultList$exprSetRGObject,
                     qualityParametersList=qPL,
                     slideNameColumn="fileName",
                     savePath=tempdir(),
                     plotOutput="pdf")

         unlink(file.path(SAVEPATH, paste(SLIDEDESCRIPTIONFILE,"_processed",sep="")))
          resultListG <- processArrayData(
                              loadPath=LOADPATH,
                               savePath=SAVEPATH,
                              slideDescriptionFile=SLIDEDESCRIPTIONFILE,
                              plotOutput="pdf",
                              hybridisationGroups = list((1:4),(5:9))
                   )
         unlink(file.path(SAVEPATH, paste(SLIDEDESCRIPTIONFILE,"_processed",sep="")))
          resultListG2 <- processArrayData(
                              loadPath=LOADPATH,
                               savePath=SAVEPATH,
                              slideDescriptionFile=SLIDEDESCRIPTIONFILE,
                              plotOutput="pdf",
                              objectsFileName = "exprSetRG.RData",
                              hybridisationGroups = "slideBySlide"
                                          )
    unlink(file.path(SAVEPATH, paste(SLIDEDESCRIPTIONFILE,"_processed",sep="")))

  SLIDEDESCRIPTIONFILE <- "genericChannelsPerFile"
```

```
spotAnnoColumns <- c("Index", "Label" , "Type" , "Name" , "ID" )
dataColumns <- c("Normalized....","Average....","Normalized....","Average....")
names(dataColumns) <- c("greenForeground","greenBackground",
                     "redForeground","redBackground")

resultGenericChannel <- processArrayData(
                              spotIdentifier="Index",
                              loadPath=LOADPATH,
                              savePath=SAVEPATH,
                              slideDescriptionFile=SLIDEDESCRIPTIONFILE,
                              normalisationMethod="none",
                              channelColumn="channel",
                              fileNameColumn="files",
                              slideNameColumn="name",
                              type="genericOneFilePerChannel",
                              spotAnnoColumns=spotAnnoColumns,
                              dataColumns=dataColumns
                          )

  unlink(file.path(SAVEPATH, paste(SLIDEDESCRIPTIONFILE,"_processed",sep="")))
SLIDEDESCRIPTIONFILE <- "genericChannelsPerFileTwo"
dataColumns <- c("Integral..QL.","Bkg..QL.", "Integral..QL.","Bkg..QL.")
names(dataColumns) <- c("greenForeground","greenBackground",
                     "redForeground","redBackground")

resultGenericChannelTwo <- processArrayData(
                              spotIdentifier="ID",
                              loadPath=LOADPATH,
                              savePath=SAVEPATH,
                              slideDescriptionFile=SLIDEDESCRIPTIONFILE,
                              normalisationMethod="vsn",
                              channelColumn="channel",
                              fileNameColumn="files",
                              slideNameColumn="name",
                              subtractBackground=TRUE,
                              type="genericOneFilePerChannel",
                              spotAnnoColumns=spotAnnoColumns,
                              dataColumns=dataColumns
                          )
```

processArrayDataObject

Automated processing and normalisation of an arrayData-object (part of function processArrayData)

Description

Automated processing and normalisation of an arrayData-object.

Usage

```
processArrayDataObject(arrayDataObject, spotIdentifier = "Name", verbose = TRUE, normalisat
```

Arguments

`arrayDataObject`	object of class `arrayData-class`; required; default missing.
`spotIdentifier`	character string; required; default "Name". `spotIdentifier` specifies the column in the image analysis result files which contain possibly non-unique spot or gene identifiers.
`verbose`	logical; required; default: TRUE
`normalisationMethod`	character string; required; default: "vsn"; cf. `normalise`
`subtractBackground`	logical; required; default: `FALSE`
`spotsRemovedBeforeNormalisation`	vector of character strings; required; default `NULL`. All spots which match the string(s) will be excluded already before normalisation and will not be present in the data objects at all.
`spotsRemovedAfterNormalisation`	vector of character strings; required; default: `NULL`; cf. `spotsRemovedBeforeNormalisation`.
`subGroups`	`NULL` or character string; required; cf. `normalise` default: `NULL`
`channelsSeparately`	logical; required; cf. `normalise` default `FALSE`
`hybridisationGroups`	list of numeric vectors; cf. `normalise` required; default: NULL

Value

A list of objects, i.e. an "exprSetRGObject" "arrayDataObject" with corresponding class types `exprSetRG-class` and `arrayData-class`.

Author(s)

Andreas Buness <a.buness@dkfz.de>

See Also

`processArrayData`, `normalise`, `exprSetRG-class`, `arrayData-class`

Examples

```
LOADPATH <- file.path(.path.package("arrayMagic"), "extdata")
 SLIDEDESCRIPTIONFILE <- "slideDescription"

slideDescription <- readpDataSlides(
                      loadPath=LOADPATH,
                      slideDescriptionFile=SLIDEDESCRIPTIONFILE
                    )
 arrayDataObject <- readIntensities(
                      loadPath=LOADPATH,
                      slideDescription=slideDescription
                   )

resultList <- processArrayDataObject( arrayDataObject=arrayDataObject )
```

`qualityDiagnostics` *qualityDiagnostics*

Description

Several quality diagnostic plots are generated. The distributions of the (normalised) intensities, as well as overall similarities between hybridisations and several other quality parameters are graphically visualised (cf. `qualityParameters`.)

Usage

```
qualityDiagnostics(arrayDataObject, exprSetRGObject, qualityParametersList, slideNameColumn
```

Arguments

`arrayDataObject`	required; default: missing
`exprSetRGObject`	required, default: missing
`qualityParametersList`	required; default: missing
`slideNameColumn`	optional; default: missing; specifies a column of `pDataSlide(exprSetRGObject)`.
`savePath`	required; default: "."
`completeOutput`	required; default: `FALSE`
`verbose`	logical; required; default: TRUE
`plotOutput`	character string specifying output plotOutput; either "screen" or "pdf"; default: "pdf"

Details

Value

Author(s)

Andreas Buness <a.buness@dkfz.de>

See Also

`qualityParameters`, `exprSetRG-class`, `arrayData-class`

Examples

`qualityParameters` *Calculation of quality characteristics for DNA chip hybridisations*

Description

Usage

```
qualityParameters(arrayDataObject, exprSetRGObject, spotIdentifier = "Name", slideNameColum
```

Arguments

`arrayDataObject`	object of type arrayData; required; default: missing
`exprSetRGObject`	object of type exprSetRG; required; default: missing
`spotIdentifier`	character string; required; specifies a column of `getSpotAttr(arrayDataObject)`; the column is used to determine spot replicates; default: "Name"
`slideNameColumn`	character string; required; specifies a column of `getHybAttr(arrayDataObject)`; the column is used to extract the names of the hybridisations; if not available the hybridisations are consecutively numbered; default: "slideName"
`identifiersToBeSkipped`	vector of character strings of spot identifiers to be excluded from calculations
`resultFileName`	character string; results are stored in a file if supplied; default: missing
`verbose`	logical; default `TRUE`

Details

Value

returns a list of results, i.e. a data frame `qualityParameters` containing several scores for each hybridisation, a matrix `slideDistance`, a matrix `slideDistanceLogRaw`, a matrix `slideDistanceGreen`, a matrix `slideDistanceGreenLogRaw`, a matrix `slideDistanceRed`, a matrix `slideDistanceRedLogRaw`, and an integer `replicateSpots`, i.e. the number of spot replicates.

The matrix `slideDistanceLogRaw` contains a calculated distance (similarity) for each pair of slides$_{ij}$, i.e. the median absolute deviation (mad) taken over all spots of the log ratio of the raw data; the matrix `slideDistance` contains the same, i.e. the mad taken over all spots of the difference of the "log-ratios" ("log-ratios": the difference of the normalised expression values of the two channels on the slide). Similarly the matrices `slideDistanceGreen`, `slideDistanceGreenLogRaw`, `slideDistanceRed`, and `slideDistanceRedLogRaw` contain calculated distances for each pair of slides$_{ij}$ based on the mad of the difference of the same channel (normalised or logged) taken over all spots.

A brief summary of all parameters given in the data frame `qualityParameters`:

`width` a robust measure of the variance, i.e. the median absolute deviation of the difference of the normalised channels taken over all spots

`medianDistance` a robust measure for the typical distance (similarity) of one slide with all other slides, i.e. the median of the "distances" between slides (c.f. `slideDistance`))

`correlation(LogRaw)` of the expression values between the two normalised (log raw) channels of the slide taken over all spots

`meanSignalGreen` the mean taken over all spots of the green raw data channel

`meanSignalRed` the mean taken over all spots of the red raw data channel

`meanSignal` mean taken over all spots of the raw data of both channels,

`signalRangeGreen` the range between the 10th and 95th percentile of the signal intensities given in the green raw data channel

`signalRangeRed` the range between the 10th and 95th percentile of the signal intensities given in the red raw data channel

`backgroundRangeGreen` the range between the 10th and 95th percentile of the background intensities given in the green raw data channel

`backgroundRangeRed` the range between the 10th and 95th percentile of the background intensities given in the red raw data channel

`signalToBackgroundGreen` the ratio of the median signal intensity and the median background intensity given in the green raw data channel

`signalToBackgroundRed` the ratio of the median signal intensity and the median background intensity given in the red raw data channel

`spotReplicatesConcordanceGreen(LogRaw)` the median of the standard deviations of all spot replicates for each unique identifier of the normalised (log raw) green channel is calculated; in case of duplicates, i.e. `replicateSpots == 2`, the Pearson and Spearman correlation is calculated instead

`spotReplicatesConcordanceGreen(LogRaw)` the median of the standard deviations of all spot replicates for each unique identifier of the normalised (log raw) green channel is calculated; in case of duplicates, i.e. `replicateSpots == 2`, the Pearson and Spearman correlation is calculated instead

`greenvsAllGreen` and `redvsAllRed` the correlation between each channel is measured against the averaged (median) channel over all hybridisations (e.g. a virtual reference)

Author(s)

Andreas Buness <a.buness@dkfz.de>

See Also

`qualityDiagnostics`

Examples

```
spotIdentifierVec <- c("A","A","Blank","B","B","Blank")
hybNames <- "H1"
R1 <- N1 <- c(1,1,9,2,2,10)
R2 <- N2 <- c(2,2,7,4,4,8)
rawDataIntensityValues <- array(0, dim=c(6,2,1))
rawDataIntensityValues[,1,] <- R1
```

```
rawDataIntensityValues[,2,] <- R2
dimnames(rawDataIntensityValues) <- list(NULL, c("green","red"), NULL)
spotAttr <- data.frame(Name=I(spotIdentifierVec))
hybAttr <- data.frame(slideName=I(hybNames))
arrayDataObject <- new("arrayData", intensities=rawDataIntensityValues, hybAttrList=list(red=hyb
indGreen <- 1
indRed <- 2
channels <- matrix( c(indGreen,indRed), nrow=length(indGreen), byrow=FALSE )
colnames(channels) <- c("green","red")
exprSetRGObject <- new("exprSetRG",
exprs <- matrix(c(R1,R2), nrow=6, byrow=FALSE), phenoData=
    new("phenoData", pData=data.frame(matrix(0,nrow=2,ncol=1)),
        varLabels=list(rep("varLabel1",1))), channels=channels)
Re1 <- qualityParameters(arrayDataObject=arrayDataObject, exprSetRGObject=exprSetRGObject, ident
stopifnot(all.equal.numeric(as.numeric(Re1$qualityParameters["H1",c("correlation")]),c(1)))
stopifnot(Re1$replicateSpots==2)
```

`readIntensities` *readIntensities*

Description

The function takes the `data.frame` `slideDescription` as input and reads the listed image analysis raw data files of column `slideNameColumn`; see also `readpDataSlides`. The raw data information is returned as an object of class `arrayData`. Note: All image analysis quantification (=raw data) files have to correspond to the same type of microarray.

Usage

```
readIntensities(slideDescription, fileNameColumn="fileName", slideNameColumn, channelColumn
```

Arguments

`slideDescription`
: data frame; required; default: missing. The data frame must contain at least one column; this column has to be named `fileNameColumn`. It may additionally contain a column named `slideNameColumn`.

`fileNameColumn`
: character string; required; default: "fileName". `fileNameColumn` specifies the column which contains all image quantification result files in the data frame `slideDescription`.

`slideNameColumn`
: character string; optional; default missing; refers to the data frame `slideDescription`. If `slideNameColumn` is missing the value is set to `fileNameColumn`.

`channelColumn`
: named vector of character strings; optional; default `NULL`. If the data frame `slideDescription` contains information for each channel of every slide/hybridisation separately, the `channelColumn` vector contains the column name of the data frame `slideDescription` used for the coding. If length(channelColumn) == 1 the character strings "green" and "red"

	are assumed to be used for the coding, otherwise names(channelColumn) must contain: c(”channelColumnName”,”green”,”red”).
`loadPath`	character string; required; default: ”.”. The path is used to load the image quantification result files; note: ”.” refers to the working directory.
`spotIdentifier`	character string; optional; default missing. `spotIdentifier` specifies the column in the image analysis result files which contain possibly non-unique spot or gene identifiers.
`type`	character string; required; possible values: ”GenePix”, ”ScanAlyze”, ”generic” and ”genericOneFilePerChannel”; cf. Details section. Note: value ”generic” requires `spotAnnoColumns`, `dataColumns` and possibly `skip`, value ”genericOneFilePerChannel” additionally requires the argument ”channelColumn”; whereas ”GenPix” and ”ScanAlyze” use predefined values if not otherwise specified. default: ”GenePix”.
`spotAnnoColumns`	vector of character strings; the column names of the image analysis data file, which contain the same information for all files. The argument `spotIdentifier` is automatically added to the vector if not given; default: `NULL`
`dataColumns`	named vector of character strings; the column names of the image analysis data file, which contain the raw intensities values of each spot; names(dataColumns) must contain: c(”greenForeground”,”greenBackground”, ”redForeground”,”redBackground”); default: `NULL`
`skip`	integer; default: `NULL`; number of lines skipped in each image analysis data file.
`verbose`	logical; required; default `TRUE`

Details

Details on the argument `type`: `type="GenePix"` defines `spotAnnoColumns = c("Block", "Column", "Row", "Name", "ID")` and `dataColumns = c("F532.Median", "B532.Median", "F635.Median", "B635.Median")`, `names(dataColumns) = c("greenForeground","greenBackground","r` and `skip = grep("Block...Column", imageFile) - 1`, whereas `type="ScanAlyze"` defines `spotAnnoColumns = c( "HEADER", "SPOT", "GRID", "ROW", "COL" )` and `dataColumns = c("CH1I", "CH1B", "CH2I", "CH2B")`, `names(dataColumns) = c("greenForeground","greenBackgrou` and `skip = 0` unless otherwise specified in the arguments.

Value

object of class `arrayData`

Author(s)

Andreas Buness <a.buness@dkfz.de>

See Also

`readpDataSlides`, `arrayData-class`

Examples

`readpDataSlides` *Reads the description of all slides*

Description

The function reads the `slideDescriptionFile`, which contains all information row-wise for each slide/hybridisation or for each channel in a tab-deliminated text file. The file is read up to the first line containing solely "white space".

Usage

```
readpDataSlides(slideDescriptionFile = "slideDescription.txt", loadPath = ".", deleteBlanks
```

Arguments

`slideDescriptionFile`	character string; required; default "slideDescription.txt". The first line of the file must contain all column names, in particular the column named `fileNameColum` and possibly additionally a column named `slideNameColumn`; cf. `readIntensities`.
`loadPath`	character string; required; default: ".". The path is used to load the `slideDescriptionFile`; note: "." refers to the working directory.
`deleteBlanks`	logical; required; default: TRUE. Any blank character is removed from the text body the `slideDescriptionFile`
`verbose`	logical; required; default: TRUE

Value

`data.frame`

Author(s)

Andreas Buness <a.buness@dkfz.de>

See Also

`readIntensities`, `processArrayData`

Examples

```
        LOADPATH <- file.path(.path.package("arrayMagic"), "extdata")
         SLIDEDESCRIPTIONFILE <- "slideDescription"

         resultObject <- readpDataSlides(
                              loadPath=LOADPATH,
                              slideDescriptionFile=SLIDEDESCRIPTIONFILE
                   )
```

`removeSpots` *Remove specified spots from objects*

Description

All elements matching the strings in `spotsToBeRemoved` are taken out of the object `arrayDataObject` and the corresponding ones out of the object `exprSetRGObject` as well, if the object is supplied.

Usage

```
removeSpots(arrayDataObject, exprSetRGObject=NULL, spotsToBeRemoved=NULL, spotIdentifier="Na
```

Arguments

`arrayDataObject`	object of type `arrayData`; required; default: missing
`exprSetRGObject`	object of type `exprSetRG`; optional; default: `NULL`
`spotIdentifier`	character string, i.e. name of the column of `getSpotAttr(arrayDataObject)` used for matching; default "Name"
`spotsToBeRemoved`	vector of character strings; default: `NULL`

Details

Value

A named list containing an object of type `arrayData` labelled "arrayDataObject", an object of type `exprSetRG` labelled "exprSetRGObject" and an integer specifying the number of removed items labelled "nrOfRemovedItems".

Author(s)

Andreas Buness <a.buness@dkfz.de>

See Also

`arrayData-class`, `exprSetRG-class`

Examples

```
intensities <- array(data=runif(600),dim=c(100,2,3))
dimnames(intensities) <- list(NULL, c("green","red"), NULL)
arrayDataObject <- new("arrayData", intensities=intensities, spotAttr=data.frame(Name=I(rep(c("x","y
res <- removeSpots(arrayDataObject, spotsToBeRemoved=c("x","z"))
stopifnot( dim(getIntensities(res[["arrayDataObject"]]))[1] == 3*20 )
```

`rowFtests` *F-test and t-test for rows of a matrix*

Description

F-test and t-test for rows of a matrix

Usage

```
rowFtests(x, fac)
rowttests(x, fac)
```

Arguments

`x`	Matrix
`fac`	Factor, with `length(fac)=ncol(x)`. For `rowttests`, `fac` must have exactly two levels.

Value

A list with the test statistics, p-values, degrees of freedom.

Author(s)

Wolfgang Huber <w.huber@dkfz.de>

See Also

`mt.teststat`

Examples

```
x  = matrix(runif(1e5), ncol=100)
k2 = floor(runif(ncol(x))*2)
k7 = floor(runif(ncol(x))*7)

tt = rowttests(x, factor(k2))
ft = rowFtests(x, factor(k7))

if(require(multtest)) {
  fs = mt.teststat(x, k7, test="f")
  stopifnot(all(abs(fs - ft$statistic) < 1e-6))

  ts = mt.teststat(x, k2, test="t.equalvar")
  stopifnot(all(abs(ts - tt$statistic) < 1e-6))
}
```

`savepng` *Save the contents of the current graphics device to a PDF or PNG file*

Description

Save the contents of the current graphics device to a PDF or PNG file

Usage

```
savepng(fn, dir, width=480, asp=1)
savepdf(fn, dir, width=6, asp=1)
```

Arguments

`fn`	character: name of the output file (without extension .png or .pdf)
`dir`	character: directory to which the file should be written. If `NULL`, use the filename in `fn` only.
`width`	numeric: width of the image in pixels (png) or inches (pdf)
`asp`	numeric: aspect ratio; height=width*asp

Details

Value

Character: name of the written file.

Author(s)

Wolfgang Huber `http://www.dkfz.de/mga/whuber`

See Also

`dev.copy`,`pdf`,`png`

Examples

```
  x = seq(0, 20*pi, len=1000)
  plot(x*sin(x), x*cos(x), type="l")

  try({   ## on some machines, png or pdf may not be available
  savepdf("spiral", dir=tempdir())
  ## Not run:
  savepng("spiral", dir=tempdir())

## End(Not run)
  })
```

`shorth` *shorth*

Description

Given a sample of real values, find the midpoint of the ”shorth”

Let f be the empirical distribution function, and w be the width of the shorth (typically 0.5). Find q that minimizes

$$f^{-1}(q+w) - f^{-1}(q)$$

Arguments

`z`

Details

Value

Author(s)

Examples

`simpleApply` *simpleApply*

Description

Note: very slow. Attention: Be careful with `funcResultDimensionality` ! `func` is applied to all subsets of `arrayObject` defined by `dimensions`, i.e. for every element i of `arrayObject`[`dimensions`] the function `func` is applied to `arrayObject`[i]. `func` must be unary. Due to the recursive definition the function might not only be slow but also very memory intensive. In simple situation this can give you more control as `apply` offers; cf. the examples.

Usage

```
simpleApply(arrayObject, dimensions, func, funcResultDimensionality, DEBUG=FALSE)
```

Arguments

`arrayObject`	array object
`dimensions`	increasing numeric vector
`func`	unary function
`funcResultDimensionality`	numeric (vector) specifying the dimensionality of the result of `func`
`DEBUG`	logical; required; default FALSE; to trace the recursive calling you may use DEBUG=TRUE

Value

an array of dim=c(dim(arrayObject[dimensions]), funcResultDimensionality); possibly use `aperm` to rearrange the dimensions

Author(s)

Andreas Buness <a.buness@dkfz.de>

Examples

```
a <- array(c(1:30),dim=c(3,2,5))
r <- simpleApply(a, 1, function(x){return(x[2,5])}, 1)
stopifnot( all(r == matrix(data=c(28:30))))

r <- simpleApply(a, 2, function(x){return(x[,])}, c(3,5))
stopifnot( all( a == aperm(r,c(2,1,3)) ) )

vec <- 1:10; dim(vec) <- c(10,1)
mat <- matrix(data=rep(1:10,4),nrow=10,ncol=4,byrow=FALSE)
r <- simpleApply(mat,1,function(y){return(mean(y))},1)
stopifnot(all(r==vec))

r <- simpleApply(mat, 1:2, function(x) return(x), 1)
stopifnot( all(r[,,1] == mat) )

r <- simpleApply(a, c(1,3) , function(x) return(x), dim(a)[2])
stopifnot( all(aperm(r[,,],c(1,3,2)) == a) )

r <- simpleApply(a, 1:2, function(x) return(x[2]), 1)
stopifnot( all(r[,,] == a[,,2]) )

r <- simpleApply(a, 1, function(x) return(x), c(dim(a)[2],dim(a)[3]))
stopifnot( all( r== a ) )
```

slideMerge *slideMerge*

Description

The mean of the expression values is calculated separately for each channel. If no `se.exprs` values are given in `exprSetRGObject`, `se.exprs` is set to the standard deviation (possibly `NA`). If available it is set to the root-mean-square of the given se.exprs values.

Usage

```
slideMerge(exprSetRGObject, slideMergeColumn, sampleAnnotationColumns, verbose=TRUE)
```

Arguments

`exprSetRGObject`	object of class exprSetRG; required; default missing
`slideMergeColumn`	character string specifying the variable of the `phenoData` object of the `exprSetRGObject` which is used to determine replicas; required; default missing
`sampleAnnotationColumns`	vector of character strings; optional; default missing; A vector which contains all `phenoData` variables relevant for further analysis. The annotation should be consistent for replicas. By default the argument `sampleAnnotationColumns` is missing and all `phenoData` variables are used.
`verbose`	logical; required; default: TRUE

Details

Value

modified `exprSetRGObject`

Author(s)

Andreas Buness <a.buness@dkfz.de>

See Also

`spotMerge`

Examples

```
indGreen=1:2
indRed=3:4
channels <- matrix( c(indGreen,indRed), nrow=length(indGreen), byrow=FALSE )
colnames(channels) <- c("green","red")
exprsMatrix <- matrix(rep(1:10,4),nrow=10,ncol=4,byrow=FALSE)
phenoMatrix <- matrix(c(c(1,2),c(3,3),c(5,5)),nrow=2,ncol=3,byrow=FALSE)
```

```
colnames(phenoMatrix) <- c("one","two","usedForMerge")
phenoMatrix <- rbind(phenoMatrix,phenoMatrix)
eSA <- new("exprSetRG", exprs=exprsMatrix, phenoData=
           new("phenoData", pData=data.frame(phenoMatrix),
               varLabels=as.list(colnames(phenoMatrix))),
           channels=channels)
eSM <- slideMerge(exprSetRGObject=eSA, slideMergeColumn="usedForMerge")
eSAOne <- slideSubset(eSA,j=c(1))
stopifnot( all(exprs(eSAOne) == exprs(eSM) ))
stopifnot( all( se.exprs(eSM) == 0 ) )
```

`spatialLayout` *spatialLayout*

Description

All `values` are mapped on a matrix representing the the spatial layout defined by `row`, `column` and `block` and possibly `numberOfValues`.

Usage

```
spatialLayout(value, row, col, block, numberOfValues, nrOfBlocksPerRow = 4, mapping = 0)
```

Arguments

`value`	numeric vector; required; default missing
`row`	integer vector; required; default missing
`col`	integer vector; required; default missing
`block`	integer vector; required; default missing
`numberOfValues`	integer; not required; the argument `numberOfValues` allows to determine the correct spatial layout if not all values for all positions are passed to the function; default missing
`nrOfBlocksPerRow`	integer; required; default 4
`mapping`	integer; either zero or one; default zero, which corresponds to "ScanAlyze" and "GenePix"; cf. source code

Value

A matrix representing the spatial layout of all `values`. The matrix is labelled as class `imageMatrix` and `matrix`.

Author(s)

Andreas Buness <a.buness@dkfz.de>

See Also

plot.imageMatrix

Examples

```
value <- rep(c(1,rep(0,49)),10)
block <- as.integer(gl(10,50))
col <- rep(c(1:10),50)
row <- rep(as.integer(gl(5,10)),10)
sL <- spatialLayout(value=value,row=row,col=col,block=block,nrOfBlocksPerRow=2)
plot.imageMatrix(sL)
value <- value[-(201:250)]
block <- block[-(201:250)]
col <- col[-(201:250)]
row <- row[-(201:250)]
sL <- spatialLayout(value=value,row=row,col=col,block=block,nrOfBlocksPerRow=2,numberOfValues=500)
plot.imageMatrix(sL)
```

spotMerge *spotMerge*

Description

The mean of replicate spots with identical number of replicates is calculated for each channel separately. (`spotMerge` does require an equal number of replicates) All other spots need to be eliminated with help of the argument `spotsToBeRemoved`. The mean is calculated for expression levels `exprs` in `exprSetRGObject` and for the `intensities` and `weights` in `arrayDataObject`. Existing `se.exprs` as part of `exprSetRGObject` is discared so far. The standard deviation of the spot merge operation for the expression values is returned as `se.exprs`. The the corresponding rows of `spotAttr` in `arrayDataObject` are concatenated and form a single row.

Usage

```
spotMerge(exprSetRGObject, arrayDataObject, spotIdentifier="Name", spotsToBeRemoved=c("Blan
```

Arguments

`arrayDataObject`	object of type arrayData required; default missing
`exprSetRGObject`	object of type exprSetRG required; default missing
`spotIdentifier`	character string; required; default "Name"
`spotsToBeRemoved`	vector of character strings, all spots which match the strings are removed from the analysis before merging; required; default: c("Blank")

Details

Value

A list containing modified `exprSetRGObject` and modified `arrayDataObject`; cf. Description.

Author(s)

Andreas Buness <a.buness@dkfz.de>

See Also

`processArrayData`, `slideMerge`, `exprSetRG-class`, `arrayData-class`

Examples

```
intensities <- array(data=runif(600),dim=c(100,2,3))
dimnames(intensities) <- list(NULL, c("green","red"), NULL)
spotAttr <- data.frame(Name=I(rep(c("x","y","k","l","z"),20)),
                       Zahl=rep(c(1,2,3,4,5),20),
                       Index=c(1:100))
arrayDataObject <- new("arrayData", intensities=intensities, weights=intensities[,1,],
                       spotAttr=spotAttr, hybAttrList=NULL)
indGreen=1:3
indRed=4:6
channels <- matrix( c(indGreen,indRed), nrow=length(indGreen), byrow=FALSE )
colnames(channels) <- c("green","red")
exprMatrix <- matrix(data=1,nrow=100,ncol=6,byrow=FALSE)
pD <- data.frame(matrix(0,nrow=6,ncol=1))
exprSetRGObject <- new("exprSetRG", exprs=exprMatrix, se.expr=exprMatrix,
                          phenoData=new("phenoData",
                            pData= pD,varLabels=list(rep("varLabel1",1))),
                          channels=channels)
resultList <- spotMerge(arrayDataObject=arrayDataObject,exprSetRGObject=exprSetRGObject, spotsToBe
resultExprSetRG <- resultList[["exprSetRGObject"]]
stopifnot( dim(exprs(resultExprSetRG))[1] == 3 )
stopifnot( dim(exprs(resultExprSetRG))[2] == 6 )
stopifnot(all(exprs(resultExprSetRG) == 1))
stopifnot((all(se.exprs(resultExprSetRG) == 0)))
resultArrayData <- resultList[["arrayDataObject"]]
stopifnot( all(getIntensities(resultArrayData)[,1,] == getWeights(resultArrayData)) )
nameColumns <- grep("Name", colnames(getSpotAttr(resultArrayData)))
zahlColumns <- grep("Zahl", colnames(getSpotAttr(resultArrayData)))
stopifnot( getSpotAttr(resultArrayData)[1,nameColumns] == rep ("y", 20 ) )
stopifnot( getSpotAttr(resultArrayData)[1,zahlColumns] == rep (2, 20 ) )
```

`visualiseHybridisations` *Microarray intensity data are visualised with respect to their spatial layout*

Description

Two colour microarray intensity data are visualised with respect to their spatial layout. Missing values (`NA`) as well as `-Inf` etc. are not visualised. They appear to be white, i.e. in the colour of the background. `-Inf` might be a result of the (log-)transformation of the data; cf. argument `transFunc`.

Usage

```
visualiseHybridisations(arrayDataObject, exprSetRGObject, type="raw", hybridisations, slideN
```

Arguments

`arrayDataObject`	object of class `arrayData`; required; default: missing
`exprSetRGObject`	object of class `exprSetRG`; required, if `type="normalised"` or codetype="normalisedLogRatio default: missing
`type`	vector of character strings; required; default "raw"; valid strings are "raw", "rawLogRatio", "rawLogRatioBackgroundSubtracted", "normalised" or "normalisedLogRatio"
`hybridisations`	vector of integers; optional; default: missing, i.e. all
`slideNameColumn`	character string which specifies a column of `getHybAttr(arrayDataObject)` which itself contains of the names of the hybridisations; optional; default: missing
`numberOfSpots`	integer; optional; default: missing; but required if the values in `arrayDataObject` do not contain all spots of the slide; cf. `spatialLayout`.
`nrOfBlocksPerRow`	integer; cf. `spatialLayout`; default: 4
`mappingColumns`	list; required; the list elements "Block", "Column", "Row" define the corresponding column names given in `getSpotAttr(arrayDataObject)`; the "Block" element is optional; default: list(Block="Block", Column="Column", Row="Row"); cf. `spatialLayout`
`transFunc`	transformation function applied to the raw data, in particular in combination with type == "raw", possibilities include for example log or rank; (note: type == "rawLogRatio" and type == "rawLogRatioBackgroundSubtracted" require log) required; default: log
`savePath`	character string; required; default: "."
`plotOutput`	character string; required; default: "pdf"

Value

None

Author(s)

Andreas Buness <a.buness@dkfz.de>

See Also

`exprSetRG-class`, `arrayData-class`

Examples

`visualiseQualityParameters` *Graphical representation of quality parameters*

Description

Graphical representation of quality parameters

Usage

```
visualiseQualityParameters(qualityParameters, savePath, fileName="visualiseQualityParameter
```

Arguments

`qualityParameters`	see `qualityParameters`
`fileName`	character string; required; default: "visualiseQualityParametersOutput.pdf"
`savePath`	character string; optional; default: missing
`plotOutput`	character string; required; either "screen" or "pdf"; default: "pdf"

Details

For details on the specific parameters see `qualityParameters`. Lines are only used for easier detection of outliers. The ordering of the hybridisation is somehow arbitrary, and only reflects the (initial) ordering. The "correlation" plot for example graphically represents the correlation coefficients calculated between the green and red channel for each hybridisation. The horizontal blue line is drawn at the height of one.

Value

A graphical representation of quality parameters.

Author(s)

Andreas Buness <a.buness@dkfz.de>

See Also

`qualityParameters`

Examples

`write.htmltable` *Write a data frame into an html table within a html page*

Description

Write a data frame into an html table

Usage

```
write.htmltable(x, filename, title="", sortby, decreasing=TRUE)
```

Arguments

`x`	data frame.
`filename`	file name.
`title`	title of html page.
`sortby`	name of column by which to sort the table rows.
`decreasing`	logical. Should the sort order be increasing or decreasing?

Details

Value

The function is called for its side effect: writing a file.

Author(s)

Wolfgang Huber `http://www.dkfz.de/mga/whuber`

See Also

Examples

`writeToFile` *Writes a textual representation of an exprSetRG and/or arrayData object*

Description

writeToFile generates a well-formated tab-deliminated output file of data stored in `exprSetRG` and/or `arrayData` objects. Different "views" on either a matching pair of an exprSetRG- and an arrayData-object or of a single object itself are possible, cf. argument `channels`. The slide or channel annotation, as well as the spot annotation is aligned to the data block if given. For further restriction and control of the output, e.g. slide subsets or ordering of slides, you may use the subsetting operations offered by the `exprSetRG-class` and `arrayData-class` beforehand (e.g. slideSubset and []).

Usage

```
writeToFile(arrayDataObject, exprSetRGObject, additionalDataMatrix, rowSelection, slideName
```

Arguments

`arrayDataObject`	arrayDataObject; optional/required, cf. `channels`; default missing
`exprSetRGObject`	exprSetRGObject; optional/required, cf. `channels`; default missing
`additionalDataMatrix`	class `matrix`; optional; default missing; colnames must be supplied
`rowSelection`	vector of indexes; optional; default missing
`slideNameColumn`	character string; optional; default missing; must refer to a valid column in `getHybAttr`(arrayDataObject) or in `pDataSlide`(exprSetRGobject)
`channels`	vector of character strings; by default c("logRatio"); valid character strings are: "logRatio", "green", "red", "rawGreen", "rawRed", "rawGreenRelative", "rawRedRelative", "se.exprsLogRatio", "se.exprsGreen", "se.exprsRed". "Raw"-types must not mix with not "raw"-types and vice versa and "logRatio"-types must not mix with "colour"-types. The "raw" types require the argument `arrayDataObject`, all other types requires at least the argument `exprSetRGObject`.
`fileName`	character string; required; default: "dumpedData.txt"
`savePath`	character string; required; default: "."
`fullOutput`	logical; adds phenoData information at the top of the table ; by default `TRUE`
`coding`	logical; adds integer-coded-phenoData information at the top to the table only if is `TRUE` and if `fullOutput` is `TRUE`; default: `FALSE`

Details

Value

The function is called for its side effect.

Author(s)

Andreas Buness <a.buness@dkfz.de>

See Also

`exprSetRG-class`, `arrayData-class`

Examples

Index