beadarray

April 20, 2011

`backgroundCorrectSingleSection`
Background correct an array-section

Description

Function to perform background correction using a defined set of foreground and background intensities.

Usage

```
backgroundCorrectSingleSection(BLData, array = 1, fg="Grn", bg="GrnB", newName =
```

Arguments

`BLData`	a `beadLevelData` object
`array`	the number of the array-section to be corrected
`fg`	the name under which the foreground intensities are stored
`bg`	the name under which the background intensities are stored
`newName`	Name to store the corrected intensities

Details

After the foreground and background intensities are retrieved, the function calculates foreground - background

Value

`beadLevelData` object with modified beadData slot for the particular section

Author(s)

Mark Dunning

Examples

```
##This will take the "Grn" and "GrnB" data for each section and save the corrected intens

data(BLData)

head(BLData[[1]])

for(i in 1:10){
 BLData = backgroundCorrectSingleSection(BLData, array=i)
}


head(BLData[[1]])
```

BASHCompact *BASH - Compact Defect Analysis*

Description

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

Usage

```
BASHCompact(BLData, array, neighbours = NULL, useLocs = TRUE, transFun = logGree
```

Arguments

BLData	BeadLevelList
array	integer specifying which strip/array to plot
neighbours	A Neighbours matrix. Optional - if left NULL, it will be computed.
useLocs	logical value, specifiying whether the .locs file (if present) should be used to determine neighbours.
transFun	function to use to transform data prior to running BASH.
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.

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

`BASH`, `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, transFun = logGreenChannelTransform, neighbours = NUL
```

Arguments

`BLData`	`BeadLevelList`
`array`	integer specifying which strip/array to plot
`transFun`	Function to use to transform data prior to running BASH.
`neighbours`	A Neighbours matrix. Optional - if left NULL, it will be computed, using default `generateNeighbours` settings.

useLocs	Logical value, specifiying whether the .locs file (if present) should be used to determine neighbours.
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

BASH, 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, transFun = logGreenChannelTransform, neighbours = NU
```

Arguments

BLData	BeadLevelList
array	integer specifying which strip/array to plot
transFun	Function to use to transform data prior to running BASH.
neighbours	A Neighbours matrix. Optional - if left NULL, it will be computed, using default generateNeighbours settings.
useLocs	Logical value, specifiying whether the .locs file (if present) should be used to determine neighbours.
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, generateNeighbours,

Examples

```
data(BLData)
extended <- BASHExtended(BLData, 1)
```

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, transFun = logGreenChannelTransform, compact = TRUE, diffuse
```

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.
transFun	function to use to transform data prior to running BASH
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.
useLocs	Logical - If TRUE then a .locs file corresponding to the array is sought and, if found, used to identify the neighbouring beads. If FALSE the neighbours are infered algorithmically. See generateNeighbours for more details.

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

J. M. Cairns, M. J. Dunning, M. E. Ritchie, R. Russell, and A. G. Lynch (2008). BASH: a tool for managing BeadArray spatial artefacts. Bioinformatics 15; 24(24)

Examples

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

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

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

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

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

---

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

Examples

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

---

beadStatusVector *Classify each bead according to its control status*

Description

Using the control annotation specified for the array, the function will classify each bead as belonging to a control group, or being a regular probe.

Usage

```
beadStatusVector(BLData, array = 1, controlProfile = NULL)
```

Arguments

`BLData`	a `beadLevelData` object
`array`	the numeric id of the section
`controlProfile`	an optional control profile data frame

Details

The function requires a control profile data frame which must be specified or can be created automatically using the annotation of the `beadLevelData` object.

Value

a vector of character strings giving the status of each bead

Author(s)

Mark Dunning

Examples

```
data(BLData)

data(controlProfile)

statVec = beadStatusVector(BLData, controlProfile=controlProfile)

table(statVec)
```

BLData *beadLevelData object from an example experiment*

Description

BLData is an object of class beadLevelData which contains data from an experiment with 10 arrays.

Usage

```
data(BLData)
```

See Also

beadLevelData

BSData *ExpressionSetIllumina object for the example experiment*

Description

BSData is an object of class ExpressionSetIllumina which a summarized version of the bead-level data distributed with the package

Usage

```
data(BSData)
```

`calculateDetection` *Calculate detection scores*

Description

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

Usage

```
calculateDetection(BSData, status=fData(BSData)$Status, negativeLabel="negative"
```

Arguments

`BSData`	An ExpressionSetIllumina object
`status`	character vector giving probe types
`negativeLabel`	character giving identifer for negative controls

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.

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

```
##This example data does not have a Status column defined, so we have to determine it fro

data(BSData)
data(controlProfile)

status = rep("regular", nrow=dim(BSData)[1])

negIDs = controlProfile[which(controlProfile[,2] == "negative"),1]

status[match(negIDs, featureNames(BSData))] = "negative"
```

```
det = calculateDetection(BSData, status)

Detection(BSData) = det
```

`calculateOutlierStats` *Outlier distribution stats*

Description

Function that determines the outlier beads on an array and how they are distributed among the segments

Usage

```
calculateOutlierStats(BLData, array = array, transFun = logGreenChannelTransform
```

Arguments

`BLData`	a `beadLevelData` object
`array`	the number of the array of interest
`transFun`	how the section data is to be transformed prior to calculating outliers
`outlierFun`	a function for calculating outliers
`useLocs`	use locs and sdf information (if available) to determine section layout
`nSegments`	manually set how many segments the section is divided into

Details

A section of an expression BeadChip (e.g. the Humanv3 or HumanHT-12) is made up of 9 physcially-separate segments. A useful QA check is to see how the outliers are distributed among these segments. Outliers are beads that have outlying intensities according to some rule that the user can specify. The default (as used by Illumina) is to exclude beads that are more than 3 median absolute deviations from the median. Once outliers are determined, the coordinates for these outliers are binned into segments by assuming that the segments are evenly spaced across the section surface.

Note that sections from Sentrix Array Matrix do not have segments, so the results may not be informative

Value

vector with the percentage of beads found in each segment that were determined to be outliers

Author(s)

Mark Dunning

Examples

```
data(BLData)

##Artificial example, there are no segments on this type of BeadArray

calculateOutlierStats(BLData, array=1, nSegments=10, useLocs=FALSE)

calculateOutlierStats(BLData, array=2, nSegments=10, useLocs=FALSE)

calculateOutlierStats(BLData, array=3, nSegments=10, useLocs=FALSE)
```

---

checkRegistration *Perform check for misregistered array segments.*

Description

Occasionally arrays the registration of an array can go wrong, with the bead centres found in the wrong place in an image. The effective result of this is a scrambling of the bead IDs. In order to check for this we can examine the within bead-type variance across the array. In cases where registration has failed we would expect to see a large jump in this value compared to correctly registered arrays. This function computes this statistic for each array segment (since each segment is registered independently) and returns them to the user for inspection.

Usage

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

Arguments

BLData	An object of class beadLevelData.
array	Integer specifying the index of the arrays to be checked. Can be a vector to process multiple arrays e.g. 1:12.

Value

Returns a list with an entry for each section specified by the array argument. Within each of these entries contains a vector storing the mean within bead-type for each segment of the array. If the array is two-colour then there will be seperate entries for the green and red channel as these are registered separately by the scanner.

Author(s)

Mike Smith

References

Smith ML, Dunning MJ, Tavare S, Lynch AG. Identification and correction of previously unreported spatial phenomena using raw Illumina BeadArray data. BMC Bioinformatics (2010) 11:208

`beadLevelData-class` *Class "beadLevelData"*

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("beadLevelData")`, but are usually created by `readIllumina`.

Slots/List Components

Objects of this class contain the following slots

`beadData`:	A list of arrays, indexed by array name. Each item in this list is itself a list, containing enviromen
`sectionData`:	a list containing information. Each item in the list is a data frame containing one row for each sec
`experimentData`:	a list containing the annotation of the platform, link to the sdf file and type of data (slide or Sentri
`history`:	Character vector storing the operations performed on this object.

Methods

show(beadLevelData) Printing method for BeadLevelList

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

`sectionNames(object,arrays=NULL)` Returns the strip/array names from a

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

Accessing data from the class

`getBeadData` retrieve data

`insertBeadData` Input or modify existing data

Author(s)

Mark Dunning, Mike Smith

See Also

`readIllumina`

Examples

```
data(BLData)

sectionNames(BLData)

head(BLData[[1]])
```

```
getBeadData(BLData, array=1, what="Grn")[1:10]
```

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 dat
`phenoData`:	an 'AnnotatedDataFrame' containing experimental information.
`arrayInfo`:	a list containing array information.
`annotation`:	character storing annotation package information.

Methods

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

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

Author(s)

Mark Dunning and Matt Ritchie

ExpressionSetIllumina-class

Class "ExpressionSetIllumina"

Description

Container for high-throughput assays and experimental metadata. `ExpressionSetIllumina` class is derived from `eSet`, and requires matrices `exprs`, `se.exprs`, `nObservations`, `Detection` as assay data members. The slots `featureData`, `phenoData` are accessed in the usual manner using `fData` and `pData` functions.

For ExpressionSetIllumina objects created from bead-level data (using the summarize function), a QC slot is used to contain any quality control data that was present in the beadLevelData object. This is a change from previous versions of beadarray, where the intensities of the control probes themselves were stored in this slot. From version 2.0.0 onwards, control probes are stored in the assayData slot with the regular probes and the featureData slot has a reference for which rows correspond to controls.

The ExpressionSetIllumina class is able to accomodate different channels when created from bead-level data. The channelNames function may be used to find out what channels are present in the object. The channel function can be used to select a particular channel, returning an ExpressionSetIllumina object.

Author(s)

Mark Dunning

---

illuminaChannel-class

Class "illuminaChannel"

Description

A class to define how illumina bead-level data are summarized

Details

From beadarray version 2.0 onwards, users are allowed more flexibility in how to create summarized data from bead-level data. The illuminaChannel is a means of allowing this flexibility by defininining how summarization will be performed on each array section in the bead-level data object. The three keys steps applied to each section are; 1) use a transform function to get the quantities to be summarized (one value per bead). The most common use-case would be to extract the Green channel intensities and possibly perform a log2 transformation. 2) remove any outliers from this list of values 3) split the values according to ArrayAddressIDs and apply the definied exprFun and varFun to the quantities belonging to each ArrayAddress.

Slots/List Components

Objects of this class contain the following slots

transFun:	function to transform the data from each array-section.
outlierFun:	A function for identifying outliers from a list of bead intensiites and associated ArrayAddressIDs .
exprFun:	A function for producing a single summary of expression level from a vector of bead-type intensities. e
varFun:	A function for producing a single summary of variability from a vector of bead-type intensities. e.g. sd
name:	Character vector that defines a name for the channel

Author(s)

Mark Dunning

See Also

summarize

Examples

```
greenChannel = new("illuminaChannel", greenChannelTransform, illuminaOutlierMethod, mean,
redChannel = new("illuminaChannel", redChannelTransform, illuminaOutlierMethod, mean, sd,

logRatio = new("illuminaChannel", logRatioTransform, illuminaOutlierMethod, mean, sd, "M"

data(BLData)

BSData = summarize(BLData, channelList = list(greenChannel))
```

---

controlProbeDetection

Percentage of beads detected

Description

Function to calculate the percentage of beads matching a defined set of control types that are detected above background level on an array-section.

Usage

```
controlProbeDetection(BLData, transFun = logGreenChannelTransform, array = 1, co
```

Arguments

BLData	a beadLevelData object
transFun	transformation to be applied to data
array	a numeric index of the array section
controlProfile	optional data frame defining ArrayAddressIDs belonging to each control type
tagsToDetect	vector of character strings defined which control types to interrogate
negativeTag	character string defining which control type to use as background
detThresh	numeric value for threshold for detection

Details

Details of the controls on the array-section can be inferred from the annotation of the beadLevelData object or supplied as a data frame. The first column of the data frame should contain ArrayAddressIDs, with the control type of the each ID in the second column. The strings supplied in the tagsToDetect and negativeTag parameters should be present in this column.

The ArrayAddressIDs that correspond to the specified tags are matching to the ArrayAddressIDs for the chosen array and intensities for all beads are extracted. 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.

The percentage reported is the percentage of beads of each control type that are detected at the defined threshold.

Author(s)

Mark Dunning

Examples

```
data(BLData)
data(controlProfile)

for(i in 1:10){
print(controlProbeDetection(BLData, array = i, controlProfile=controlProfile, tagsToDetec
}
```

`controlProfile` *Control annotation for Illumina expression chips*

Description

Data frame defining control information for the example `BLData` object included with the beadarray package. The first column contains ArrayAddressIDs to be found in the `beadLevelData` object and which control type they represent.

Usage

```
data(controlProfile)
```

Examples

```
library(beadarray)
data(controlProfile)
head(controlProfile)
table(controlProfile)
```

`convertBeadLevelList` *Convert a BeadLevelList object into a beadLevelData object*

Description

As of beadarray version 2.0 the BeadLevelList class has been deprecated and replaced by the beadLevelData class. Whilst these are superficially similar, the way the data are stored is quite different, meaning most functionality within the package is no longer compatible with the original BeadLevelList class.

This function converts any object that is of the old BeadLevelList class into a beadLevelData object.

Usage

```
convertBeadLevelList(BeadLevelList)
```

Arguments

`BeadLevelList` An object of class BeadLevelList

Value

Returns an object of class beadLevelData.

Author(s)

Mike Smith

See Also

`beadLevelData-class`

`createTargetsFile` *A function to generate a targets file given a directory of Illumina bead-level files*

Description

This function, when pointed to a directory containing Illumina bead-level files (e.g. txt, idat, locs, tif) will return a simple targets file of the sort expected by beadarray. Note that a user created targets file is likely to be of greater value.

Usage

```
createTargetsFile(dir = NULL, nochannels = 1, channel1 = "Grn", channel2 = "Red"
```

Arguments

dir	dir: The directory containing the Illumina bead-level files. By default, will search the working directory.
nochannels	nochannels: Does the directory contain 1 or 2 channel arrays? Setting this argument to be null will result in the function making its best guess.
channel1	channel1: The string indicating that files are associated with the first channel (usually Grn).
channel2	channel2: The string indicating that files are associated with the second channel (usually Red).
txtsuff	txtsuff: The suffix of files containing the bead-level intensities (usually txt, but occasionally csv).
imgsuff	imgsuff: The suffix of files containing the images.
locssuff	locssuff: The suffix of files containing the precise bead locations (usually locs).
xmlsuff	xmlsuff: The suffix of files containing the meta-data (usually xml).
verbose	verbose: Determines whether or not the function reports on its progress as it goes along.
special	special: Files with names containing special words (such as fiducial) are ignored.
ColourConfusionStop	ColourConfusionStop: This determines the behaviour of the function if there is a discrepancy between the number of channels specified, and the number apparently present.
metricsflag	codemetricsflag: This gives the key word that can be used to identify metrics files.
metsep	metsep: This gives the cell separator used in the metrics file.
metricsection	metricsection: This gives the column heading used in the metrics file to indicate array section names.
metricchip	metricchip: This gives the column heading used in the metrics file to indicate the chip name.

Details

This function bases its resultant targets file on the files with suffix txtsuff.

Value

This returns a dataframe containing

Author(s)

Andy Lynch

See Also

readIlluminaData()

Examples

```
#createTargetsFile(verbose=T)
```

`deprecatedFunctions` *Deprecated Functions*

Description

Deprecated functions kept so they package ArrayQualityMetrics can still be installed. All have been replaced and will be removed in future updates.

Author(s)

Mark Dunning

`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,]
```

`expressionQCPipeline` *Flexible bead-level QC pipeline*

Description

Function to produce various QC plots and HTML summary pages for bead-level data.

Usage

```
expressionQCPipeline(BLData, transFun = logGreenChannelTransform, qcDir = "QC",
```

Arguments

`BLData`	a `beadLevelData` object
`transFun`	what transformation function to apply
`qcDir`	a directory to write output to
`plotType`	desired file extension for plots (jpeg or png)
`horizontal`	if TRUE imageplots and outlier plots are produced with longest edge on x axis
`controlProfile`	a data frame defining all control types. not required if annotation information is stored in the bead-level object
`overWrite`	if FALSE any plots that exist in the directory will not be recreated
`nSegments`	how many segments each section is divided into
`outlierFun`	a function to removed outliers
`tagsToDetect`	which control types to used in the detection metrics
`zlim`	the range of the imageplots
`boxplotFun`	what transformation function to be used in boxplots
`imageplotFun`	what transformation function to be used for imageplots
`positiveControlTags`	character strings defining which positive controls to plot
`hybridisationTags`	additional control types to be plotted
`negativeTag`	character string to identify which control type in the control profile corresponds to negative controls
`...`	other plot arguments

Details

This function is a convient way of automatically generating QC plots for each section within a `beadLevelData` object. The following plots are produced for each section. i) scatter plots of all bead observation of the positive controls. See `poscontPlot`. ii) Further scatter plots of other controls of interest using `poscontPlot`. iii) imageplot (`imageplot`) of section data after applying transformation function iv) plot of outlier locations using specified outlier function. A HTML page displaying all the plots is produced.

After plots have been produced for each section, makeQCTable is run to make a table of mean and standard deviations for the defined control types, followed by the results of calculateOutlierStats and controlProbeDetection for each section and written to a HTML page in the requested directory.

The function should be able to run automatically for expression data that has its annotation stored using setAnnotation or using readIllumina. Otherwise the controlProfile data frame can be used to define the control types on the array and their associated ArrayAddressIDs. Similarly, the function assumes single-channel data but a transformation function can be passed.

Author(s)

Mark Dunning

See Also

poscontPlot imageplot outlierplot controlProbeDetection

Examples

```
data(BLData)
data(controlProfile)

#Not Run
#expressionQCPipeline(BLData, controlProfile=controlProfile, positiveControlTags=c("house
```

---

generateNeighbours *Generate matrix of neighbouring beads*

Description

Generates a neighbours matrix from either a .locs file or the X and Y coordinates in a beadLevelData object.

Usage

```
generateNeighbours(BLData, array = 1, useLocs = TRUE, window = 30, margin = 10,
```

Arguments

BLData	An object of class beadLevelData
array	integer specifying which strip/array to process
useLocs	logical value, specifiying whether the .locs file (if present) should be used to determine neighbours.
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.

If the `.locs` file is present and `useLocs = TRUE` then the ordering of the `.locs` file is used to infere the grid layout. This is far quicker than the alternative and is thus recommended, but can only be used on BeadChip platforms. If the data is from a Sentrix Array `useLocs` is automatically set to `FALSE` and the following algorithm is applied instead.

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.

Value

A matrix with 6 columns. Each row corresponds to a bead in the passed `beadLevelData` and the six entries are the indices of the 6 neighbouring beads. Values of NA indicate that the neighbouring bead is missing, either due to failing Illumina's decoding or being at the edge of the array.

Author(s)

Jonathan Cairns, Mike Smith

References

Lynch AG, Smith ML, Dunning MJ, Cairns JM, Barbosa-Morais NL, Tavare S. beadarray, BASH and HULK - tools to increase the value of Illumina BeadArray experiments. In A. Gusnato, K.V. Mardia, & C.J. Fallaize (eds), Statistical Tools for Challenges in Bioinformatics. 2009 pp. 33-37. Leeds, Leeds University Press.

See Also

`HULK`, `BASH`

Examples

```
data(BLData);
neighbours <- generateNeighbours(BLData, array = 1, useLocs = FALSE);
```

---

getAnnotation *Storage of annotation information*

Description

An interface to set or retrieve information about the annotation of a `beadLevelData` object.

Usage

```
getAnnotation(BLData)
setAnnotation(BLData, annoName)
```

Arguments

BLData	a beadLevelData object
annoName	character string to set annotation

Details

A character string is used to specify the annotation with the currently supported values being; Humanv4, Humanv3, Humanv2, Humanv1, Mousev2, Mousev1, Mousev1p1 and Ratv1. This string is used within beadarray to retrieve control probe IDs within particular QC functions.

Value

setAnnotation returns a beadLevelData object with the annotation stored.

getAnnotation returns the name of the annotation that is currently being stored.

Author(s)

Mark Dunning

---

getBeadData *Get raw data from a beadLevelData object*

Description

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

Usage

```
getBeadData(BLData, what="Grn", array=1)
```

Arguments

BLData	BeadLevelList
what	character string specifying the values to retrieve.
array	integer specifying the section/array to use

Value

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

Author(s)

Mark Dunning

Examples

```
data(BLData)
summary(getBeadData(BLData))
```

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 = 1, neighbours = NULL, invasions = 20, useLocs = TRUE, weigh
```

Arguments

BLData	An object of class beadLevelData
array	integer specifying which strip/array to process
neighbours	A Neighbours matrix. Optional - if left NULL, it will be computed.
invasions	Integer - Number of invasions used when identifying neighbouring beads.
useLocs	If information from an associated locs is to be used
weightName	Column name where bead weights are to be taken from
transFun	Transformation function

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 corrected intensities object are returned. These corrected intensities can be saved in the original beadLevelData object using insertBeadData

Value

List where each entry is either a vector of corrected intensities or NULL, depending upon which arrays were specified.

Author(s)

Mike Smith

References

Lynch AG, Smith ML, Dunning MJ, Cairns JM, Barbosa-Morais NL, Tavare S. beadarray, BASH and HULK - tools to increase the value of Illumina BeadArray experiments. In A. Gusnato, K.V. Mardia, & C.J. Fallaize (eds), Statistical Tools for Challenges in Bioinformatics. 2009 pp. 33-37. Leeds, Leeds University Press.

See Also

BASH

Examples

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

---

illuminaOutlierMethod

Identifier outliers on an array section

Description

Implementation of the illumina method for excluding outliers using a 3 MAD (median absolute deviation) cutoff for each bead type

Usage

```
illuminaOutlierMethod(inten, probeList, n = 3)
```

Arguments

inten	a list of intensities
probeList	the IDs corresponding to each intensity value
n	number of MADs cutoff used

Details

This function is called within the summarisation routine of beadarray to exclude outliers from an array-section prior to summary. The intensities are not assumed to be on any particular scale and can result from any user-defined transformation function.

Value

the positions in the original vector that were determined to be outliers

Author(s)

Mark Dunning

Examples

```
data(BLData)

oList = illuminaOutlierMethod(logGreenChannelTransform(BLData, 1), getBeadData(BLData, ar
```

`imageplot` *imageplot for beadLevelData object*

Description

Generates an image plot for data from a `beadLevelData` object.

Usage

```
imageplot(BLData, array = 1, transFun = logGreenChannelTransform, squareSize = N
```

Arguments

`BLData`	`beadLevelData`
`array`	integer specifying what section to plot
`transFun`	Function that defines how values from the BLData object are to be transformed prior to plotting.
`squareSize`	Numeric specifying how many pixels in the original image make up each square in the imageplot. If NULL, the function will guess a suitable value from the data.
`useLocs`	If `TRUE` the function will read the locs file associated with the section in order to include the physical properties of the section in the plot
`horizontal`	If `TRUE` the image will be plotted so that the longest edge of the section is on the x axis.
`low`	colour to use for lowest intensity
`high`	colour to use for highest intensity
`ncolors`	The number of colour graduations between high and low
`zlim`	numerical vector of length 2 giving the extreme values of 'z' to associate with colours 'low' and 'high'.
`legend`	logical, if `TRUE`, `zlim` and range of data is added to plot.
`...`	other arguments to plot

Details

Produces a standard imageplot for the specified section. The default, transformation `logGreenChannelTransform`, takes the log2 of the green channel. For two channel data, the red channel or log ratio can be plotted by `logRedChannelTransform` or `logRatioTransform` functions can be used. The user can also specify their own functions.

The default plotting orientation is such that the longest edge of the section is along the x axis. If `horizontal = FALSE`, the longest edge will by on the y axis and should match how the corresponding TIFF image from the BeadScan directory is orientated.

If `locs = TRUE` and locs file were made available to `readIllumina`, the segments that the section is comprised of will be visible (For expression BeadChips, each section is made of nine physically separate segments). The squareSize parameter will also be set appropriately.

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 and Mark Dunning

Examples

```
data(BLData)

##Be default the first array is plotted
imageplot(BLData, horizontal = FALSE)

##Can use the squaresize parameter

imageplot(BLData, horizontal=FALSE, squareSize=10)

imageplot(BLData, array=3, horizontal=FALSE, squareSize=10)

##User can specifiy what colours to represent low and high intensity

imageplot(BLData, array=3, horizontal=FALSE, squareSize=10, low="lightgreen", high="darkg
```

`insertBeadData` *Add or modify data in a beadLevelData object*

Description

Add or modify data in a `beadLevelData` object.

Usage

```
insertBeadData(BLData, array = 1, what, data)
```

Arguments

BLData	An object of class beadLevelData.
array	Positive integer specifying what section should be modified.
what	Name of the data that is being modified. It 'what' doesn't exist then a new entry is created using the name specified in this argument.
data	A numeric vector to be stored, the same length as the number of beads in the section specified by the array argument.

Details

This function allows the beadData slot of the beadLevelData object to be modified for a given array.

Value

Returns an object of class beadLevelData.

Author(s)

Mike Smith

---

insertSectionData *Modify the sectionData slot*

Description

A function to modify the sectionData slot of a beadLevelData object. Data can be be added if it is a data frame with a number of rows equal to the number of sections in the beadLevelData object.

Usage

```
insertSectionData(BLData, what, data)
```

Arguments

BLData	a beadLevelData object
what	a character string specifiying a name for the new data
data	a data frame containing the data we wish to add

Details

This function allows users to modifiy the per\-section information that is included in the sectionData slot. Typical usage would be to store quality control data that has been computed.

Value

a modified beadLevelData object with the new data attached to sectionData

Author(s)

Mark Dunning

Examples

```
data(BLData)

data(controlProfile)


qct = makeQCTable(BLData, controlProfile=controlProfile)

BLData = insertSectionData(BLData, what="ProbeQC", data = qct)
```

`makeQCTable` *Tabulate QC scores*

Description

Function to make a table of quality control scores for every section in a `beadLevelData` object. Either the annotation of the data needs to be specified, or a control profile data frame that lists ArrayAddress IDs and control types. The supplied summary functions are applied to each control type on each section.

Usage

```
makeQCTable(BLData, transFun = logGreenChannelTransform, controlProfile = NULL,
```

Arguments

`BLData`	a `beadLevelData` object
`transFun`	a function to be applied to the `beadLevelData` object prior to tabulation
`controlProfile`	an optional data frame that specifies ID and type of controls to be used to generate the table
`summaryFns`	list of functions to apply to each control type on every section
`channelSuffix`	optional character string to append to the column names of the resulting table

Details

For each section in turn, the function groups together IDs of the same control type (e.g. housekeeping), and uses an `lapply` with the specified summary functions. A transformation function is applied to `BLData` prior to the summary, with the default being to take the log2 of the green channel.

If the annotation of the `beadLevelData` has been set by `readIllumina` or `setAnnotation` then the controlProfile data frame is calculated automatically and the `controlProfile` argument may be omitted.

Value

A matrix with one row per section and one column for each combination of control type and summary function.

Author(s)

Mark Dunning

Examples

```
data(BLData)

data(controlProfile)

table(controlProfile[,2])

qct = makeQCTable(BLData, controlProfile=controlProfile)

barplot(qct[,1])
```

---

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, status=fD
```

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"`, `neqc`, `rsn` and `"vst"`
`T`	A target distribution vector used when `method="rankInvariant"` normalisation. If `NULL`, the mean is used.
`status`	character vector giving probe types (used in neqc normalisation only)
`negctrl`	character vector giving negative control probes (used in neqc normalisation only)
`regular`	character vector giving regular probes (used in neqc normalisation only)
`...`	further arguments to be passed to `lumiT` or `neqc`

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). The rsn option calls code directly from `lumi`.

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.

The neqc normalisation is described in Shi et al (2010).

Value

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

Author(s)

Matt Ritchie and Mark Dunning

References

Shi. W, Oshlack. A, Smyth, GK, (2010) Optimizing the noise versus bias trade-off for Illumina whole genome expression BeadChips. Nucleic Acids Research

Lin, S.M., Du, P., Kibbe, W.A., (2008) 'Model-based Variance-stabilizing Transformation for Illumina Microarray Data', Nucleic Acids Res. 36, e11

Examples

```
data(BSData)
BSData.norm = normaliseIllumina(BSData, method="quantile", transform="none")

BSData.rsn = normaliseIllumina(BSData, method="rsn", transform="none")
```

---

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

Description

Retrieves the number of beads on selected sections from a beadLevelData object.

Usage

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

Arguments

object	beadLevelData
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 array sections.

Author(s)

Matt Ritchie

Examples

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

---

outlierplot *Plot outlier locations*

Description

Function to plot where the outliers are located on a given array

Usage

```
outlierplot(BLData, array = array, transFun = logGreenChannelTransform, outlierF
```

Arguments

BLData	a beadLevelData object
array	the number of the array to plot
transFun	a function defining how to transform the data prior to calculating outliers
outlierFun	function that will identify outliers
horizontal	if TRUE the longest edge of the array section will be on the x axis
nSegments	How many segments the section is divided into. If this argument is left as the default value (NULL) the code will attempt to extract this information from the relevant .sdf file. If it can't be found then the segments will not be indicated on the final plot.
lowOutlierCol	what colour to plot outliers below the median
highOutlierCol	what colour to plot outliers above the median
outlierPch	plotting character for the outliers
main	an optional title for the plot

Details

The function calls the specified outlier function to determine the outliers on the array and then plots their location. Points are coloured according the intensity of the bead is above or below the median for that bead-type.

Value

plot produced on current graphical device

Author(s)

Mark Dunning and Mike Smith

Examples

```
data(BLData)
outlierplot(BLData, array=1, horizontal = FALSE)
```

---

beadIntensityPlots *Plotting the intensities of selected beads on a section*

Description

The function will plot the intensities of selected beads on a specified array

Usage

```
plotBeadIntensities(BLData, array = 1, BeadIDs, transFun = logGreenChannelTransf
```

Arguments

BLData	a beadLevelData object
array	numeric specifying which array to plot the intensities from
BeadIDs	what ArrayAddress IDs to be plotted
transFun	function specifying what transformation to be applied to the beadLevelData prior to plotting
cols	a vector of colours to be used to plot each ID. If NULL the rainbow function is used to generate colours.
...	other argument that may be passed along to plot.

Details

The function will take all data from the specified section, apply the transformation (the default is to do log2) and then find the subset of beads that have the specified ID. These IDs should match the numeric ArrayAddress IDs that are stored in the beadLevelData object.

Value

Plot is produced on current graphical device.

Author(s)

Mark Dunning

Examples

```
data(BLData)

randIDs = sample(getBeadData(BLData, array=1, what="ProbeID"),10)

plotBeadIntensities(BLData, array=1, BeadIDs = randIDs)
```

---

plotBeadLocations *Plot bead locations*

Description

Can plot where specified beads, or bead types were located on the array surface

Usage

```
plotBeadLocations(BLData, ProbeIDs = NULL, BeadIDs = NULL, array = 1, SAM = FALS
```

Arguments

BLData	a beadLevelData object
ProbeIDs	a list of ArrayAddress IDs to plot
BeadIDs	a list of beads (rows in the beadLevelData object) to plot
array	the number of the section to plot
SAM	if TRUE the array is treated as a Sentrix Array Matrix (hexagonal)
xCol	column name for the x coordinates
yCol	column name for the x coordinates
xlab	optional label for the x axis
ylab	optional label for the y axis
horizontal	if TRUE the longest edge of the array surface will be plotted on the x axis
main	an optional title for the plot
...	any arguments to be passed to plots

Value

plot to current graphical device

Author(s)

Mark Dunning

Examples

```
data(BLData)

##Plot location of first 100 beads as they are listed in beadLevelData object

plotBeadLocations(BLData, BeadIDs = 1:100, array=1, horizontal = FALSE)
```

---

plotChipLayout *Function to Plot the Layout of an Illumina BeadChip*

Description

Using the values obtained from a sentrix descriptor object generated by simpleXMLparse, a plot is generated showing samples and sections.

Usage

```
plotChipLayout(SD, subsC = NULL, sampC = NULL, sectC = NULL, desectC = "red", ma
```

Arguments

SD	A sentrix descriptor object such as that generated by simpleXMLparse from an Illumina sdf file
subsC	The colour for the array substrate. Will be extracted from the sentrix descriptor object if null.
sampC	The colour for the samples. Will be extracted from the sentrix descriptor object if null.
sectC	The colour for the sections. Will be extracted from the sentrix descriptor object if null.
desectC	The colour with which to outline the decode sections
markC	The colour for plotting decoding and analytical markers
main	A title to give the figure (e.g. chip name)
samplab	labels for the samples. Will be extracted from the sentrix descriptor object if null.
sectlab	labels for the sections (if appropriate). Will be extracted from the sentrix descriptor object if null.

Value

returns a plot to the current device (recommend this is tall and thin)

Author(s)

Andy Lynch

Examples

```
## SD<-simpleXMLparse(mysdf)
## plotChipLayout(SD)
```

`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)
```

`plotTIFF` *Produce plots of the Illumina tiff images*

Description

Produces a plot of an Illumina tiff image, which can be useful for observing spatial artifacts on an array and checking the alignment of spot centres features in the image.

Usage

```
plotTIFF(tiff, xrange = c(0, ncol(tiff)-1), yrange = c(0, nrow(tiff)-1), high =
```

Arguments

`tiff`	Intended to the the result of `readTIFF`, but in reality can be any matrix
`xrange`	Range of X coordinates to plot.
`yrange`	Range of Y coordinates to plot.
`high`	Colour to plot the brightest pixels in the image.
`low`	Colour to plot the dimmest pixels.
`mid`	If specified the colour gradient will go from low to mid to high. If not specified then the gradient simply goes from low to high.
`ncolours`	Specify how many steps there should be in the gradient between the high and low colours
`log`	If TRUE the pixel values are logged before the colour gradient is created.
`values`	When set to TRUE each pixel in the image has it's value displayed over it. This should only be used when displaying a very small number of pixels as the text very quickly covers the entire image.
`textCol`	If `values` is `TRUE` this argument specifies the colour of the text.
`...`	Other graphical parameters specified in `par`

Details

This can be very slow, especially when the Cairo graphics library is being used. When using the Cairo library, if one is plotting a large tiff with 10s of millions of pixels, the plotting time increases from around 20 seconds to 5 minutes on an Intel Xeon E5420.

If running on a Linux system it is recommended to use:

```
x11(type = "Xlib")
```

before running `plotTIFF()`, on order to force the quicker plotting mechanism.

Of course it is debatable whether it is useful to plot all of those pixels, given that there are far more than can be displayed on a normal screen, and future revisions of the code may address this.

Value

A plot is produced on the current graphical device.

Author(s)

Mike Smith

`poscontPlot` *Plot the positive 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.

Usage

```
poscontPlot(BLData, array = 1, transFun = logGreenChannelTransform, positiveCont
```

Arguments

`BLData`	a `beadLevelData` object
`array`	The section to be plotted
`transFun`	What transformation function to be applied prior to plotting
`positiveControlTags`	What identifiers to be used as positive controls
`colList`	vector of colours to be used to each positive control
`controlProfile`	an optional data frame with columns defining the ArrayAddress IDs and control\-type for all controls on the platform.
`...`	other arguments to plot

Details

Function for plotting the observed intensites for all replicates of the specified control probes on a given array\-section. The identity of the control probes can be specified by passing a ControlProfile data frame, with the first column being a vector of ArrayAddress IDs and the second column being a corresponding set of characters tags. The beads to be plotted are found by matching the `positiveControlTags` argument to these character tags. Users with expression data can have the ControlProfile data frame defining automatically within the function, provided the annotation of the `beadLevelData` object has been defined by `readIllumina` or `setAnnotation`.

Value

Plot to current graphical device

Author(s)

Mark Dunning

References

www.illumina.com/downloads/GX_QualityControl_TechNote.pdf

Examples

```
###Load the example beadLevelData and associated controlProfile

data(BLData)

data(controlProfile)

poscontPlot(BLData, array=2,controlProfile=controlProfile, positiveControlTags = c("house

poscontPlot(BLData, array=2,controlProfile=controlProfile, positiveControlTags = c("house

poscontPlot(BLData, array=2,controlProfile=controlProfile, positiveControlTags = c("house
```

---

quickSummary *Create summary values for specified IDs*

Description

A utility function for quickly creating summary values for particular IDs (e.g. control IDs) on a given section.

Usage

```
quickSummary(BLData, array = 1, transFun = logGreenChannelTransform, reporterIDs
```

Arguments

BLData	a beadLevelData object
array	Which section to summarize
transFun	a transformation to be applied prior to summarization
reporterIDs	vector specifying the set of IDs to be summarized
reporterTags	a vector that divides the supplied IDs into categories
reporterFun	a function used to summarize each category

Details

The function can be used to calculate summarized values for particular control types on a section. The IDs for all controls are supplied in the reporterIDs argument along with which control type they belong to in the reporterTags argument. A summarized value for each control type is then calculated with the specified function (default is mean).

Author(s)

Mark Dunning

Examples

```
data(BLData)

data(controlProfile)

head(controlProfile)

table(controlProfile[,2])

quickSummary(BLData, array=1,reporterIDs = controlProfile[,1], reporterTags = as.characte
```

---

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",
                        nObservations = "Avg_NBEADS", Detection="Detection Pval"
                    qc.sep="\t", qc.skip=8, controlID="ProbeID",
                    qc.columns = list(exprs="AVG_Signal", se.exprs="BEAD_STDERR"
nObservations="Avg_NBEADS", Detection="Detection Pval"),
    annoPkg=NULL, dec=".", quote="", annoCols = c("TargetID", "PROBE_ID","SYMBOL
```

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)
annoCols	additional columns containing annotation to be read from the file

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

Examples

```
##Read the example data from
##http://www.switchtoi.com/datasets/asuragenmadqc/AsuragenMAQC_BeadStudioOutput.zip
##To follow this example, download the zip file

##dataFile = "AsuragenMAQC-probe-raw.txt"

##qcFile = "AsuragenMAQC-controls.txt"

##BSData = readBeadSummaryData(dataFile=dataFile, qcFile=qcFile, controlID="ProbeID",skip
```

---

readIllumina *Read bead-level Illumina data*

Description

Reads bead-level ouput by Illumina's BeadScan software.

Usage

```
readIllumina(dir = ".", useImages = FALSE, illuminaAnnotation = NULL, sectionN
```

Arguments

`dir`	Directory from which to read the data, the default being the current working directory.
`useImages`	Boolean value specifying if bead intensities should be read directly from the source text files or calculated using the bead centre coordinates and image files.
`illuminaAnnotation`	Character string specifiying the Illumina platform on which the data were generated. This is optional and will help to automate some analyses on expression data. Currently the choices for this argument are Humanv4, Humanv3, Humanv2, Humanv1, Mousev2, Mousev1, Mousev1p1, Ratv1.
`sectionNames`	Optional vector of character strings corresponding to section names to be read in. Typically these are the 10 digit numeric ID of Illumina chip followed by an underscore and capital letter
`metricsFile`	optional name of a metrics file to be read in from the directory. Usually called Metrics.txt
`...`	other arguments to pass when reading bead-level text files

Details

The bead-level data can be generated by any Illumina assay (expression, genotyping, methylation) and via BeadChips or Sentrix Array Matrix. However, some operations within the package are optimised for expression data. For optimal performance, BeadScan needs to be modified to output coordinates for each bead and to include outliers. See http://www.compbio.group.cam.ac.uk/Resources/illumina/index.html for details.

If present, the function will automatically read the following files from the directory

.txt Text file that lists the ID, coordinates and intensity for every decoded bead on an array section. The intensities have been subjected to a local background correction. If `useImages = FALSE` these intensities will be used as a starting point for analysis

.sdf Illumina\'s Sample Description File for the entire chip or SAM. This is used within beadarray to determine the physical properties of a section.

.locs Locations of all beads on the array (i.e. including all those that could not be decoded)

Metrics.txt Illumina\'s metrics that are produced at the time of scanning.

Separate functionality exists to read and manipulate TIFF images that may be found in the same directory. See `readTIFF`.

Value

Returns an object of class `beadLevelData`

Author(s)

Mike Smith, Mark Dunning, Andy Lynch

`readLocsFile` *Read ".locs" file.*

Description

Reads the binary Illumina bead location files and returns a matrix of the coordinate pairs for every bead on the array.

Usage

```
readLocsFile(fileName)
```

Arguments

`fileName`	A string containing the name of the ".locs" file to be read.

Details

The locs file contains bead centre locations for every bead on the array, unlike the bead level text files, with contain just the beads that were decoded. Reading these can be useful if one wants to verify that the image registration was successful, or is interested in the locations of the undecoded beads.

The locs file itself is in a binary format, with each of the bead locations stored as a pair of doubles. The first 2 bytes contain header information, with the 3rd byte containing the number of probes on the array. The location information begins with the 4th byte.

Value

Returns a two column matrix of bead coordinates, one row per bead.

Author(s)

Mike Smith

`readTIFF` *Read the Illumina tiff images*

Description

Reads Illumina tiff images and produces a matrix of pixel values.

Usage

```
readTIFF(fileName, path = NULL, verbose = FALSE)
```

Arguments

`fileName`	String specifying the name of the tiff image to be read.
`path`	String specifying the path to the desired image. The default value of NULL means the current working directory will be used.
`verbose`	If TRUE then details from the header of the tiff are printed as it is read. These include things like the byte order, the number of pixels in the image, the number of tags in the header etc. Defaults to FALSE as this is generally not of interest.

Details

This function has been specifically written to read the grayscale tiff images which are produced by the Illumina scanners. It is not generic enough to read all tiff files, although this functionality may be added in the future.

Given that the raw images can be quite large, functionality has also been included to read tiffs that have been compressed as either .bz2 or .gz files. Identification is performed based on the file extension and it is assumed that each tif is compressed individually. Support for zip files may be added in the future.

Value

Returns a matrix with the same dimensions as the pixels in the tiff file to be read in.

Author(s)

Mike Smith

sectionNames *Gets the section names from a beadLevelData Object*

Description

Retrieves the section names from a beadLevelData object.

Usage

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

Arguments

object	Object of class beadLevelData
arrays	integer (scalar or vector) specifying the sections/arrays to retrieve the names of. When NULL the names of all sections/arrays are returned.

Details

sectionNames retrieves the name of the sections from the sectionInfo slot.

Value

A character vector containing the names of the individual sections.

Author(s)

Mark Dunning

Examples

```
data(BLData)
sectionNames(BLData)
```

setWeights *Set weights from BASH*

Description

Function for committing the weights calculated by BASH into the beadLevelData object

Usage

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

Arguments

BLData	a beadLevelData object
wts	the wts component of the BASH output
array	a vector of arrays that we want to set the weights for
combine	if TRUE combine the weights with existing weights (if they exist)
wtName	name of column to assign weights to

Value

Modified beadLevelData object

Author(s)

Mark Dunning

---

showArrayMask *Show Array Mask*

Description

Function to display beads masked by BASH. The masked beads are assumed to have a weight of 0 in the specified weights column.

Usage

```
showArrayMask(BLData,array = 0, override = FALSE, wtsName = "wts", transFun = lo
```

Arguments

BLData	A BeadLevelList object.
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.
wtsName	name under which the bead weights are stored
array	numeric index of the array to plot
transFun	function to transform intensities prior to calculating outliers
outlierFun	function to remove outliers
horizontal	if TRUE the resulting image is plotting with the longest edge along the x axis

Details

showArrayMask plots the beads on an array that have been assigned a weight of 0 by BASH in red, and beads determined to be outliers in black.

Value

None returned

Author(s)

Jonathan Cairns and Mark Dunning

Examples

```
data(BLData)

bsh = BASH(BLData, array=1)

BLData = setWeights(BLData, wts=bsh$wts[[1]], array=1)

showArrayMask(BLData,1)
```

---

summarize *Create a summarized object*

Description

Function to summarize the data in a `beadLevelData` object into a form more ameanable for downstream analysis (with the same number of observations for each bead type).

Usage

```
summarize(BLData, channelList, probeIDs=NULL, useSampleFac = TRUE, sampleFac
```

Arguments

`BLData`	An object of class beadLevelData
`channelList`	List of objects of class illuminaChannel
`probeIDs`	Vector of ArrayAddressIDs to be included in the summarized object
`useSampleFac`	if TRUE sections belonging to the same biological sample will be combined
`sampleFac`	optional character vector giving which a sample identifer for each section
`weightNames`	name of column in the `beadLevelData` to take extract weights
`removeUnMappedProbes`	if TRUE and annotation information is stored in the `beadLevelData` object, any ArrayAddressIDs that cannot be mapped to ILMN IDs will be removed.

Details

From beadarray version 2.0 onwards, users are allowed more flexibility in how to create summarized data from bead-level data. The `illuminaChannel` is a means of allowing this flexibility by defininining how summarization will be performed on each array section in the bead-level data object. The three keys steps applied to each section are; 1) use a transform function to get the quantities to be summarized (one value per bead). The most common use-case would be to extract the Green channel intensities and possibly perform a log2 transformation. 2) remove any outliers from this list of values 3) split the values according to ArrayAddressIDs and apply the definied `exprFun` and `varFun` to the quantities belonging to each ArrayAddress.

Some Illumina chips have multiple sections for the same biological sample; for example the HumanWG-6 chip or the Infinium genotyping chips. For such cases it may be more convenient to produce a summarized object where each column in the output is a different biological sample. This is especially important for genotpying chips where different SNPs are interrogated on the different sections, making a section-based summary problematic.

If the `useSampleFac` argument is set to TRUE, beadarray will try and combine sections belonging to the same sample. If the location of the sdf file for the chip is sucessfully stored in the `experimentData` slot of the `beadLevelData` object, the sdf will be interrogated to determine how samples were allocated to the chip. Otherwise the user can specifiy a sample factor that is the same length as the number of sections. If the sample factor is not supplied, or cannot be determined, then beadarray will summarize each section separately.

During the course of the summary, ArrayAddressIDs present in the `beadLevelData` object will be converted to Illumina IDs (prefix ILMN) if the annotation of the object was set by `readIllumina` or `setAnnotation`. The rownames of the resulting ExpressionSetIllumina will be set to these new IDs, and the `featureData` slot will contain the original and new IDs. Any control probes present in the `beadLevelData` object will retain their original ArrayAddressID and the Status vector in `featureData` will report if each probe is a control or regular probe. Some ArrayAddressIDs present in the `beadLevelData` object may be neither regular probes will ILMN IDs, or control probes. These are internal controls used by Illumina and can be stopped from appearing in the summarized object by choosing the `removeUnmappedProbes = TRUE` option.

The user can specify a vector of ArrayAddressIDs to be summarized using the `probeIDs` argument. Otherwise, a unique set of IDs is derived from all the array sections in the `beadLevelData` object.

Value

Returns an object of class ExpressionSetIllumina

Author(s)

Mark Dunning

Examples

```
data(BLData)

myMean = function(x) mean(x,na.rm=TRUE)
mySd = function(x) sd(x,na.rm=TRUE)

greenChannel = new("illuminaChannel", logGreenChannelTransform, illuminaOutlierMethod, my

bsd = summarize(BLData, channelList = list(greenChannel))
```

bsd

---

transformFunctions *Functions for transforming the data store in a list("beadLevelData") object for easier visualisation or summarisation.*

Description

Functions for transforming the data store in a beadLevelData object for easier visualisation or summarisation.

Usage

```
logGreenChannelTransform(BLData, array)
logRedChannelTransform(BLData, array)
logRatioTransform(BLData, array)
```

Arguments

BLData	beadLevelData object
array	numeric specifying the section to be transformed

Details

beadarray aims to support the whole range of data that can be generated by the Illumina BeadArray technology and allows users to build upon the functionality in the package to make pipeline to automtically process their own data and develop new methodologies. Therefore we have made the quality assessment and summarisation tools general enough to take any kind of values that can be derived from the beadLevelData object. This is achieved by the definition of transformation functions that can be used throughout the package whenever a function is operating on data on a per-section basis.

The default transformation is to take the data from the Green channel (column Grn in the beadLevelData object) and perform a log2 transformation and is the default to functions such as boxplot or imageplot.

Users with two channel data (e.g. data from methylation and SNP assays) can use the logRedChannelTransform function which instead extracts the red channel on the log2 scale or logRatioTransform.

Value

A numeric vector with the same length as he number of beads recorded for the section

Author(s)

Mark Dunning

Examples

```
data(BLData)

BLData[[1]]

head(BLData[[1]])

log2(getBeadData(BLData, array=1,what="Grn")[1:10])

logGreenChannelTransform(BLData, array=1)[1:10]
```

Index