Package 'simpleaffy'

October 17, 2020

Title Very simple high level analysis of Affymetrix data

Version 2.64.0

Author Crispin J Miller

Description Provides high level functions for reading Affy .CEL files, phenotypic data, and then computing simple things with it, such as t-tests, fold changes and the like. Makes heavy use of the affy library. Also has some basic scatter plot functions and mechanisms for generating high resolution journal figures...

Maintainer Crispin Miller <cmiller@picr.man.ac.uk>

Depends R (>= 2.0.0), methods, utils, grDevices, graphics, stats, BiocGenerics (>= 0.1.12), Biobase, affy (>= 1.33.6), genefilter, gcrma

Imports methods, utils, grDevices, graphics, stats, BiocGenerics, Biobase, affy, genefilter, gcrma

License GPL (>= 2)

URL http://www.bioconductor.org, http://bioinformatics.picr.man.ac.uk/simpleaffy/

LazyLoad yes

biocViews Microarray, OneChannel, QualityControl, Preprocessing, Transcription, DataImport, DifferentialExpression, Annotation, ReportWriting, Visualization

git_url https://git.bioconductor.org/packages/simpleaffy

git_branch RELEASE_3_11

git_last_commit 0dfde26

git_last_commit_date 2020-04-27

Date/Publication 2020-10-16

R topics documented:

`all.present` *Filter by PMA call*

Description

must be present in at least no arrays to be called present

Usage

```
  ## S3 method for class 'present'
all(x,calls,no = "all")
```

Arguments

x	An object to filter
calls	A matrix of PMA calls
no	How many in a row to pass the filter? If 'all' then all must be present

Value

A probesetid

Author(s)

Crispin J Miller

Examples

```
  ## Not run:
     all.present(eset,calls,dim(calls)[2])

## End(Not run)
```

---

all.present.in.group *Filter by PMA call*

Description

Filters an object by PMA calls. Must be called present in at leset 'no' elements in at least one of the replicate sets in the factor 'group'

Usage

```
  ## S3 method for class 'present.in.group'
all(x,group,members,calls,no = "all")
```

Arguments

x	An object to filter
group	The factor to filter by
members	The members in the group to check. If null, checks all possible ones
calls	A matrix of PMA calls
no	How many in a row to pass the filter? If 'all' then all must be present

Value

A probesetid

Author(s)

Crispin J Miller

Examples

```
## Not run:
   all.present.in.group(eset,calls,"line",dim(calls)[2])

## End(Not run)
```

`bg.correct.sa` *Simpleaffy Implementation of Mas5 Background Correction*

Description

Implements the MAS5.0 background correction functions as described in Affy's 'Statistical Algorithms Description Document'.

Usage

```
bg.correct.sa(unnormalised,grid=c(4,4))
```

Arguments

`unnormalised`	An unnormalised AffyBatch object
`grid`	The dimensions of the grid to divide the chip into for background correction.

Value

An AffyBatch object

Author(s)

Crispin J Miller

References

http://bioinformatics.picr.man.ac.uk/ http://www.affymetrix.com/support/technical/technotes/statistical_reference_guide.pdf

See Also

http://www.affymetrix.com/support/technical/technotes/statistical_reference_guide.pdf

Examples

```
## Not run:
  eset.bg.mas <- bg.correct.sa(eset);

## End(Not run)
```

blue.white.red.cols *Generate colourings for heatmaps*

Description

Produces standard colourings for heatmaps.

Usage

```
blue.white.red.cols(x)
red.black.green.cols(x)
red.yellow.white.cols(x)
```

Arguments

x	How many colours to make

Value

A vector of colors

Author(s)

Crispin J Miller

See Also

hmap hmap.eset hmap.pc

Examples

```
## Not run:
     cols <- blue.white.red.cols(21)

## End(Not run)
```

call.exprs *Generate Expression Summaries for Affymetrix Data*

Description

Generates expression summaries and normalizes Affymetrix data using either MAS5.0, GCRMA or RMA algorithms.

Usage

```
call.exprs(x, algorithm = "rma", do.log = TRUE, sc = 100, method = NA)
```

Arguments

x	an AffyBatch object
algorithm	one of '"rma","rma-R","gcrma", "mas5", "mas5-R". "rma" and "mas5" make use of a native C-library and are faster than "rma-R" and "mas5-R".
do.log	return logged data if true
sc	if the mas5 algorithm is being used, sets the target intensity to which the chips should be scaled.
method	The algorithm used to normalise the data. Has no effect for "rma", defaults to quantile normalisation for "rma" and no normalisation for "mas5"

Value

An AffyBatch object containing expression summaries.

Author(s)

Crispin J Miller

References

http://bioinformatics.picr.man.ac.uk/

See Also

read.affy, expresso, justRMA, justMAS

Examples

```
  ## Not run:
    eset.rma  <- call.exprs(eset,"rma");
    eset.mas5 <- call.exprs(eset,"mas5");

## End(Not run)
```

---

detection.p.val *Calculate Detection p-values*

Description

Calculate MAS5 detection pvalues and Present Marginal Absent calls. This is an implementation based on the algorithm described in Liu, Mei et al. (2002) 'Analysis of high density expression microarrays with signed-rank call algorithms', Bioinformatics 18(12) pp1593-1599.

Usage

```
detection.p.val(x, tau = NULL,calls=TRUE,alpha1=NULL,alpha2=NULL,ignore.saturated=TRUE)
```

Arguments

x	An unnormalised AffyBatch object
tau	Errrmmm... tau
alpha1	Present-Marginal threshold
alpha2	Marginal-Absent threshold
calls	if true, generate PMA calls
ignore.saturated	if true do the saturation correction described in the paper, with a saturation level of 46000

Value

A list:

pval	A matrix of detection p values
call	A matrix of PMA calls

Note

alpha1 and alpha2 are parameters that change according to the chip type you are using. If they are not specified, the function uses the current QC environment to find them, and attempts to set one up if it is not there. This is done with an internal call to the function `setQCEnvironment`. If it is unable to find the appropriate config files, this will cause an error. See `setQCEnvironment` for more details.

Author(s)

Crispin J Miller

References

http://bioinformatics.picr.man.ac.uk/

See Also

`setQCEnvironment`

Examples

```
## Not run:
  dpv <- detection.p.val(eset);

## End(Not run)
```

`get.annotation` *Get annotation data for a gene list*

Description

Takes a vector of probeset names and a CDF name. Produces a table of annotations, containing gene name, description, sequence accession number and unigene accession number for each probeset. In addition, write.annotation is a utility function that outputs the annotation data in a form suitable for loading into excel and results.summary takes the outut of pairwise.comparison or pairwise.filter and spits out a table with the means of the replicates the fold-change between them (log2) and t-test p-values. This is followed by a table of annotation (produced by get.annotation).

Usage

```
get.annotation(x, cdfname,verbose=FALSE)
write.annotation(summary,file="results/annotation.table.xls")
results.summary(results,cdfname)
```

Arguments

`x`	a vector of probe names
`cdfname`	the name of the chip (as produced by cdfName(AffyBatch)
`verbose`	print out information if problems are found looking things up in the annotation data
`summary`	a table of data to write in a format appropriate to read into Excel
`file`	a table delimited file
`results`	a PairComp object, as produced by pairwise.comparison and pairwise.filter

Value

A table containing annotation data

Author(s)

Crispin J Miller

References

http://bioinformatics.picr.man.ac.uk/

Examples

```
 ## Not run:
   pw       <- pairwise.comparison(eset.rma,"group",c("A","P"))
   pw.filtered <- pairwise.filter(pw)
   summary     <- results.summary(pw.filtered,"hgu133a")
   write.annotation(file="spreadsheet.xls",summary)

## End(Not run)
```

get.array.indices *Find arrays in an AffyBatch object defined by their phenoData*

Description

Given an AffyBatch object, looks at its phenoData slot to find the factor, or column specified by 'group' and searches that column for entries supplied in 'members'. Returns the indices of these rows. For example, in a six chip AffyBatch object, x, with a column 'treatment' containing 'c','c',t1','t2','t1','t2', a call to get.array.indices(x,\"treatment\",c(\"c\",\"t1\")) would return c(1,2,3,5).

Usage

```
get.array.indices(x,group,members)
```

Arguments

x	An ExpressionSet or AffyBatch object.
group	The name of the pData column to use.
members	The labels within the pData column to match against.

Author(s)

Crispin J Miller

Examples

```
  ## Not run:
    indices3 <- get.array.indices(eset.rma,"group","A")

## End(Not run)
```

get.array.subset *Get a subset of arrays from an affybatch object, split by phnotypic data*

Description

Looks at a factor in the phenotypic data for an AffyBatch or ExpressionSet object and uses it to select a subset of arrays, as defined by 'members'.

Usage

```
get.array.subset(x,group,members)
```

Arguments

x	An ExpressionSet or AffyBatch object.
group	The name of the pData column to use.
members	The labels within the pData column to match against.

Author(s)

Crispin J Miller

See Also

get.array.subset.affybatch get.array.subset.exprset

Examples

```
## Not run:
  subset1 <- get.array.subset.affybatch(eset.rma,"group","A")
  subset2 <- get.array.subset.exprset(eset.rma,"group",c("A","P"))
  subset3 <- get.array.subset(eset.rma,"group","A")

## End(Not run)
```

---

get.array.subset.affybatch

Get a subset of arrays from an affybatch object, split by phnotypic data

---

Description

Looks at a factor in the phenotypic data for an AffyBatch or ExpressionSet object and uses it to select a subset of arrays, as defined by 'members'.

Usage

```
get.array.subset.affybatch(x, group, members)
get.array.subset.exprset(x, group, members)
```

Arguments

x	An AffyBatch or ExpressionSet object.
group	The name of the pData column to use.
members	The labels within the pData column to match against.

Details

Subsetting an AffyBatch object by array is achieved using [x,], while the same is achieved for an ExpressionSet by [,x]. Hence the two different functions. In general the generic method get.array.subset should be used - since it sorts this all out automatically.

Value

An AffyBatch or ExpressionSet (as appropriate) containing the selected subset of chips.

Author(s)

Crispin J Miller

Examples

```
## Not run:
  subset1 <- get.array.subset.affybatch(eset.rma,"group","A")
  subset2 <- get.array.subset.exprset(eset.rma,"group",c("A","P"))
  subset3 <- get.array.subset(eset.rma,"group","A")

## End(Not run)
```

`get.fold.change.and.t.test` *Compute fold change and t-test statistics between two experimental groups*

Description

Generate fold changes (and possibly means) for a pair of experimental groups

Usage

```
get.fold.change.and.t.test(x,group,members,logged = TRUE,a.order=NULL,b.order=NULL,method=c("un
```

Arguments

`x`	an `ExpressionSet` object.
`group`	column in pData(x).
`members`	labels in group.
`logged`	is the AffyBatch data logged?
`a.order`	For a pairwise comparison the ordering of the first group of replicates
`b.order`	For a pairwise comparison the ordering of the second group of replicates
`method`	What method should be used to calculate the average for the fold-change - can be either "logged","unlogged","median"

Details

Given an ExpressionSet object, generate quick stats for pairwise comparisons between a pair of experimental groups. If a.order and b.order are specified then a paired sample t-test will be conducted between the groups, with the arrays in each group sorted according to the ordering specified.

The fold-changes are computed from the average values across replicates. By default this is done using the mean of the unlogged values. The parameter, method allows the mean of the logged values or the median to be used instead. T-tests are always computed with the logged data.

Value

An object of class PairComp

Author(s)

Crispin J Miller

References

http://bioinformatics.picr.man.ac.uk/

Examples

```
  ## Not run:
    pc <- get.fold.change.and.t.test(eset.rma,"group",c("A","P"))

## End(Not run)
```

`hmap.eset` *Draw a heatmap from an AffyBatch object*

Description

Given either an `AffyBatch` draw a heatmap.

Usage

```
hmap.eset(x,probesets,samples=1:length(sampleNames(x)),scluster=standard.pearson,pcluster=standa
```

Arguments

`x`	The `AffyBatch` object to get the expression data from
`probesets`	What probesets to plot, defaults to all of them
`samples`	Which samples to plot
`scluster`	The function to use to cluster the samples by. Can also be a dendrogram object.
`pcluster`	The function to use to cluster the probesets by. Can also be a dendrogram object.
`slabs`	Labels for the sample axis
`plabs`	Labels for the probeset axis defaults to geneNames(x)
`col`	Vector of colour values to use (see below)
`min.val`	The minimum intensity to plot
`max.val`	The maximum intensity to plot
`scale`	Scale each gene's clouring based on standard deviation (See below)
`spread`	If the data is scaled, how many standard deviations (or fold changes) either way should we show. If no scaling, then does nothing
`by.fc`	If the data is scaled, scale by s.d. or by fold.change?
`sdev`	A vector of standard deviaitions for each gene to be plotted. If no value is supplied these are worked out from the data.
`show.legend`	Draw a scale on the graph and show the title if supplied
`title`	The title of the graph
`cex`	Character expansion

Details

Takes an AffyBatch object and plots a heatmap. At its simplest, all that is required is an AffyBatch object (as calculated by call.exprs) and a vector supplying the probesets to plot. These can be specified by name, as an integer index or as a vector of TRUEs and FALSES. The function will try to do something sensible with the labels. If it fails you will need to specify this with plabs. The function will then draw a heatmap, coloured blue-white-red in increasing intensity, scaled so that 100

Col can be used to change the colouring. "bwr" specifies blue-white-red, "rbg" specifies red-black-green, and "ryw" specifies red-yellow-white. Alternatively, a vector of arbitrary colours can be supplied (try rainbow(21), for example).

The clustering method can also be changed by supplying, either, a function that takes a matrix of expression values and returns an hclust or dendrogram object, or alternatively, an hclust or dendrogram object itself. Setting either of these to NULL will stop the heatmap being clustered on that axis.

Scaling is somewhat more complex. If scale is TRUE, then each gene is coloured independently, on a scale based on its standard deviation. By default this is calculated for the samples that are being plotted, unless a value is supplied for sdev – in which case this should be a vector of standard deviations, one for each probeset being plotted (and in the same order). This scaling is done after the clustering. For more details on how all of this works see the website http://bioinf.picr.man.ac.uk/simpleaffy and also look at hmap.pc which uses the scaling to plot transcripts identified as being differentially expressed.

Value

Returns an (invisible) list containing the dendrograms used for samples and probesets

Author(s)

Crispin J Miller

See Also

hmap.pc blue.white.red.cols standard.pearson

Examples

```
## Not run:
  eset.mas <- call.exprs(eset,"mas5")
  hmap.eset(eset.mas,1:100,1:6,col="rbg")

## End(Not run)
```

---

hmap.pc *Draw a heatmap from an PairComp object*

Description

Given either a PairComp object draw a heatmap.

Usage

```
hmap.pc(x,eset,samples=rownames(pData(x)),scluster=standard.pearson,pcluster=standard.pearson,sl
```

Arguments

x	The PairComp object to get the probeset list (and other data) from
eset	The AffyBatch object containing expression data
samples	Which samples to plot – defaults to those used to calculate 'x', but can be any of the samples in eset
scluster	The function to use to cluster the samples by. Can also be a dendrogram object.
pcluster	The function to use to cluster the probesets by. Can also be a dendrogram object.
slabs	Labels for the sample axis
plabs	Labels for the probeset axis
col	Vector of colour values to use (see below)
scale	Scale each gene's clouring based on standard deviation (See below)
spread	If the data is scaled, how many standard deviations (or fold changes) either way should we show. If no scaling, then does nothing
by.fc	If the data is scaled, do it by fold change?
gp	The column in the expression set's pData object used to select the samples to plot. By default this is the one used to calculate x.
mbrs	The members of the 'group' column that we wish to plot. By default these are the pair used to calculate x. If 'all' is supplied then all samples are used.
show.legend	Draw a scale on the graph and show the title if supplied
title	The title of the graph
cex	Character expansion

Details

Takes a PairComp object and an AffyBatch object and plots a heatmap. At its simplest, all that is required are these two objects. The function will then draw a heatmap, coloured red-black-green in increasing intensity, scaled for each gene based on standard deviation. The legend shows how these colours translate into intensity.

Col can be used to change the colouring. "bwr" specifies blue-white-red, "rbg" specifies red-black-green, and "ryw" specifies red-yellow-white. Alternatively, a vector of arbitrary colours can be supplied (try `rainbow(21)`, for example).

Scaling is somewhat complex. If scale is TRUE, then each gene is coloured independently, on a scale based on its standard deviation. This is calculated as follows: 'group' supplies a column in the pData object of 'eset' that is used to collect samples together (generally as replicate groups). 'members' supplies the entries within this column that are to be used. (Unless specified, the function uses the same value for 'group' and 'members' used to calculate the PairComp object). The function uses these data to calculate the standard deviation for each probeset within each set of replicates, and then calculates the average sd for each gene. This is then used to scale the data so that each probeset is plotted on a scale that shows the number of standard deviations away from the mean it is for that sample. For more details on how all of this works see the website http://bioinf.picr.man.ac.uk/simpleaffy.

Alternatively, by setting by.fc to FALSE, scaling can be done simply in terms of fold-change, in which case, spread defines the maximum and minimum fold changes to show.

Value

Returns an (invisible) list containing the dendrograms used for samples and probesets

Author(s)

Crispin J Miller

See Also

hmap.eset blue.white.red.cols standard.pearson

Examples

```
## Not run:
  pc <- pairwise.comparison(eset.mas,group="group",members=c("a","b"),spots=eset)
  pf <- pairwise.filter(pc)
  hmap.pc(pf,eset.mas)

## End(Not run)
```

---

journalpng *Produce a device for producing artwork for presentations and journals*

Description

journalpng generates a device to print a 4 x 4 inch 300 dpi figure (by default). screenpng does the same, but 72dpi.

Usage

```
journalpng(file="figure.png",width=4, height=4,res=300)
screenpng(file="figure.png",width=4, height=4,res=72)
```

Arguments

file	the file to write the figure to
width	the width of the figure
height	its height
res	resolution in dots-per-inch

Value

A table containing annotation data

Author(s)

Crispin J Miller

References

http://bioinformatics.picr.man.ac.uk/

Examples

```
## Not run:
  journalpng(file="results/figure1.png"); # starts a new device
  trad.scatter.plot(exprs(eset)[,1],exprs(eset)[,2])
  dev.off();    # writes the file at this point.

## End(Not run)
```

---

justMAS *Generate Expression calls using a C implementation of the MAS 5.0 Algorithm*

Description

Implements the MAS5.0 background correction, expression summary and scaling functions as described in Affy's 'Statistical Algorithms Description Document'

Usage

```
justMAS(unnormalised,tgt=100,scale=TRUE)
```

Arguments

unnormalised	An unnormalised AffyBatch object
tgt	The target intensity to scale array to, if scaling.
scale	Scale the data to the specified target intensity.

Details

Uses a C code implementation of the MAS5.0 algorithm (As described in Affymetrix's 'Statistical Algorithms Reference Guide' - see http://www.affymetrix.com, and in Hubbell et al. (2002) Robust Estimators for expression analysis. Bioinformatics 18(12) 1585-1592). Note that this function returns log2 data.

Value

An AffyBatch object, with, in addition, scale-factors for each array stored in the object's description@preprocessing@ slot, and the target intensity the arrays were scaled to in description@preprocessing@tgt

Author(s)

Crispin J Miller

References

http://bioinformatics.picr.man.ac.uk/

See Also

http://www.affymetrix.com/support/technical/technotes/statistical_reference_guide.pdf

Examples

```
   ## Not run:
     eset.mas <- justMAS(eset.mas);

## End(Not run)
```

`PairComp-class` *Class "PairComp" Represents the results of pairwise comparison between two experimental factors*

Description

Holds fold-change, ttest p-score and detection p-value calls(if used) between a pair of experimental factors.

Slots

`means`**:** Object of class `"matrix"` Mean values for each of the experimental factors.

`fc`**:** Object of class `"numeric"` Fold change between the means.

`tt`**:** Object of class `"numeric"` P-score between the factors.

`calls`**:** Object of class `"matrix"` Detection p-values for each probeset on each array.

`group`**:** Object of class `"character"` The name of the factor that was compared.

`members`**:** Object of class `"character"` A list containing the two levels compared between.

`pData`**:** Object of class `"pData"` The phenoData for the members that were compared.

`calculated.from`**:** Object of class `"ExpressionSet"` The original expression set that was being compared.

Methods

[`signature(x = "PairComp")`: get the values for the specified gene(s).

[<- `signature(x = "PairComp")`: not supported.

calls `signature(object = "PairComp")`: the detection.p.values.

fc `signature(object = "PairComp")`: the fold-changes.

group `signature(object = "PairComp")`: the name of the group that was compared.

means `signature(object = "PairComp")`: the means of the two experimental factors that were compared.

members `signature(object = "PairComp")`: the members of that group that were compared.

pairwise.filter `signature(object = "PairComp")`: Take a `PairComp` object and filter it to yield probesets that pass the specified criteria.

tt `signature(object = "PairComp")`: the results of a ttest between groups.

pData `signature(object = "pData")`: The phenoData from the members that were compared.

calculated.from `signature(object = "ExpressionSet")`: The original expression set.

Author(s)

Crispin Miller

`pairwise.comparison` *Compute pairwise comparison statistics between two experimental groups*

Description

Generate fold changes, t-tests and means for a pair of experimental groups

Usage

```
pairwise.comparison(x,group,members=NULL,spots=NULL,a.order=NULL,b.order=NULL,method="unlogged"
```

Arguments

`x`	an `ExpressionSet` object.
`group`	column in pData(x).
`members`	labels in group.
`spots`	unnormalised AffyBatch data for this experiment - if included, results in PMA calls and detection p-values being generated
`a.order`	For a comparison with matched pairs, the ordering of the first group of replicates
`b.order`	For a comparison with matched pairs, the ordering of the second group of replicates
`method`	What method should be used to calculate the average for the fold-change - can be either "logged","unlogged","median"
`logged`	Whether the input data is logged or not

Details

Given an ExpressionSet object, generate quick stats for pairwise comparisons between a pair of experimental groups. If a.order and b.order are specified then a paired sample t-test will be conducted between the groups, with the arrays in each group sorted according to the ordering specified. By default, the function assumes that the expression values are logged (this can be changed with the parameter "logged"). The fold-changes are computed from the average values across replicates. Unless you specify otherwise, this is done using the mean of the unlogged values (i.e. logged data is first unlogged, the mean calculated, and the result re-logged). The parameter "method", allows the mean of the logged values or their median to be used instead. T-tests are always computed with the logged data.

Value

A Pairwise comparison object.

Author(s)

Crispin J Miller

References

http://bioinformatics.picr.man.ac.uk/

Examples

```
  ## Not run:
    pc <- pairwise.comparison(eset.rma,"group",c("A","P"))

## End(Not run)
```

`pairwise.filter` *Filter pairwise comparison statistics between two experimental groups*

Description

Given the results of a pairwise.comparison, filter the resulting gene list on expression level, PMA calls (if available), fold change and t-test statistic.

min.exp and min.exp.no allow the data to be filtered on intensity (where min.exp.no specifies the minimum number of arrays that must be above the threshold 'min.exp' to be allowed through the filter).

PMA filtering is done when min.present.number is greater than 0.

min.present.no allows arrays to be filtered by PMA call. A number or 'all' must be specified. If a number, then the at least this many arrays must be called present, if 'all', then all arrays must be called present.

present.by.group specifies whether PMA filtering is to be done on a per-group basis or for all arrays at once. If 'false' then the experiment is treated as a single group (i.e. a probeset passes the filter if it is called present on at least min.present.number arrays in the whole experiment. If 'true' then it must be called present on at least this many arrays in one or more groups. See the vignette for more details.

Usage

```
pairwise.filter(object,min.exp=log2(100),min.exp.no=0,min.present.no=0,present.by.group=T,fc=1.
```

Arguments

`object`	a 'PairComp' object
`min.exp`	Filter genes using a minimum expression cut off
`min.exp.no`	A gene must have an expression intensity greater than 'min.exp' in at least this number of chips
`min.present.no`	A gene must be called present on at least this number of chips
`present.by.group`	If true, then the probeset must be called Present on at least min.present.number arrays in any of the replicate sets used to generate the PairComp object being filtered. If false, then must be called present on at least min.present.no of the arrays in the whole experiment
`fc`	A gene must show a log2 fold change greater than this to be called significant
`tt`	A gene must be changing with a p-score less than this to be called significant

Value

A 'PairComp' object filtered to contain only the genes that pass the specified filter parameters.

Author(s)

Crispin J Miller

References

http://bioinformatics.picr.man.ac.uk/

Examples

```
  ## Not run:
    pc <- pairwise.comparison(eset.rma,"group",c("A","P"))
    pf <- pairwise.filter(pc,tt=0.01);

## End(Not run)
```

---

plot.pairwise.comparison

Plots a PairComp object

Description

Draws a scatter plot between means from a pairwise comparison. Colours according to PMA calls and identifies 'signficant' genes yielded by a filtering

Usage

```
## S3 method for class 'pairwise.comparison'
plot(x,y=NULL,labels=colnames(means(x)),showPMA=TRUE,type="scatter",...)
```

Arguments

x	A PairComp object
y	A PairComp object
labels	A list containing x and y axis labels
showPMA	True if PMA calls are to be identified
type	Can be 'scatter', 'ma' or 'volcano'
...	Additional arguments to plot

Details

Takes a PairComp object (as produced by `pairwise.comparison` and plots a scatter plot between the sample means. If PMA calls are present in the `calls` slot of the object then it uses them to colour the points. Present on all arrays: red; absent on all arrays: yellow; present in all some arrays; orange. In addition, if a second `PairComp` object is supplied, it identifies spots in that object, by drawing them as black circles. This allows, for example, the results of a `pairwise.filter` to be plotted on the same graph.

If type is 'scatter' does a simple scatter plot. If type is 'volcano' does a volcano plot. If type is 'ma' does an MA plot.

Author(s)

Crispin J Miller

See Also

`pairwise.comparison` `pairwise.filter` `trad.scatter.plot`

Examples

```
## Not run:
  pc <- pairwise.comparison(eset.mas,group="group",members=c("a","b"),spots=eset)
  pf <- pairwise.filter(pc)
  plot(pc,pf)

## End(Not run)
```

---

plot.qc.stats *Plots a QCStats object*

Description

Generates a visual summary of the various QC statistics recommended by Affymetrix in their 'Data Analysis Fundamentals' handbook.

Arguments

`x`	A `QCStats` object
`fc.line.col`	The colour to mark fold change lines with
`sf.ok.region`	The colour to mark the region in which scale factors lie within appropriate bounds
`chip.label.col`	The colour to label the chips with
`sf.thresh`	Scale factors must be within this fold-range
`gdh.thresh`	Gapdh ratios must be within this range
`ba.thresh`	beta actin must be within this range
`present.thresh`	The percentage of genes called present must lie within this range
`bg.thresh`	Array backgrounds must lie within this range
`label`	What to call the chips

main	The title for the plot
usemid	If true use 3'/M ratios for the GAPDH and beta actin probes
cex	Value to scale character size by (e.g. 0.5 means that the text should be plotted half size)
...	Other parameters to pass through to plot

Details

A lot of information is presented in this one figure. By default, each array is represented by a seperate line in the figure. The central vertical line corresponds to 0 fold change, the dotted lines on either side correspond to 3 fold up and down regulation. The blue bar represents the region in which all arrays have scale factors within, by default, three-fold of each other. Its position is found by calculating the mean scale factor for all chips and placing the center of the region such that the borders are -1.5 fold up or down from the mean value.

Each array is plotted as a line from the 0-fold line to the point that corresponds to its scale factor. If the ends of all of the lines are in the blue region, their scale-factors are compatible. The lines are coloured blue if OK, red if not.

The figure also shows GAPDH and beta-actin 3'/5' ratios. These are represented as a pair of points for each chip. Affy state that beta actin should be within 3, gapdh around 1. Any that fall outside these thresholds (1.25 for gapdh) are coloured red; the rest are blue.

Written along the left hand side of the figure are the number of genes called present on each array and the average background. These will vary according to the samples being processed, and Affy's QC suggests simply that they should be similar. If any chips have significantly different values this is flagged in red, otherwise the numbers are displayed in blue. By default, 'significant' means that %-present are within 10% of each other; background intensity, 20 units. These last numbers are somewhat arbitrary and may need some tweaking to find values that suit the samples you're dealing with, and the overall nature of your setup.

Finally, if BioB is not present on a chip, this will be flagged by printing 'BioB' in red.

In short, everything in the figure should be blue - red highlights a problem!

Usage

plot.qc.stats(x, fc.line.col = "black", sf.ok.region = "light blue", chip.label.col = "black", sf.thresh = 3, gdh.thresh = 1.25, ba.thresh = 3, present.thresh = 10, bg.thresh = 20, label = NULL,title="QC Stats",spread=c(-8,8),usemid=F,type="l",cex=1, ...)

Author(s)

Crispin J Miller

See Also

qc

Examples

```
data(qcs)
plot(qcs)
```

qc *Generate QC stats from an AffyBatch object*

Description

Generate QC metrix for Affymetrix data.

Usage

```
qc(unnormalised, ...)
```

Arguments

unnormalised	An AffyBatch object with nowt done to it
...	Any other parameters

Details

Affymetrix recommend a series of QC metrics that should be used to check that arrays have hybridised correctly and that sample quality is acceptable. These are discussed in the document 'QC and Affymetrix data' accompanying this package, and on the web at http://bioinformatics.picr.man.ac.uk. They are described in detail in the 'Expression Analysis Fundamentals' manual available from Affymetrix.

Before using this function you are strongly encouraged to read the 'QC and Affymetrix data' document, which contains detailed examples.

This function takes an AffyBatch object and generates a QCStats object containing a set of QC metrics. See qc.affy for more details.

Author(s)

Crispin J Miller

See Also

qc.affy setQCEnvironment

Examples

```
  ## Not run:
    qcs <- qc(eset,eset.mas)

## End(Not run)
  data(qcs)
  ratios(qcs)
  avbg(qcs)
  maxbg(qcs)
  minbg(qcs)
  spikeInProbes(qcs)
  qcProbes(qcs)
  percent.present(qcs)
  plot(qcs)
  sfs(qcs)
```

```
target(qcs)
ratios(qcs)
```

`qc.affy` *Generate Affymetrix Style QC Statistics*

Description

Generate QC data for Affymetrix arrays

Usage

```
qc.affy(unnormalised,normalised=NULL,tau=0.015,logged=TRUE,
cdfn=cdfName(unnormalised))
```

Arguments

`unnormalised`	An unnormalised raw `AffyBatch` object to call qc stats on
`normalised`	The same one, processed using `justMAS` (contains scale factors etc.). If not supplied, then the object gets calculated internally.
`tau`	used by detection p value
`logged`	True if the data is logged
`cdfn`	The cdf name for the array - can be used to specify a different set of probes to the default

Details

Affymetrix recommend a series of QC metrics that should be used to check that arrays have hybridised correctly and that sample quality is acceptable. These are discussed in the document 'QC and Affymetrix data' accompanying this package, and on the web at http://bioinformatics.picr.man.ac.uk. They are described in detail in the 'Expression Analysis Fundamentals' manual available from Affymetrix.

This function takes an (unnormalised) AffyBatch object, and (optionally) an ExprSet, with MAS expression calls produced by `call.exprs` and generates QC metrics. If the MAS calls are not supplied these are claculated internally.

Value

A QCStats object describing the supplied `AffyBatch`

Author(s)

Crispin J Miller

Examples

```
    ## Not run:
       qcs <- qc(eset)

## End(Not run)
  data(qcs)
  ratios(qcs)
  avbg(qcs)
  maxbg(qcs)
  minbg(qcs)
  spikeInProbes(qcs)
  qcProbes(qcs)
  percent.present(qcs)
  plot(qcs)
  sfs(qcs)
  target(qcs)
  ratios(qcs)
```

`qc.get.alpha1` *Get or set the alpha values for the current QC environment*

Description

Alpha1 and Alpha2 are used to define the P/M/A thresholds for detection calling algorithm see - `detection.p.val`. These are array dependent, these functions set or get their values. Tau is a constant parameter within the calculation and is not array specific.

Usage

```
qc.get.alpha1()
qc.set.alpha1(value)
qc.get.alpha2()
qc.set.alpha2(value)
qc.get.tau()
```

Arguments

`value`	A double representing the alpha1 or alpha2 threshold for defining detection calls. See `detection.p.val` for more details.

Value

`qc.set.alpha1` and `qc.set.alpha2` return nothing. `qc.get.alpha1` and `qc.get.alpha2` return a double.

Author(s)

Crispin J Miller

References

http://bioinformatics.picr.man.ac.uk/

See Also

detection.p.val

Examples

```
setQCEnvironment("hgu133plus2cdf")
qc.get.alpha1()
qc.get.alpha2()
qc.set.alpha1(0.05)
qc.get.alpha1()
qc.set.alpha2(0.05)
qc.get.alpha2()
```

---

qc.get.array — *Get or set the name of the array for which the current QC environment is valid. Currently not used for anything important; is a free text identifier.*

Description

The array name is simply a free text name for the array of interest.

Usage

```
qc.get.array()
qc.set.array(name)
```

Arguments

name	a free text name for the array of interest

Value

a string

Author(s)

Crispin J Miller

References

http://bioinformatics.picr.man.ac.uk/

See Also

setQCEnvironment

Examples

```
qc.set.array("plus2")
qc.get.array()
```

qc.get.probes *Retrieve QC probeset names for the current array type*

Description

Get the names of probesets used to calculate 3'/5' ratios for the current array type. qc.get.spikes is used to set the spike probe names (i.e. bioB, bioC, etc.)

Usage

```
qc.get.probes()
qc.get.probe(name)
qc.add.probe(name,probeset)
```

Arguments

name	A name for the given probeset. By default, this is the probeset identifier
probeset	A probeset ID

Value

A character array of probeset IDs, or the requested probeset ID, as appropriate.

Author(s)

Crispin J Miller

References

http://bioinformatics.picr.man.ac.uk/

See Also

setQCEnvironment qc.get.spikes

Examples

```
setQCEnvironment("hgu133plus2cdf")
qc.get.probes()
qc.add.probe("my.name","a.probesetid_at")
qc.add.probe("another.name","another.probesetid_at")
qc.get.probes()
```

qc.get.ratios *Retrieve pairs of probesets used for calculating 3'/5' ratios*

Description

Get the names of the qc probesets used to define the 3'/5' ratios.

Usage

```
qc.get.ratios()
qc.get.ratio(name)
qc.add.ratio(name,probeset1,probeset2)
```

Arguments

name	A name for the given ratio calculation (such as gapdh3/5)
probeset1	A probeset ID
probeset2	A probeset ID

Value

A list, each element with a name like gapdh3/5 and comprising of a two-element character vector of probeset IDs.

Author(s)

Crispin J Miller

References

http://bioinformatics.picr.man.ac.uk/

See Also

setQCEnvironment qc.get.probes

Examples

```
setQCEnvironment("hgu133plus2cdf")
qc.get.ratios()
qc.add.ratio("a.name","probeset1.id","probeset2.id")
qc.get.ratio("a.name")
```

qc.get.spikes *Retrieve QC spike probeset names for the current array type*

Description

Get the names of spike probesets for bioB, bioC, etc. ratios for the current array type. qc.get.probes is used to define the 3'/5' ratio probesets

Usage

```
qc.get.spikes()
qc.get.spike(name)
qc.add.spike(name,probeset)
```

Arguments

name	A name for the given probeset. By default, this is the probeset identifier
probeset	A probeset ID

Value

A character array of probeset IDs, or the requested probeset ID, as appropriate.

Author(s)

Crispin J Miller

References

http://bioinformatics.picr.man.ac.uk/

See Also

setQCEnvironment qc.get.probes

Examples

```
qc.get.spikes()
qc.add.spike("my.name","a.probesetid_at")
qc.add.spike("another.name","another.probesetid_at")
qc.get.spikes()
```

`qc.have.params` *Does simpleaffy have a QC definition file for the specified array?*

Description

Simpleaffy requires a definition file describing the qc probes, spikes, alpha values, etc. for the array of interest. This is used to initialize the QC environment for the array (usually implicitly within the `qc` function), by a call to `setQCEnvironment`. This function can be used to see if the specified array has a definition file.

Usage

```
qc.have.params(name)
```

Arguments

`name`	The 'clean' CDF name of the array (i.e. the result of calling `cleancdfname` on the `cdfName` of the `AffyBatch` object containing the array data of interest.

Value

True or False

Author(s)

Crispin J Miller

References

http://bioinformatics.picr.man.ac.uk/

See Also

`setQCEnvironment`, `qc`, `qc.ok`, `cdfName`, `cleancdfname`

Examples

```
qc.have.params("nosucharraycdf")
qc.have.params("hgu133plus2cdf")
setQCEnvironment("hgu133plus2cdf")
qc.have.params(cleancdfname("HG-U133_Plus_2"))
```

qc.ok *Has simpleaffy's QC environment been set up?*

Description

Simpleaffy requires a definition file describing the qc probes, spikes, alpha values, etc. for the array of interest. This is used to initialize the QC environment for the array (usually implicitly within the qc function), by a call to setQCEnvironment. This function can be used to check if the qc environment has been set up for the current session

Usage

```
qc.ok()
```

Value

True or False

Author(s)

Crispin J Miller

References

http://bioinformatics.picr.man.ac.uk/

See Also

setQCEnvironment qc qc.have.params cdfName

Examples

```
qc.ok()
setQCEnvironment("hgu133plus2cdf")
qc.ok()
```

qc.read.file *Read a file defining the QC parameters for a specified array and set up the QC Environment*

Description

Affymetrix define a series of QC parameters for their arrays. Many of these rely on specific probeset that differ between arrays and are used to calculate things like 3'/5' ratios. See qc.affy for more details. This is usually done by a call to setQCEnvironment; the function described here is the one that does the actual loading of the configuration file. See the package vignette for details of the config file's syntax.

Usage

```
qc.read.file(fn)
```

Arguments

fn	full path and name of the file to load

Value

none.

Author(s)

Crispin J Miller

References

http://bioinformatics.picr.man.ac.uk/

See Also

setQCEnvironment

Examples

```
fn <- system.file("extdata","hgu133plus2cdf.qcdef",package="simpleaffy")
qc.read.file(fn)
qc.get.spikes()
qc.get.probes()
qc.get.ratios()
```

---

qcs *an example QC Stats object*

Description

This datasets gives sample qc data for 25 array hgu133a comparison between two cell lines (MCF7 and MCF10A) and a variety of protocols.

Usage

```
qcs
```

Format

a QCStats object

Examples

```
data(qcs)
plot(qcs)
```

QCStats-class *Class "QCStats"*

Description

Holds Quality Control data for a set of Affymetrix arrays

Objects from the Class

Objects can be created by calls of the form `qc(AffyBatch)`.

Slots

`scale.factors`: Object of class `"numeric"` Scale factors used to scale the chips to the specified target intensity

`target`: Object of class `"numeric"` The target intensity to which the chips were scaled

`percent.present`: Object of class `"numeric"` Number of genes called present

`average.background`: Object of class `"numeric"` The average background for the arrays

`minimum.background`: Object of class `"numeric"` The minimum background for the arrays

`maximum.background`: Object of class `"numeric"` The maximum background for the arrays

`bioBCalls`: Object of class `"character"` The detection PMA (present / marginal / absent) calls of bioB spike-in probes

`spikes`: Object of class `"list"` spiked in probes (e.g. biob, bioc...)

`qc.probes`: Object of class `"list"` qc probes (e.g. gapdh 3,5,M,...)

`arraytype`: The `cdfName` of the `AffyBatch` object used to create the object

Methods

avbg `signature(object = "QCStats")`: average background

maxbg `signature(object = "QCStats")`: maximum background

minbg `signature(object = "QCStats")`: minimum background

spikeInProbes `signature(object = "QCStats")`: the spike-in QC probes

qcProbes `signature(object = "QCStats")`: the gapdh and actin QC probes

percent.present `signature(object = "QCStats")`: no probesets called present

plot `signature(x = "QCStats")`: Plot a QCStats object

sfs `signature(object = "QCStats")`: scale factors

target `signature(object = "QCStats")`: target scaling

ratios `signature(object = "QCStats")`: 5'3' and 5'M ratios for QC Probes

arrayType `signature(object = "QCStats")`: The type of array for which this QC stats object was generated

Author(s)

Crispin J Miller

See Also

`qc`

`read.affy` *Read a Set of .CEL Files and Phenotypic Data*

Description

Reads the specified file, which defines phenotypic data for a set of .CEL files. Reads the specified files into an `AffyBatch` object and then creates a `phenoData` object, defining the experimental factors for those chips.

Usage

```
read.affy(covdesc = "covdesc",path=".", ...)
```

Arguments

`covdesc`	A white space delimited file suitable for reading as a `data.frame`. The first column (with no column name) contains the names(or paths to) the .CEL files to read. Remaining columns (with names) represent experimental factors for each chip. these become elements of the `phenoData` object.
`...`	extra functions to pass on to `ReadAffy`
`path`	The path to prefix the filenames with before calling `ReadAffy`

Value

An AffyBatch object

Author(s)

Crispin J Miller

References

http://bioinformatics.picr.man.ac.uk/

See Also

`ReadAffy`, `AffyBatch` `data.frame` `phenoData`

Examples

```
  ## Not run:
    eset <- read.affy(); # read a set of CEL files
    eset.rma <- call.exprs(eset,"rma");

## End(Not run)
```

`read.affy.mixed` *Read a Set of .CEL Files and Phenotypic Data from mixed chip types*

Description

Reads the specified file, which defines phenotypic data for a set of .CEL files. Reads the specified files into an `AffyBatch` object and then creates a `phenoData` object, defining the experimental factors for those chips. This function deals with different array types by generating a pseudo arrayset containing only the probes in common. It does this by finding the smallest chip type in the set, and using this as a template. Probesets that aren't shared are set to 0. Other probesets are copied in. Note that this means that spots that were in one place on one array, appear to be at a different place on another. What this does to position specific background correction algorithms (such as mas5) is left as an exercise to the reader). Beware...

Usage

```
read.affy.mixed(covdesc = "covdesc",path=".", ...)
```

Arguments

`covdesc`	A white space delimited file suitable for reading as a `data.frame`. The first column (with no column name) contains the names(or paths to) the .CEL files to read. Remaining columns (with names) represent experimental factors for each chip. these become elements of the `phenoData` object.
`...`	extra functions to pass on to `ReadAffy`
`path`	The path to prefix the filenames with before calling `ReadAffy`

Value

An AffyBatch object

Author(s)

Crispin J Miller

References

http://bioinformatics.picr.man.ac.uk/

See Also

`ReadAffy`, `AffyBatch` `data.frame` `phenoData`

Examples

```
## Not run:
  eset <- read.affy.mixed(); # read a set of CEL files

  eset.rma <- call.exprs(eset,"rma");

## End(Not run)
```

`setQCEnvironment` *Establish the appropriate QC environment for the specified array*

Description

Affymetrix define a series of QC parameters for their arrays. Many of these rely on specific probeset that differ between arrays and are used to calculate things like 3'/5' ratios. See `qc.affy` for more details. These functions are used to set up the appropriate QC environment for the specified array. This is done by loading a configuration file, either from the packages data directory, or from the specified path. See the package vignette for details of the config file's syntax.

Usage

```
setQCEnvironment(array,path=NULL)
```

Arguments

`array`	This should be the 'clean' cdf name of the array as generated by `cleancdfname` in the affy package.
`path`	Path to the file. By default, checks the package's own data directory - only needed if a definiinition file is being specified manually, as described in the vignette.

Details

The usual way to get the 'clean' cdfname is as follows: `cleancdfname(cdfName(eset))`, where `eset` is an `AffyBatch` object.

Value

none.

Author(s)

Crispin J Miller

References

http://bioinformatics.picr.man.ac.uk/

See Also

`qc`

Examples

```
setQCEnvironment("hgu133plus2cdf")
setQCEnvironment(cleancdfname("HG-U133_Plus_2"))
```

simpleaffy-deprecated *Does simpleaffy have a QC definition file for the specified array?*

Description

The underlying implementation of simpleaffy has changed significantly and it now represents QC parameters differently. In particular, it loads only the QC data for the specified array type. A call to any of these functions loads the appropriate environment specifed by name. They therefore been deprecated and WILL disappear from simpleaffy in the future.

Usage

```
getTao(name)
getAlpha1(name)
getAlpha2(name)
getActin3(name)
getActinM(name)
getActin5(name)
getGapdh3(name)
getGapdhM(name)
getGapdh5(name)
getAllQCProbes(name)
getBioB(name)
getBioC(name)
getBioD(name)
getCreX(name)
getAllSpikeProbes(name)
haveQCParams(name)
```

Arguments

name	The 'clean' CDF name of the array (i.e. the result of calling cleancdfname on the cdfName of the AffyBatch object containing the array data of interest.

Details

Each of these functions has been replaced by a new function of the form qc.get.. In order to support ratios other than gapdh and beta-actin, the appropriate way to get ratios is now to use qc.get.ratios, which will return a table containing all suggested ratio calculations for the array. Similarly, qc.get.spikes will return a table containing all spike probesets for the array.

Value

None.

Author(s)

Crispin J Miller

References

http://bioinformatics.picr.man.ac.uk/

See Also

setQCEnvironment qc qc.ok cdfName cleancdfname qc.get.ratios qc.get.spikes qc.get.probes

Examples

```
#old
getBioB("hgu133plus2cdf")
getActin3("hgu133plus2cdf")
getActinM("hgu133plus2cdf")
getActin5("hgu133plus2cdf")
#new
setQCEnvironment("hgu133plus2cdf")
qc.get.spikes()["bioB"]
r <- qc.get.probes()
r["actin3"]
r["actinM"]
r["actin5"]
```

---

standard.pearson *A clustering function based on pearson correlation*

Description

Given a matrix of values, uses hclust and cor to generate a clustering based on 1-Pearson correlation

Usage

```
standard.pearson(x)
```

Arguments

x	A matrix of data

Value

The result of performing an hclust

Author(s)

Crispin J Miller

See Also

hmap hmap.eset hmap.pc

Examples

```
  ## Not run:
y <- standard.pearson(x)

## End(Not run)
```

`trad.scatter.plot` *Does a Traditional Scatter Plot of Expression Data*

Description

Plots expression data as a scatter plot with optional fold-change lines

Usage

```
trad.scatter.plot(x, y, add = FALSE, fc.lines = log2(c(2, 4, 6, 8)), draw.fc.lines = TRUE, draw.fc.li
```

Arguments

`x`	x coords
`y`	y coords
`add`	add this data to an existing graph
`fc.lines`	Vector of intervals at which to draw fold-change lines
`draw.fc.lines`	Draw fold change lines?
`draw.fc.line.labels`	Label the fold change lines with the fold changes they represent?
`fc.line.col`	The colour to draw fold change lines
`pch`	Plotting character to use for the scatter data (see `plot` for more details)
`xlim`	Range for the xaxis
`ylim`	Range for the yaxis
`...`	Additional parameters to pass through to the underlying `plot` function

Author(s)

Crispin J Miller

References

http://bioinformatics.picr.man.ac.uk/

See Also

`plot`

Examples

```
  ## Not run:
    trad.scatter.plot(exprs(eset.rma)[,1],exprs(eset.rma)[,4])

## End(Not run)
```

Index