Ringo

April 19, 2010

`image.RGList` *Function to visualize spatial distribution of raw intensities*

Description

Function to visualize spatial distribution of raw intensities on NimbleGen Oligoarrays. Requires `RGList` with component `genes` complete with `genes$X` and `genes$X` coordinates of probes on array. `arrayImage` is a synonym of `image.RGList`.

Usage

```
## S3 method for class 'RGList':
image(x,arrayno,channel=c("red","green","logratio"),
         mycols=NULL, mybreaks=NULL, dim1="X", dim2="Y",
         ppch=20, pcex=0.3, verbose=TRUE, ...)
```

Arguments

`x`	object of class `RGList` containing red and green channel raw intensities; possibly result of `readNimblegen`.
`arrayno`	integer; which array to plot; one of `1:ncol(x$R)`
`channel`	character; which channel to plot, either `red`, `green` or the `logratio` `log2(red)-log2(green)`
`mycols`	vector of colors to use for image; if `NULL` defaults to `colorRampPalette(c("White", "Yellow", "Red"))(10)`
`mybreaks`	optional numeric vector of breaks to use as argument `breaks` in `image.default`; default `NULL` means take `length(mycols)+1` quantiles of the data as breaks.
`dim1`	string; which column of the 'genes' element of the supplied `RGList` indicates the first dimension of the reporter position on the microarray surface; for example this column is called 'X' with some NimbleGen arrays and 'Row' with some Agilent arrays.
`dim2`	string; which column of the 'genes' element of the supplied `RGList` indicates the second dimension of the reporter position on the microarray surface; for example this column is called 'Y' with some NimbleGen arrays and 'Col' with some Agilent arrays.
`ppch`	which symbol to use for intensities; passed on as `pch` to `points..default`

pcex	enlargement factor for intensity symbols; passed on as cex to points.default
verbose	logical; extended output to STDOUT?
...	further arguments passed on to plot.default and points.default

Value

invisibly returns NULL; function is called for its side effect, this is producing the plot

Author(s)

Joern Toedling

See Also

readNimblegen,plot.default,points

Examples

```
exDir <- system.file("exData",package="Ringo")
exRG <- readNimblegen("example_targets.txt","spottypes.txt",path=exDir)
image(exRG, 1, channel="red", mycols=c("black","darkred","red"))
## this example looks strange because the example data files only
## includes the probe intensities of probes mapped to the forward
## strand of chromosome 9.
## you can see these probes are distributed all over the array
```

---

asExprSet *converts a Ringo MAList into an ExpressionSet*

Description

Function to convert an object of class MAList into an object of class ExpressionSet. Note that the otherwise optional targets component is required in this case to generate the phenoData of the new ExpressionSet.

Usage

```
asExprSet(from, idColumn="PROBE_ID")
```

Arguments

from	object of class MAList to convert into an ExpressionSet
idColumn	string; indicating which column of the genes data.frame of the MAList holds the identifier for reporters on the microarray. This column, after calling make.names on it, will make up the unique featureNames of the resulting ExpressionSet.

Value

an object of class ExpressionSet

Note

There is a more general function for converting MALists to ExpressionSets in the package `convert`. This function here is solely intended for converting Ringo-generated `MALists` into `ExpressionSets`.

Author(s)

Joern Toedling

See Also

`ExpressionSet`, `preprocess`

Examples

```
exDir <- system.file("exData",package="Ringo")
exRG  <- readNimblegen("example_targets.txt","spottypes.txt",path=exDir)
exMA  <- preprocess(exRG, "none", returnMAList=TRUE)
exX   <- asExprSet(exMA)
```

---

autocor *Function to compute auto-correlation of probe intensities*

Description

Function to compute auto-correlation of probe intensities at specified offsets from the original positions.

Usage

```
autocor(x, probeAnno, chrom, samples = NULL, lag.max = 2000,
        lag.step = 100, cor.method = "pearson",
        channel = c("red","green","logratio"),
        idColumn = "ID", verbose = TRUE)
```

Arguments

`x`	an object either of class `ExpressionSet` containing the normalized probe intensities or of class `RGList` containing the raw intensities.
`probeAnno`	Object of class `probeAnno` holding chromosomal match positions and indices of reporters in data matrix.
`chrom`	character; chromosome to compute the autocorrelation for
`samples`	which samples of the data to use; if more than 1 for each probe the mean intensity over these samples is taken.
`lag.max`	integer; maximal offset from the original position, the auto-correlation is to be computed for.
`lag.step`	integer; step size of lags between 0 and maximal lag.
`cor.method`	character; which type of correlation to compute, translates to argument `method` of function `cor`
`channel`	character; in case `x` is an `RGList`, which channel to plot, either `red`, `green` or the `logratio` `log2(red)-log2(green)`

idColumn	string; indicating which column of the genes data.frame of the RGList holds the identifier for reporters on the microarray. Character entries of the index elements of the probeAnno will be matched against these identifiers. If the index elements of the probeAnno are numeric or x is of class ExpressionSet, this argument will be ignored.
verbose	logical; extended output to STDOUT

Details

the lags, i.e. the offsets from the original position, the auto-correlation is to be computed for, are constructed from the given arguments as `seq(0, lag.max, by=lag.step)`.

Value

Object of class `autocor.result`, a numeric vector of auto-correlation values at the offsets specified in argument `lags`. The lag values are stored as the `names` of the vector. Argument `chrom` is stored as attribute `chromosome` of the result.

Author(s)

Joern Toedling

See Also

`cor`,`plot.autocor.result`

Examples

```
exDir <- system.file("exData",package="Ringo")
load(file.path(exDir,"exampleProbeAnno.rda"))
load(file.path(exDir,"exampleX.rda"))
exAc <- autocor(exampleX, probeAnno=exProbeAnno,
                chrom="9", lag.max=1000)
plot(exAc)
```

---

cherByThreshold *Function to identify chers based on thresholds*

Description

Given a vector of probe positions on the chromosome, a vector of smoothed intensities on these positions, and a threshold for intensities to indicated enrichment, this function identifies *Cher*s (ChIP-enriched regions) on this chromosome.

This function is called by the function `findChersOnSmoothed`.

Usage

```
cherByThreshold(positions, scores, threshold, distCutOff,
  minProbesInRow = 3)
```

Arguments

positions	numeric vector of genomic positions of probes
scores	scores (intensities) of probes on those positions
threshold	threshold for scores to be called a cher
distCutOff	maximal positional distance between two probes to be part of the same cher
minProbesInRow	integer; minimum number of enriched probes required for a cher; see details for further explanation.

Details

Specifying a minimum number of probes for a cher (argument minProbesInRow) guarantees that a cher is supported by a reasonable number of measurements in probe-sparse regions. For example, if there's only one enriched probe within a certain genomic 1kb region and no other probes can been mapped to that region, this single probe does arguably not provide enough evidence for calling this genomic region enriched.

Value

A LIST with n components, where the first n components are the cher clusters, each one holding the scores and, as their names, the genomic positions of probes in that cluster.

Author(s)

Joern Toedling

See Also

findChersOnSmoothed

Examples

```
## example with random generated data:
rpos <- cumsum(round(runif(200)*5))
rsco <- rnorm(200)+0.2
plot(rpos, rsco, type="l", col="seagreen3", lwd=2)
rug(rpos, side=1, lwd=2); abline(h=0, lty=2)
rchers <- cherByThreshold(rpos, rsco, threshold=0, distCutOff=2)
sapply(rchers[-length(rchers)], function(thisClust){
 points(x=as.numeric(names(thisClust)), y=thisClust, type="h", lwd=2,
col="gold")})
```

---

cher-class *Class "cher" - ChIP-enriched region*

Description

An object of class cher (ChIP-enriched region) holds characteristics of an enriched genomic region from ChIP chip data.

Objects from the Class

Objects can be created by calls of the form `new("cher", name, chromosome, start, end, cellType, antibody, maxLevel, score, probes, extras, ...)`.

Slots

`name`: character vector of length 1 unequivocally describing the cher, e.g. "Suz12.Nudt2.upstream.cher"

`chromosome`: character vector of length one, naming the chromosome of the region, e.g. "9"

`start`: `integer`, region start position on the chromosome, e.g. 34318900

`end`: `integer`, region end position on the chromosome, e.g. 34320100

`cellType`: `character` vector describing the cell type the ChIP chip experiment has been done on, e.g. "HeLa" or "human"

`antibody`: `character` vector describing the antibody or characteristic for which fragments were supposedly enriched in immuno-precipitation step, e.g. "Suz12" for the protein Suz12

`maxLevel`: `numeric`, maximal (smoothed) probe level in the cher, e.g. 2.00

`score`: `numeric` of a cher score, currently we use the sum of smoothed probe levels (log fold changes), e.g. 69.16

`probes`: `vector` of probe identifiers of all probes with match positions in the `cher`

`extras`: `list` of further elements used to annotate the `cher`; examples of such that are used in `Ringo` are:

- **typeUpstream** optional character vector of features that this `cher` is located upstream of, e.g. the transcriptional start site of "ENST00000379158". See `relateChers` for details.
- **typeInside** optional character vector of features that this `cher` is located inside of
- **distMid2TSS** optional named numeric vector of distances of the `cher`'s middle position to features, e.g. TSSs of features upstream and inside; names are the features to which the distances are given; only meaningful in combination with `typeUpstream` and `typeInside`; e.g. 55 with name "ENST00000379158"
- **upSymbol** optional character vector of gene symbols of features the cher is located upstream of; supplements `typeUpstream`; e.g. "Nudt2"
- **inSymbol** optional character vector of gene symbols of features the cher is located upstream of; supplements `typeInside`.
- **...** further list elements can be added using the `update` method.

Methods

initialize create a new `cher`; see section `examples` below

plot calls `chipAlongChrom` to plot the `cher`; see `plot.cher` for more details

update signature(`cher`,...); updates elements of the `cher` object; The further arguments in '...' are interpreted. Arguments corresponding to defined slot names of the `cher` result in the value by that slot being replaced by the specified value for the argument; argument names that do not correspond to slot names of the object result in list elements of the `extras` list of the `cher` being replaced by the given values for these arguments or the values are appended to the current `extras` list and the argument names make up the list names of the appended arguments. See section `examples` below for an example how to use this method.

cellType obtain or replace the description of the cell type, the ChIP-enriched regions was found in with this antibody

probes obtain the vector of probes involved in a ChIP-enriched region

cherList

A list in which each element is of class `cher`, is called a `cherList`. This class, however, is rarely used (yet).

Note

The `cher` class used to be an S3 list before.

The term 'cher' is shorthand for 'ChIP-enriched region'. We think this term is more appropriate than the term 'peak' commonly used in ChIP-chip context. Within such regions the actual signal could show two or more actual signal peaks or none at all (long plateau).

Author(s)

Joern Toedling, Tammo Krueger

See Also

`plot.cher`, `findChersOnSmoothed`, `relateChers`

Examples

```
## how to create a cher object from scratch
cherNudt2 <- new("cher", name="nudt2.cher", chromosome=9,
                 start=34318954, end=34319944, antibody="Suz12",
                 maxLevel=2.00, score=69.2, upSymbol="NUDT2")
                 #extras=list(upSymbol="NUDT2"))
cherNudt2
str(cherNudt2)

## use the update method (note:this update is biologically meaningless)
cher2 <- update(cherNudt2, cellType="HeLa", downSymbol="P53",
                probes=c("probe1","probe2"))
cher2; str(cher2)

## plot a cher object
exDir <- system.file("exData",package="Ringo")
load(file.path(exDir,"exampleProbeAnno.rda"))
load(file.path(exDir,"exampleX.rda"))
smoothX <- computeRunningMedians(exampleX, probeAnno=exProbeAnno,
     modColumn = "Cy5", allChr = "9", winHalfSize = 400)
plot(cherNudt2, smoothX, probeAnno=exProbeAnno, gff=exGFF, extent=5000)
```

---

chipAlongChrom *Visualize ChIP intensities along the chromosome*

Description

This function can visualize the array intensities from a ChIP chip experiment for a chromosomal region or the whole chromosome. It's based on the `plotAlongChrom` function from the package `tilingArray`, but provides a different visualization.

Usage

```
## S4 method for signature 'ExpressionSet,probeAnno':
plot(x, y, ...)

chipAlongChrom(eSet,  probeAnno, chrom, xlim, ylim,
 samples = NULL, paletteName = "Set2", colPal = NULL,
 ylab = "Fold change [log]", ipch = 16, ilwd = 3, ilty = 1,
 icex = 3, gff = NULL,
 featureExclude=c("chromosome", "nucleotide_match","insertion"),
 zeroLine = TRUE, sampleLegend = TRUE, sampleLegendPos = "topleft",
 featureLegend = FALSE, maxInterDistance = 200,  coord = NULL,
 highlight, main, ...)
```

Arguments

eSet	An expression set containing the (normalized) ChIP intensities, e.g. the result objects from functions preprocess and computeRunningMedians.
x	Corresponds to argument eSet when calling the S4 method
probeAnno	An object of class probeAnno holding genomic position, index and gene association of probes on array.
y	Corresponds to argument probeAnno when calling the S4 method
chrom	character; the chromosome to visualize
xlim	start and end genomic coordinates on the chromosome to visualize
ylim	minimum and maximum probe intensities for the plot, if missing (default) set to range(exprs(eSet))
samples	numeric; which samples from the eSet are to be shown. Default is to show all samples in the eSet,
paletteName	character; Name of the RColorBrewer palette to use for sample colors. If the number of samples is greater than the palette size, random colors are taken.
colPal	vector of colors to use for the sample intensities. This is alternative to the argument paletteName for specifying which colors to use.
ylab	character; label for the y-axis, passed on to the plotting function as ylab
ipch	plot character to use
icex	character expansion to use for plotting symbol
ilwd	line width of plotted data lines
ilty	line type of plotted data lines; passed on to par(lty).
gff	data frame containing annotation for genomic feature to be used to further annotate the plot.
featureExclude	character vector specifying the feature types in the data.frame gff that should not be shown in the plot
zeroLine	logical; should a dashed horizontal line at y=0 be put into the plot?
sampleLegend	logical; should a sample legend be put into the plot?
sampleLegendPos	character; where to put the sample legend; one of 'topleft' (default), 'bottomleft', 'topright', or 'bottomrigth'

featureLegend
: logical; should a feature legend be put beneath the plot?

maxInterDistance
: numeric; only used when itype is either "r" or "u"; specifies the maximal distance up to which adjacent probe positions should be connected by a line.

coord
: optional integer of length 2; can be used instead of xlim to specify the start and end coordinates of the genomic region to plot

highlight
: optional list specifying a genomic region to be highlighted in the shown plot

main
: optional main title for the plot; if not specified: the default is 'Chromosome coordinate [bp]'

...
: further parameters passed on to grid.polyline and grid.points

Value

invisible list of probe positions (element x) and probe levels (element y) in the selected genomic region.

Note

The S4 method is provided as a mere convenience wrapper.

When plotting a new 'grid' plot in an active x11 window that already contains a plot, remember to call grid.newpage() before.

Author(s)

Joern Toedling <joern.toedling@curie.fr>

See Also

ExpressionSet-class, probeAnno-class, grid.points, plotAlongChrom in package tilingArray

Examples

```
### load data
ringoExampleDir <- system.file("exData",package="Ringo")
load(file.path(ringoExampleDir,"exampleProbeAnno.rda"))
load(file.path(ringoExampleDir,"exampleX.rda"))

### show a gene that is well represented on this microarray
plot(exampleX, exProbeAnno, chrom="9",
     xlim=c(34318000,34321000), ylim=c(-2,4), gff=exGFF)

### this should give you the same result as:
chipAlongChrom(exampleX, chrom="9", xlim=c(34318000,34321000),
               ylim=c(-2,4), probeAnno=exProbeAnno, gff=exGFF)
```

compute.gc *Compute the GC content of DNA and probe sequences*

Description

Simple auxiliary function to compute the GC content of a given set of DNA sequences, such as microarray probe sequences.

Usage

```
compute.gc(probe.sequences, digits = 2)
```

Arguments

probe.sequences	character vector of DNA or probe sequences of which the GC content is to be computed
digits	integer specifying the desired precision

Value

a numeric vector with sequence-wise GC contents; the names of this vector are the names of the supplied probe.sequences.

Author(s)

Joern Toedling

See Also

Function basecontent in package matchprobes for a more general function to compute base occurrence in sequences

Examples

```
ex.seqs <- c("gattaca", "GGGNTT", "ggAtT", "tata","gcccg")
names(ex.seqs) <- paste("sequence",1:5,sep="")
compute.gc(ex.seqs)
```

computeRunningMedians
Function to compute running medians on a tiling expression set

Description

Function to compute running medians (or other quantiles) on a tiling expression set.

Usage

```
computeRunningMedians(xSet, probeAnno, modColumn = "Cy5",
 allChr, winHalfSize = 400, min.probes = 5, quant = 0.5,
 combineReplicates = FALSE, nameSuffix = ".sm", checkUnique=TRUE,
 uniqueCodes=c(0), verbose = TRUE)
```

Arguments

xSet	Object of class ExpressionSet holding the normalized probe intensity data
probeAnno	Environment holding the genomic positions of probes in the ExpressionSet
modColumn	Column of the ExpressionSet's phenoData holding the samples' difference of interest
allChr	Character vector of all chromosomes in genome; if not specified (defaul) all chromosomes annotated in the supplied probeAnno are used.
winHalfSize	Half the size of the window centered at a probe position, in which all other probes contribute to the calculation of the median.
min.probes	integer; if less probes are in the sliding window, NA instead of the median is returned. This meant to avoid to computing non-meaningful medians. If unwanted, set this to 1 or less
quant	numeric; which quantile to use for the smoothing. The default 0.5 means compute the median over the values in the sliding window.
combineReplicates	logical; should the median not be computed over individual samples in the ExpressionSet, but should samples be combined according to the column modColumn of the phenoData. The median is then computed across all probe levels and samples of the same type in the window. The resulting ExpressionSet has so many columns as are there different entries in the column modColumn
nameSuffix	character; suffix attached to the sample labels of the supplied ExpressionSet xSet for the sample names of the resulting ExpressionSet.
checkUnique	logical; indicates whether the uniqueness indicator of probe matches from the probeAnno environment should be used.
uniqueCodes	numeric; which numeric codes in the chromosome-wise match-uniqueness elements of the probeAnno environment indicate uniqueness?
verbose	logical; detailed progress output to STDOUT?

Value

An object of class ExpressionSet, holding smoothed intensity values for the probes of the supplied ExpressionSet. The number of results samples is the number of levels in the supplied modColumn of the supplied ExpressionSet's phenoData.

Author(s)

Joern Toedling <joern.toedling@curie.fr>

See Also

ExpressionSet, sliding.quantile, probeAnno-class

Examples

```
exDir <- system.file("exData",package="Ringo")
load(file.path(exDir,"exampleProbeAnno.rda"))
load(file.path(exDir,"exampleX.rda"))
smoothX <- computeRunningMedians(exampleX, probeAnno=exProbeAnno,
                                 winHalfSize = 400)
combX <- combine(exampleX, smoothX)
if (interactive()){
 grid.newpage()
 plot(combX, exProbeAnno, chrom="9", xlim=c(34318000,34321000),
      ylim=c(-2,4), gff=exGFF)
}
```

---

computeSlidingT *Function to compute sliding T statistics on a tiling expression set*

Description

Function to compute sliding (regularized) one- or two-sample T statistics on a tiling expression set.

Usage

```
computeSlidingT(xSet, probeAnno, allChr = c(1:19, "X", "Y"), test = "one.sample"
```

Arguments

xSet	Object of class ExpressionSet holding the normalized probe intensity data
probeAnno	Environment holding the genomic positions of probes in the ExpressionSet
allChr	Character vector of all chromosomes in genome
test	character; one of one.sample or two.sample
grouping	factor vector of length equal to number of samples, currently not required
winHalfSize	Half the size of the window centered at a probe position, in which all other probes contribute to the calculation of the mean and standard deviation.
min.probes	integer; if less probes are in the sliding window, NA instead of the mean and sd is returned. This is meant to avoid to computing non-meaningful means and standard deviations. If unwanted, set this to 1 or less
checkUnique	logical; indicates whether the uniqueness indicator of probe matches from the probeAnno environment should be used.
uniqueCodes	numeric; which numeric codes in the chromosome-wise match-uniqueness elements of the probeAnno environment indicate uniqueness?
verbose	logical; detailed progress output to STDOUT?

Value

An object of class ExpressionSet, holding the T statistics values for the probes of the supplied ExpressionSet. The number of results samples is the number of levels in the supplied factor grouping.

Note

the option `two.sample` is not implemented yet

Author(s)

Joern Toedling

See Also

`sliding.meansd`

Examples

```
exDir <- system.file("exData",package="Ringo")
load(file.path(exDir,"exampleProbeAnno.rda"))
load(file.path(exDir,"exampleX.rda"))
tX <- computeSlidingT(exampleX, probeAnno=exProbeAnno,
           allChr=c("9"), winHalfSize=400)
sampleNames(tX) <- "t-Stat_Suz12vsTotal"
if (interactive()){
  grid.newpage()
  plot(cbind2(exampleX, tX), exProbeAnno, chrom="9",
       xlim=c(34318000,34321000), ylim=c(-2,8.5), gff=exGFF,
       paletteName="Paired")
}
```

---

corPlot *Function to plot correlation of different samples*

Description

This function can be used to visualize the (rank) correlation in expression data between different samples or sample groups.

Usage

```
corPlot(eset, samples = NULL, grouping = NULL, ref = NULL,
   useSmoothScatter = TRUE, ...)
```

Arguments

`eset`	object of class `ExpressionSet` holding the array data, or a numeric matrix instead
`samples`	which samples' expression shall be correlated to each other; either a numeric vector of sample numbers in the `ExpressionSet` or a character vector that must be contained in the `sampleNames` of the `ExpressionSet`, default `NULL` means take all samples in the `ExpressionSet`
`grouping`	an optional factor vector defining if the correlation should be assessed between groups of samples, rather than individual samples. If two or more samples are assigned into the same group, the mean over these samples' expression values is taken before computing correlation. Default NULL means assess correlation between individual samples only.

ref	reference than only applies if argument grouping is given; see relevel
useSmoothScatter	logical; should the function smoothScatter be used? given; see relevel
...	additional arguments, not used yet

Value

No useful return. The function is called for its side-effect to produce the pairs plot.

Author(s)

Joern Toedling

See Also

ExpressionSet, relevel, pairs, smoothScatter

Examples

```
data(sample.ExpressionSet)
if (interactive())
corPlot(sample.ExpressionSet,
 grouping=paste(sample.ExpressionSet$sex,
 sample.ExpressionSet$type, sep="."))
```

---

exportCherList *Function to export cherList into a file*

Description

Function to export cherList into a file of gff or BED format. This files can be imported as tracks into genome browsers.

Usage

```
exportCherList(object, filename = "chers.gff", format = "gff3", ...)
```

Arguments

object	an object of class cherList
filename	character; path to file to be written
format	Format of exported file; currently only "gff3" and "bed" are supported
...	further arguments to be passed on to the trackSet method

Details

First converts the cherList into an object of class trackSet from package **rtracklayer** and then calls the export method as defined for a trackSet.

Value

returns invisible NULL; called for the side effect of writing the file filename.

Author(s)

Joern Toedling

See Also

Class `trackset` in package **rtracklayer**

Examples

```
## Not run:
  exDir <- system.file("exData",package="Ringo")
  load(file.path(exDir,"exampleProbeAnno.rda"))
  load(file.path(exDir,"exampleX.rda"))
  smoothX <- computeRunningMedians(exampleX, probeAnno=exProbeAnno,
       modColumn = "Cy5", allChr = "9", winHalfSize = 400)
  chersX <- findChersOnSmoothed(smoothX, probeAnno=exProbeAnno,
       thresholds=0.45, allChr="9", distCutOff=600, cellType="human")
  exportCherList(chersX, file="chers.gff")

## End(Not run)
```

`extractProbeAnno` *Build probeAnno from match positions in an RGList*

Description

This function can be used to build a probeAnno object from the reporter match positions given in the 'genes' slot of an RGList if present, as is the case with some ChIP-chip microarray platforms, e.g. with certain Agilent ones after reading in the data with `read.maimages(...,"agilent")`.

Usage

```
extractProbeAnno(object, format = "agilent", ...)
```

Arguments

`object`	an object that holds the data and the probe match positions, currently can only be of class `RGList`
`format`	in which format are the reporter match positions stored in the object; see details; currently only "agilent" is implemented
`...`	further arguments that are passed on to the function `posToProbeAnno`

Details

Which information is used for creating the probeAnno is specified by the argument `format`.

agilent expects that the object is of class `RGList`. The 'genes' element of the object is taken. This element is expected to have at least a column 'ProbeName', which stores the unique reporter/probe identifiers, and a column 'SystematicName', which holds the probe match position in the format "chr<chromosome>:coordinate1-coordinate2", e.g."chr1:087354051-087354110".

Value

An object of class `probeAnno` holding the mapping between reporters and genomic positions.

Author(s)

Joern Toedling

See Also

`posToProbeAnno`, `probeAnno-class`

`features2Probes` *Function for mapping genomic features to probes*

Description

This function creates a mapping between annotated genomic features and probes on the array whose matching genomic positions are stored in a probeAnno environment.

Usage

```
features2Probes(gff, probeAnno, upstream = 5000, checkUnique = TRUE, uniqueCodes
```

Arguments

`gff`	`data.frame` holding genomic feature annotation
`probeAnno`	Object of class `environment` holding the genomic positions of probes in the ExpressionSet
`upstream`	up to how many bases upstream of annotated genomic features should probes be counted as related to that feature (see details)
`checkUnique`	logical; indicates whether the uniqueness indicator of probe matches from the probeAnno environment should be used.
`uniqueCodes`	numeric; which numeric codes in the chromosome-wise match-uniqueness elements of the probeAnno environment indicate uniqueness?
`mem.limit`	integer value; what is the maximal allowed size of matrices during the computation; see `regionOverlap`
`verbose`	logical; detailed progress output to STDOUT?

Value

The results is a list of length equal to the number of rows in the provided `gff`, the data.frame of genomic features. The `names` of the list are the names specified in the `gff`. Each element of the list is specified by the probes mapping into the genomic region from `upstream` bases upstream of the feature's start site to the feature's end site. The entries itself are either `NULL`, if no probe was mapped into this region, or a named numeric vector, with its values being the distances of the probes' middle positions to the feature's start site (which depends on the strand the feature is on) and its names being the identifiers of these probes.

Note

This resulting mapping is not used excessively by other Ringo functions, so creating this mapping is optional at this time, but it may simplify subsequent gene/transcript-based analyses.

Here, the term *feature* describes a genomic entity such as a gene, transcript, non-coding RNA or a similar feature annotated to a genome. It does NOT refer to oligo-nucleotide or cDNA probes on the microarray.

Author(s)

Joern Toedling

See Also

regionOverlap

Examples

```
ringoExampleDir <- system.file("exData",package="Ringo")
load(file.path(ringoExampleDir,"exampleProbeAnno.rda"))
trans2Probe <- features2Probes(exGFF, exProbeAnno)
trans2Probe[exGFF$name[match("NUDT2", exGFF$symbol)]]
exGFF[match(names(trans2Probe)[listLen(trans2Probe)>0],exGFF$name),]
trans2Probe[listLen(trans2Probe)==1]
```

---

findChersOnSmoothed

Find ChIP-enriched regions on smoothed ExpressionSet

Description

Given an ExpressionSet of smoothed probe intensities, an environment with the mapping of probes to chromosomes, and a vector of thresholds for calling genomic sites enriched, this function finds the 'chers' (ChIP-enriched regions) consisting of enriched genomic positions, with probes mapped to them. 'Adjacent' enriched positions are condensed into a single Cher.

Usage

```
findChersOnSmoothed(smoothedX, probeAnno, thresholds, allChr = NULL,
   distCutOff = 600, minProbesInRow = 3, cellType = NULL,
   antibodyColumn=NULL, checkUnique = TRUE, uniqueCodes = c(0),
   verbose = TRUE)
```

Arguments

smoothedX	Object of class ExpressionSet holding the smoothed probe intensities, e.g. the result of function computeRunningMedians.
probeAnno	environment containing the probe to genome mapping
thresholds	numeric vector of threshold above which smoothed probe intensities are considered to correspond to enriched probes. The vector has to be of length equal the number of samples in smoothedX, with a single threshold for each sample.

`allChr` character vector of all chromosomes on which enriched regions are sought. Every chromosome here has to have probes mapped to it in the `probeAnno` environment. By default (`NULL`) the `chromosomeNames` of the probeAnno object are used.

`distCutOff` integer; maximum amount of base pairs at which enriched probes are condensed into one Cher.

`minProbesInRow` integer; minimum number of enriched probes required for a Cher; see `details` for further explanation.

`cellType` character; name of cell type the data comes from, is either a. of length one indicating the column of `pData(smoothedX)` that holds the cell type OR b. of length one indicating the common cell type for all samples in the `ExpressionSet` OR c. of length equal to `ncol(smoothedX)` specifying the cell type of each sample individually.

`antibodyColumn` the name or number of the column of the `pData(smoothedX)` that holds the description of the antibody used for each sample. This information is used to annotate found ChIP-enriched regions accordingly. If `NULL` (default), the `sampleNames` of `smoothedX` are used.

`checkUnique` logical; indicates whether the uniqueness indicator of probe matches from the probeAnno environment should be used.

`uniqueCodes` numeric; which numeric codes in the chromosome-wise match-uniqueness elements of the probeAnno environment indicate uniqueness?

`verbose` logical; extended output to STDOUT?

Details

Specifying a minimum number of probes for a Cher (argument `minProbesInRow`) guarantees that a Cher is supported by a reasonable number of measurements in probe-sparse regions. For example, if there's only one enriched probe within a certain genomic 1kb region and no other probes can been mapped to that region, this single probe does arguably not provide enough evidence for calling this genomic region enriched.

Value

A list of class `cherList`, holding objects of class `cher` that were found on the supplied data.

Author(s)

Joern Toedling

See Also

`cherByThreshold`,`computeRunningMedians`, `relateChers`

Examples

```
exDir <- system.file("exData",package="Ringo")
load(file.path(exDir,"exampleProbeAnno.rda"))
load(file.path(exDir,"exampleX.rda"))
smoothX <- computeRunningMedians(exampleX, probeAnno=exProbeAnno,
     modColumn = "Cy5", allChr = "9", winHalfSize = 400)
```

```
chersX <- findChersOnSmoothed(smoothX, probeAnno=exProbeAnno,
     thresholds=0.45, allChr="9", distCutOff=600, cellType="human")
if (interactive())
  plot(chersX[[1]], smoothX, probeAnno=exProbeAnno, gff=exGFF)
chersX <- relateChers(chersX, exGFF)
as.data.frame.cherList(chersX)
```

`ftr2xys` *Convert a NimbleScan ftr-file into a xys-file*

Description

Auxiliary function to convert a NimbleScan `feature-report file` into a xys-file that can be used with the function `read.xysfiles` of package `oligo`.

Usage

```
ftr2xys(ftr.file, path=getwd())
```

Arguments

`ftr.file`	character; file path of feature report file to convert into an xys file
`path`	file path to directory where the xys-file should be written to; defaults to the current working directory

Details

The output file is names as the input ftr file; with the file extension `.ftr` replaced by `.xys`.

Value

Function returns only `NULL` invisibly and is only called for its side effect to write the xys-file into the current working directory.

Note

This function should only be used with one-color Nimblegen microarrays and when the correct xys-file of the raw data is not available. The output file can be used with the function `read.xysfiles` of package `oligo`.

Author(s)

Joern Toedling

Examples

```
 ## Not run:
   sapply(list.files(pattern=".ftr$"),ftr2xys)
   library(oligo)
   fs = read.xysfiles(list.xysfiles())

## End(Not run)
```

getFeats *Utility function to extract all features from a cherList*

Description

This is a small utility function for extracting all related features from a *cherList*, a list of *ChIP-enriched regions*.

Usage

```
getFeats(cl)
```

Arguments

cl	object of class cherList, a list of cher objects

Value

a character vector containing the names of all features that were associated to any ChIP-enriched region in the list before, using the function relateChers

Author(s)

Joern Toedling

See Also

relateChers,cher-class

nonzero-methods *Methods for Function nonzero*

Description

Auxiliary functions to retrieve the indices of non-zero elements in sparse matrices.

Value

A two-column matrix. Each row gives the row and column index of a non-zero element in the supplied matrix x.

Methods

x = "dgCMatrix" returns the indices of non-zero elements in matrices of class dgCMatrix

x = "matrix.csr" returns the indices of non-zero elements in matrices of class matrix.csr

x = "matrix" returns the indices of non-zero elements in matrices of base class matrix; equivalent to which(x != 0, arr.ind=TRUE)

Note

Originally we used the `matrix.csr` class from SparseM, but we have switched to the class `dgCMatrix` from package Matrix, as that package is part of the R distribution bundle now.

The idea is to have a function similar to `which(x != 0, arr.ind=TRUE)` if `x` is a matrix.

See Also

`dgCMatrix-class`

Examples

```
(A <- matrix(c(0,0,0,0,0,1,0,0,0,0,0,0,0,0,-34),
             nrow=5, byrow=TRUE))
str(A.dgc <- as(A, "dgCMatrix"))
nonzero(A.dgc)
A2.dgc <- Matrix::cBind(A.dgc, A.dgc)
as.matrix(A2.dgc)
nonzero(A2.dgc)
```

---

plot.autocor.result

Plots auto-correlation of probe intensities

Description

Function to plot the auto-correlation of probe intensities computed by function `autocor`.

Usage

```
## S3 method for class 'autocor.result':
plot(x,
plot.title = "ChIP: Autocorrelation of Intensities", ...)
```

Arguments

`x`	an object of class `autocor.result`, the result of function `autocor`
`plot.title`	`main` title of the plot
`...`	further arguments passed on to `plot.default`, see details

Details

The following arguments to `plot.default` are already defined in the function and thus cannot be specified by the user as further arguments in `...`: `type, lwd, xlab, ylab, col`. Argument `main` is specified in `plot.title`.

Value

invisible `NULL`

Author(s)

Joern Toedling

See Also

autocor

Examples

```
## see the help page of 'autocor' for an example
```

---

plotBM *Visualization of a binary matrix*

Description

This function produces simple, heatmap-like visualizations of binary matrices.

Usage

```
plotBM(x, boxCol = "darkblue", reorder = FALSE, frame = TRUE, ...)
```

Arguments

x	Binary matrix to visualize
boxCol	Color to use for boxes of '1's
reorder	logical; states whether the rows shall be reordered according to the size of the category
frame	logical; states whether a frame should be drawn around the visualization. In contrast to the frame drawn in plot.default, there is no gap between the visualization and this frame.
...	further arguments passed on to plot.default

Details

For reordering, each row is interpreted as a binary matrix, for example a row z=(1,0,0,1) would be interpreted as the binary number 1001 = 9 in the decimal system. Rows are then reordered by the frequency of each binary number with the rows that correspond to the most frequent binary number shown at the top in the visualization.

Value

The function invisibly returns the (reordered) matrix x, but its mainly called for its side effect of producing the visualization.

Note

An alternative way to display such matrices are given by heatmap or, the simpler version thereof, image. However, image files produced with this functions tend to be very large. This function uses plot.default and polygon which results in much smaller file sizes and is sufficient for binary matrices.

Author(s)

Joern Toedling

See Also

polygon,colors

Examples

```
A <- matrix(round(runif(80)), ncol=4, byrow=TRUE)
dimnames(A)=list(letters[seq(nrow(A))],
                 as.character(as.roman(seq(ncol(A)))))
show(A)
plotBM(A, reorder=FALSE)
plotBM(A, reorder=TRUE)
```

---

plot.cher *Plot identified Chers*

Description

Function for plotting identified *Cher*s (ChIP-enriched regions).

Usage

```
## S4 method for signature 'cher,ExpressionSet':
plot(x, y, probeAnno, samples=NULL, extent = 1000, gff = NULL, ...)
```

Arguments

x	object of class cher
y	data object of class ExpressionSet that was used for function findChersOnSmoothed
probeAnno	object of class probeAnno holding the reporter/probe to genome mappings
samples	which samples to plot, either a numeric vector of entries in 1 to ncol(dat), or character vector with entries in sampleNames(dat) or NULL meaning plot the levels from all samples in the ExpressionSet
extent	integer; how many base pairs to the left and right should the plotted genomic region be extended
gff	data frame with gene/transcript annotation
...	further arguments passed on to function chipAlongChrom

Value

called for generating the plot; invisible(NULL)

Author(s)

Joern Toedling

See Also

chipAlongChrom, cher-class

`posToProbeAnno` *Function for creating a probeAnno environment*

Description

This function allows the user to create a probeAnno environment that holds the mapping between probes on the array and their genomic match position(s). As input, the function takes either a.) one of NimbleGen's POS file or a similar file that holds the mapping of probes to the genome. OR b.) a `data.frame` holding this information

Usage

```
posToProbeAnno(pos, chrNameColumn = "CHROMOSOME",
   probeColumn = "PROBE_ID", chrPositionColumn = "POSITION",
   lengthColumn = "LENGTH", sep="\t", genome="unknown",
   microarrayPlatform="unknown", verbose = TRUE, ...)
```

Arguments

`pos`	either a file-name that specifies the path to the POS or other mapping file OR a data.frame holding the mapping
`chrNameColumn`	name of the column in the file or `data.frame` that holds the chromosome name of the match
`probeColumn`	name of the column that holds the matching probe's unique identifier
`chrPositionColumn`	name of the column that holds the match genomic position/coordinate on the chromosome
`lengthColumn`	name of the column that holds the length of the match position, in case of perfect match should correspond to the sequence length of the probe
`sep`	string; denotes the seperator between elements in the supplied mappings file `pos`; passed on to function `scan`; ignored if `pos` is not a filename.
`genome`	string; denotes genome (and assembly) the reporters have been mapped to for this probeAnno object, e.g. "M. musculus (mm9)"
`microarrayPlatform`	string; denotes the commercial or custom microarray platform/design that holds the reporters whose mapping is stored in this probeAnno object, e.g. "NimbleGen MOD SUZ12"
`verbose`	logical; should progress be written to STDOUT?
`...`	further arguments passed on to function `scan`, which is used for reading in the file `pos`.

Details

The default column names correspond to the column names in a NimbleGen POS file.

For custom mappings, using the tools Exonerate, BLAT or MUMmer, the scripts directory of this package holds Perl scripts to generate such a POS file from the respective output files.

Value

The results is an object of class `probeAnno`.

Author(s)

Joern Toedling

See Also

`probeAnno-class`, `scan`

Examples

```
exPos <- read.delim(file.path(system.file("exData",package="Ringo"),
                    "MOD_2003-12-05_SUZ12_1in2.pos"),
                    header=TRUE,as.is=TRUE)
str(exPos)
exProbeAnno <- posToProbeAnno(exPos,
    genome="M. musculus (assembly mm8)",
    microarrayPlatform="NimbleGen 2005-06-17_Ren_MM5Tiling_Set1")
## is equivalent to
exProbeAnno2 <- posToProbeAnno(file.path(
   system.file("exData",package="Ringo"),"MOD_2003-12-05_SUZ12_1in2.pos"),
   genome="M. musculus (assembly mm8)",
   microarrayPlatform="NimbleGen 2005-06-17_Ren_MM5Tiling_Set1")
ls(exProbeAnno)
chromosomeNames(exProbeAnno2)
```

`preprocess` *Preprocess Raw ChIP-chip Intensities*

Description

Calls one of various (`limma`) functions to transform raw probe intensities into (background-corrected) normalized log ratios (M-values).

Usage

```
preprocess(myRG, method="vsn", ChIPChannel="R", inputChannel="G",
           returnMAList=FALSE, idColumn="PROBE_ID", verbose=TRUE, ...)
```

Arguments

`myRG`	object of class `RGList`
`method`	string; denoting which normalization method to choose, see below for details
`ChIPChannel`	string; which element of the `RGList` holds the ChIP result, see details
`inputChannel`	string; which element of the `RGList` holds the untreated *input* sample; see details
`returnMAList`	logical; should an MAList object be returned? Default is to return an ExpressionSet object.

idColumn	string; indicating which column of the genes data.frame of the RGList holds the identifier for reporters on the microarray. This column, after calling make.names on it, will make up the unique featureNames of the resulting ExpressionSet. If argument returnMAList is TRUE, this argument is ignored.
verbose	logical; progress output to STDOUT?
...	further arguments to be passed on normalizeWithinArrays and normalizeBetweenArray

Details

The procedure and called limma functions depend on the choice of method.

loess Calls normalizeWithinArrays with method="loess".

vsn Calls normalizeBetweenArrays with method="vsn".

Gquantile Calls normalizeBetweenArrays with method="Gquantile".

Rquantile Calls normalizeBetweenArrays with method="Rquantile".

median Calls normalizeWithinArrays with method="median".

nimblegen Scaling procedure used by Nimblegen. Yields scaled log-ratios by a two step procedure: srat = log2(R) - log2(G) srat = srat - tukey.biweight(srat)

Gvsn Learns vsn model on green channel intensities only and applies that transformation to both channels before computing fold changes.

Rvsn Learns vsn model on red channel intensities only and applies that transformation to both channels before computing fold changes.

none No normalization of probe intensities, takes raw log2(R)-log2(G) as component M and (log2(R)+log2(G))/2 as component A; uses normalizeWithinArrays with method="none".

Mostly with two-color ChIP-chip, the ChIP sample is marked with the red Cy5 dye and for the untreated *input* sample the green Cy3 dye is used. In that case the RGListmyRG's element R holds the ChIP data, and element G holds the input data. If this is not the case with your data, use the arguments ChIPChannel and inputChannel to specify the respective elements of myRG.

Value

Returns normalized, transformed values as an object of class ExpressionList or MAList.

Note

Since Ringo version 1.5.6, this function does not call limma's function backgroundCorrect directly any longer. If wanted by the user, background correction should be indicated as additional arguments passed on to normalizeWithinArrays or normalizeBetweenArrays, or alternatively call backgroundCorrect on the RGList before preprocessing.

Author(s)

Joern Toedling

See Also

normalizeWithinArrays, normalizeBetweenArrays, malist, ExpressionSet, vsnMatrix

Examples

```
exDir <- system.file("exData",package="Ringo")
exRG <- readNimblegen("example_targets.txt","spottypes.txt",
                      path=exDir)
exampleX <- preprocess(exRG)
sampleNames(exampleX) <- make.names(paste(exRG$targets$Cy5,"vs",
                                    exRG$targets$Cy3,sep="_"))
print(exampleX)
### compare VSN to NimbleGen's tukey-biweight scaling
exampleX.NG <- preprocess(exRG, method="nimblegen")
sampleNames(exampleX.NG) <- sampleNames(exampleX)
if (interactive())
  corPlot(cbind(exprs(exampleX),exprs(exampleX.NG)),
    grouping=c("VSN normalized","Tukey-biweight scaled"))
```

---

probeAnno-class *Class "probeAnno"*

Description

A class that holds the mapping between reporters/probes on a microarray and their genomic match position(s) in a chosen genome.

Objects from the Class

Objects can be created by calls of the form `new("probeAnno", map, arrayName, genome)`.

Slots

`map`**:** Object of class `"environment"` This map consists of four vectors for each chromosome/strand, namely, say for chromosome 1:

1.start genomic start coordinates of probe matches on chromosome 1

1.end genomic start coordinates of probe matches on chromosome 1

1.index identifier of probes matching at these coordinates

1.unique vector of the same length as the three before; encoding how many matches the corresponding probe has in the given file or `data.frame`. An entry of '0' indicates that the probe matching at this position has only this one match.

`arrayName`**:** Object of class `"character"`, the name or identifier of the microarray design, e.g. `2005-06-17_Ren_MM5Tiling_Set1`

`genome`**:** Object of class `"character"`, which genome the reporters have been mapped to

Methods

arrayName obtain the microarray platform name

arrayName<- replace the microarray platform name

[get elements from the `map` environment

[<- assign elements to the `map` environment

chromosomeNames obtain a character vector holding the names of the chromosomes for which the probeAnno objects holds a mapping.

get get elements from the `map` environment

initialize create mew probeAnno object

ls list elements of the `map` environment

genome obtain the description of the genome the reporters were mapped to

genome<- replace the description of the genome the reporters were mapped to

as signature(from="environment"); function to coerce old-style 'probeAnno' environments to new-style 'probeAnno' objects. Simply creates a new object with the old environment in its `map` slot

Note

'probeAnno' objects used to be environments and still are used as such in package `tilingArray`

Author(s)

Joern Toedling <`joern.toedling@curie.fr`>; Wolfgang Huber

See Also

posToProbeAnno

Examples

```
pa <- new("probeAnno")
pa["X.start"] <- seq(5000,10000,by=1000)
if (interactive()) show(pa)
pa2 <- posToProbeAnno(file.path(system.file("exData",package="Ringo"),
                                "MOD_2003-12-05_SUZ12_1in2.pos"))
arrayName(pa2) <- "NimbleGen MOD_2003-12-05_SUZ12_1in2"
genome(pa2) <- "H. sapiens (hg18)"
show(pa2)
head(pa2["9.start"])
```

---

quantilesOverPositions

show ChIP-chip data aligned over genome features, e.g. TSSs

Description

Function to show the ChIP-chip data aligned over certain genome features, for example transcription start sites (TSSs).

Usage

```
quantilesOverPositions(xSet, selGenes, g2p,
                       positions = seq(-5000, 10000, by = 250),
                       quantiles = c(0.1, 0.5, 0.9))
```

Arguments

xSet	an ExpressionSet holding the ChIP-chip data
selGenes	character; vector of genome features, e.g. transcripts, to use for the plot
g2p	A list object containing the mapping between genome positions and probes on the microarray. Created with the function features2Probes.
positions	Numeric vector of positions related to the coordinates of the genome features, such as in which distances of the TSS the values should be computed over the aligned data
quantiles	numeric; which quantiles to compute over the aligned data

Value

An object of class qop, which can be visualized by its plot method.

Author(s)

Joern Toedling

See Also

features2Probes, qop-class

Examples

```
ringoExampleDir <- system.file("exData",package="Ringo")
load(file.path(ringoExampleDir,"exampleProbeAnno.rda"))
trans2Probe <- features2Probes(exGFF, exProbeAnno)
load(file.path(ringoExampleDir,"exampleX.rda"))
exampleSX <- computeRunningMedians(exampleX, probeAnno=exProbeAnno,
   modColumn = "Cy5", allChr = "9", winHalfSize = 400)
exampleC <- findChersOnSmoothed(exampleSX, probeAnno=exProbeAnno,
   thresholds=0.2, allChr="9", distCutOff=600, cellType="human")
exampleC <- relateChers(exampleC, exGFF)
exampleQop <- quantilesOverPositions(exampleSX,
   selGenes=getFeats(exampleC), quantiles=c(0.5, 0.9),
   g2p=trans2Probe, positions=seq(-4000, 1000, by=250))
show(exampleQop)
plot(exampleQop, ylim=c(-0.5, 2.1))
```

---

readNimblegen *Function to read in Nimblegen Intensity Text Files*

Description

Function to read in Nimblegen Intensity Text Files into an RGList. Calls some other functions for actual reading of data. This function is to be used with two-color NimbleGen array data. Use the function read.xysfiles of the oligo package for single-color data.

Usage

```
readNimblegen(hybesFile, spotTypesFile, path = getwd(),
 headerPattern="# software=NimbleScan",verbose = TRUE, ...)
```

Arguments

hybesFile	Name of the file describing the arrays. In limma this file would be called targets file.
spotTypesFile	spot types also used by limma
path	Path to directoy that hold the files hybesFile, spotTypesFile and also the intensity files. Set this to NULL if you prefer the arguments hybesFile, spotTypesFile and the file-name entries of the hybes file to be treated as absolute or relative file paths themselves.
headerPattern	string; pattern used to identify explantory header lines in the supplied pair-format files
verbose	logical; progress output to STDOUT?
...	further arguments passed on the readNgIntensitiesTxt

Value

Returns raw intensity values in form of an RGList.

Author(s)

Joern Toedling

See Also

rglist, readTargets

Examples

```
exDir <- system.file("exData",package="Ringo")
exRG <- readNimblegen("example_targets.txt","spottypes.txt",path=exDir)
print(exRG)
```

---

regionOverlap *Function to compute overlap of genomic regions*

Description

Given two data frames of genomic regions, this function computes the base-pair overlap, if any, between every pair of regions from the two lists.

Usage

```
regionOverlap(xdf, ydf, chrColumn = "chr", startColumn = "start",
              endColumn = "end", mem.limit=1e8)
```

Arguments

xdf	data.frame that holds the first set of genomic regions
ydf	data.frame that holds the first set of genomic regions
chrColumn	character; what is the name of the column that holds the chromosome name of the regions in xdf and ydf
startColumn	character; what is the name of the column that holds the start position of the regions in xdf and ydf
endColumn	character; what is the name of the column that holds the start position of the regions in xdf and ydf
mem.limit	integer value; what is the maximal allowed size of matrices during the computation

Value

Originally, a matrix with nrow(xdf) rows and nrow(ydf) columns, in which entry X[i,j] specifies the length of the overlap between region i of the first list (xdf) and region j of the second list (ydf). Since this matrix is very sparse, we use the dgCMatrix representation from the Matrix package for it.

Note

The function only return the absolute length of overlapping regions in base-pairs. It does not return the position of the overlap or the fraction of region 1 and/or region 2 that overlaps the other regions.

The argument mem.limit is not really a limit to used RAM, but rather the maximal size of matrices that should be allowed during the computation. If larger matrices would arise, the second regions list is split into parts and the overlap with the first list is computed for each part. During computation, matrices of size nrow(xdf) times nrow(ydf) are created.

Author(s)

Joern Toedling

See Also

dgCMatrix-class

Examples

```
  ## toy example:
  regionsH3ac <- data.frame(chr=c("chr1","chr7","chr8","chr1","chrX","chr8"), start=c(100
  regionsH4ac <- data.frame(chr=c("chr1","chr2","chr7","chr8","chr9"),
start=c(500,100,50,80,100), end=c(700, 200, 250, 120,200))

  ## compare the regions first by eye
  ##  which ones do overlap and by what amount?
  regionsH3ac
  regionsH4ac

  ## compare it to the result:
  as.matrix(regionOverlap(regionsH3ac, regionsH4ac))
  nonzero(regionOverlap(regionsH3ac, regionsH4ac))
```

`relateChers` *Relate found Chers to genomic features*

Description

This function relates found 'cher's (ChIP-enriched regions) to annotated genomic features, such as transcripts.

Usage

```
relateChers(pl, gff, upstream = 5000, verbose = TRUE)
```

Arguments

`pl`	Object of class `cherList`
`gff`	`data.frame` holding genomic feature annotation
`upstream`	up to how many bases upstream of annotated genomic features should chers be counted as related to that feature (see details)
`verbose`	logical; extended output to STDOUT?

Details

chers will be counted as related to genomic features, if

- their middle position is located between start and end position of the feature
- their middle position is located not more than argument `upstream` bases upstream of the feature start

.

One can visualize such cher-feature relations as a graph using the Bioconductor package `Rgraphviz`. See the script 'graphChers2Transcripts.R' in Ringo's scripts directory for an example.

Value

An object of class `cherList` with for each cher the elements `typeUpstream` and `typeInside` filled in with the names of the features that have been related to.

Author(s)

Joern Toedling

Examples

```
# see findChersOnSmoothed for an example
```

`sliding.meansd` *Compute mean and standard deviation of scores in a sliding window*

Description

This functions is used to slide a window of specified size over scores at given positions. Computed is the mean and standard deviation over the scores in the window.

Usage

```
sliding.meansd(positions, scores, half.width)
```

Arguments

`positions`	numeric; sorted vector of (genomic) positions of scores
`scores`	numeric; scores to be smoothed associated to the `positions`
`half.width`	numeric, half the window size of the sliding window

Value

Matrix with three columns:

`mean`	means over scores in running window centered at the positions that were specified in argument `positions`.
`sd`	standard deviations over scores in running window centered at the positions that were specified in argument `positions`.
`count`	number of points that were considered for computing the mean and standard deviation at each position

Author(s)

Joern Toedling and Oleg Sklyar

See Also

`sliding.quantile`

Examples

```
set.seed(123)
sampleSize <- 10
ap <- cumsum(1+round(runif(sampleSize)*10))
as <- c(rnorm(floor(sampleSize/3)),
        rnorm(ceiling(sampleSize/3),mean=1.5),
        rnorm(floor(sampleSize/3)))
sliding.meansd(ap, as, 20)
ap
mean(as[1:3])
sd(as[1:3])
```

`sliding.quantile` *Compute quantile of scores in a sliding window*

Description

This functions is used to slide a window of specified size over scores at given positions. Computed is the quantile over the scores in the window.

Usage

```
sliding.quantile(positions, scores, half.width, prob = 0.5,
                 return.counts = TRUE)
```

Arguments

`positions`	numeric; sorted vector of (genomic) positions of scores
`scores`	numeric; scores to be smoothed associated to the `positions`
`half.width`	numeric, half the window size of the sliding window
`prob`	numeric specifying which quantile is to be computed over the scores in the window; default 0.5 means compute the median over the scores.
`return.counts`	logical; should the number of points, e.g. probes, that were used for computing the median in each sliding window also returned?

Value

Matrix with two columns:

`quantile`	medians over running window centered at the positions that were specified in argument `positions`.
`count`	number of points that were considered for computing the median at each position

These positions are given as `row.names` of the resulting vector. If argument `return.counts` is `FALSE`, only a vector of the medians is returned, with the positions as `names`.

Author(s)

Oleg Sklyar and Joern Toedling

See Also

`quantile`

Examples

```
sampleSize <- 1000
ap <- cumsum(1+round(runif(sampleSize)*10))
as <- c(rnorm(floor(sampleSize/3)),
        rnorm(ceiling(sampleSize/3),mean=1.5),
        rnorm(floor(sampleSize/3)))
arm <- sliding.quantile(ap, as, 20)
arq <- sliding.quantile(ap, as, 20, prob=0.25)
```

```
plot(ap, as, pch=20, xlab="position",ylab="level")
points(ap, arm[,1], type="l", col="red", lwd=2)
points(ap, arq[,1], type="l", col="green", lwd=2)
legend(x="topleft", legend=c("median","1st quartile"),
       col=c("red","green"), lty=1, lwd=2)
```

`twoGaussiansNull` *Estimate a threshold from Gaussian mixture distribution*

Description

Function to estimate a threshold from Gaussian mixture distribution. The data is assumed to follow a mixture of two Gaussian distributions. The one Gaussian with the lower mean value is assumed to be the null distribution and probe levels are assigned p-values based on this null distribution. The threshold is then the minimal data value with an adjusted p-value smaller than a specified threshold.

Usage

```
twoGaussiansNull(x, p.adj.method = "BY", max.adj.p = 0.1, var.equal = FALSE, ...
```

Arguments

`x`	numeric vector of data values
`p.adj.method`	method for adjusting the p-values for multiple testing; must be one of `p.adjust.methods`
`max.adj.p`	which adjusted p-value to use as upper limit for estimating the threshold
`var.equal`	logical; is the variance of the two Gaussians assumed to be equal or different
`...`	further arguments passed on to function `Mclust`

Details

This function uses the package `mclust` to fit a mixture of two Gaussians to the data. The threshold is then estimated from the fitted Gaussian with the lower mean value.

Value

Single numeric value. The threshold that is the minimal data value with an adjusted p-value smaller than a specified threshold.

Note

Thanks to Richard Bourgon for pointing out the necessity of providing this method as an alternative way of estimating the threshold.

Author(s)

Joern Toedling, Aleksandra Pekowska

See Also

`mclust`, `p.adjust`

Examples

```
exDir <- system.file("exData",package="Ringo")
load(file.path(exDir,"exampleProbeAnno.rda"))
load(file.path(exDir,"exampleX.rda"))
smoothX <- computeRunningMedians(exampleX, probeAnno=exProbeAnno,
   modColumn = "Cy5", allChr = "9", winHalfSize = 400)

## compare the two different ways of estimating the threshold
y0a <- apply(exprs(smoothX), 2, upperBoundNull)
y0b <- apply(exprs(smoothX), 2, twoGaussiansNull)

hist(exprs(smoothX)[,1], n=10, main=NA,
     xlab="Smoothed expression level [log2]")
abline(v=c(y0a, y0b), col=c("blue","orange"), lwd=2)
legend(x="topright", col=c("blue","orange"), lwd=2,
       legend=c(expression(paste(y[0]," Non-parametric")),
                expression(paste(y[0]," Gaussian"))))
```

---

upperBoundNull *function to estimate upper limit of null distribution*

Description

The data is assumed to arise from a mixture of two distributions, a symmetric null distribution with its mode close to zero, and an alternative distribution that is stochastically larger than the null. This function tries to pinpoint the minimum of data values that are more likely to arise from the alternative distribution, i.e. an upper bound for values following the null distribution.

Usage

```
upperBoundNull(x, modeMethod = "shorth", limits = c(-1, 1), prob = 0.99, ...)
```

Arguments

`x`	numeric vector of data values
`modeMethod`	character string; which method to use for estimating the mode of the null distribution; see details
`limits`	numeric of length 2; data values within this range are used for estimating the mode of the null distribution
`prob`	quantile of the null distribution that will be used as an upper bound
`...`	additional arguments that are passed on to the function for mode estimation

Details

For estimating the mode of the null distribution, current options are

"shorth" the function `shorth`

"half.range.mode" the function `half.range.mode`

"null" does not estimate the mode from the data, but sets it to 0

Value

a single numeric value which is the estimated upper bound for the null distribution.

Note

This way of estimating the null distribution is mentioned in the PhD thesis of Richard Bourgon.

Author(s)

Joern Toedling, based on suggestions by Richard Bourgon

See Also

shorth, half.range.mode

Examples

```
exDir <- system.file("exData",package="Ringo")
load(file.path(exDir,"exampleProbeAnno.rda"))
load(file.path(exDir,"exampleX.rda"))
smoothX <- computeRunningMedians(exampleX, probeAnno=exProbeAnno,
     modColumn = "Cy5", allChr = "9", winHalfSize = 400)
apply(exprs(smoothX), 2, upperBoundNull)
```

Index