beadarray

November 11, 2009

R topics documented:

`ArrayMask` *Array Mask*

Description

Functions to edit or display array masks.

Usage

```
addArrayMask(BLData, array, SAM = FALSE, nrow = 50, ncol = 50, high = "red",
        low = "yellow", zlim = c(7,15), override = FALSE)
removeArrayMask(BLData, array, SAM = FALSE, nrow = 50, ncol = 50, high = "red",
        low = "yellow", zlim = c(7,15), override = FALSE)
showArrayMask(BLData,array, SAM = FALSE, elim = TRUE, override = FALSE)
clearArrayMask(BLData,array)
```

Arguments

`BLData`	A BeadLevelList object.
`array`	The number of an array in the BeadLevelList object.
`SAM`	Logical. If `TRUE`, display a hexagonal overlay where appropriate.
`elim`	Logical. If TRUE, plot eliminated beads (with blue crosses).
`nrow, ncol, high, low, zlim`	Arguments passed to `imageplot` - see the help for `imageplot` for details.

override — Logical. Plotting a large mask can cause slowdown problems. By default, if more than 200 000 beads are masked, the current mask will not be plotted. You can force the mask to be plotted by setting this argument to TRUE, however beware as this may cause slower systems to freeze.

Details

These functions are used to manipulate the mask on a single array.

addArrayMask adds beads in a specified region to the mask (i.e. sets their weights to 0).

removeArrayMask removes beads in a specified region from the mask (i.e. sets their weights to 0)

On calling either of these functions, an imageplot is displayed. Click on the plot to define vertices of a polygon, in order. Having specified the last vertex, right-click to close the polygon. A plot is then produced of the beads currently masked on the array (in grey) and the polygon just defined (in red), with a menu prompting you to accept the displayed region - if you do so, then all beads in the polygon then be masked or unmasked as appropriate. Alternatively, you can right-click on the image without defining any vertices, thus leaving the mask unchanged.

If making the change would result in all beads of a certain probe ID being completely covered by the mask, then the functions return a warning message, and the beads eliminated in this way are highlighted with blue crosses on the plot.

showArrayMask plots the beads on an array which have been masked, over a plot of outliers.

clearArrayMask clears the mask on an array, removing all weights associated with it.

Value

None returned

Author(s)

Jonathan Cairns

See Also

listEliminatedProbes

Examples

```
##data(BLData)
##addArrayMask(BLData,1)
##showArrayMask(BLData,1)
```

---

arrayNames — *Gets the strip/array names from a BeadLevelList Object*

Description

Retrieves the strip/array names from a BeadLevelList object.

Usage

```
arrayNames(object, arrays=NULL)
```

Arguments

object	BeadLevelList
arrays	integer (scalar or vector) specifying the strips/arrays to retrieve the names of. When NULL the names of all strips/arrays are returned.

Details

arrayNames retrieves the name of the strip(s)/array(s) from the arrayInfo slot.

Value

A character vector containing the names of the individual strips(s)/array(s).

Author(s)

Matt Ritchie

Examples

```
data(BLData)
arrayNames(BLData)
```

---

backgroundControlPlot

QA measures based on bead-level negative controls

Description

Function for plotting the bead-level intensities for all the negative controls that are placed on an array. Typically there are around a thousand of these controls, each replicated 30 times. The sequences used for these controls should not target any part of the genome and therefore we should not observe any signal.

Usage

```
backgroundControlPlot(BLData, array = 1, plot = FALSE)
```

Arguments

BLData	A BeadLevelList for an Illumina expression chip
array	The number of the array in BLData that we want QA of.
plot	if TRUE a diagnostic plot will be produced

Details

For QA, we report the mean and variance of all negative controls (of all bead-types) after first removing outiers using a 3 MAD cut-off. To retrieve the IDs of the negative controls, we make use of the annotation slot stored with the BeadLevelList object. It is therefore important that this information is accurate. A plot of all negative control bead-types can also be produced, where each bead-type is represented by a vertical line covering the inter-quartile range and ordered according to mean intensity. Too many high intensity values for the negatives could indicate a poor quality array.

Value

The function returns the mean (AveNeg) and variance (VarNeg) of all negative control beads and a diagnositc plot if requested.

Author(s)

Mark Dunning and Andy Lynch

See Also

`calculateBeadLevelScores`

---

backgroundCorrect *Background correct a BeadLevelList object*

Description

Adapted from the 'limma' backgroundCorrect function to correct the foreground intensities of a `BeadLevelList` object using the background values.

Usage

```
backgroundCorrect(object, method = "subtract", offset = 0, verbose = FALSE)
```

Arguments

`object`	a BeadLevelList object
`method`	character string specifying correction method. Possible values are '"none"', '"subtract"', '"half"', '"minimum"', '"edwards"', '"normexp"', '"rma"')
`offset`	numeric value to add to the intensities
`verbose`	logical. Used when method = '"normexp"'. If TRUE, the parameters estimated by the model are output.

Details

Below is an excerpt from the 'limma' backgroundCorrect man page:

If 'method="none"' then the corrected intensities are equal to the foreground intensities, i.e., the background intensities are treated as zero. If 'method="subtract"' then this function simply subtracts the background intensities from the foreground intensities which is the usual background correction method.

The remaining methods are all designed to produce positive corrected intensities. If 'method="half"' then any intensity which is less than 0.5 after background subtraction is reset to be equal to 0.5. If 'method="minimum"' then any intensity which is zero or negative after background subtraction is set equal to half the minimum of the positive corrected intensities for that array. If 'method="edwards"' the method of Edwards (2003) is used. If 'method="normexp"' or '"rma"', a normal-exponential convolution model is fitted to the intensities, using different estimation procedures. See Smyth (2005) for further details on normexp.

The 'offset' can be used to add a constant to the intensities before log-transforming, so that the log-ratios are shrunk towards zero at the lower intensities. This may eliminate or reverse the usual 'fanning' of log-ratios at low intensities associated with local background subtraction.

End of excerpt.

As a result of both having identical function names this function can conflict with the backgroundCorrect method in 'limma'. If both packages are loaded, the function from whichever package was loaded last takes precedence. If the 'beadarray' backgroundCorrect() function is masking that from 'limma', one can directly call the 'limma' method using the command "limma::backgroundCorrect()". Alternatively, one can detach the 'beadarray' package using "detach(package:beadarray)". Similar techniques can be used if 'limma' is masking the 'beadarray' method.

Value

A BeadLevelList object in which the 'G' (and 'R', if present) intensities for each array are background corrected. Note that the 'Gb' (and 'Rb' intensities) are not removed.

Author(s)

Mark Dunning and Mike Smith based on the limma function

References

Edwards, D. E. (2003). Non-linear normalization and background correction in one-channel cDNA microarray studies, Bioinformatics, 19, 825-833.

Smyth, G. K. (2005). Limma: linear models for microarray data. In: Bioinformatics and Computational Biology Solutions using R and Bioconductor, R. Gentleman, V. Carey, S. Dudoit, R. Irizarry, W. Huber (eds.), Springer, New York, pages 397-420.

Examples

```
data(BLData)

#default is to simply subtract Rb from R
BLData.bc = backgroundCorrect(BLData)

#Use 'minimum' method to stop negative values appearing
BLData.min = backgroundCorrect(BLData, method="minimum")
```

BASHCompact *BASH - Compact Defect Analysis*

Description

Creates a list of probes marked as being in compact defects.

Usage

```
BASHCompact(BLData, array, neighbours = NULL, log = TRUE, maxiter = 10, cutoff =
```

Arguments

BLData	BeadLevelList
array	integer specifying which strip/array to plot
neighbours	A Neighbours matrix. Optional - if left NULL, it will be computed.
log	Logical - If TRUE, find outliers on the log scale.
maxiter	Integer - Maximum number of iterations.
cutoff	Integer - Size a cluster must be to be labelled a compact defect.
cinvasions	Integer - Number of invasions used when closing the image.
...	Additional arguments to be passed to findAllOutliers (e.g. what = "R")

Details

BASHCompact finds "compact defects" on an array. A compact defect is defined as a large connected cluster of outliers.

This function first finds the outliers on an array. This is done via the function findAllOutliers.

Next, using the Neighbours matrix and a Flood Fill algorithm, it determines which beads are in large connected clusters of outliers (of size larger than cutoff). These beads are then temporarily removed and the process repeated with the remaining beads. The repetition continues until either no large clusters of outliers remain, or until we have repeated the process maxiter times (and in this case, a warning will be given). In this way, we obtain a list of defective probes.

Finally, we "close" the image, to fill in small gaps in the defect image. This consists of a "dilation" and an "erosion". In the dilation, we expand the defect image, by adding beads adjacent to defective beads into the defect image. This is repeated cinvasions times. In the erosion, we contract the defect image, by removing beads adjacent to non-defective beads from the defect image. (Erosion of the defect image is equivalent to a dilation of the non-defective image.)

Value

A vector consisting of the BeadIDs of beads labelled as compact defects.

Author(s)

Jonathan Cairns

References

Mayte Suarez-Farinas, Maurizio Pellegrino, Knut M. Wittkwosky and Marcelo O. Magnasco (2007).
Harshlight: A "corrective make-up" program for microarray chips. R package version 1.8.0. http://asterion.rockefeller.edu

See Also

BASHDiffuse, generateE, generateNeighbours,

Examples

```
data(BLData)
o <- BASHCompact(BLData, 1)
o <- BASHCompact(BLData, 1, cinvasions = 10) ##increased no of closure invasions
o <- BASHCompact(BLData, 1, cutoff = 12) ##only larger defects will be found with this se
```

BASHDiffuse *BASH - Diffuse Defect Analysis*

Description

Creates a list of probes marked as being in diffuse defects.

Usage

```
BASHDiffuse(BLData, array, neighbours = NULL, E = NULL, n = 3, compact = NULL, s
```

Arguments

BLData	BeadLevelList
array	integer specifying which strip/array to plot
neighbours	A Neighbours matrix. Optional - if left NULL, it will be computed, using default generateNeighbours settings.
E	Numerical vector - The error image to use. Optional - if left blank, it will be computed, using generateE using bgfilter = "median".
n	Specify a cut-off for outliers as n median absolute deviations (MADs) from the median. The default value is 3
compact	Vector - Optional. BeadIDs of beads in compact defects to remove from the analysis.
sig	Numerical - Significance level of binomial test.
invasions	Integer - Number of invasions to use to find the kernel (see below).
cutoff	Integer - Size a cluster must be to be labelled a diffuse defect.
cinvasions	Integer - Number of invasions used when closing the image.
twotail	Logical - If TRUE, then we analyse positive and negative outliers separately, and then combine the diffuse defect images at the end.

Details

BASHDiffuse finds "diffuse defects" on an array. A diffuse defect is defined as a region containing an unusually large number of (not necessarily connected) outliers.

Firstly, we consider the error image E, and find outlier beads on this image. Outliers for a particular bead type are determined using a 3 MAD cut-off from the median.

We now consider an area around each bead (known as the "kernel"). The kernel is found by an invasion process using the neighbours matrix - we choose the beads which can be reached from the central bead in cinvasions steps.

We count how many beads are in the kernel, and how many of these are marked as outliers. Using a binomial test, we work out if there are significantly more outliers in the kernel than would be expected if the outliers were equally distributed over the entire array. If so, then the central bead is marked as a diffuse defect.

Lastly, we run a clustering algorithm and a closing algorithm similar to those in BASHCompact.

Value

A vector consisting of the BeadIDs of beads considered diffuse defects.

Author(s)

Jonathan Cairns

References

Mayte Suarez-Farinas, Maurizio Pellegrino, Knut M. Wittkwosky and Marcelo O. Magnasco (2007).
Harshlight: A "corrective make-up" program for microarray chips. R package version 1.8.0. http://asterion.rockefeller.edu

See Also

BASHCompact, generateE, generateNeighbours,

Examples

```
data(BLData)
o <- BASHDiffuse(BLData, 1)
o <- BASHDiffuse(BLData, 1, sig = 0.00001) ##stricter significance value, perhaps more us
o <- BASHDiffuse(BLData, 1, cutoff = 12) ##only larger defects will be found with this se
```

BASHExtended *BASH - Extended Defect Analysis*

Description

Returns a score, which assesses the extent to which the background is changing across the array/strip.

Usage

```
BASHExtended(BLData, array, neighbours = NULL, E = NULL, E.BG = NULL)
```

Arguments

BLData	BeadLevelList
array	integer specifying which strip/array to plot
neighbours	A Neighbours matrix. Optional - if left NULL, it will be computed, using default generateNeighbours settings.
E	Numerical vector - The error image to use. Optional - if left blank, it will be computed, using generateE (with bgfilter = "none", i.e. no background filter applied).
E.BG	Numerical vector - The background error image to use. Optional - if left blank, it will be computed from E, using default BGFilter settings (i.e. method = "median").

Details

BASHExtended assesses the change of background across an array.

The error image used should not be background filtered (as opposed to the error image used in BASHDiffuse). Here, E is the error image

Value

Scalar (Extended defect score)

Author(s)

Jonathan Cairns

References

Mayte Suarez-Farinas, Maurizio Pellegrino, Knut M. Wittkwosky and Marcelo O. Magnasco (2007).
Harshlight: A "corrective make-up" program for microarray chips. R package version 1.8.0. http://asterion.rockefeller.edu

See Also

BASH, generateE, BGFilter, generateNeighbours,

Examples

```
data(BLData)
an <- arrayNames(BLData)
extended <- NULL

for(i in 1:length(an))
{
        extended[i] <- BASHExtended(BLData, i)
}
```

BASH *BASH - BeadArray Subversion of Harshlight*

Description

BASH is an automatic detector of physical defects on an array. It is designed to detect three types of defect - COMPACT, DIFFUSE and EXTENDED.

Usage

```
BASH(BLData, array, compact = TRUE, diffuse = TRUE, extended = TRUE, log = TRUE,
```

Arguments

BLData	BeadLevelList
array	integer specifying which strip/array to plot. Alternatively you can supply a vector of strip/array IDs, and BASH will analyse each in turn.
compact	Logical - Perform compact analysis?
diffuse	Logical - Perform diffuse analysis?
extended	Logical - Perform extended analysis?
log	Logical - Perform analyses on the log scale? (recommended)
cinvasions	Integer - number of invasions used whenever closing the image - see BASHCompact

dinvasions	Integer - number of invasions used in diffuse analysis, to find the kernel - see BASHDiffuse
einvasions	Integer - number of invasions used when filtering the error image - see BGFilter.
bgcorr	One of "none", "median", "medianMAD" - Used in diffuse analysis, this determines how we attempt to compensate for the background varying across an array. For example, on a SAM array this should be left at "median", or maybe even switched to "none", but if analysing a large beadchip then you might consider setting this to "medianMAD". (this code is passed to the method argument of BGFilter).
maxiter	Integer - Used in compact analysis - the max number of iterations allowed. (Exceeding this results in a warning.)
compcutoff	Integer - the threshold used to determine whether a group of outliers is in a compact defect. In other words, if a group of at least this many connected outliers is found, then it is labelled as a compact defect.
compdiscard	Logical - should we discard compact defect beads before doing the diffuse analyis?
diffcutoff	Integer - this is the threshold used to determine the minimum size that clusters of diffuse defects must be.
diffsig	Probability - The significance level of the binomial test performed in the diffuse analysis.
diffn	Numerical - when finding outliers on the diffuse error image, how many MADs away from the median an intensity must be for it to be labelled an outlier.
difftwotail	Logical - If TRUE, then in the diffuse analysis, we consider the high outlier and low outlier images seperately.

Details

The BASH pipeline function performs three types of defect analysis on an image.

The first, COMPACT DEFECTS, finds large clusters of outliers, as per BASHCompact. The outliers are found using findAllOutliers(). We then find which outliers are clustered together. This process is iterative - having found a compact defect, we remove it, and then see if any more defects are found.

The second, DIFFUSE DEFECTS, finds areas which are densely populated with outliers (which are not necessarily connected), as per BASHDiffuse. To make this type of defect more obvious, we first generate an ERROR IMAGE, and then find outliers based on this image. (The error image is calculated by using method = "median" and bgfilter = "medianMAD" in generateE, unless ebgcorr = FALSE in which case we use bgfilter = "median".) Now we consider a neighbourhood around each bead and count the number of outlier beads in this region. Using a binomial test we determine whether this is more that we would expect if the outliers were evenly spread over the entire array. If so, we mark it as a diffuse defect. (A clustering algorithm similar to the compact defect analysis is run to reduce false positives.)

After each of these two analyses, we "close" the image, filling in gaps.

The third, EXTENDED DEFECTS, returns a score estimating how much the background is changing across an array, as per BASHExtended. To estimate the background intensity, we generate an error image using the median filter (i.e. generateE with method = "median" and bgfilter = "median"). We divide the variance of this by the variance of an error image without using the median filter, to obtain our extended score.

It should be noted that to avoid repeated computation of distance, a "neighbours" matrix is used in the analysis. This matrix describes which beads are close to other beads. If a large number of beads

are missing (for example, if beads with ProbeID = 0 were discarded) then this algorithm may be affected.

For more detailed descriptions of the algorithms, read the help files of the respective functions listed in "see also".

Value

The output is a list with three attributes:

wts: A list, where the ith object in the list corresponds to the weights for array i.

ext: A vector of extended scores (null if the extended analysis was disabled)

call: The function you used to call BASH.

Author(s)

Jonathan Cairns

References

Mayte Suarez-Farinas, Maurizio Pellegrino, Knut M. Wittkwosky and Marcelo O. Magnasco (2007).
Harshlight: A "corrective make-up" program for microarray chips. R package version 1.8.0. http://asterion.rockefeller.edu

See Also

BASHCompact, BASHDiffuse, BASHExtended, generateE, generateNeighbours,

Examples

```
data(BLData)
output <- BASH(BLData,array=1:4)
boxplot(output$ext) #view spread of extended scores
for(i in 1:4)
{
        BLData <- setWeights(BLData, output$wts[[i]], i) #apply BASH weights to B
}

#diffuse test is stricter
output <- BASH(BLData, diffsig = 0.00001,array=1)

#more outliers on the error image are used in the diffuse analysis
output <- BASH(BLData, diffn = 2,array=1)

#only perform compact & diffuse analyses (we will only get weights)
output <- BASH(BLData, extended = FALSE,array=1)

#attempt to correct for background.
output <- BASH(BLData, bgcorr = "median",array=1)
```

`beadarrayUsersGuide` *View beadarray User's Guide*

Description

Finds the location of the beadarray User's Guide and opens it.

Usage

```
beadarrayUsersGuide(view=TRUE, topic="beadlevel")
```

Arguments

`view`	logical, should the document be opened using the default PDF document reader? (default is `TRUE`)
`topic`	character string specifying topic (`"beadlevel"`, `"beadsummary"` or `"BASH"`)

Details

The function `vignette("beadarray")` will find the short beadarray vignette which describes how to obtain the more detailed user's guide on the analysis of raw `"beadlevel"` data, `"beadsummary"` data or how to use the `"BASH"` method for detecting spatial artefacts.

Value

Character string giving the file location.

Author(s)

Matt Ritchie

See Also

`limmaUsersGuide`

Examples

```
beadarrayUsersGuide(view=FALSE)
beadarrayUsersGuide(view=FALSE, topic="beadsummary")
```

`beadResids` *Calculates per strip/array bead-level residuals*

Description

Calculates the per bead residuals for a given strip/array using data from a `BeadLevelList`.

Usage

```
beadResids(BLData, what="G", array=1, log=TRUE,
                  method="illumina", n=3, trim=0.05)
```

Arguments

`BLData`	`BeadLevelList`
`what`	character string specifying which intensities to use in the calculation of residuals. See `getArrayData` for a list of possibilities
`array`	integer specifying the strip/array to use
`log`	if TRUE then use log2 intensities of each bead
`method`	character string specifying the summarisation method (see help page for `createBeadSummaryDat` for further details).
`n`	numeric value defining a cut-off for the number of median absolute deviations (MADs) from the median to use for determining outliers. The default value is 3. Only used when `method="illumina"` (see `createBeadSummaryData` help page for further details).
`trim`	fraction of intensities to remove from the bead summary calculations when `method="trim"`, or the fraction of intensities to set to the `trim` and 1-`trim` percentile intensities when `method="winsorize"`. Default value is 0.05. Only used when `what="residR"`, `"residG"` or `"residM"`.

Details

Calculates the residuals, i.e. the differences between the summary values obtained from `createBeadSummaryData` and the individual values for each bead.

Value

A vector containing the residual values.

Author(s)

Matt Ritchie

Examples

```
data(BLData)
summary(beadResids(BLData, log=TRUE))
```

`BGFilter` *Background Filter*

Description

Performs various image transforms, based on statistics from local beads.

Usage

```
BGFilter(E = NULL, neighbours, invasions = 20, method = "median")
```

Arguments

`E`	Error Image
`neighbours`	A Neighbours matrix. Required.
`invasions`	Integer - Number of invasions. This argument is passed to the function BGfilter.
`method`	Method for computing local statistics. Options are `"median"`, `"mean"`, `"MAD"`, `"medianMAD"`

Details

This function transforms an error image based on a local statistic.

To obtain our statistic, we use an invasion process. Links between beads are defined in the neighbours matrix. We define the local beads as those which can be reached in `invasions` steps from the first bead, and then collect their values.

`method = "median"` subtracts the local median from each error intensity.

`method = "mean"` subtracts the local mean from each error intensity.

`method = "MAD"` divides each bead's intensity by the MAD (median absolute devation from the median) of local beads.

`method = "medianMAD"` subtracts the local median from each error intensity, and then divides each intensity by the local MAD.

Value

A vector - the updated error image.

Author(s)

Jonathan Cairns

See Also

`BGFilter`

Examples

```
data(BLData)
E <- generateE(BLData,1,bgfilter = "none")
neighbours <- generateNeighbours(BLData,1)
E.MAD <- BGFilter(E, neighbours, method = "MAD")
E.median <- BGFilter(E, neighbours, method = "median")
```

`BGFilterWeighted` *Weighted Background Filter*

Description

Finds local weighted means at each bead.

Usage

```
BGFilterWeighted(E = NULL, neighbours, invasions = 20, weights = NULL)
```

Arguments

`E`	Error Image
`neighbours`	A Neighbours matrix. Required.
`invasions`	Integer - Number of invasions. This argument is passed to the function BGfilter.
`weights`	Numerical vector - A vector of weights, from 0 to 1, to consider in the analysis. (see below.)

Details

This function finds the weighted mean of local bead intensities, using intensities from the given error image.

To obtain our weighted mean for each bead, we use an invasion process. Links between beads are defined in the neighbours matrix. We define the local beads as those which can be reached in `invasions` steps from the first bead, and then collect their error values.

We take a weighted mean of these error values, where the weights are calculated by taking the product of: a) 1/(the number of steps required to get to the bead from the central bead) b) (if supplied) the weights defined through the `weights` parameter.

This weighted mean is then assigned to the central bead.

Value

A vector - the weighted means. (NB: Whilst `BGFilter` manipulates the error image and returns an updated error image, e.g. subtracting the local median, this function does not - it merely returns the local weighted means.)

Author(s)

Jonathan Cairns

See Also

`BGFilter`

Examples

```
data(BLData)
E <- generateE(BLData,1,method = "mean")
neighbours <- generateNeighbours(BLData,1)
##bgf <- BGFilterWeighted(E, neighbours)
```

BLData — *BeadLevelList objec from an example experiment*

Description

BLData is an object of class BeadLevelList which contains data from an experiment with 4 arrays.

Usage

```
data(BLData)
```

See Also

BeadLevelList

boxplotBeads — *Box plot of bead intensities*

Description

Function to produce box plots of the bead intensities from selected strips/arrays from a BeadLevelList object.

Usage

```
boxplotBeads(BLData, whatToPlot = "G", arrays = NULL, log = TRUE,
          varwidth = TRUE, method = "illumina", n = 3, trim = 0.05, ...)
```

Arguments

BLData	BeadLevelList
whatToPlot	character string specifying which intensities to plot. See getArrayData for a list of the possibilities.
arrays	integer (scalar or vector) specifying the strips/arrays to plot. If NULL, all the strips/arrays are plotted.
log	if TRUE log2 intensities are plotted
varwidth	logical, indicating whether box widths should be proportional to the number of values in each box plot
method	character string specifying the summarisation method to use (only applicable when whatToPlot="residG", "residR" or "residM"). Refer to help createBeadSummaryData help page for further information.
n	numeric value defining a cut-off for the number of median absolute deviations (MADs) from the median to use for determining outliers. The default value is 3. Only applicable when whatToPlot="residG", "residR" or "residM" and method="illumina". Refer to help createBeadSummaryData help page for further information.

trim	fraction of intensities to remove from the bead summary calculations. Only applicable when whatToPlot="residG", "residR" or "residM". Refer to createBeadSummaryData help page for further information.
...	further graphical parameters to the boxplot function from the graphics package

Details

Produces box plots of the specified intensities for selected strips/arrays.

Value

A plot is produced on the current graphical device

Author(s)

Matt Ritchie

Examples

```
data(BLData)

boxplotBeads(BLData)
```

BSData *ExpressionSetIllumina object for the example experiment*

Description

BSData is an object of class ExpressionSetIllumina which contains the data from the example Human6 version 1 BeadChips analysed in the bead-summary user guide.

Usage

```
data(BSData)
```

See Also

class.ExpressionSetIllumina

`calculateBeadLevelScores` *Quality assessment for expression chips*

Description

A collection of functions for tabulating and plotting various quality control measurements derived from the bead-level data for Illumina expression chips. Currently, Humanv1, Humanv2, Humanv3, Mousev1, Mousev1p1, Mousev2 and Rat chips are supported.

Usage

```
calculateBeadLevelScores(BLData, path = "QC", log = TRUE, plot = FALSE,
                         replacePlots=TRUE, writeToFile = TRUE)
```

Arguments

`BLData`	A BeadLeveList for an expression chip. The annotation slot of the object should define the type of chip
`path`	Specifies the directory where diagnostics plot are to be saved in
`log`	(used for outlier calculations) if TRUE calculate outliers on the log2 scale. If FALSE calculate outliers on the original scale
`plot`	if TRUE then diagnostic plots will be generated
`writeToFile`	Argument describing whether results of QA assessment should be output to html, txt or not output at all
`replacePlots`	if TRUE any plots that have already been created will be replaced

Details

For these QA tools we make use of the controls probes that Illumina use on their expression chips to detect the presence, or lack of, expression. See www.illumina.com/downloads/GX_QualityControl_TechNote.pdf for an overview of these controls. Illumina provide a means to visualize these controls, but the values reported are *after outlier removal* and there is no way to infer how many outliers are removed. Therefore, one does not get a true impression of the quality of an array. For instance, low intensity observations for positive controls may indicate a spatial defect.

For our QA measurements we perform a detection score calculation the same as Illumina, except on per-bead observations for each control type rather than the summarized values. Specifically, we test each bead observation of a given control bead-type for detection by computing a p-value: $1 - R/N$, where R is the relative rank of the bead intensity when compared to the N negative controls. Thus, if a particular bead has higher intensity than all the negative controls it will be assigned a value of 0. After these p-values have been calculated for all replicates of the bead type we report the percentage of beads with p-values lower than a set threshold of 0.05 (currently in favour in the Illumina literature). The percentage of beads that are detected at a set threshold is then reported. Another adaptation is to change the bead-types used as a reference in the calculation rather than the negative controls. For example, there are a series of sample-independent controls that have probe sequences complementary to oligonucleotides spiked into the hybridization solution and hence should always have detectable signal. For some of these bead types (six on the Human6 V3), the concentration is either "medium", "low" or "high", with the intention that there should be a predictable gradient

between the controls. Thus, we test if the bead-types with a medium concentration are detected compared to the low controls and similar for the medium and high controls.

The purpose of calculateBeadLevelScores is to calculate the following QA measures for all arrays in the BeadLevelList objects and return them in the arrayInfo slot of the BeadLevelList object. We also record the number of outliers found on the array.

If the plot argument to calculateBeadLevelScores is set to TRUE, then a number of diagnostic plots will be produced for each and compiled into a HTML page for that array. The location of these completed pages is specified by the path argument. Finally, if writeToFile is set to html, a html page compiling all the QA measures for the chip will be created.

Value

A modified version of BeadLevelList is created with the QA measures stored in a qcScores slot.

`HkpDet`	%age of housekeeping control beads that are detected compared to the negative controls.
`BioDet`	%age of biotin labelling control beads that are detected compared to the negative controls.
`LowDet`	%age of "low" control beads that are detected compared to the negative controls.
`MedDet`	%age of "medium" control beads that are detected compared to the negative controls.
`HighDet`	%age of "high" control beads that are detected compared to the negative controls.
`MvsL`	%age of "medium" control beads that are detected compared to the "low" controls.
`HvsM`	%age of "high" control beads that are detected compared to the "medium" controls.

Author(s)

Mark Dunning and Andy Lynch

See Also

`outlierPlot`, `lmhPlot`, `poscontPlot`, `backgroundControlPlot`

`calculateDetection` *Calculate detection scores*

Description

Function to calculate detection scores for summarized data if they are not available.

Usage

```
calculateDetection(BSData)
```

Arguments

`BSData`	An ExpressionSetIllumina object

Details

The function implements Illumina's method for calculating the detection scores for all bead types on a given array. Within an array, Illumina discard negative control bead-types whose summary values are more than three MADs from the median for the negative controls. Illumina then rank the summarized intensity for each other bead-type against the summarized values for the remaining negative control bead-types and calculate a detection p-value 1-R/N, where R is the relative rank of the bead intensity when compared to the N remaining negative controls. Thus, if a particular bead has higher intensity than all the negative controls it will be assigned a value of 0. This calculation is repeated for all arrays stored in the BSData object. The annotation slot of the BSData object needs to be set correctly in order for the function to find the IDs of the negative control beads.

Value

Matrix of detection scores with the same dimensions as the exprs matrix of BSData. This matrix can be stored in a BSData object using the Detection function

Author(s)

Mark Dunning and Andy Lynch

Examples

```
##BSData@annotation = "Humanv3"
##Detection(BSData) = calculateDetection(BSData)
```

---

chooseClusters *Choose Clusters*

Description

Find large clusters of beads.

Usage

```
chooseClusters(IDs, neighbours, cutoff = 8)
```

Arguments

IDs	IDs of beads to be clustered.
neighbours	A Neighbours matrix - obtained from generateNeighbours.
cutoff	Integer - threshold for the minimum size a cluster must be.

Details

This function will find which beads are in large clusters. Using a flood fill algorithm, it finds clusters of beads, determines the size of each, and then returns only the beads in clusters of size greater than cutoff. It is primarily used in BASHCompact and BASHDiffuse.

Value

Vector of bead IDs. (This will be a subset of the argument `IDs`)

Author(s)

Jonathan Cairns

See Also

`BASHCompact`, `BASHDiffuse`, `closeImage`

Examples

```
data(BLData)
neighbours <- generateNeighbours(BLData,1)
o <- findAllOutliers(BLData,1,log = TRUE)
##clusters8 <- chooseClusters(o, neighbours)
##clusters12 <- chooseClusters(o, neighbours, cutoff = 12) ## only ##larger clusters

##x11()
##plotBeadLocations(BLData,array=1,BeadIDs = clusters8, pch = ".")
```

BeadLevelList-class

Class "BeadLevelList"

Description

A class for storing red and green channel foreground and background intensities from an Illumina experiment.

Objects from the Class

Objects can be created by calls of the form `new("BeadLevelList")`, but are usually created by `readIllumina`.

Slots/List Components

Objects of this class contain the following slots

`beadData`:	an environment for storing the raw bead-level data. Each row correspond to a bead and columns the data
`phenoData`:	an 'AnnotatedDataFrame' containing experimental information.
`arrayInfo`:	a list containing array information.
`annotation`:	character storing annotation package information.

Methods

show(BeadLevelList) printing method for BeadLevelList

initialize `signature(.Object = "BeadLevelList")`

dim `dim(object)` The dimension of the `BeadLevelList` object

`copyBeadLevelList(object)` Creates a new copy of a `BeadLevelList` object

arrayNames(object,arrays=NULL) Returns the strip/array names from a BeadLevelList object for selected arrays

combineBeadLevelLists(object1,object2) Combines two BeadLevelList objects into one

getArrayData(object,what="G",log=TRUE) Retrieves the what intensities on the log scale from the BeadLevelList

numBeads(object,arrays=NULL) Returns the number of beads on selected arrays

pData(object) Returns a data.frame with samples as rows, variables as columns

phenoData(object) Returns an object containing phenotypic information on both variable values and variable meta-data

Author(s)

Mark Dunning and Matt Ritchie

See Also

readIllumina

ExpressionSetIllumina *Class to Contain Objects Describing High-Throughput Illumina Expression BeadArrays.*

Description

Container for high-throughput assays and experimental metadata. ExpressionSetIllumina class is derived from eSet, and requires matrices exprs, se.exprs, NoBeads, Detection as assay data members.

Extends

Directly extends class eSet.

Creating Objects

new('ExpressionSetIllumina', phenoData = [AnnotatedDataFrame], exprs = [matrix], se.exprs = [matrix], NoBeads = [matrix], Detection = [matrix], annotation = [character], featureData = [AnnotatedDataFrame], experimentData = [MIAME], ...) ExpressionSetIllumina instances are usually created through new("ExpressionSetIllumina", ...). Arguments to new include exprs, se.exprs, NoBeads, Detection, phenoData, experimentData, and annotation. phenoData, experimentData, and annotation can be missing, in which case they are assigned default values.

Slots

Inherited from eSet:

assayData: Contains matrices with equal dimensions, and with column number equal to nrow(phenoData). assayData must contain a matrix exprs with rows representing features (e.g., genes) and columns representing samples, a matrix se.exprs describing the standard error of each gene, and matrices NoBeads and Detection to describe the number of beads used to produce the summary and a probability of a gene being expressed above background. The contents of these matrices are not enforced by the class. Additional matrices of identical size may also be included in assayData. Class:AssayData

phenoData: See eSet

experimentData: See eSet

annotation: See eSet

featureData: annotation for SNPs, usually will contain a CHR and a MapInfo column for genomic localization

Methods

Class-specific methods:

exprs(ExpressionSetIllumina), exprs(ExpressionSetIllumina,matrix)<- Access and set elements named exprs in the AssayData slot.

se.exprs(ExpressionSetIllumina), se.exprs(ExpressionSetIllumina,matrix)<- Access and set elements named se.exprs in the AssayData slot.

NoBeads(ExpressionSetIllumina) Access elements named NoBeads in the AssayData slot.

Detection(ExpressionSetIllumina) Access elements named Detection in the AssayData slot.

getVariance(ExpressionSetIllumina) Calculate bead-type specific variance using se.exprs and NoBeads from the AssayData slot.

QCInfo(ExpressionSetIllumina),QCInfo(ExpressionSetIllumina,list)<- Access elements named QC in the AssayData slot.

object[(index): Conducts subsetting of matrices and phenoData and reporterInfo components

combine(ExpressionSetIllumina,ExpressionSetIllumina): performs union-like combination in both dimensions of ExpressionSetIllumina objects

show(ExpressionSetSetIllumina) See eSet

Derived from eSet:

sampleNames(ExpressionSetSetIllumina) and sampleNames(ExpressionSetSetIllumina)<-: See eSet

featureNames(ExpressionSetSetIllumina), featureNames(ExpressionSetSetIllumina, va See eSet

dims(ExpressionSetSetIllumina): See eSet

phenoData(ExpressionSetSetIllumina), phenoData(ExpressionSetSetIllumina,value)<-: See eSet

varLabels(ExpressionSetSetIllumina), varLabels(ExpressionSetSetIllumina, value)<- See eSet

varMetadata(ExpressionSetSetIllumina),varMetadata(ExpressionSetSetIllumina,value
See eSet

pData(ExpressionSetSetIllumina),pData(ExpressionSetSetIllumina,value)<-:
See eSet

varMetadata(ExpressionSetSetIllumina),varMetadata(ExpressionSetSetIllumina,value
See eSet

experimentData(ExpressionSetSetIllumina),experimentData(ExpressionSetSetIllumina,
See eSet

annotation(ExpressionSetSetIllumina),annotation(ExpressionSetSetIllumina,value)<
See eSet

storageMode(eSet),storageMode(eSet,character)<-: See eSet

Standard generic methods:

initialize(ExpressionSetSetIllumina): Object instantiation, used by `new`; not to be called directly by the user.

validObject(ExpressionSetSetIllumina): Validity-checking method, ensuring that `call`, `callProbability`, `G`, and `R` are members of `assayData`. `checkValidity(ExpressionSetSet` imposes this validity check, and the validity checks of `Biobase:eSet`.

show(ExpressionSetSetIllumina) See eSet

dim(ExpressionSetSetIllumina),ncol See eSet

ExpressionSetSetIllumina[(index): See eSet

ExpressionSetSetIllumina$,ExpressionSetSetIllumina$<- See eSet

Author(s)

Mark Dunning, based on Biobase eSet class

See Also

eSet

`closeImage` *Close Image*

Description

Find the closure of a set of beads on an array.

Usage

```
closeImage(IDs, neighbours, cinvasions = 10)
```

Arguments

`IDs`	IDs of beads to be closed.
`neighbours`	A Neighbours matrix - obtained from `generateNeighbours`.
`cinvasions`	The number of invasions used when dilating and eroding.

Details

This function "closes" the set of beads supplied, as used in the BASH functions. It dilates (expands) the image, and then erodes (contracts) it. Each is done via an invasion process - if we let the set of beads supplied be called S, then dilation considers all neighbours of beads in S, and adds them to S. Erosion finds all beads in S with neighbours outside of S, and removes them from S.

The result of this process is to close "holes" in the group of specified beads during the dilation. These are not reopened during the erosion.

Value

An updated vector of bead IDs (of which the argument `IDs` will be a subset).

Author(s)

Jonathan Cairns

See Also

`generateNeighbours`

Examples

```
data(BLData)

##This process is equivalent to one iteration of BASHCompact.
##o <- findAllOutliers(BLData,4)
##neighbours <- generateNeighbours(BLData,4)
##o.clusters <- chooseClusters(o, neighbours)
##o.compact <- closeImage(o.clusters, neighbours)
```

`combineBeadLevelLists` *Combines data from two BeadLevelList objects*

Description

Combines two `BeadLevelList` objects.

Usage

```
combineBeadLevelLists(object1, object2)
```

Arguments

`object1`	`BeadLevelList`
`object2`	`BeadLevelList`

Details

`combineBeadLevelLists` combines two `BeadLevelList` objects.

Value

A BeadLevelList object holding data from all strips/arrays and beads from the individual objects.

Author(s)

Matt Ritchie

---

copyBeadLevelList *Copies a BeadLevelList Object*

Description

Make a new copy of a BeadLevelList object.

Usage

```
copyBeadLevelList(object)
```

Arguments

object	BeadLevelList

Details

copyBeadLevelList makes a new copy of a BeadLevelList object. This is necessary beacuse the beadData slot is stored as an environment.

Value

A new BeadLevelList object containing the data from object.

Author(s)

Matt Ritchie

Examples

```
data(BLData)
BLDataNew = copyBeadLevelList(BLData)
BLData@beadData # the same bead level data is now
BLDataNew@beadData # stored in different environments
```

createBeadSummaryData
Produce bead averages

Description

Produce bead averages for each bead type used in an experiment on a specified set of strips/arrays.

Usage

```
createBeadSummaryData(BLData, log=FALSE, imagesPerArray = 1,
                what="G",  probes = NULL, arrays=NULL,
                method="illumina", n=3, trim=0.05)
```

Arguments

BLData	BeadLevelList
log	if TRUE then summarise the log2 intensities of each bead
imagesPerArray	Specifies how many images (strips) there are per array. Normally 1 for a SAM and 1 or 2 for a BeadChip. The images (strips) from the same array will be combined so that each column in the output represents a sample
what	character string specifying which intensities/values to summarise. See getArrayData for a list of possibilities.
probes	Specify particular probes to summarise. If left NULL then all the probes on the first array are used.
arrays	integer (scalar or vector) specifying the strips/arrays to summarise. If NULL, then all strips/arrays are summarised.
method	chracter string specifying the summarisation method to use. Options are "illumina", "mean", "median", "trim" and "winsorise".
n	numeric value defining a cut-off for the number of median absolute deviations (MADs) from the median to use for determining outliers. The default value is 3. Used when method="illumina"
trim	fraction of intensities to remove from the bead summary calculations when method="trim", or the fraction of intensities to set to the trim and 1-trim percentile intensities when method="winsorize". Default value is 0.05.

Details

To summarise the raw data using the default method used by Illumina (method="illumina") we first remove outliers for each bead type on each array. Outliers are beads which have an intensity greater than 3 median absolute deviations (MADs) from the bead median intensity on the original (un-logged) scale. The n argument can be changed to remove beads with intensity n MADs above or below the median. With outliers removed, the average (mean) intensities of the remaining beads are calculated along with the standard error and number of beads.

Other summarisation options are also available. When method="mean", the average and standard error of all beads for a given bead type is calculated on each array. This would be appropriate if the scanner has been set up to exclude outlier beads from the bead level .txt or .csv files.

When `method="median"`, the middle value is returned along with the median absolute deviation (rather than standard error) for each bead type. When `method="trim"`, the trimmed mean and standard error are calculated and for `method="winsorize"` the winsorised mean and standard error are returned.

By setting the `log` argument to `TRUE`, we calculate outliers and summary values on the log2-scale.

If there are any NAs or Inf values, they are ignored.

Objects which are created separately by 'createBeadSummaryData' may be joined using the `combine` function.

Value

An `ExpressionSetIllumina` object (or `NChannelSet` object for two-colour data, when `what="RG"`) in which all components are matrices with number of rows equal to the number of bead types for the experiment and number of columns equal to the number of arrays.

Author(s)

Mark Dunning and Mike Smith

See Also

`findBeadStatus`

Examples

```
#produce bead summaries for each array
data(BLData)
BSData = createBeadSummaryData(BLData, log=TRUE, what="G")
dim(BSData)
```

---

denseRegions *Find Dense Regions of Points (as used in diffuse defect analysis).*

Description

Given a list of beads, this function finds dense regions of beads on the list.

Usage

```
denseRegions(IDs, neighbours, ignore = NULL, sig = 0.0001, invasions = 10)
```

Arguments

`IDs`	Vector - IDs of beads to find dense regions of.
`neighbours`	A Neighbours matrix - obtained from `generateNeighbours`.
`ignore`	Vector - IDs of beads to be ignored during this process.
`sig`	Significance of the Binomial test performed within each kernel.
`invasions`	Integer - No of invasions used to generate the kernel.

Details

This function, given a list of bead IDs, finds regions where these marked beads are denser.

To do this, we use a "sliding kernel" technique. For each bead, we find the "kernel", a local neighbourhood of beads, obtained via invasion along links defined in the neighbours matrix. We count the number of beads in the kernel, and we also count how many of these are beads are marked. Now we test the density of this region with a binomial test.

Assuming that we expect the marked beads to be evenly distributed across the array, then the number of marked beads in the kernel should have distibution Bin(n,p) under the null hypothesis, where n is the total number of beads in the kernel, and p is the proportion of marked beads on the entire array. We test this hypothesis at a level defined by `sig`, and on rejection of the null hypothesis we label the kernel's central bead as being part of a dense region. This is performed for the kernel about each bead.

If `ignore` is specified, then these beads will be completely removed before analysis. Any links attached to a removed bead are severed.

Value

Vector - IDs of beads in dense regions.

Author(s)

Jonathan Cairns

See Also

`generateNeighbours`, `BASHDiffuse`

Examples

```
data(BLData)
E <- generateE(BLData,1)
E <- generateE(BLData,1, invasions = 10) #reduced no of invasions to increase speed.
E <- generateE(BLData,1, bgfilter = "none") #residuals (median)
```

ExpressionControlData

Control annotation for Illumina expression chips

Description

Data frames derived from the bgx files from Illumina that give details of the control probes used on Illumina expression arrays. A list structure is used with the control probes for a particular platform accessed by name. Note that the HumanHT12 arrays use the same probes and the Humanv3 and therefore the same annotation can be used.

Usage

```
data(ExpressionControlData)
```

Examples

```
library(beadarray)
data(ExpressionControlData)
names(ExpressionControlData)
ExpressionControlData[["Humanv3"]][1:10,]
```

`findAllOutliers` *Find outliers on a given strip/array*

Description

Function to find all beads which are outliers for their particular bead type on a given strip/array using Illumina's standard outlier detection method.

Usage

```
findAllOutliers(BLData, array, log=FALSE, n=3, what="G", usewts=FALSE)
```

Arguments

`BLData`	`BeadLevelList`
`array`	integer specifying which strip/array we want to find outliers on
`log`	if TRUE the intensities will be calculated on the log2 scale. Otherwise unlogged data is used.
`n`	numeric value defining a cut-off for the number of median absolute deviations (MADs) from the median to use for determining outliers. The default value is 3.
`what`	character string specifying which intensities to use. See `getArrayData` for a list of possibilities.
`usewts`	if TRUE, then beads with weights below 1 will be discarded prior to analysis.

Details

We find the outliers for each bead type on the array in turn using the `findBeadStatus` function and store the indices of the outliers found. By default, outliers for a particular bead type are determined using a 3 MAD cut-off from the median.

Value

numeric vector giving the row indices of `BLData` (in the range 1 to total number of beads on the array) of all beads that are outliers for their bead type.

Author(s)

Mark Dunning

See Also

`findBeadStatus`

Examples

```
data(BLData)
# how many outliers are there on the original scale?
length(findAllOutliers(BLData, 1))
# how many outliers are there on the log2-scale?
length(findAllOutliers(BLData, 1, log=TRUE)) #
# how many outliers are there using a 4 MAD
# cut-off from the median?
length(findAllOutliers(BLData, 1, n=4))
```

`findBeadStatus` *Find Outliers*

Description

Function finds all beads which are outliers for a given bead type

Usage

```
findBeadStatus(BLData, probes, array = 1, log = FALSE, what = "G", n = 3,
           outputValid = FALSE, intProbeID = NULL, ignoreList = NULL,
           probeIndex = NULL, startSearch = 1)
getProbeIntensities(BLData, ProbeIDs, array = 1, log = TRUE, what = "G")
```

Arguments

`BLData`	`BeadLevelList`
`probes`	numeric vector for the ProbeIDs of the bead type we want to find outliers for
`array`	integer specifying which strip/array to use
`log`	if TRUE the intensities will be calculated on the log2 scale. Otherwise unlogged data is used
`what`	character string specifying which intensities to use. Possibilities are `"G"`, `"Gb"` for single channel data and `"G"`, `"Gb"`, `"R"` and `"Rb"` for two-colour data
`n`	numeric value defining a cut-off for the number of median absolute deviations (MADs) from the median to use for determining outliers. The default value is 3.
`outputValid`	if TRUE the IDs of beads which are not outliers will be output
`intProbeID`	BLData$ProbeID coerced to vector of integers. Never change this, for internal use only
`ignoreList`	list of ProbeIDs to be omitted from the averaging procedure. These could be Illumina internal controls which are replicated many thousands of times on arrays
`probeIndex`	parameter for internal use only
`startSearch`	integer specify where to start searching for a particular ProbeID
`ProbeIDs`	numeric vector for the ProbeIDs of the bead type we want to find outliers for

Details

The intensities of each bead with ProbeID 'probe' on the specified array are found and if the 'log' parameter is set to TRUE we do a log2 transformation of these values.

The median and MAD for the bead intensities are then calculated. Outliers are beads which have intensity more than 'n' MADs from the median.

The method used by Illumina is to use un-logged intensities with 'n' = 3.

Any beads which have intensity NA are also counted as outliers.

The function returns only the outliers for a bead type unless the outputValid parameter is specified.

Value

List of beadIDs dividing the beads of this bead type into two categories.

`valid`	valid beads
`outliers`	beads which are calculated as outliers

Author(s)

Mike Smith and Mark Dunning

See Also

`findAllOutliers`

Examples

```
data(BLData)
findBeadStatus(BLData, 2, 1, outputValid=TRUE)
findBeadStatus(BLData, 2, 1, log=TRUE, outputValid=TRUE)
findBeadStatus(BLData, 23, 1, outputValid=TRUE)
findBeadStatus(BLData, 23, 1, log=TRUE, outputValid=TRUE)
```

`generateE` *Generate Error Image for BeadLevelList object*

Description

Generates an Error Image from the data in a `BeadLevelList` object.

Usage

```
generateE(BLData, array, neighbours = NULL, log = TRUE, method = "median", what
```

Arguments

BLData	BeadLevelList
array	integer specifying which strip/array to plot
neighbours	A Neighbours matrix. Optional - if left NULL, it will be computed.
log	Logical. If TRUE, compute residuals on the log scale.
method	Method for computing residuals. Options are "mean" and "median"
what	What to derive the error image from, as used in getArrayData.
bgfilter	Method passed to the function BGFilter. Options are "none", "mean", "median", "MAD" and "medianMAD".
invasions	Integer - Number of invasions. This argument is passed to the function BGfilter.

Details

generateE creates an error image, usually based on bead residuals. This output can then be fed into BASHDiffuse or BASHExtended.

If what is residG, residR, or residM, then residuals are calculated based on method. For other values of what, the residuals are not calculated.

We then apply a "background filter" to this data, using the function BGFilter with arguments bgfilter and invasions - see its help file for more details. The background filter subtracts an estimate of the local background of the error image, and/or scales by the local MAD. This step is disabled by using bgfilter = "none".

Value

An "Error Image" - a vector of length equal to the number of beads on the array.

Author(s)

Jonathan Cairns

See Also

BGFilter

Examples

```
data(BLData)
E <- generateE(BLData,1)
E <- generateE(BLData,1, invasions = 10) #reduced no of invasions to increase speed.
E <- generateE(BLData,1, bgfilter = "none") #residuals (median)
```

`generateNeighbours` *Generate Neighbours Matrix for BeadLevelList object*

Description

Generates a Neighbours matrix from the X and Y co-ordinates in a `BeadLevelList` object.

Usage

```
generateNeighbours(BLData, array, window = 30, margin = 10, thresh = 2.2)
```

Arguments

`BLData`	`BeadLevelList`
`array`	integer specifying which strip/array to plot
`window`	numeric value, specifying window size (see below)
`margin`	numeric value, specifying size of window margin (see below)
`thresh`	numeric value, which determines how large links are removed. (see below)

Details

generateNeighbours determines, for each bead on the array, which beads are next to it. It assumes that the beads are in a hexagonal lattice.

The algorithm used first links each bead to its 6 closest neighbours. It then removes the longest link if its squared length is more than `thresh` multiplied by the squared length of the next longest link. A similar process is applied to the 2nd and 3rd longest links.

Finally, any one way links are removed (i.e. a link between two beads is only preserved if each bead considers the other to be its neighbour).

To ease computation, the algorithm only computes neighbours of beads in a square window of side length `2*(window)` which travels across the array. Beads in a margin around the square, of width `(margin)`, are also considered as possible neighbours.

The Neighbours matrix is designed for use with the BASH functions.

Value

A matrix with 6 columns, and a number of rows equal to the number of beads on the array. The neighbours of bead i are found in row i. 0 denotes a deleted link. (For example, if row 15 consists of 20, 35, 0, 0, 60, 4, then bead 15 is considered a neighbour of beads 4, 20, 35 and 60.)

Author(s)

Jonathan Cairns

Examples

```
data(BLData)
neighbours <- generateNeighbours(BLData, 1)
```

`getAnnotation` *Storage of annotation information for Illumina expression chips*

Description

Illumina use several control types for QA purposes, however the IDs of these controls change between different organisms and annotation revisions. Therefore we need to store the annotation of a chip in order to perform QA on the bead-level data. The functions setAnnotation and getAnnotation are used to manage this annotation information.

Usage

```
getAnnotation(BLData)
setAnnotation(BLData, aName)
```

Arguments

`BLData`	BeadLevelList for an Illumina expression array.
`aName`	Character to define the annotation of the chip

Details

We currently store the annotation as a slot in a BeadLevelList. The value in the slot should match one of entries in `ExpressionControlData` (see example).

Value

setAnnotation returns a modified BeadLevelList with the new value for the annotation slot.

Author(s)

Mark Dunning

See Also

`ExpressionControlData`

Examples

```
data(BLData)
data(ExpressionControlData)
getAnnotation(BLData)
names(ExpressionControlData)
```

getArrayData *Get raw data from a BeadLevelList object*

Description

Retrieves the raw bead data from a BeadLevelList object for a given strip/array.

Usage

```
getArrayData(BLData, what="G", array=1,log=TRUE, method="illumina", n=3, trim=0.
```

Arguments

BLData	BeadLevelList
what	character string specifying the values to retrieve. Possibilities are "ProbeID", "GrnX", "GrnY", "G", "Gb" for single channel data and "R", "Rb", "residR", "residG", "M" (log-ratios) "residM", "A" (average log-intensities) and "beta" (=R/(R+G)) for two-colour data
array	integer specifying the strip/array to use
log	if TRUE log2 of the raw intensities are returned (ignored if what="beta")
method	character string specifying the summarisation method to use in createBeadSummaryData (see help page for further details). Only used when what="residR", "residG" or "residM".
n	numeric value specifying the number of median absolute deviations (MADs) from the median to use as a cut-off for outliers. Only used when what="residR", "residG" or "residM" and method="illumina".
trim	fraction of intensities to remove from the bead summary calculations when method="trim", or the fraction of intensities to set to the trim and 1-trim percentile intensities when method="winsorize". Default value is 0.05. Only used when what="residR", "residG" or "residM".

Details

getArrayData retrieves the raw bead data from a given array. The data is either extracted from a BeadLevelList object (e.g. "ProbeID", "GrnX", "GrnY", "G", "Gb", "R" and "Rb" or calculated from these values (e.g. "residR", "residG", "M", "residM", "A" or "beta"). When log=TRUE, intensity data is returned on the log2 scale.

Value

A vector containing the raw bead data (or residuals) for a particular array.

Author(s)

Matt Ritchie

Examples

```
data(BLData)
summary(getArrayData(BLData))
```

getVariance *Gets the bead-type variances from an ExpressionSetIllumina Object*

Description

Calculates the variance for each bead-type on each array from an ExpressionSetIllumina object.

Usage

```
getVariance(object, offset=0)
```

Arguments

object	ExpressionSetIllumina object
offset	numeric value to add to the variances to avoid very small values

Details

getVariance uses the se.exprs and NoBeads slots in assayData to calculate the variances for each bead-type on each array.

Value

A matrix containing the variances.

Author(s)

Matt Ritchie

Examples

```
data(BSData)
v = getVariance(BSData)
boxplot(as.data.frame(log2(v)), ylab="log2var", xlab=colnames(BSData), las=2)
```

HULK *HULK - Bead Array Normalization by NEighbourhood Residuals*

Description

Normalizes an probe intensities by calculating a weighted average residual based on the residuals of the surrounding probes.

Usage

```
HULK(BLData, array, neighbours = NULL, invasions = 20, what = "G")
```

Arguments

BLData	BeadLevelList
array	integer specifying which strip/array to plot
neighbours	A Neighbours matrix. Optional - if left NULL, it will be computed.
invasions	Integer - Number of invasions used when identifying neighbouring beads.
what	Specify the data in the BLData to create the residuals from. Defaults to the foreground intensities of the green channel.

Details

HULK is a method of intensity normalization based upon the BASH framework. Firstly For each bead a local neighbourhood of beads is determined, using the same process as the other BASH functions.

For each bead a weighted average residual is calculated. The average residual is calculated as the sum of the residuals for each bead in the neighbourhood, divided by 1 plus the number of invasions it took to reach that bead. This calculation is made by a call to HULKResids.

The average residuals are then subtracted from each bead and the resulting BeadLevelList object is returned.

Value

An object of class BeadLevelList

Author(s)

Mike Smith

References

Cairns JM, Dunning MJ, Ritchie ME, Russell R, Lynch AG. (2008). BASH: a tool for managing BeadArray spatial artefacts. Bioinformatics, 24(24):2921-2.

See Also

HULKResids, BASH

Examples

```
data(BLData)
o <- HULK(BLData, 1)
```

HULKResids *HULK - Residuals*

Description

Calculates an set of weighted average residuals, one for each probe, based on the residuals of the surrounding probes.

Usage

```
HULKResids(BLData, array, neighbours = NULL, invasions = 20, what = "G")
```

Arguments

BLData	BeadLevelList
array	integer specifying which strip/array to plot
neighbours	A Neighbours matrix. Optional - if left NULL, it will be computed.
invasions	Integer - Number of invasions used when identifying neighbouring beads.
what	Specify the data in the BLData to create the residuals from. Defaults to the foreground intensities of the green channel.

Details

HULKResids calculates a weighted average residual for each probe on the specified array of BLData. It makes use of the same neighbourhood calculations as other BASH functions. The average residuals are calculated as the sum of the residuals for each bead in the neighbourhood, divided by 1 plus the number of invasions it took to reach that bead. It is intended that HULKResids be called through HULK, but it is quite possible to call it as a stand alone function.

Value

A vector containing an average residual for each bead on the specified array of BLData.

Author(s)

Mike Smith

References

Cairns JM, Dunning MJ, Ritchie ME, Russell R, Lynch AG. (2008). BASH: a tool for managing BeadArray spatial artefacts. Bioinformatics, 24(24):2921-2.

See Also

HULK, BASH

Examples

```
data(BLData)
o <- HULKResids(BLData, 1)
```

imageplot *imageplot for BeadLevelList object*

Description

Generates an image plot for data from a BeadLevelList object.

Usage

```
imageplot(BLData, array = 1, nrow = 100, ncol = 100, low= NULL,
          high = NULL, ncolors = 123, whatToPlot ="G", log=TRUE,
          zlim=NULL, main=whatToPlot, method="illumina", n = 3,
          trim=0.05, legend=TRUE, SAM=FALSE, ...)
```

Arguments

BLData	BeadLevelList
array	integer specifying which strip/array to plot
nrow	integer specifying the number of rows to divide the strip/array into
ncol	integer specifying the number of columns to divide the strip/array into
low	colour to use for lowest intensity
high	colour to use for highest intensity
ncolors	The number of colour graduations between high and low
whatToPlot	character string specifying which intensities/values to plot. See getArrayData for a list of possibilities
log	if TRUE, log2 intensities are plotted
zlim	numerical vector of length 2 giving the extreme values of 'z' to associate with colours 'low' and 'high'.
main	character string for plot title
method	character string specifying the summarisation method to use. Only applicable when whatToPlot="residG", "residR" or "residM". Refer to the createBeadSummaryData help page for further information.
n	numeric value specifying the number of median absolute deviations (MADs) from the median to use as a cut-off for outliers. The default value is 3. Only applicable when whatToPlot="residG", "residR" or "residM" and method="illumina". Refer to createBeadSummaryData help page for further information.
trim	fraction of intensities to remove from the bead summary calculations. Only applicable when whatToPlot="residG", "residR" or "residM". Refer to createBeadSummaryData help page for further information.
legend	logical, if TRUE, zlim and range of data is added to plot.
SAM	logical, if TRUE, x and y coordinates are transposed.
...	other graphical parameters to plot that can be specified

Details

Because of the large number of beads on each strip/array, this function works by mapping a grid of size specified by the `nrow` and `ncol` arguments and averaging the intensities of the beads within each section of the grid.

The number of rows and columns may change the appearance of the plots. If the array is divided into too many squares it will be difficult to detect changes. We recommend using `nrow=20` and `ncol=200` for the strips on a BeadChip, and `nrow=100` `ncol=100` for arrays on a SAM.

An imageplot of the log base 2 foreground intensities is produced by default. Other values can be plotted by changing the `whatToPlot` argument. The default colour scheme ranges from white for low values to blue for high values.

As a result of both having identical function names this function can conflict with the `imageplot` method in 'limma'. If both packages are loaded, the function from whichever package was loaded last takes precedence. If the 'beadarray' imageplot() function is masking that from 'limma', one can directly call the 'limma' method using the command "limma::imageplot()". Alternatively, one can detach the 'beadarray' package using "detach(package:beadarray)". Similar techniques can be used if 'limma' is masking the 'beadarray' method.

Value

A plot is produced on the current graphical device.

Author(s)

Mike Smith, Mark Dunning

Examples

```
data(BLData)
imageplot(BLData)
```

`interactivePlots` *Interactive bead-level plotting*

Description

Generates spatial plots using bead-level data to discover artefacts on strips/arrays.

Usage

```
SAMSummary(BLData, mode = "outliers", whatToPlot = "G",  samID = NULL,
               log = TRUE, n = 3, colour = TRUE,
               scale = NULL, low = "yellow", high="red", ...)
BeadChipSummary(BLData, mode = "outliers", whatToPlot = "G", chipID = NULL,
               stripsPerChip = 12, log = TRUE, n = 3, colour = TRUE,
               scale = NULL, low = "yellow", high = "red", ...)
```

Arguments

BLData	BeadLevelList object
mode	character string either "outliers" or "intensities" specifying what to display on the plots
whatToPlot	character string specifying which intensities to plot. Possibilities are "G", "Gb" for single channel data and "G", "Gb", "R" and "Rb" for two-colour data
samID	character string specifying which SAM to plot. If NULL, data from the first SAM is plotted.
chipID	character string specifying which BeadChip to plot. If NULL, data from the first BeadChip is plotted.
stripsPerChip	integer specifying number of strips on BeadChip (8 or 12)
log	if TRUE log2 intensities of each bead are used to find outliers
n	numeric value specifying the number of median absolute deviations (MADs) from the median to use as a cut-off for outliers. The default value is 3
colour	if TRUE the hexagons will be plotted in colour
scale	numeric value giving the amount by which to divide all numbers by (eg for log2 intensities this should be 16) to transform to range 0 - 1
low	colour to use for lowest intensity
high	colour to use for highest intensity
...	other parameters to imageplot that can be specified

Details

A plot will be displayed giving a summary of each array in the experiment on the left screen and initially a blank right hand side. The left hand side is coloured according to the number of outliers found on the array or the mean intensity of the array (depending on the mode parameter). Clicking on tha particular array on the left will display a location plot of the outliers or an image plot on the right See RNews article

Value

A plot is produced on the current graphical device

Author(s)

Mark Dunning

---

lmhPlot *Plot the bead-level hybridisation controls*

Description

Function for retrieving and plotting the hybridisation controls for an expression array. We know these controls should show high signal and are therefore useful for QA purposes. Moreover, we should expect to see a gradient between the low, medium and high controls. By considering all bead observations (unlike the plots produced by BeadStudio) we get an detailed impression of array quality.

Usage

```
lmhPlot(BLData, array = 1, plot = FALSE)
```

Arguments

BLData	BeadLevelList object for an Illumina expression array which must have the annotation slot set appropriately.
array	The number of the array of interest
plot	If TRUE then a diagnostic plot will be produced, other only summary values will be returned.

Details

The annotation stored with the BLData object in the annotation is used to find the IDs of the hybridisation controls. We try and find these controls among the bead-level data for the array. If not all the control IDs can be found, then the wrong annotation may be stored for the array and the function will report an error. If found, we test the replicates of the low, medium and high controls for detection using the same criteria as used by Illumina (implemented in the calculateDetection function). However, an important difference is that we test each bead observation individually and report for each bead-type the percentage of beads detected.

The function returns five measures that can be used to evaluate the quality of the array (see below). On a good quality array, we would expect to see 100% for all these measures and a drop from 100% could indicate a defect on the array. However, it should be noted that the HvsM percentage could drop below 100% often due to the saturation effect often observed at high intensity.

If a plot is requested, the intensities of the hybridisation controls are plotted on a y-axis and grouped according to different control-type (low, medium or high concentration) on the x-axis. Some arrays may have more than one bead-type for a particular control.

Value

LowDet	%age of "low" control beads that are detected compared to the negative controls.
MedDet	%age of "medium" control beads that are detected compared to the negative controls.
HighDet	%age of "high" control beads that are detected compared to the negative controls.
MvsL	%age of "medium" control beads that are detected compared to the "low" controls.
HvsM	%age of "high" control beads that are detected compared to the "medium" controls.

Author(s)

Mark Dunning

References

see www.illumina.com/downloads/GX_QualityControl_TechNote.pdf for description of the hybridisation controls

See Also

calculateBeadLevelScores, setAnnotation, calculateDetection

`medianNormalise` *Median normalise data in a matrix*

Description

Normalises expression intensities so that the intensities or log-ratios have equal median values across a series of arrays (columns).

Usage

```
medianNormalise(exprs, log=TRUE)
```

Arguments

`exprs`	a matrix of expression values
`log`	if TRUE then do a log2 transformation prior to normalising

Details

Normalisation is intended to remove from the expression measures any systematic trends which arise from the microarray technology rather than from differences between the probes or between the target RNA samples hybridized to the arrays.

For median normalisation, the intensity for each gene is adjusted by subtracting the median of all genes on the array and then adding the median across all arrays. The effect is that each array then has the same median value.

Value

Produces a matrix of normalised intensity values (on the log2 scale by default) with the same dimensions as `exprs`.

Author(s)

Mark Dunning

Examples

```
data(BSData)
BSData.med = assayDataElementReplace(BSData, "exprs", medianNormalise(exprs(BSData)))
```

`normaliseIllumina` *Normalise Illumina expression data*

Description

Normalises expression intensities from an `ExpressionSetIllumina` object so that the intensities are comparable between arrays.

Usage

```
normaliseIllumina(BSData, method="quantile", transform="none", T=NULL, ...)
```

Arguments

`BSData`	an `ExpressionSetIllumina` object
`method`	character string specifying normalisation method (options are `"quantile"`, `"qspline"`, `"vsn"`, `"rankInvariant"`, `"median"` and `"none"`.
`transform`	character string specifying transformation to apply to the data prior to normalisation (options are `"none"`, `"log2"` and `"vst"`
`T`	A target distribution vector used when `method="rankInvariant"` normalisation. If `NULL`, the mean is used.
`...`	further arguments to be passed to `lumiT`

Details

Normalisation is intended to remove from the expression measures any systematic trends which arise from the microarray technology rather than from differences between the probes or between the target RNA samples hybridized to the arrays.

In this function, the `transform` specified by the user is applied prior to the chosen normalisation procedure.

When `transform="vst"` the variance-stabilising transformation from the 'lumi' package is applied to the data. Refer to the `lumiT` documentation for further particulars. Note that the Detection P values are only passed on when they are available (i.e. not NA)

For further particulars on the different normalisation methods options refer to the individual help pages (`?normalize.quantiles` for `"quantile"`, `?normalize.qspline` for "qspline", `?rankInvariantNormalise` for `"rankInvariant"`, `?medianNormalise` for `"median"` and `?vsn2` for `"vsn"`.

For median normalisation, the intensity for each gene is adjusted by subtracting the median of all genes on the array and then adding the median across all arrays. The effect is that each array then has the same median value.

Note: If your `BSData` object contains data already on the log-scale, be careful that you choose an appropriate `transform` to avoid transforming it twice. The same applies for the `"vst"` transformation and `"vsn"` normalisation methods which require the expression data stored in `BSData` to be on the original (un-logged) scale. When `method="vsn"`, transform must be set to `"none"`, since this method transforms and normalises the data as part of the model.

Value

An 'ExpressionSetIllumina' object which conatains the transformed and normalised expression values for each array.

Author(s)

Matt Ritchie

Examples

```
data(BSData)
BSData.norm = normaliseIllumina(BSData, method="quantile", transform="log2")
```

---

numBeads *Gets the number of beads from a BeadLevelList object*

Description

Retrieves the number of beads on selected strips/arrays from a `BeadLevelList` object.

Usage

```
numBeads(object, arrays=NULL)
```

Arguments

`object`	`BeadLevelList`
`arrays`	either `NULL` to return the bead numbers for all arrays, or a scalar or vector of integers specifying a subset of strips/arrays

Details

`numBeads` retrieves the number of beads on arrays from the arrayInfo slot.

Value

A vector containing the number of beads on individual strips/arrays.

Author(s)

Matt Ritchie

Examples

```
data(BLData)
numBeads(BLData)
numBeads(BLData, arrays=2)
```

outlierPlot *Locations of outliers on an Illumina chip*

Description

Diagnostic function that reports how many outliers are found on a specific array on a chip. We take advantage of the segmental structure of the array and break-down the number of outliers into 9 sections.

Usage

```
outlierPlot(BLData, array = array, log = FALSE, plot = FALSE)
```

Arguments

BLData	A BeadLevelList object containing the bead-level data for an Illumina experiment
array	The number of the array of interest
log	if TRUE calculate outliers on the log2 scale. If FALSE calculate outliers on the original scale
plot	if TRUE a diagnostic plot will be produced, otherwise only the numbers of outliers will be returned.

Details

The number of outliers are computed for the whole array using the Illumina default method that specifies a cut-off of 3 MADs from the median on either the log2 or original scale. These outliers are then split into 9 different sections on the array (the separation between these sections can usually be seen in the plots).

Author(s)

Mark Dunning

plotBeadDensities *plot densities of bead intensities*

Description

Function to produce smoothed density plots of the bead intensities from different bead strips/arrays.

Usage

```
plotBeadDensities(BLData, whatToPlot = "G", arrays = NULL, log = TRUE,
          type="l", col=1, xlab="Intensity", ylab="Density",
          xlim=NULL, ylim=NULL, ...)
```

Arguments

BLData	BeadLevelList
whatToPlot	character string specifying which intensities to plot. Possibilities are "G", "Gb" for single channel data and "G", "Gb", "R" and "Rb" for two-colour data
arrays	integer (scalar or vector) specifying the strip(s)/array(s) to plot. If set to NULL (default value), data from all strips/arrays are plotted
log	if TRUE log2 intensities are plotted
type	character string specifying what type of plot to draw
col	the colours for lines and points
xlab	label for x-axis
ylab	label for y-axis
xlim	numeric vector specifying x-axis limits
ylim	numeric vector specifying y-axis limits
...	further graphical parameters to plot

Details

Produces density plots of the raw intensities from a BeadLevelList When arrays="NULL", densities from all arrays are plotted.

Value

A plot is produced on the current graphical device

Author(s)

Matt Ritchie

Examples

```
data(BLData)
plotBeadDensities(BLData)
```

---

plotBeadIntensities

Plot bead intensities

Description

Function to plot the intensities of all beads of a particular type on a strip/array.

Usage

```
plotBeadIntensities(BLData, ProbeIDs, arrays, log = FALSE,
          whatToPlot="G", ProbeCols=NULL, ylim=NULL,...)
```

Arguments

BLData	BeadLevelList
ProbeIDs	numeric value giving the ProbeID(s) for the bead type(s) to plot
arrays	integer (scalar or vector) specifying the strip(s)/array(s) to plot
log	if TRUE log2 intensities are plotted
whatToPlot	character string specifying which intensities to plot. Possibilities are "G", "Gb" for single channel data and "G", "Gb", "R" and "Rb" for two-colour data
ProbeCols	vector of colour names to assign to each bead type
ylim	vector providing the upper and lower bounds for the y-axis in the plotting window
...	further graphical parameters to plot

Details

Boxplots of the specified intensities on the original or log-scale for each bead type specified by ProbeIDs on selected arrays is produced.

A vertical line separates the intensities from different strips/arrays.

Value

A plot is displayed on the current graphical device.

Author(s)

Mark Dunning

Examples

```
data(BLData)

plotBeadIntensities(BLData, arrays=1:4, ProbeIDs=c(2,1000), log=TRUE)
```

---

plotBeadLocations *Plot bead locations*

Description

Shows location of a set of beads on a strip/array. The beads can either be defined to be all beads with particular ProbeIDs or as rows in BeadLevelList.

Usage

```
plotBeadLocations(BLData, ProbeIDs = NULL, BeadIDs = NULL, array = 1,
                  SAM = FALSE, xlab = "x-coordinate",
                  ylab = "y-coordinate",
                  main = paste("Bead", ProbeIDs,"locations"), ...)
```

Arguments

BLData	BeadLevelList
ProbeIDs	a vector of ProbeIDs to plot
BeadIDs	logical/integer vector specifying which rows of data from BeadLevelList) to plot (used if ProbeIDs is NULL)
array	integer specifying the strip/array to plot
SAM	if TRUE then the data is assumed to be taken from a SAM array and therefore hexagonal
xlab	character string specifying x-axis label
ylab	character string specifying y-axis label
main	character string specifying plot title
...	further graphical parameters to plot

Details

The outline of the hexagonal array is drawn and the locations of the specified beads are overlayed.

Value

A plot is produced on the current graphical device.

Author(s)

Mark Dunning

Examples

```
data(BLData)

#plot all beads with ProbeID 2 on array 1
plotBeadLocations(BLData, array=1, ProbeIDs=2, SAM=TRUE)

#find all outliers on the first array and plot their locations
o=findAllOutliers(BLData, 1)
plotBeadLocations(BLData, BeadIDs=o, array=1, SAM=TRUE)
```

---

plotMA *Show MA plots*

Description

Function which produces an MA plot between two specified arrays.

Usage

```
plotMA(exprs, array1=1, array2=2, genesToLabel=NULL, labelCol="red", foldLine=2,
```

Arguments

exprs	a matrix of expression values
array1	integer specifying the first array to plot
array2	integer specifying the second array to plot
genesToLabel	vector of genes to highlight on the plot. These must match the rownames of exprs.
labelCol	plotting colours for highlighted genes
foldLine	a numeric value defining where to draw horizontal fold-change lines on the plot
log	if TRUE the data will be log-transformed before plotting
labelpch	plotting characters for highlighted genes
ma.ylim	numeric value specifying the range of the plot (from -ma.ylim to ma.ylim)
sampleSize	The number of genes to plot. Default is NULL, which plots every gene.
...	other graphical parameters to plot that can be specified

Details

The log2 difference in intensity (M-value, log-ratio) are plotted against the log2 average intensity (A-value) for each probe for the two arrays selected.

As a result of both having identical function names this function can conflict with the plotMA method in 'limma'. If both packages are loaded, the function from whichever package was loaded last takes precedence. If the 'beadarray' plotMA() function is masking that from 'limma', one can directly call the 'limma' method using the command "limma::plotMA()". Alternatively, one can detach the 'beadarray' package using "detach(package:beadarray)". Similar techniques can be used if 'limma' is masking the 'beadarray' method.

Value

A smoothed MA scatter plot is displayed on the current graphical device.

Author(s)

Mark Dunning

Examples

```
data(BSData)

plotMA(exprs(BSData), array1=1, array2=2)
```

plotMAXY *Scatter plots and MA-plots for all specified arrays*

Description

Produces smoothed scatter plots of M versus A and X versus Y for all pairwise comparisons from a set of arrays.

Usage

```
plotMAXY(exprs, arrays, log = TRUE, genesToLabel=NULL,
         labels=colnames(exprs)[arrays],labelCol="red",
         labelpch=16,foldLine=2,sampleSize=NULL,...)
```

Arguments

exprs	a matrix of expression values
arrays	integer vector giving the indices of the arrays (columns of exprs) to plot
log	if TRUE then all values will be log2-transformed before plotting
genesToLabel	vector of genes to highlight on the plot. These must match the rownames of exprs.
labels	vector of array names to display on the plot
labelCol	plotting colours for highlighted genes
labelpch	plotting characters for highlighted genes
foldLine	a numeric value defining where to draw horizontal fold change lines on the plot
sampleSize	The number of genes to plot. Default is NULL, which plots every gene
...	other graphical parameters to be passed

Details

This graphical tool shows differences that exist between two arrays and can be used to highlight biases between arrays as well as highlighting genes which are differentially expressed. For each bead type, we calculate the average (log2) intensity and difference in intensity (log2-ratio) for each pair of arrays.

In the lower-left section of the plot we see XY plots of the intensities for all pairwise comparisons between the arrays and in the upper right we have pairwise MA plots. Going down the first column we observe XY plots of array 1 against array 2 and array 1 against array 3 etc. Similarly, in the upper-right corner we can observe pairwise MA plots.

Author(s)

Mark Dunning

Examples

```
data(BSData)
plotMAXY(exprs(BSData), arrays=1:3)
```

`plotOnSAM` *Show variation between all 96 arrays*

Description

Function to show how quantities vary across all 96 arrays. eg mean intensity of a certain control probe

Usage

```
plotOnSAM(values, mx = max(values, na.rm = TRUE), scale = max(values,
na.rm = TRUE), min = 0, main = NULL, label = TRUE, missing_arrays = NULL, colour
```

Arguments

`values`	vector containing 96 numeric values to plot
`mx`	maximum value to display on y axis of plot
`scale`	numeric value giving the amount by which to divide all numbers by (eg for log2 intensities this should be 16) to transform to range 0 - 1
`min`	numeric value giving the minimum value to display on y axis
`main`	character string giving a title for the plot
`label`	boolean defining if the arrays are labeled on the plot
`missing_arrays`	vector of numeric values specifying the index of any arrays that have been removed from the SAM.
`colour`	if TRUE the hexagons will be plotted in colour

Details

Two plots are produced side-by-side. The first is a plot of the set of values against the index 1-96 and secondly we plot 8 x 12 hexagonal arrays with array number 1 being the hexagon in the top-left corner and array 96 in the bottom-right. The colour of hexagon is directly related to the value in v for the particular array number. An array which has a higher value in v will be coloured brighter.

Value

Plot is produced on current graphical device.

Author(s)

Mark Dunning

plotRG *Plot bead-level data: R vs G intensities*

Description

Plot R versus G intensities fom a BeadLevelList object.

Usage

```
plotRG(BLData, ProbeIDs=NULL, BeadIDs=NULL, log=TRUE, arrays=1,
                   xlim=c(8,16), ylim=c(8,16), xlab="G intensities",
                   ylab="R intensities",
                   main=arrayNames(BLData)[arrays], smooth=TRUE,
                   cols=NULL, ...)
```

Arguments

BLData	BeadLevelList
ProbeIDs	a vector of ProbeIDs to plot
BeadIDs	logical/integer vector specifying which rows of data from BeadLevelList to plot (used if ProbeIDs is NULL)
log	logical, if TRUE, take log base 2 of intensities
arrays	which array/s to plot
xlim	x-axis limits for plot
ylim	y-axis limits for plot
xlab	character string specifying x-axis label
ylab	character string specifying y-axis label
main	main plot title
smooth	logical, whether to smooth the points (only used when one array is selected for plotting)
cols	colours to use on the plot
...	further graphical parameters to plot

Details

The R and G intensities from selected beads and arrays are plotted.

Value

Plot is produced on the current graphical device.

Author(s)

Matt Ritchie

plotXY *XY plots for two samples*

Description

Function which produces an XY plot of the intensities from two specified arrays.

Usage

```
plotXY(exprs,array1=1, array2=2, genesToLabel=NULL, labelCol="red", log=TRUE,lab
```

Arguments

exprs	a matrix of expression values
array1	integer specifying the first array to plot
array2	integer specifying the second array to plot
genesToLabel	vector of genes to highlight on the plot. These must match the rownames of exprs.
labelCol	plotting colours for highlighted genes
log	if TRUE the data will be log-transformed before plotting
labelpch	plotting characters for highlighted genes
foldLine	a numeric value defining where to fold change lines on the plot
sampleSize	The number of genes to plot. Default is NULL, which plots every gene.
...	other graphical parameters to plot that can be specified

Details

Plots the array1 intensities versus the array2 intensities for all probes.

Value

A smoothed scatter plot is displayed on the current graphical device

Author(s)

Mark Dunning

Examples

```
data(BSData)

plotXY(exprs(BSData), array1=1, array2=2)
```

poscontPlot *Plot bead-level housekeeping and biotin controls*

Description

Function for retrieving and plotting the biotin and housekeeping controls for an expression array. We know these controls should show high signal and are therefore useful for QA purposes. The housekeeping control targets a bead-type believed to be universally expressed whereas the biotin control targets the biotin used for staining. By considering all bead observations (unlike the plots produced by BeadStudio) we get an detailed impression of array quality.

Usage

```
poscontPlot(BLData, array = 1, plot = FALSE)
```

Arguments

BLData	A BeadLevelList object for an Illumina expression array
array	The number of the array of interest
plot	if TRUE a diagnostic plot will be produced, otherwise only summary values are returned.

Details

The IDs for the housekeeping and biotin controls are retrieved making use of the annotation information stored for the array. If this information is incorrect, or missing, then the function will not proceed correctly. Valid names for the annotation slot must match the names of the ExpressionControlData object provided with beadarray. Assuming the controls can be identified, we then perform a detection test for each bead observation by computing a p-value: of 1-R/N, where R is the relative rank of the bead intensity when compared to the N negative controls. Thus, if a particular bead has higher intensity than all the negative controls it will be assigned a value of 0. After these p-values have been calculated for all replicates of the bead type we report the percentage of beads with p-values lower than a set threshold of 0.05 (currently in favour in the Illumina literature). The percentage of beads that are detected at a set threshold is then reported for the housekeeping and biotin controls respecitvely.

If plot is TRUE, the values of all bead observations are plotted for each bead-type respectively grouped together according to the type of control (housekeeping or biotin). Any beads with low intensity may be due to array defects and would be impossible to detect with summarized data only. If available, the intensities of any high stringency probes will also be plotted.

An important point to note that the utility of these housekeeping controls is dependant on the probe sequences used targeting the intended genes. If a particular housekeeping control shows lower expression, then it is worth checking the probe sequence by a BLAT search of by checking pre-complied tables from http://www.compbio.group.cam.ac.uk/Resources/Annotation/index.html.

Value

HkpDet	%age of housekeeping control beads that are detected compared to the negative controls.
BioDet	%age of biotin labelling control beads that are detected compared to the negative controls.

Author(s)

Mark Dunning and Andy Lynch

References

www.illumina.com/downloads/GX_QualityControl_TechNote.pdf

See Also

calculateBeadLevelScores, setAnnotation

---

probePairsPlot *QA plot using perfect match and mismatch controls*

Description

QA using the hybridisations controls that are identical expect for a few bases. Hence lower signal should be observed for the mismatch probes.

Usage

```
probePairsPlot(BLData, array = 1)
```

Arguments

BLData	A BeadLevelList object for an Illumina expression array
array	Number of the array in BLData that we want QA for

Details

Firstly, the annotation of the control probes found on the BeadLevelList object is retrieved. We then search the annoation for the perfect and mismatch controls (labeled phage_lambda_genome:pm and phage_lambda_genome:mm2). For each perfect match we find the corresponding mismatch control by comparing the sequences and plot the perfect match control next to the mismatch control.

Value

Plot on the current graphical device with the perfect match beads in red and mismatches in purple. The labels on the x-axis show the ID of the controls.

Author(s)

Mark Dunning

See Also

calculateBeadLevelScores

qcBeadLevel *Generate simple diagnostic plots for Illumina bead-level data*

Description

Generate simple diagnostic plots for Illumina bead-level data

Usage

```
qcBeadLevel(object, whatToPlot="G", RG=FALSE, log=TRUE, nrow = 100, ncol= 100,
                    colDens=1, colBox=0, html=TRUE,
                    fileName="qcsummary.htm", plotdir=NULL,
                    experimentName=NULL, targets=NULL, ...)
```

Arguments

object	BeadLevelList
whatToPlot	character string or vector specifying which intensities to plot. Possibilities are "G", "Gb" for single channel data and "G", "Gb", "R", "Rb" and "M" for two-colour data
RG	if TRUE, plot R vs G intensities per array. Default value is FALSE. Only useful for two-channel data
log	if TRUE log2 intensities are plotted
nrow	integer specifying the number of rows to divide the image into (used by imageplot function)
ncol	integer specifying the number of columns to divide the image into (used by imageplot)
colDens	colours for density plots (default is 1)
colBox	colours for box plot (default is 0)
html	logical scalar. If TRUE an html summary page is generated. If FALSE, no summary page is generated.
fileName	name of html summary page. Default is "qcsummary.htm".
plotdir	optional character string specifying the filepath where the plots will be saved. Defaults to the current working directory.
experimentName	name to appear on HTML report (default is NULL).
targets	data.frame containing sample information
...	further arguments that can be passed to the plotting functions.

Details

This function creates boxplots, smoothed histogram (density) plots and imageplots of raw bead-level intensity data.

An html page which displays the results, is created when html=TRUE. The html page name is specified by the fileName argument.

Author(s)

Matt Ritchie

Examples

```
#data(BLData)
#qcBeadLevel(BLData)
```

---

rankIvariantNormalise

Rank Invariant normalise data in a matrix

Description

Normalise expression matrix using rank invariant genes

Usage

```
rankInvariantNormalise(exprs, T)
```

Arguments

exprs	a matrix of expression values
T	A target distribution vector to normalise the data to. The default is NULL in which case the average quantiles are used

Details

Using the `normalize.invariantset` function from the affy package, we find a list of rank invariant genes whose rank does not change significantly between the columns of `exprs`. We then fit a normalising curve to each array using the values of the rank invariant genes of the array and a target distribution.

The target distribution may be specified by the user and by default is the vector of average quantiles across all arrays.

Value

Matrix of normalised expression data with the same dimensions as `exprs`.

Author(s)

Mark Dunning

Examples

```
data(BSData)

BSData.ri  = assayDataElementReplace(BSData, "exprs", rankInvariantNormalise(exprs(BSData
```

readBeadSummaryData

Read BeadStudio gene expression output

Description

Function to read the output of Illumina's BeadStudio software into beadarray

Usage

```
readBeadSummaryData(dataFile, qcFile=NULL, sampleSheet=NULL,
                    sep="\t", skip=8, ProbeID="ProbeID",
                    columns = list(exprs = "AVG_Signal", se.exprs="BEAD_STDERR",
                        NoBeads = "Avg_NBEADS", Detection="Detection Pval"),
                    qc.sep="\t", qc.skip=8, controlID="ProbeID",
                    qc.columns = list(exprs="AVG_Signal", se.exprs="BEAD_STDERR"
                        NoBeads="Avg_NBEADS", Detection="Detection Pval"),
                    annoPkg=NULL, dec=".", quote="")
```

Arguments

dataFile	character string specifying the name of the file containing the BeadStudio output for each probe on each array in an experiment (required). Ideally this should be the 'SampleProbeProfile' from BeadStudio.
qcFile	character string giving the name of the file containing the control probe intensities (optional). This file should be either the 'ControlProbeProfile' or 'ControlGeneProfile' from BeadStudio.
sampleSheet	character string used to specify the file containing sample infomation (optional)
sep	field separator character for the dataFile ("\t" for tab delimited or "," for comma separated)
skip	number of header lines to skip at the top of dataFile. Default value is 8.
ProbeID	character string of the column in dataFile that contains identifiers that can be used to uniquely identify each probe
columns	list defining the column headings in dataFile which correspond to the matrices stored in the assayData slot of the final ExpressionSetIllumina object
qc.sep	field separator character for qcFile
qc.skip	number of header lines to skip at the top of qcFile
controlID	character string specifying the column in qcFile that contains the identifiers that uniquely identify each control probe
qc.columns	list defining the column headings in qcFile which correspond to the matrices stored in the QCInfo slot of the final ExpressionSetIllumina object
annoPkg	character string specifying the name of the annotation package (only available for certain expression arrays at present)
dec	the character used in the dataFile and qcFile for decimal points
quote	the set of quoting characters (disabled by default)

Details

This function can be used to read gene expression data exported from versions 1,2 and 3 of the Illumina BeadStudio application. The format of the BeadStudio output will depend on the version number. For example, the file may be comma or tab separated of have header information at the top of the file. The parameters `sep` and `skip` can be used to adapt the function as required (i.e. skip=7 is appropriate for data from earlier version of BeadStudio, and skip=0 is required if header information hasn't been exported.

The format of the BeadStudio file is assumed to have one row for each probe sequence in the experiment and a set number of columns for each array. The columns which are exported for each array are chosen by the user when running BeadStudio. At a minimum, columns for average intensity standard error, the number of beads and detection scores should be exported, along with a column which contains a unique identifier for each bead type (usually named "ProbeID").

It is assumed that the average bead intensities for each array appear in columns with headings of the form 'AVG_Signal-ARRAY1', 'AVG_Signal-ARRAY2',...,'AVG_Signal-ARRAYN' for the N arrays found in the file. All other column headings are matched in the same way using the character strings specified in the `columns` argument.

NOTE: With version 2 of BeadStudio it is possible to export annotation and sequence information along with the intensities. We _don't_ recommend exporting this information, as special characters found in the annotation columns can cause problems when reading in the data. This annotation information can be retrieved later on from other Bioconductor packages.

The default object created by readBeadSummaryData is an `ExpressionSetIllumina` object.

If the control intensities have been exported from BeadStudio ('ControlProbeProfile') this may be read into beadarray as well. The `qc.skip`, `qc.sep` and `qc.columns` parameters can be used to adjust for the contents of the file. If the 'ControlGeneProfile' is exported, you will need to set `controlID="TargetID"`.

Sample sheet information can also be used. This is a file format used by Illumina to specify which sample has been hybridised to each array in the experiment.

Note that if the probe identifiers are non-unique, the duplicated rows are removed. This may occur if the 'SampleGeneProfile' is exported from BeadStudio and/or `ProbeID="TargetID"` is specified (the "ProbeID" column has a unique identifier in the 'SampleProbeProfile', whereas the "TargetID" may not, as multiple beads can target the same transcript).

Value

An `ExpressionSetIllumina` object.

Author(s)

Mark Dunning and Mike Smith

See Also

`ExpressionSetIllumina`

Examples

```
##code to read the example BeadStudio (version 2) output distributed with the package
#dataFile = "SampleProbeProfile.txt"
#sampleSheet = "SampleSheet.csv"
#qcFile = "ControlGeneProfile.txt"
#BSData =readBeadSummaryData(dataFile, qcFile=qcFile, sampleSheet=sampleSheet, controlID=
```

readBGX *Read Illumina .bgx file into R*

Description

Reads in an unzipped Illumina .bgx file, which provides further information on each bead type, including the controls.

Usage

```
readBGX(filename, path=".", sep="\t", quote="", header=TRUE,
           probeStart="\\[Probes\\]", controlStart="\\[Controls\\]", ...)
```

Arguments

filename	character vector specifying name of unzipped bgx file
path	character string specifying the location of the bgx file
sep	separator character (default is tab, "\t")
quote	character string specifying the quoting characters (disabled by default with quote=""). See scan for further information.
header	logical, TRUE if the bgx file has a header, FALSE otherwise
probeStart	character string, below which the information for the beads of interest appear. Default value is "\ $Probes\backslash$ "
controlStart	character string, below which the information for the control beads appear. Default value is "\ $Controls\backslash$ "
...	further arguments to read.table

Details

The .bgx file is a zip file which contains information about each probe on an expression BeadArray.

To read in the file, you first need to unzip it. To do this, replace the .bgx extension with .zip (for example rename HumanRef-8_V2_0_R0_11223162_A.bgx as HumanRef-8_V2_0_R0_11223162_A.zip) and then unzip this file (which should leave one file HumanRef-8_V2_0_R0_11223162_A for our example). The unzipped file is tab delimited file and should be readable using readBGX. At present this should work for Human and Rat expression arrays. For Mouse arrays, the .bgx has a more complicated structure.

Value

data.frame containing information about each bead type (probe sequence, ID, control status, etc)

Author(s)

Matt Ritchie

Examples

```
#human8bgx = readBGX("HumanRef-8_V2_0_R0_11223162_A", fill=TRUE)
#colnames(human8bgx)
#summary(human8bgx$Status)
#human6bgx = readBGX("HumanWG-6_V2_0_R0_11223189_A", fill=TRUE)
#ratbgx = readBGX("RatRef-12_V1_0_R0_11222119_A", fill=TRUE)
```

---

readIllumina *Read bead-level data into R*

Description

Uses .csv or .txt and TIFFs (where available) to load information about each bead on each array in a BeadChip or SAM experiment.

Usage

```
readIllumina(arrayNames=NULL, path=".", textType=".txt",
        annoPkg=NULL, useImages=TRUE,
        singleChannel=TRUE, targets=NULL,
        imageManipulation="sharpen", backgroundSize=17,
        storeXY=TRUE, sepchar="_", dec=".", metrics=FALSE,
        metricsFile="Metrics.txt", backgroundMethod="subtract",
        offset=0, normalizeMethod="none",
        tiffExtGrn="_Grn.tif", tiffExtRed="_Red.tif", ...)
```

Arguments

`arrayNames`	character vector containing names of arrays to be read in. If `NULL`, all arrays that can be found in the current working directory will be read in.
`path`	character string specifying the location of files to be read by the function
`textType`	character string specifiying the extension of the files which store the bead-level information. Typically `".txt"`, `".csv"` or `"_perBeadFile.txt"`.
`annoPkg`	character string specifying the annotation package for the arrays being read in (only available for certain expression arrays at present). Default value is `"illuminaProbeIDs"` which is not an annotation package, and indicates that Illumina bead IDs have been used to identify each bead.
`useImages`	logical. If TRUE, the foregound and background values are retrieved from the TIFFs. When FALSE, the intensity values in the text files are used. Note that background values will not be available (set to 0) when FALSE, as the current option in BeadScan is to store background corrected intensities.
`singleChannel`	logical. Set to TRUE if the data is single channel (Green images only) or FALSE for two-colour (both Green and Red data available).
`targets`	`data.frame` containing sample information
`imageManipulation`	character string specifying which image analysis method to use. The current options are `"none"` (no image manipulation or `"sharpen"` (the Illumina sharpening mask will be used prior to the foreground averages being calculated).

backgroundSize
: integer value which defines the size of the n x n region (in pixels) used to calculate local background values. Default value is 17

storeXY
: logical scalar, indicating whether the xy coordinates should be stored

sepchar
: character string which separates the row and column positions in the file names (default value is "_")

dec
: character used in the files for decimal points. The default value is "."

metrics
: logical scalar, indicating whether the scanner metrics file metricsFile is to be read in

metricsFile
: name of the scanner metrics file ("Metrics.txt" by default)

backgroundMethod
: method to use for background correcting the data. Options are "none", "subtract", "half", "minimum", "edwards", "normexp" or "rma"

offset
: numeric value to add to intensities

normalizeMethod
: method to use to normalize the background corrected bead-level data. Options are "none", "quantile" and "vsn". Note that the normalization occurs at the bead-level and is only available for two-colour data at this stage

tiffExtGrn
: character string specifying the file extension of the Cy3 (Green) images. Default is "_Grn.tif"

tiffExtRed
: character string specifying the file extension of the Cy5 (Red) images (where present). Default is "_Red.tif"

...
: other arguments

Details

This function can be used to read in bead-level information from the raw .tif and .csv or .txt files output by BeadScan.

Note that the .txt or .csv files must contain the raw data for each bead on each array/strip, not the summarised data. The .csv or .txt files specify the location and identity of each bead on the array and must contain columns for the x and y position of each bead as well as a ProbeID. For two-colour arrays, this information is required for each channel.

The foreground and background intensities of each bead are calculated from the images when useImages=TRUE. For the foreground calculations the sharpening mask used by Illumina is applied prior to averaging over the 9 pixels in a 3 x 3 square closest to the bead centre by default (imageManipulation="sharpen"). If imageManipulation="none", no sharpening is performed. The local background estimate for each bead is calculated by averaging the 5 minimum pixels in a 17 x 17 square around each bead centre from the unsharpened image. The size of this window is controlled by the backgroundSize argument. If a bead is too close to the edge of the image, it is ignored.

When useImages=FALSE, the intensities from the .txt or .csv files are stored as the foreground values for each bead. Note that the values stored in these files have already undergone a local background adjustment, so the background values are set to 0.

To keep track of the samples hybridised to each array, we recommend using a targets data.frame, which lists each strip/array in the rows, and experimental information about each strip/array in the columns (sample, array name, etc.) Targets information can easily be created and saved in tab delimited text format, read in using read.table and passed to readIllumina using the targets argument.

The pairs of strips from a BeadChip can be combined when the data is summarised with `createBeadSummaryData`.

The function creates a `BeadLevelList` containing foreground and background intensities for each bead on each array.

NOTE: Reading in bead-level data, particularly with the TIFFs is memory intensive. For example, reading in text and image data from a Human-6 BeadChip requires several Gigabytes of RAM. If you have limited memory, it is recommnded that you read in the data using the `useImages=FALSE` option.

Value

BeadLevelList object

Author(s)

Mark Dunning, Mike Smith

Examples

```
#BLData = readIllumina()

#targets = read.table("targets.txt", header=T)

#targets
#May take a while to run
#BLData.s = readIllumina(arrayNames=target$Institute.Sample.Label, targets=targets, image

#Create foreground intensities without using sharpening. Should take less time
#BLData.ns = readIllumina(arrayNames=targets$Institute.Sample.Label, targets=targets, ima
```

`readQC` *Read Illumina control intensities*

Description

Reads the standard format of Illumina control intensities output by BeadStudio

Usage

```
readQC(file, sep="\t", skip=8, controlID = "ProbeID", columns = list(exprs = "AV
```

Arguments

`file`	character string giving the name of the file output by BeadStudio containing the control probe intensities. This file should be either the 'ControlProbeProfile' or 'ControlGeneProfile'.
`sep`	a character string for the file separator
`skip`	number of lines of header information to ignore in the file
`controlID`	character string specifying the column that contains the (unique) control probe IDs

columns	a vector of column names to read from the file
dec	the character used in the file for decimal points
quote	the set of quoting characters (disabled by default)

Details

The format of the quality control files differs slightly between BeadStudio versions 1 and later versions. This function is able to read in data in either format

Note that if the control identifiers are non-unique, the duplicated rows are removed. This may occur if the 'ControlProbeProfile' is exported from BeadStudio and `controlID="TargetID"` is specified (the "ProbeID" column has a unique identifier in the 'ControlProbeProfile', whereas the "TargetID" may not, as multiple beads can be of the same type).

Once read in, the control intensities can be used for quality assessment purposes.

Value

`readQC` produces an `assayData` object with a list of items defined by the `columns` parameter. The row names of each matrix are given by the `controlID` argument .

Author(s)

Mark Dunning

Examples

```
##Code to read the example quality control file included with the
#package.
#QC = readQC("ControlGeneProfile.txt", controlID="TargetID")
#the average expression of each control can then be accessed by the $ operator
#QC$exprs
```

---

setWeights *Set BeadLevelList Weights*

Description

Replaces the weights of a BeadLevelList with user-specified ones.

Usage

```
setWeights(BLData, wts, array, combine=FALSE)
```

Arguments

BLData	BeadLevelList
wts	either a numerical vector of weights to use, or 0 or 1 to set all weights to 0 or 1.
array	integer specifying the strip/array to use
combine	logical. If `TRUE`, the new weights specified by `wts` are combined with the existing weights by storing the minimum of the two for each bead. If `FALSE` the new weights replace any existing weights.

Details

This function replaces the weights column, `wts`, on the specified array, with user-specified values.

Only rows with `wts != 1` are used in `createBeadSummaryData`.

Value

`BeadLevelList` object, with updated `wts` values.

Author(s)

Mark Dunning

Examples

```
data(BLData)
BLData <- setWeights(BLData,1,1) ##set all weights to 1
BLData <- setWeights(BLData,0,1) ##set all weights to 0
```

---

viewBeads *View Beads*

Description

View an image of the beads in a certain region, optionally with links between neighbours or certain beads highlighted.

Usage

```
viewBeads(BLData, array, x, y, xwidth = 100, ywidth = 100, neighbours = NULL, ma
```

Arguments

`BLData`	`BeadLevelList`
`array`	integer specifying which strip/array to plot
`x`	numeric value - x co-ordinate to centre on
`y`	numeric value - y co-ordinate to centre on
`xwidth`	numeric value - width of square
`ywidth`	numeric value - width of square
`neighbours`	Neighbours matrix (optional) - if specified, links will be drawn between neighbours. (See `generateNeighbours`.)
`mark`	integral vector (optional) - a list of beadIDs to highlight.
`markcol`	The colour used for the highlighted beads.
`markpch`	The colour used for the highlighted beads.
`inten`	logical - if true, plot coloured circles, with shades corresponding to intensities. Intensities are retrieved using `getArrayData`, and the arguments below.
`low`	Colour used for low intensities.
`high`	Colour used for high intensities.

what	Data to be used - passed to getArrayData.
log	Data to be used - passed to getArrayData.
zlim	Limits to be used. Supply in form c(0,5).
...	Additional arguments passed to getArrayData.

Details

viewBeads plots the beads within the defined square region.

Specifying a neighbours matrix will result in links between neighbours being plotted. Specifying a mark vector of beadIDs will result in the beads with these beadIDs being highlighted with a blue circular border.

Value

Outputs to the active graphical device.

Author(s)

Jonathan Cairns

See Also

generateNeighbours

Examples

```
##data(BLData)
##o <- findAllOutliers(BLData,2)

##x11()
##viewBeads(BLData, 2, 1000,1200,250,250, mark = o) ## outliers in this ##region are mark
##x11()
##viewBeads(BLData, 2, 1000,1200,250,250, mark = o, inten = FALSE) ## ##removing the inte

##neighbours <- generateNeighbours(BLData, 2)
##x11()
##viewBeads(BLData, 2, 1000,1200,250,250, neighbours) ## observe that ##there are many mi
```

Index