Package ‘affy’

Version 1.52.0

Title Methods for Affymetrix Oligonucleotide Arrays

Author Rafael A. Irizarry <rafa@jhu.edu>, Laurent Gautier
<lgautier@gmail.com>, Benjamin Milo Bolstad
<bmb@bmbolstad.com>, and Crispin Miller
<cmiller@picr.man.ac.uk> with contributions from Magnus
Astrand <Magnus.Astrand@astrazeneca.com>, Leslie M. Cope
<ccope@mts.jhu.edu>, Robert Gentleman, Jeff Gentry, Conrad
Halling <challing@agilixcorp.com>, Wolfgang Huber, James
MacDonald <jmacdon@u.washington.edu>, Benjamin I. P. Rubinstein,
Christopher Workman <workman@cbs.dtu.dk>, John Zhang

Maintainer Rafael A. Irizarry <rafa@jhu.edu>

Depends R (>= 2.8.0), BiocGenerics (>= 0.1.12), Biobase (>= 2.5.5)

Imports affyio (>= 1.13.3), BiocInstaller, graphics, grDevices,
methods, preprocessCore, stats, utils, zlibbioc

Suggests tkWidgets (>= 1.19.0), affydata, widgetTools

LinkingTo preprocessCore

Description The package contains functions for exploratory
oligonucleotide array analysis. The dependence on tkWidgets
only concerns few convenience functions. ‘affy’ is fully
functional without it.

License LGPL (>= 2.0)

generateExprVal.method.avgdiff.R
generateExprVal.method.liwong.R generateExprVal.method.mas.R
generateExprVal.method.medianpolish.R
normalize.constant.R normalize.contrasts.R
normalize.invariantset.R normalize.loess.R normalize.qspline.R
R topics documented:

**biocViews**  Microarray, OneChannel, Preprocessing

**LazyLoad**  yes

**NeedsCompilation**  yes

R topics documented:

- .setAffyOptions
- affy-deprecated
- affy-options
- affy.scalevalue.exprSet
- AffyBatch-class
- AffyRNAdeg
- barplot.ProbeSet
- bg.adjust
- bg.correct
- cdfenv.example
- cdfFromBioC
- cleancdfname
- debug.affy123
- expresso
- expressoWidget
- fit.li.wong
- generateExprSet-method
- generateExprVal
- generateExprVal.method.avgdiff
- generateExprVal.method.playerout
- hlog
- justRMA
- list.celfiles
- MAplot
- mas5
- mas5calls
- merge.AffyBatch
- mva.pairs
- normalize-methods
- normalize.constant
- normalize.contrasts
- normalize.invariantset
- normalize.loess
- normalize.qspline
- normalize.quantiles
- normalize.quantiles.robust
- pairs.AffyBatch
- plot.ProbeSet
- plotDensity
- plotLocation
- pmcorrect
- ppsetApply
- probeMatch-methods
- probeNames-methods
- ProbeSet-class
.setAffyOptions

Description
~~ Set the options for the package

Usage
.setAffyOptions(affy.opt = NA)

Arguments

affy.opt A list structure of options. If NA, the default options are set.

Details
See the vignettes to know more. This function could disappear in favor of a more general one the package Biobase.

Value
The function is used for its side effect. Nothing is returned.

Author(s)
Laurent

Examples

affy.opt <- getOption("BioC")$affy
.setAffyOptions(affy.opt)
Description

These functions are provided for compatibility with older versions of affy only, and will be defunct at the next release.

Details

The following functions are deprecated and will be made defunct; use the replacement indicated below:

- loess.normalize: `normalize.loess`
- maffy.normalize
- multiloess
- simplemultiLoess

Note

The affy package options are contained in the Bioconductor options. The options are:

- `use.widgets`: a logical used to decide on the default of widget use.
- `compress.cel`: a logical
- `compress.cdf`: a logical
- `probes.loc`: a list. Each element of the list is itself a list with two elements `what` and `where`. When looking for the information about the locations of the probes on the array, the elements in the list will be looked at one after the other. The first one for which `what` and `where` lead to the matching locations information is used. The element `what` can be one of `package`, `environment` or `file`. The element `where` depends on the corresponding element `what`:
  - if `package`: location for the package (like it would be for the argument `lib.loc` for the function `library`.)
  - if `environment`: an environment to look for the information (like the argument `env` for the function `get`).
  - if `file`: a character with the path in which a CDF file can be found.
Examples

```r
## get the options
opt <- getOption("BioC")
affy.opt <- opt$affy

## list their names
names(affy.opt)

## set the option compress.cel
affy.opt$compress.cel <- TRUE
options(BioC=opt)
```

affy.scalevalue.exprSet

*Scale normalization for exprSets*

Description

Normalizes expression values using the method described in the Affymetrix user manual.

Usage

```r
affy.scalevalue.exprSet(eset, sc = 500, analysis="absolute")
```

Arguments

- **eset**: An `ExpressionSet` object.
- **sc**: Value at which all arrays will be scaled to.
- **analysis**: Should we do absolute or comparison analysis, although "comparison" is still not implemented.

Details

This function was implemented from the Affymetrix technical documentation for MAS 5.0. It can be downloaded from the website of the company. Please refer to this document for details.

Value

A normalized `ExpressionSet`.

Author(s)

Laurent
Description

This is a class representation for Affymetrix GeneChip probe level data. The main component are the intensities from multiple arrays of the same CDF type. It extends eSet.

Objects from the Class

Objects can be created using the function `read.affybatch` or the wrapper `ReadAffy`.

Slots

cdfName: Object of class character representing the name of CDF file associated with the arrays in the AffyBatch.
nrow: Object of class integer representing the physical number of rows in the arrays.
ncol: Object of class integer representing the physical number of columns in the arrays.
assayData: Object of class AssayData containing the raw data, which will be at minimum a matrix of intensity values. This slot can also hold a matrix of standard errors if the 'sd' argument is set to TRUE in the call to ReadAffy.
phenoData: Object of class AnnotatedDataFrame containing phenotypic data for the samples.
annotation A character string identifying the annotation that may be used for the ExpressionSet instance.
protocolData: Object of class AnnotatedDataFrame containing protocol data for the samples.
featureData Object of class AnnotatedDataFrame containing feature-level (e.g., probeset-level) information.
experimentData: Object of class "MIAME" containing experiment-level information.
%cclassVersion__: Object of class Versions describing the R and Biobase version number used to create the instance. Intended for developer use.

Extends

Class "eSet", directly.

Methods

cdfName signature(object = "AffyBatch"): obtains the cdfName slot.
pm<- signature(object = "AffyBatch"): replaces the perfect match intensities.
pm signature(object = "AffyBatch"): extracts the pm intensities.
mm<- signature(object = "AffyBatch"): replaces the mismatch intensities.
mm signature(object = "AffyBatch"): extracts the mm intensities.
probes signature(object = "AffyBatch", which): extract the perfect match or mismatch probe intensities. Uses which can be "pm" and "mm".
exprs signature(object = "AffyBatch"): extracts the expression matrix.
exprs<- signature(object = "AffyBatch", value = "matrix"): replaces the expression matrix.
AffyBatch-class

se.exprs signature(object = "AffyBatch"): extracts the matrix of standard errors of expression values, if available.

se.exprs<- signature(object = "AffyBatch", value = "matrix"): replaces the matrix of standard errors of expression values.

[<- signature(x = "AffyBatch"): replaces subsets.

[ signature(x = "AffyBatch"): subsets by array.

boxplot signature(x = "AffyBatch"): creates a boxplots of log base 2 intensities (pm, mm or both). Defaults to both.

hist signature(x = "AffyBatch"): creates a plot showing all the histograms of the pm,mm or both data. See plotDensity.

computeExprSet signature(x = "AffyBatch", summary.method = "character"): For each probe set computes an expression value using summary.method.

featureNames signature(object = "AffyBatch"): return the probe set names also referred to as the Affymetrix IDs. Notice that one can not assign featureNames. You must do this by changing the cdfenvs.

geneNames signature(object = "AffyBatch"): deprecated, use featureNames.

getCodeInfo signature(object = "AffyBatch"): retrieve the environment that defines the location of probes by probe set.

image signature(x = "AffyBatch"): creates an image for each sample.

indexProbes signature(object = "AffyBatch", which = "character"): returns a list with locations of the probes in each probe set. The affyID corresponding to the probe set to retrieve can be specified in an optional parameter genenames. By default, all the affyIDs are retrieved. The names of the elements in the list returned are the affyIDs, which can be "pm", "mm", or "both". If "both" then perfect match locations are given followed by mismatch locations.

signature(object = "AffyBatch", which = "missing") (i.e., calling indexProbes without a "which" argument) is the same as setting "which" to "pm".

intensity<- signature(object = "AffyBatch"): a replacement method for the exprs slot, i.e. the intensities.

intensity signature(object = "AffyBatch"): extract the exprs slot, i.e. the intensities.

length signature(x = "AffyBatch"): returns the number of samples.

pmindex signature(object = "AffyBatch"): return the location of perfect matches in the intensity matrix.

mmindex signature(object = "AffyBatch"): return the location of the mismatch intensities.

dim signature(x = "AffyBatch"): Row and column dimensions.

ncol signature(x = "AffyBatch"): An accessor function for ncol.

nrow signature(x = "AffyBatch"): an accessor function for nrow.

normalize signature(object = "AffyBatch"): a method to normalize. The method accepts an argument method. The default methods is specified in package options (see the main vignette).

normalize.methods signature(object = "AffyBatch"): returns the normalization methods defined for this class. See normalize.

probeNames signature(object = "AffyBatch"): returns the probe set associated with each row of the intensity matrix.

probeset signature(object = "AffyBatch", genenames=NULL, locations=NULL): Extracts ProbeSet objects related to the probe sets given in genenames. If an alternative set of locations defining pms and mms a list with those locations should be passed via the locations argument.
AffyRNAdeg

Function to assess RNA degradation in Affymetrix GeneChip data.

Description

Uses ordered probes in probeset to detect possible RNA degradation. Plots and statistics used for evaluation.

Usage

AffyRNAdeg(abatch, log.it=TRUE)
summaryAffyRNAdeg(rna.deg.obj, signif.digits=3)
plotAffyRNAdeg(rna.deg.obj, transform = "shift.scale", cols = NULL, ...)

Note

This class is better described in the vignette.

See Also

related methods merge.AffyBatch, pairs.AffyBatch, and eSet

Examples

if (require(affydata)) {
  ## load example
data(Dilution)

  ## nice print
  print(Dilution)

  pm(Dilution)[1:5,]
  mm(Dilution)[1:5,]

  ## get indexes for the PM probes for the affyID "1900_at"
mypmindex <- pmindex(Dilution,"1900_at")
  ## same operation using the primitive
  mypmindex <- indexProbes(Dilution, which="pm", genenames="1900_at")[[1]]

  ## get the probe intensities from the index
  intensity(Dilution)[mypmindex, ]

  description(Dilution) ##we can also use the methods of eSet
  sampleNames(Dilution)
  abstract(Dilution)
}

bg.correct signature(object = "AffyBatch", method="character") applies background correction methods defined by method.

updateObject signature(object = "AffyBatch", ..., verbose=FALSE): update, if necessary, an object of class AffyBatch to its current class definition. verbose=TRUE provides details about the conversion process.
Arguments

abatch An object of class AffyBatch-class.

log.it A logical argument: If log.it=T, then probe data is log2 transformed.

rna.deg.obj Output from AffyRNAdeg.

signif.digits Number of significant digits to show.

transform Possible choices are "shift.scale","shift.only", and "neither". "Shift" vertically lagger the plots for individual chips, to make the display easier to read. "Scale" normalizes so that standard deviation is equal to 1.

cols A vector of colors for plot, length = number of chips.

... further arguments for plot function.

Details

Within each probeset, probes are numbered directionally from the 5’ end to the 3’ end. Probe intensities are averaged by probe number, across all genes. If log.it=FALSE and transform="Neither", then plotAffyRNAdeg simply shows these means for each chip. Shifted and scaled versions of the plot can make it easier to see.

Value

AffyRNAdeg returns a list with the following components:

sample.names names of samples, derived from affy batch object

means.by.number average intensity by probe position

ses standard errors for probe position averages

slope from linear regression of means.by.number

pvalue from linear regression of means.by.number

Author(s)

Leslie Cope

Examples

if (require(affydata)) {
  data(Dilution)
  RNAdeg<-AffyRNAdeg(Dilution)
  plotAffyRNAdeg(RNAdeg)
}
**Description**

Displays the probe intensities in a ProbeSet as barplots

**Usage**

```r
## S3 method for class 'ProbeSet'
barplot(height, xlab = "Probe pair", ylab = "Intensity",
        main = NA, col.pm = "red", col.mm = "blue", beside = TRUE, names.arg = "pp",
        ask = TRUE, scale, ...)
```

**Arguments**

- `height`: an object of class `ProbeSet`.
- `xlab`: label for x axis.
- `ylab`: label for y axis.
- `main`: main label for the figure.
- `col.pm`: color for the 'pm' intensities.
- `col.mm`: color for the 'mm' intensities.
- `beside`: bars beside each others or not.
- `names.arg`: names to be plotted below each bar or group of bars.
- `ask`: ask before plotting the next barplot.
- `scale`: put all the barplot to the same scale.
- `...`: extra parameters to be passed to `barplot`.

**Examples**

```r
if (require(affydata)) {
  data(Dilution)
  gn <- geneNames(Dilution)
  pps <- probeset(Dilution, gn[1][1])

  barplot.ProbeSet(pps)
}
```
**bg.adjust**

**Background adjustment (internal function)**

**Description**

An internal function to be used by `bg.correct.rma`.

**Usage**

```r
bg.adjust(pm, n.pts = 2^14, ...)  
bg.parameters(pm, n.pts = 2^14)
```

**Arguments**

- `pm` a pm matrix
- `n.pts` number of points to use in call to `density`
- `...` extra arguments to pass to `bg.adjust`.

**Details**

Assumes PMs are a convolution of normal and exponential. So we observe X+Y where X is background and Y is signal. `bg.adjust` returns E[Y|X+Y, Y>0] as our background corrected PM. `bg.parameters` provides ad hoc estimates of the parameters of the normal and exponential distributions.

**Value**

a matrix

**See Also**

- `bg.correct.rma`

---

**bg.correct**

**Background Correction**

**Description**

Background corrects probe intensities in an object of class `AffyBatch`.

**Usage**

```r
bg.correct(object, method, ...)  
bg.correct.rma(object,...)  
bg.correct.mas(object, griddim)  
bg.correct.none(object, ...)
```
Arguments

object    An object of class AffyBatch.
method    A character that defines what background correction method will be used. Available methods are given by bg.correct.methods.
griddim   grid dimension used for mas background estimate. The array is divided into griddim equal parts. Default is 16.
...       arguments to pass along to the engine function.

Details

The name of the method to apply must be double-quoted. Methods provided with the package are currently:

* bg.correct.none: returns object unchanged.
* bg.correct.chipwide: noise correction as described in a ‘white paper’ from Affymetrix.
* bg.correct.rma: the model based correction used by the RMA expression measure.

They are listed in the variable bg.correct.methods. The user must supply the word after "bg.correct", i.e none, subtractmm, rma, etc...

More details are available in the vignette.

R implementations similar in function to the internal implementation used by bg.correct.rma are in bg.adjust.

Value

An AffyBatch for which the intensities have been background adjusted. For some methods (RMA), only PMs are corrected and the MMs remain the same.

Examples

```r
if (require(affydata)) {
  data(Dilution)

  ##bgc will be the bg corrected version of Dilution
  bgc <- bg.correct(Dilution, method="rma")

  ##This plot shows the tranformation
  plot(pm(Dilution)[,1],pm(bgc)[,1],log="xy",
       main="PMs before and after background correction")
}
```

Description

Example cdfenv (environment containing the probe locations).

Usage

data(cdfenv.example)
Format

An environment `cddfenv.example` containing the probe locations

Source

Affymetrix CDF file for the array Hu6800

dcdfFromBioC  Functions to obtain CDF files

dcdfFromBioC

dcdfFromLibPath

dcdfFromEnvironment

Description

A set of functions to obtain CDF files from various locations.

Usage

```r
cdfFromBioC(cdfname, lib = .libPaths()[1], verbose = TRUE)
cdfFromLibPath(cdfname, lib = NULL, verbose=TRUE)
cdfFromEnvironment(cdfname, where, verbose=TRUE)
```

Arguments

- `cdfname` name of the CDF.
- `lib` install directory for the CDF package.
- `where` environment to search.
- `verbose` logical controlling extra output.

Details

These functions all take a requested CDF environment name and will attempt to locate that environment in the appropriate location (a package's data directory, as a CDF package in the .libPaths(), from a loaded environment or on the Bioconductor website. If the environment can not be found, it will return a list of the methods tried that failed.

Value

The CDF environment or a list detailing the failed locations.

Author(s)

Jeff Gentry
**cleancdfname**  
*Clean Affymetrix’s CDF name*

**Description**

This function converts Affymetrix’s names for CDF files to the names used in the annotation package and in all Bioconductor.

**Usage**

```r
cleancdfname(cdfname, addcdf = TRUE)
```

**Arguments**

- `cdfname`: A character denoting Affymetrix’s CDF file name
- `addcdf`: A logical. If TRUE it adds the string “cdf” at the end of the cleaned CDF name. This is used to name the cdfenvs packages.

**Details**

This function takes a CDF filename obtained from an Affymetrix file (from a CEL file for example) and convert it to a convention of ours: all small caps and only alphanumeric characters. The details of the rule can be seen in the code. We observed exceptions that made us create a set of special cases for mapping CEL to CDF. The object `mapCdfName` holds information about these cases. It is a `data.frame` of three elements: the first is the name as found in the CDF file, the second the name in the CEL file and the third the name in Bioconductor. `mapCdfName` can be loaded using `data(mapCdfName)`.

**Value**

A character

**Examples**

```r
cdf.tags <- c("HG_U95Av2", "HG-133A")
for (i in cdf.tags)
  cat(i, "becomes", cleancdfname(i), "\n")
```

**debug.affy123**  
*Debugging Flag*

**Description**

For developmental use only
from raw probe intensities to expression values

**Description**

Goes from raw probe intensities to expression values

**Usage**

```r
expresso(
  afbatch,
  # background correction
  bg.correct = TRUE,
  bgcorrect.method = NULL,
  bgcorrect.param = list(),
  # normalize
  normalize = TRUE,
  normalize.method = NULL,
  normalize.param = list(),
  # pm correction
  pmcorrect.method = NULL,
  pmcorrect.param = list(),
  # expression values
  summary.method = NULL,
  summary.param = list(),
  summary.subset = NULL,
  # misc.
  verbose = TRUE,
  widget = FALSE)
```

**Arguments**

- `afbatch`: an `AffyBatch` object.
- `bg.correct`: a boolean to express whether background correction is wanted or not.
- `bgcorrect.method`: the name of the background adjustment method.
- `bgcorrect.param`: a list of parameters for `bgcorrect.method` (if needed/wanted).
- `normalize`: normalization step wished or not.
- `normalize.method`: the normalization method to use.
- `normalize.param`: a list of parameters to be passed to the normalization method (if wanted).
- `pmcorrect.method`: the name of the PM adjustment method.
- `pmcorrect.param`: a list of parameters for `pmcorrect.method` (if needed/wanted).
- `summary.method`: the method used for the computation of expression values.
summary.param a list of parameters to be passed to the summary.method (if wanted).
summary.subset a list of 'affyids'. If NULL, an expression summary value is computed for every-
thing on the chip.
verbose logical value. If TRUE, it writes out some messages.
widget a boolean to specify the use of widgets (the package tkWidget is required).

Details
Some arguments can be left to NULL if the widget=TRUE. In this case, a widget pops up and let the
user choose with the mouse. The arguments are: AffyBatch, bgcorrect.method, normalize.method,
pmcorrect.method and summary.method.
For the mas 5.0 and 4.0 methods ones need to normalize after obtaining expression. The function
affy.scalevalue.exprSet does this.
For the Li and Wong summary method notice you will not get the same results as you would get
with dChip. dChip is not open source so it is not easy to reproduce. Notice also that this iterative
algorithm will not always converge. If you run the algorithm on thousands of probes expect some
non-convergence warnings. These are more likely when few arrays are used. We recommend using
this method only if you have 10 or more arrays. Please refer to the fit.li.wong help page for more
details.

Value
An object of class ExpressionSet, with an attribute pps.warnings as returned by the method
computeExprSet.

See Also
AffyBatch

Examples
if (require(affydata)) {
  data(Dilution)

  eset <- expresso(Dilution, bgcorrect.method="rma",
                   normalize.method="constant", pmcorrect.method="pmonly",
                   summary.method="avgdiff")

  ##to see options available for bg correction type:
  bgcorrect.methods()
}

expressoWidget  A widget for users to pick correction methods

Description
This widget is called by expresso to allow users to select correction methods that will be used to
process affy data.
Usage

```
expressoWidget(BGMethods, normMethods, PMMethods, expMethods, BGDefault,
               normDefault, PMDefault, expDefault)
```

Arguments

- **BGMethods**: a vector of character strings for the available methods that can be used as a background correction method of affy data.
- **normMethods**: a vector of character strings for the available methods that can be used as a normalization method of affy data.
- **PMMethods**: a vector of character strings for the available methods that can be used as a PM correction method of affy data.
- **expMethods**: a vector of character strings for the available methods that can be used as a summary method of affy data.
- **BGDefault**: a character string for the name of a default background correction method.
- **normDefault**: a character string for the name of a default normalization method.
- **PMDefault**: a character string for the name of a default PM correction method.
- **expDefault**: a character string for the name of a default summary method.

Details

The widget will be invoked when expresso is called with argument "widget" set to TRUE. Default values can be changed using the drop down list boxes. Double clicking on an option from the drop-down list makes an selection. The first element of the list for available methods will be the default method if no default is provided.

Value

The widget returns a list of selected correction methods.

- **BG**: background correction method
- **NORM**: normalization method
- **PM**: PM correction method
- **EXP**: summary method

Author(s)

Jianhua Zhang

References

Documentations of affy package

See Also

```
expresso
```
Examples
if(interactive()){
  require(widgetTools)
  expressoWidget(c("mas", "none", "rma"), c("constant", "quantiles"),
    c("mas", "pmonly"), c("liwong", "playerout"))
}

fit.li.wong Fit Li and Wong Model to a Probe Set

Description
Fits the model described in Li and Wong (2001) to a probe set with I chips and J probes.

Usage
fit.li.wong(data.matrix, remove.outliers=TRUE, normal.array.quantile=0.5,
  normal.resid.quantile=0.9, large.threshold=3, large.variation=0.8,
  outlier.fraction=0.14, delta=1e-06, maxit=50,
  outer.maxit=50, verbose=FALSE, ...
)
li.wong(data.matrix, remove.outliers=TRUE, normal.array.quantile=0.5,
  normal.resid.quantile=0.9, large.threshold=3, large.variation=0.8,
  outlier.fraction=0.14, delta=1e-06, maxit=50,
  outer.maxit=50, verbose=FALSE)

Arguments

- data.matrix: an I x J matrix containing the probe set data. Typically the i,j
  entry will contain the PM-MM value for probe pair j in chip i. Another possible
  use, is to use PM instead of PM-MM.
- remove.outliers: logical value indicating if the algorithm will remove outliers
  according to the procedure described in Li and Wong (2001).
- large.threshold: used to define outliers.
- normal.array.quantile: quantile to be used when determining what a normal SD is. probes
  or chips having estimates with SDs bigger than the quantile normal.array.quantile
  of all SDs x large.threshold.
- normal.resid.quantile: any residual bigger than the normal.resid.quantile quantile of all
  residuals x large.threshold is considered an outlier.
- large.variation: any probe or chip describing more than this much total variation
  is considered an outlier.
- outlier.fraction: this is the maximum fraction of single outliers that can be in the same probe
  or chip.
- delta: numerical value used to define the stopping criterion.
maxit  maximum number of iterations when fitting the model.
outer.maxit  maximum number of iterations of defined outliers.
verbose  logical value. If TRUE information is given of the status of the algorithm.
...  additional arguments.

Details
This is Bioconductor’s implementation of the Li and Wong algorithm. The Li and Wong PNAS 2001 paper was followed. However, you will not get the same results as you would get with dChip. dChip is not open source so it is not easy to reproduce.

Notice that this iterative algorithm will not always converge. If you run the algorithm on thousands of probes expect some non-convergence warnings. These are more likely when few arrays are used. We recommend using this method only if you have 10 or more arrays.

Please refer to references for more details.

Value
li.wong returns a vector of expression measures (or column effects) followed by their respective standard error estimates. It was designed to work with express which is no longer part of the package.

fit.li.wong returns much more. Namely, a list containing the fitted parameters and relevant information.

theta  fitted thetas.
phi  fitted phis.
sigma.eps  estimated standard deviation of the error term.
sigma.theta  estimated standard error of theta.
sigma.phi  estimated standard error of phis.
theta.outliers  logical vector describing which chips (thetas) are considered outliers (TRUE).
phi.outliers  logical vector describing which probe sets (phis) are considered outliers (TRUE)
convergence1  logical value. If FALSE the algorithm did not converge when fitting the phis and thetas.
convergence2  logical value. If FALSE the algorithm did not converge in deciding what are outliers.
iter  number of iterations needed to achieve convergence.
delta  difference between thetas when iteration stopped.

Author(s)
Rafael A. Irizarry, Cheng Li, Fred A. Wright, Ben Bolstad

References

generateExprSet-method

generate a set of expression values

Description

Generate a set of expression values from the probe pair information. The set of expression is returned as an ExpressionSet object.

Usage

computeExprSet(x, pmcorrect.method, summary.method, ...)
generateExprSet.methods()
upDate.generateExprSet.methods(x)

Arguments

x a AffyBatch holding the probe level informations to generate the expression values, for computeExprSet, and for upDate.generateExprSet.methods it is a character vector..

pmcorrect.method
the method used to correct PM values (see section 'details').

summary.method
the method used to generate the expression value (see section 'details').

... any of the options of the normalization you would like to modify.

Details

An extra argument ids= can be passed. It must be a vector of affyids. The expression values will only be computed and returned for these affyids.

The different methods available through this mechanism can be accessed by calling the method generateExprSet.methods with an object of call Cel.container as an argument.

In the Affymetrix design, MM probes were included to measure the noise (or background signal). The original algorithm for background correction was to subtract the MM signal to the PM signal. The methods currently included in the package are "bg.correct.subtractmm", "bg.correct.pmonly" and "bg.correct.adjust".

To alter the available methods for generating ExprSets use upDate.generateExprSet.methods.
See Also

method generateExprSet of the class AffyBatch

Examples

if (require(affydata)) {
  data(Dilution)

  ids <- c( "1000_at","1001_at")

  eset <- computeExprSet(Dilution, pmcorrect.method="pmonly",
                           summary.method="avgsd", ids=ids)
}

---

**generateExprVal**

*Compute a summary expression value from the probes intensities*

**Description**

Compute a summary expression value from the probes intensities

**Usage**

```r
express.summary.stat(x, pmcorrect, summary, ...)
express.summary.stat.methods() # vector of names of methods
upDate.express.summary.stat.methods(x)
```

**Arguments**

- `x` a (ProbeSet)
- `pmcorrect` the method used to correct the PM values before summarizing to an expression value.
- `summary` the method used to generate the expression value.
- `...` other parameters the method might need... (see the corresponding methods below...)

**Value**

Returns a vector of expression values.

**Examples**

```r
if (require(affydata)) {
  data(Dilution)

  p <- probeset(Dilution, "1001_at")[[1]]

  par(mfcol=c(5,2))
  mymethods <- express.summary.stat.methods()
  nmet <- length(mymethods)
  nc <- ncol(pm(p))
```
layout(matrix(c(1:nc, rep(nc+1, nc)), nc, 2), width = c(1, 1))

barplot(p)

results <- matrix(0, nc, nmet)
rownames(results) <- paste("sample", 1:nc)
colnames(results) <- mymethods

for (i in 1:nmet) {
  ev <- express.summary.stat(p, summary=mymethods[[i]], pmcorrect="pmonly")
  if (mymethods[[i]] != "medianpolish")
    results[, i] <- 2^ev$exprs
  else
    results[, i] <- ev$exprs
}

dotchart(results, labels=paste("sample", 1:nc))

---

**generateExprVal.method.avgdiff**

*Generate an expression value from the probes informations*

---

**Description**

Generate an expression from the probes

**Usage**

```r
generateExprVal.method.avgdiff(probes, ...)
generateExprVal.method.medianpolish(probes, ...)
generateExprVal.method.liwong(probes, ...)
genereateExprVal.method.mas(probes, ...)
```

**Arguments**

- **probes**
  - a matrix of probe intensities with rows representing probes and columns representing samples. Usually `pm(probeset)` where `probeset` is a of class `ProbeSet`.

- **...**
  - extra arguments to pass to the respective function.

**Value**

A list containing entries:

- **exprs**
  - The expression values.

- **se.exprs**
  - The standard error estimate.

**See Also**

`generateExprSet-methods, generateExprVal.method.playerout, fit.li.wong`
Examples

data(SpikeIn) ##SpikeIn is a ProbeSets
probes <- pm(SpikeIn)
avgdiff <- generateExprVal.method.avgdiff(probes)
medianpolish <- generateExprVal.method.medianpolish(probes)
liwong <- generateExprVal.method.liwong(probes)
playerout <- generateExprVal.method.playerout(probes)
mas <- generateExprVal.method.mas(probes)

concentrations <- as.numeric(sampleNames(SpikeIn))
plot(concentrations,avgdiff$exprs,log="xy",ylim=c(50,10000),pch="a",type="b")
points(concentrations,medianpolish$exprs,pch="m",col=2,type="b",lty=2)
points(concentrations,liwong$exprs,pch="l",col=3,type="b",lty=3)
points(concentrations,playerout$exprs,pch="p",col=4,type="b",lty=4)
points(concentrations,mas$exprs,pch="p",col=4,type="b",lty=4)

---

generateExprVal.method.playerout

Generate an expression value from the probes informations

Description

Generate an expression from the probes

Usage

generateExprVal.method.playerout(probes, weights=FALSE, optim.method="L-BFGS-B")

Arguments

- probes: a list of probes slots from PPSet.container
- weights: Should the resulting weights be returned?
- optim.method: see parameter 'optim' for the function optim

Details

A non-parametric method to weight each perfect match probe in the set and to compute a weighted mean of the perfect match values. One will notice this method only makes use of the perfect matches. (see function playerout.costfunction for the cost function).

Value

A vector of expression values.

Author(s)

Laurent <laurent@cbs.dtu.dk>
(Thanks to E. Lazaridris for the original playerout code and the discussions about it)
References

Emmanuel N. Lazaridis, Dominic Sinibaldi, Gregory Bloom, Shrikant Mane and Richard Jove
A simple method to improve probe set estimates from oligonucleotide arrays, Mathematical Biosciences, Volume 176, Issue 1, March 2002, Pages 53-58

---

hlog  

**Hybrid Log**

**Description**

Given a constant \( c \) this function returns \( x \) if \( x \) is less than \( c \) and \( \text{sign}(x) \times (c \times \log(|x|/c) + c) \) if its not. Notice this is a continuous odd ( \( f(-x)=-f(x) \) ) function with continuous first derivative. The main purpose is to perform log transformation when one has negative numbers, for example for PM-MM.

**Usage**

\[ \text{hlog}(x, \text{constant}=1) \]

**Arguments**

- \( x \) a number.
- \( \text{constant} \) the constant \( c \) (see description).

**Details**

If \( \text{constant} \) is less than or equal to 0 \( \log(x) \) is returned for all \( x \). If \( \text{constant} \) is infinity \( x \) is returned for all \( x \).

**Author(s)**

Rafael A. Irizarry

---

justRMA  

**Read CEL files into an ExpressionSet**

**Description**

Read CEL files and compute an expression measure without using an AffyBatch.
Usage

```r
just.rma(..., filenames = character(0),
  phenoData = new("AnnotatedDataFrame"),
  description = NULL,
  notes = "",
  compress = getOption("BioC")$affy$compress.cel,
  rm.mask = FALSE, rm.outliers = FALSE, rm.extra = FALSE,
  verbose=FALSE, background=TRUE, normalize=TRUE,
  bgversion=2, destructive=FALSE, cdfname = NULL)
```

```r
justRMA(..., filenames=character(0),
  widget=getOption("BioC")$affy$use.widgets,
  compress=getOption("BioC")$affy$compress.cel,
  celfile.path=getwd(),
  sampleNames=NULL,
  phenoData=NULL,
  description=NULL,
  notes="",
  rm.mask=FALSE, rm.outliers=FALSE, rm.extra=FALSE,
  hdf5=FALSE, hdf5FilePath=NULL,verbose=FALSE,
  normalize=TRUE, background=TRUE,
  bgversion=2, destructive=FALSE, cdfname = NULL)
```

Arguments

... file names separated by comma.
filenames file names in a character vector.
phenoData an AnnotatedDataFrame object.
description a MIAME object.
notes notes.
compress are the CEL files compressed?
rm.mask should the spots marked as 'MASKS' set to NA?
rm.outliers should the spots marked as 'OUTLIERS' set to NA?
rm.extra if TRUE, then overrides what is in rm.mask and rm.outliers.
hdf5 use of hdf5 ? (not available yet)
hdf5FilePath a filename to use with hdf5 (not available yet).
verbose verbosity flag.
widget a logical specifying if widgets should be used.
celfile.path a character denoting the path ReadAffy should look for cel files.
sampleNames a character vector of sample names to be used in the AffyBatch.
normalize logical value. If TRUE, then normalize data using quantile normalization.
background logical value. If TRUE, then background correct using RMA background correction.
bgversion integer value indicating which RMA background to use: 1: use background similar to pure R rma background given in affy version 1.0 - 1.0.2
2: use background similar to pure R rma background given in affy version 1.1 and above
destructive

logical value. If TRUE, then works on the PM matrix in place as much as possible, good for large datasets.

cdfname

Used to specify the name of an alternative cdf package. If set to NULL, then the usual cdf package based on Affymetrix’ mappings will be used.

Details

justRMA is a wrapper for just.rma that permits the user to read in phenoData, MIAME information, and CEL files using widgets. One can also define files where to read phenoData and MIAME information.

If the function is called with no arguments justRMA(), then all the CEL files in the working directory are read, converted to an expression measure using RMA and put into an ExpressionSet. However, the arguments give the user great flexibility.

phenoData is read using read.AnnotatedDataFrame. If a character is given, it tries to read the file with that name to obtain the AnnotatedDataFrame object as described in read.AnnotatedDataFrame. If left NULL and widget=FALSE (widget=TRUE is not currently supported), then a default object is created. It will be an object of class AnnotatedDataFrame with its pData being a data.frame with column x indexing the CEL files.

description is read using read.MIAME. If a character is given, it tries to read the file with that name to obtain a MIAME instance. If left NULL but widget=TRUE, then widgets are used. If left NULL and widget=FALSE, then an empty instance of MIAME is created.

The arguments rm.masks, rm.outliers, rm.extra are passed along to the function read.celfile.

Value

An ExpressionSet object, containing expression values identical to what one would get from running rma on an AffyBatch.

Author(s)

In the beginning: James MacDonald <jmacdon@med.umich.edu> Supporting routines, maintenance and just.rma: Ben Bolstad <bmb@bmbolstad.com>

See Also

rma, read.affybatch

list.celfiles

List the Cel Files in a Directory/Folder

Description

This function produces a vector containing the names of files in the named directory/folder ending in .cel or .CEL.

Usage

list.celfiles(...)
MAplot

Arguments

... arguments to pass along to `list.files`

Value

A character vector of file names.

See Also

`list.files`

Examples

`list.celfiles()`

---

**MAplot**

Relative M vs. A plots

Description

Create boxplots of M or M vs A plots. Where M is determined relative to a specified chip or to a pseudo-median reference chip.

Usage

```r
MAplot(object,...)  
Mbox(object,...)  
ma.plot(A, M, subset = sample(1:length(M), min(c(10000, length(M)))),  
show.statistics = TRUE, span = 2/3, family.loess = "gaussian",  
cex = 2, plot.method = c("normal","smoothScatter","add"),  
add.loess = TRUE, lwd = 1, lty = 1, loess.col = "red", ...)
```

Arguments

- **object**
  - an `AffyBatch-class`
- **...**
  - additional parameters for the routine.
- **A**
  - a vector to plot along the horizontal axis.
- **M**
  - a vector to plot along vertical axis.
- **subset**
  - a set of indices to use when drawing the loess curve.
- **show.statistics**
  - logical. If TRUE, some summary statistics of the M values are drawn.
- **span**
  - span to be used for loess fit.
- **family.loess**
  - "guassian" or "symmetric" as in `loess`.
- **cex**
  - size of text when writing summary statistics on plot.
- **plot.method**
  - a string specifying how the plot is to be drawn. "normal" plots points, "smoothScatter" uses the `smoothScatter` function. Specifying "add" means that the MAplot should be added to the current plot.
- **add.loess**
  - add a loess line to the plot.
- **lwd**
  - width of loess line.
- **lty**
  - line type for loess line.
- **loess.col**
  - color for loess line.
mas5

MAS 5.0 expression measure

Description

This function converts an instance of AffyBatch into an instance of ExpressionSet using our implementation of Affymetrix’s MAS 5.0 expression measure.

Usage

mas5(object, normalize = TRUE, sc = 500, analysis = "absolute", ...)

Arguments

- object: an instance of AffyBatch
- normalize: logical. If TRUE scale normalization is used after we obtain an instance of ExpressionSet
- sc: Value at which all arrays will be scaled to.
- analysis: should we do absolute or comparison analysis, although "comparison" is still not implemented.
- ...: other arguments to be passed to expresso.

Details

This function is a wrapper for expresso and affy.scalevalue.exprSet.

Value

ExpressionSet

The methods used by this function were implemented based upon available documentation. In particular a useful reference is Statistical Algorithms Description Document by Affymetrix. Our implementation is based on what is written in the documentation and, as you might appreciate, there are places where the documentation is less than clear. This function does not give exactly the same results. All source code of our implementation is available. You are free to read it and suggest fixes.

For more information visit this URL: http://stat-www.berkeley.edu/users/bolstad/

See Also

expresso, affy.scalevalue.exprSet

Examples

if (require(affydata)) {
  data(Dilution)
  MAplot(Dilution)
  Mbox(Dilution)
}

See Also

mva.pairs
Examples

```r
if (require(affydata)) {
  data(Dilution)
  eset <- mas5(Dilution)
}
```

---

**mas5calls**

**MAS 5.0 Absolute Detection**

---

**Description**

Performs the Wilcoxon signed rank-based gene expression presence/absence detection algorithm first implemented in the Affymetrix Microarray Suite version 5.

**Usage**

- `mas5calls(object,...)`
- `mas5calls.AffyBatch(object, ids = NULL, verbose = TRUE, tau = 0.015, alpha1 = 0.04, alpha2 = 0.06, ignore.saturated=TRUE)`
- `mas5calls.ProbeSet(object, tau = 0.015, alpha1 = 0.04, alpha2 = 0.06, ignore.saturated=TRUE)`
- `mas5.detection(mat, tau = 0.015, alpha1 = 0.04, alpha2 = 0.06, exact.pvals = FALSE, cont.correct = FALSE)`

**Arguments**

- `object` an object of class `AffyBatch` or `ProbeSet`.
- `ids` probeset IDs for which you want to compute calls.
- `mat` an n-by-2 matrix of paired values (pairs in rows), PMs first col.
- `verbose` logical. It `TRUE`, status of processing is reported.
- `tau` a small positive constant.
- `alpha1` a significance threshold in (0, alpha2).
- `alpha2` a significance threshold in (alpha1, 0.5).
- `exact.pvals` logical controlling whether exact p-values are computed (irrelevant if n<50 and there are no ties). Otherwise the normal approximation is used.
- `ignore.saturated` if `TRUE`, do the saturation correction described in the paper, with a saturation level of 46000.
- `cont.correct` logical controlling whether continuity correction is used in the p-value normal approximation.
- `...` any of the above arguments that applies.
This function performs the hypothesis test:

H0: median(Ri) = \(\tau\), corresponding to absence of transcript

H1: median(Ri) > \(\tau\), corresponding to presence of transcript

where \(R_i = (PM_i - MM_i) / (PM_i + MM_i)\) for each \(i\) a probe-pair in the probe-set represented by data.

Currently exact.pvals=TRUE is not supported, and cont.correct=TRUE works but does not give great results (so both should be left as FALSE). The defaults for \(\tau\), alpha1 and alpha2 correspond to those in MAS5.0.

The p-value that is returned estimates the usual quantity:

\[
\text{Pr}(\text{observing a more "present looking" probe-set than data} \mid \text{data is absent})
\]

So that small p-values imply presence while large ones imply absence of transcript. The detection call is computed by thresholding the p-value as in:

call "P" if \(p\)-value < alpha1 call "M" if alpha1 <= \(p\)-value < alpha2 call "A" if alpha2 <= \(p\)-value

This implementation has been validated against the original MAS5.0 implementation with the following results (for exact.pvals and cont.correct set to F):

Average Relative Change from MAS5.0 p-values:38% Proportion of calls different to MAS5.0 calls:1.0%

where "average/proportion" means over all probe-sets and arrays, where the data came from 11 bacterial control probe-sets spiked-in over a range of concentrations (from 0 to 150 pico-mols) over 26 arrays. These are the spike-in data from the GeneLogic Concentration Series Spikein Dataset.

Clearly the p-values computed here differ from those computed by MAS5.0 – this will be improved in subsequent releases of the affy package. However the p-value discrepancies are small enough to result in the call being very closely aligned with those of MAS5.0 (99 percent were identical on the validation set) – so this implementation will still be of use.

The function mas5.detect is no longer the engine function for the others. C code is no available that computes the Wilcox test faster. The function is kept so that people can look at the R code (instead of C).

Value

mas5.detect returns a list containing the following components:

- **pval**
  a real p-value in \([0,1]\) equal to the probability of observing probe-level intensities that are more present looking than data assuming the data represents an absent transcript; that is a transcript is more likely to be present for p-values closer 0.

- **call**
  either "P", "M" or "A" representing a call of present, marginal or absent; computed by simply thresholding pval using alpha1 and alpha2.

The mas5calls method for AffyBatch returns an ExpressionSet with calls accessible with exprs(obj) and p-values available with assayData(obj)["se.exprs"]. The code mas5calls for ProbeSet returns a list with vectors of calls and p-values.

Author(s)

Crispin Miller, Benjamin I. P. Rubinstein, Rafael A. Irizarry
References

Examples
if (require(affydata)) {
  data(Dilution)
  PACalls <- mas5calls(Dilution)
}

merge.AffyBatch  merge two AffyBatch objects

Description
merge two AffyBatch objects into one.

Usage
## S3 method for class 'AffyBatch'
merge(x, y, annotation = paste(annotation(x),
      annotation(y)), description = NULL, notes = character(0), ...)

Arguments
  x        an AffyBatch object.
y        an AffyBatch object.
annotation a character vector.
description a character OR miame, eventually NULL.
notes a character vector.
...        additional arguments.

Details
To be done.

Value
A object if class AffyBatch.
### mva.pairs

**M vs. A Matrix**

#### Description

A matrix of M vs. A plots is produced. Plots are made on the upper triangle and the IQR of the Ms are displayed in the lower triangle.

#### Usage

```r
mva.pairs(x, labels=colnames(x), log.it=TRUE, span=2/3, family.loess="gaussian", digits=3, line.col=2, main="MVA plot", cex=2, ...)
```

#### Arguments

- `x`: a matrix containing the chip data in the columns.
- `labels`: the names of the variables.
- `log.it`: logical. If TRUE, uses log scale.
- `span`: span to be used for loess fit.
- `family.loess`: "gaussian" or "symmetric" as in `loess`.
- `digits`: number of digits to use in the display of IQR.
- `line.col`: color of the loess line.
- `main`: an overall title for the plot.
- `cex`: size for text.
- `...`: graphical parameters can be given as arguments to `mva.plot`.

#### See Also

- `pairs`

#### Examples

```r
x <- matrix(rnorm(4000),1000,4)
x[,1] <- x[,1]^2
dimnames(x) <- list(NULL,c("chip 1","chip 2","chip 3","chip 4"))
mva.pairs(x, log=FALSE, main="example")
```
normalize-methods

Normalize Affymetrix Probe Level Data - methods

Description

Method for normalizing Affymetrix Probe Level Data

Usage

normalize.methods(object)
bgcorrect.methods()
upDate.bgcorrect.methods(x)
pmcorrect.methods()
upDate.pmcorrect.methods(x)

Arguments

object An AffyBatch.
x A character vector that will replace the existing one.

Details

If object is an AffyBatch object, then normalize(object) returns an AffyBatch object with the intensities normalized using the methodology specified by getOption("BioC")$affy$normalize.method. The affy package default is quantiles.

Other methodologies can be used by specifying them with the method argument. For example to use the invariant set methodology described by Li and Wong (2001) one would type: normalize(object, method="invariantset").

Further arguments passed by ..., apart from method, are passed along to the function responsible for the methodology defined by the method argument.

A character vector of nicknames for the methodologies available is returned by normalize.methods(object), where object is an AffyBatch, or simply by typing normalize.AffyBatch.methods. If the nickname of a method is called "loess", the help page for that specific methodology can be accessed by typing ?normalize.loess.

For more on the normalization methodologies currently implemented please refer to the vignette ‘Custom Processing Methods’.

To add your own normalization procedures please refer to the customMethods vignette.

The functions: bgcorrect.methods, pmcorrect.methods, provide access to internal vectors listing the corresponding capabilities.

See Also

AffyBatch-class, normalize.

Examples

if (require(affydata)) {
data(Dilution)
normalize.methods(Dilution)
generateExprSet.methods()
bgcorrect.methods()}
normalize.constant

pmcorrect.methods()
}

normalize.constant

Scale probe intensities

Description

Scale array intensities in a AffyBatch.

Usage

normalize.AffyBatch.constant(abatch, refindex=1, FUN=mean, na.rm=TRUE)
normalize.constant(x, refconstant, FUN=mean, na.rm=TRUE)

Arguments

abatch an instance of the AffyBatch-class.
x a vector of intensities on a chip (to normalize to the reference).
refindex the index of the array used as a reference.
refconstant the constant used as a reference.
FUN a function generating a value from the intensities on an array. Typically mean or median.
na.rm parameter passed to the function FUN.

Value

An AffyBatch with an attribute "constant" holding the value of the factor used for scaling.

Author(s)

L. Gautier <laurent@cbs.dtu.dk>

See Also

AffyBatch
normalize.contrasts

Normalize intensities using the contrasts method

Description

Scale chip objects in an AffyBatch-class.

Usage

normalize.AffyBatch.contrasts(abatch, span=2/3, choose.subset=TRUE, subset.size=5000, verbose=TRUE, family="symmetric", type=c("together","pmonly","mmonly","separate"))

Arguments

- abatch: an AffyBatch-class object.
- span: parameter to be passed to the function loess.
- choose.subset: Boolean. Defaults to TRUE.
- subset.size: Integer. Number of probesets to use in each subset.
- verbose: verbosity flag.
- family: parameter to be passed to the function loess.
- type: a string specifying how the normalization should be applied.

Value

An object of the same class as the one passed.

See Also

maffy.normalize

normalize.invariantset

Invariant Set normalization

Description

Normalize arrays in an AffyBatch using an invariant set.

Usage

normalize.AffyBatch.invariantset(abatch, prd.td = c(0.003, 0.007), verbose = FALSE, baseline.type = c("mean","median","pseudo-mean","pseudo-median"), type = c("separate","pmonly","mmonly","together"))

normalize.invariantset(data, ref, prd.td=c(0.003,0.007))
**Arguments**

- `abatch`: an `AffyBatch` object.
- `data`: a vector of intensities on a chip (to normalize to the reference).
- `prd.td`: cutoff parameter (details in the bibliographic reference).
- `baseline.type`: specifies how to determine the baseline array.
- `type`: a string specifying how the normalization should be applied. See details for more.
- `verbose`: logical indicating printing throughout the normalization.

**Details**

The set of invariant intensities between `data` and `ref` is found through an iterative process (based on the respective ranks the intensities). This set of intensities is used to generate a normalization curve by smoothing.

The `type` argument should be one of "separate", "pmonly", "mmonly", "together" which indicates whether to normalize only one probe type (PM, MM) or both together or separately.

**Value**

Respectively a `AffyBatch` of normalized objects, or a vector of normalized intensities, with an attribute "invariant.set" holding the indexes of the "invariant" intensities.

**Author(s)**

L. Gautier <laurent@cbs.dtu.dk> (Thanks to Cheng Li for the discussions about the algorithm.)

**References**


**See Also**

`normalize` to normalize `AffyBatch` objects.

---

**normalize.loess**

*Scale microarray data*

**Description**

Normalizes arrays using loess.

**Usage**

```r
normalize.loess(mat, subset = sample(1:(dim(mat)[1]), min(c(5000, nrow(mat)))), epsilon = 10^-2, maxit = 1, log.it = TRUE, verbose = TRUE, span = 2/3, family.loess = "symmetric")
normalize.AffyBatch.loess(abatch,type=c("together","pmonly","mmonly","separate"), ...)
```
Arguments

mat  a matrix with columns containing the values of the chips to normalize.
abatch  an AffyBatch object.
subset  a subset of the data to fit a loess to.
epsilon  a tolerance value (supposed to be a small value - used as a stopping criterion).
maxit  maximum number of iterations.
log.it  logical. If TRUE it takes the log2 of mat
verbose  logical. If TRUE displays current pair of chip being worked on.
span  parameter to be passed the function loess
family.loess  parameter to be passed the function loess. "gaussian" or "symmetric" are acceptable values for this parameter.
type  A string specifying how the normalization should be applied. See details for more.
...  any of the options of normalize.loess you would like to modify (described above).

Details

The type argument should be one of "separate","pmonly","mmonly","together" which indicates whether to normalize only one probe type (PM,MM) or both together or separately.

See Also

normalize

Examples

if (require(affydata)) {
  #data(Dilution)
  #x <- pm(Dilution[,1:3])
  #mva.pairs(x)
  #x <- normalize.loess(x,subset=1:nrow(x))
  #mva.pairs(x)
}

normalize.qspline Normalize arrays

Description

normalizes arrays in an AffyBatch each other or to a set of target intensities
normalize.AffyBatch.qspline

Usage

normalize.AffyBatch.qspline(abatch, type=c("together", "pmonly", "mmonly", "separate"), ...)

normalize.qspline(x, target = NULL, samples = NULL, fit.iters = 5, min.offset = 5, spline.method = "natural", smooth = TRUE, spar = 0, p.min = 0, p.max = 1.0, incl.ends = TRUE, converge = FALSE, verbose = TRUE, na.rm = FALSE)

Arguments

x a data.matrix of intensities
abatch an AffyBatch
target numerical vector of intensity values to normalize to. (could be the name for one of the celfiles in 'abatch').
samples numerical, the number of quantiles to be used for spline. if (0,1], then it is a sampling rate.
fit.iters number of spline interpolations to average.
min.offset minimum span between quantiles (rank difference) for the different fit iterations.
spline.method specifies the type of spline to be used. Possible values are "fmm", "natural", and "periodic".
smooth logical, if ‘TRUE’, smoothing splines are used on the quantiles.
spar smoothing parameter for ‘splinefun’, typically in (0,1].
p.min minimum percentile for the first quantile.
p.max maximum percentile for the last quantile.
incl.ends include the minimum and maximum values from the normalized and target arrays in the fit.
converge (currently unimplemented)
verbose logical, if ‘TRUE’ then normalization progress is reported.
na.rm logical, if ‘TRUE’ then handle NA values (by ignoring them).
type a string specifying how the normalization should be applied. See details for more.
... optional parameters to be passed through.

Details

This normalization method uses the quantiles from each array and the target to fit a system of cubic splines to normalize the data. The target should be the mean (geometric) or median of each probe but could also be the name of a particular chip in the abatch object.

Parameters setting can be of much importance when using this method. The parameter fit.iter is used as a starting point to find a more appropriate value. Unfortunately the algorithm used do not converge in some cases. If this happens, the fit.iter value is used and a warning is thrown. Use of different settings for the parameter samples was reported to give good results. More specifically, for about 200 data points use samples = 0.33, for about 2000 data points use samples = 0.05, for about 10000 data points use samples = 0.02 (thanks to Paul Boutros).

The type argument should be one of "separate", "pmonly", "mmonly", "together" which indicates whether to normalize only one probe type (PM,MM) or both together or separately.
**normalize.quantiles**

**Value**

a normalized AffyBatch.

**Author(s)**

Laurent and Workman C.

**References**


---

**normalize.quantiles**  
Quantile Normalization

**Description**

Using a normalization based upon quantiles, this function normalizes a matrix of probe level intensities.

**Usage**

normalize.AffyBatch.quantiles(abatch, type=c("separate","pmonly","mmonly","together"))

**Arguments**

- abatch: an AffyBatch object.
- type: A string specifying how the normalization should be applied. See details for more.

**Details**

This method is based upon the concept of a quantile-quantile plot extended to n dimensions. No special allowances are made for outliers. If you make use of quantile normalization either through rma or expresso please cite Bolstad et al, Bioinformatics (2003).

The type argument should be one of "separate","pmonly","mmonly","together" which indicates whether to normalize only one probe type (PM,MM) or both together or separately.

**Value**

A normalized AffyBatch.

**Author(s)**

Ben Bolstad, <bmbolstad.com>
References


See Also

normalize

normalize.quantiles.robust

Robust Quantile Normalization

Description

Using a normalization based upon quantiles, this function normalizes a matrix of probe level intensities. Allows weighting of chips

Usage

normalize.AffyBatch.quantiles.robust(abatch, 
  type = c("separate","pmonly","mmonly","together"), 
  weights = NULL, 
  remove.extreme = c("variance","mean","both","none"), 
  n.remove = 1, use.median = FALSE, 
  use.log2 = FALSE)

Arguments

- **abatch**: an `AffyBatch` object.
- **type**: a string specifying how the normalization should be applied. See details for more.
- **weights**: a vector of weights, one for each chip.
- **remove.extreme**: if weights is NULL, then this will be used for determining which chips to remove from the calculation of the normalization distribution. See details for more info.
- **n.remove**: number of chips to remove.
- **use.median**: if TRUE, the use the median to compute normalization chip; otherwise uses a weighted mean.
- **use.log2**: work on log2 scale. This means we will be using the geometric mean rather than ordinary mean.
This method is based upon the concept of a quantile-quantile plot extended to n dimensions. Note that the matrix is of intensities not log intensities. The function performs better with raw intensities. Choosing variance will remove chips with variances much higher or lower than the other chips, mean removes chips with the mean most different from all the other means, both removes first extreme variance and then an extreme mean. The option none does not remove any chips, but will assign equal weights to all chips.

The type argument should be one of "separate","pmonly","mmonly","together" which indicates whether to normalize only one probe type (PM,MM) or both together or separately.

Value
a matrix of normalized intensities

Note
This function is still experimental.

Author(s)
Ben Bolstad, <bmb@bmbolstad.com>

See Also
normalize, normalize.quantiles

Description
Plot intensities using the function 'pairs'

Usage
```r
## S3 method for class 'AffyBatch'
pairs(x, panel=points, ..., transfo=I, main=NULL, oma=NULL,
font.main = par("font.main"),
cex.main = par("cex.main"), cex.labels = NULL,
lower.panel=panel, upper.panel=NULL, diag.panel=NULL,
font.labels = 1, rowlattop = TRUE, gap = 1)
```

Arguments
- **x**: an AffyBatch object.
- **panel**: a function to produce a plot (see pairs).
- **...**: extra parameters for the 'panel' function.
- **transfo**: a function to transform the intensity values before generating the plot. 'log' and 'log2' are popular choices.
- **main**: title for the plot.
oma see 'oma' in par.
font.main see pairs.
cex.main see pairs.
cex.labels see pairs.
lower.panel a function to produce the plots in the lower triangle (see pairs).
upper.panel a function to produce the plots in the upper triangle (see pairs).
diag.panel a function to produce the plots in the diagonal (see pairs).
font.labels see pairs.
row1attp row1attp see pairs.
gap see pairs.

Details

Plots with several chips can represent zillions of points. They require a lot of memory and can be very slow to be displayed. You may want to try to split of the plots, or to plot them in a device like 'png' or 'jpeg'.

plot.ProbeSet  plot a probe set

Description

Plot intensities by probe set.

Usage

## S3 method for class 'ProbeSet'
plot(x, which=c("pm", "mm"), xlab = "probes", type = "l", ylim = NULL, ...)

Arguments

x a ProbeSet object.
which get the PM or the MM.
xlab x-axis label.
type plot type.
ylim range of the y-axis.
... optional arguments to be passed to matplot.

Value

This function is only used for its (graphical) side-effect.

See Also

ProbeSet

Examples

data(SpikeIn)
plot(SpikeIn)
plotDensity

Description

Plots the non-parametric density estimates using values contained in the columns of a matrix.

Usage

```r
plotDensity(mat, ylab = "density", xlab="x", type="l", col=1:6,
    na.rm = TRUE, ...)

plotDensity.AffyBatch(x, col = 1:6, log = TRUE,
    which=c("pm","mm","both"),
    ylab = "density",
    xlab = NULL, ...)
```

Arguments

- `mat`: a matrix containing the values to make densities in the columns.
- `x`: an object of class `AffyBatch`.
- `log`: logical value. If TRUE the log of the intensities in the `AffyBatch` are plotted.
- `which`: should a histogram of the PMs, MMs, or both be made?
- `col`: the colors to use for the different arrays.
- `ylab`: a title for the y axis.
- `xlab`: a title for the x axis.
- `type`: type for the plot.
- `na.rm`: handling of NA values.
- `...`: graphical parameters can be given as arguments to `plot`.

Details

The list returned can be convenient for plotting large input matrices with different colors/line types schemes (the computation of the densities can take some time).

To match other functions in base R, this function should probably be called `matdensity`, as it is sharing similarities with `matplot` and `matlines`.

Value

It returns invisibly a list of two matrices ‘x’ and ‘y’.

Author(s)

Ben Bolstad and Laurent Gautier
Examples

if (require(affydata)) {
  data(Dilution)
  plotDensity(exprs(Dilution), log="x")
}

---

plotLocation

Plot a location on a cel image

Description

Plots a location on a previously plotted cel image. This can be used to locate the physical location
of probes on the array.

Usage

plotLocation(x, col="green", pch=22, ...)

Arguments

x  a ‘location’. It can be obtained by the method of AffyBatch indexProbes, or
    made elsewhere (basically a location is n rows and two columns array. The first
    column corresponds to the x positions and the second columns corresponds to
    the y positions of n elements to locate).

col  colors for the plot.

pch  plotting type (see function plot).

...  other parameters passed to the function points.

Author(s)

Laurent

See Also

AffyBatch

Examples

if (require(affydata)) {
  data(Dilution)

  ## image of the celfile
  image(Dilution[, 1])

  ## genenames, arbitrarily pick the 101th
  n <- geneNames(Dilution)[101]

  ## get the location for the gene n
  l <- indexProbes(Dilution, "both", n)[[1]]
  ## convert the index to X/Y coordinates
  xy <- indices2xy(l, abatch=Dilution)
pmcorrect

## plot
plotLocation(xy)
}

pmcorrect

### PM Correction

**Description**

Corrects the PM intensities in a **ProbeSet** for non-specific binding.

**Usage**

pmcorrect.pmonly(object)

pmcorrect.subtractmm(object)

pmcorrect.mas(object, contrast.tau=0.03, scale.tau=10, delta=2^(-20))

**Arguments**

- **object**
  An object of class **ProbeSet**.
- **contrast.tau**
  A number denoting the contrast tau parameter in the MAS 5.0 pm correction algorithm.
- **scale.tau**
  A number denoting the scale tau parameter in the MAS 5.0 pm correction algorithm.
- **delta**
  A number denoting the delta parameter in the MAS 5.0 pm correction algorithm.

**Details**

These are the pm correction methods performed by Affymetrix MAS 4.0 (subtractmm) and MAS 5.0 (mas). See the Affymetrix Manual for details. pmonly does what you think: does not change the PM values.

**Value**

A **ProbeSet** for which the pm slot contains the corrected PM values.

**References**

Affymetrix MAS 4.0 and 5.0 manual

**Examples**

```r
if (require(affydata)) {
  data(Dilution)
  gn <- geneNames(Dilution)
  pps <- probeset(Dilution, gn[1][[1]]

  pps.pmonly <- pmcorrect.pmonly(pps)
  pps.subtractmm <- pmcorrect.subtractmm(pps)
  pps.mas5 <- pmcorrect.mas(pps)
}
```
Apply a function over the ProbeSets in an AffyBatch

**Description**

Apply a function over the ProbeSets in an AffyBatch

**Usage**

```r
ppsetApply(abatch, FUN, genenames = NULL, ...)
```

```r
ppset.ttest(ppset, covariate, pmcorrect.fun = pmcorrect.pmonly, ...)
```

**Arguments**

- **abatch**: an object inheriting from AffyBatch.
- **ppset**: an object of class ProbeSet.
- **covariate**: the name a covariate in the slot phenoData.
- **pmcorrect.fun**: a function to correct PM intensities.
- **FUN**: a function working on a ProbeSet.
- **genenames**: a list of Affymetrix probesets ids to work with. All probe set ids used when NULL.
- **...**: optional parameters to the function FUN.

**Value**

Returns a list of objects, or values, as returned by the function FUN for each ProbeSet it processes.

**Author(s)**

Laurent Gautier <laurent@cbs.dtu.dk>

**See Also**

ProbeSet-class

**Examples**

```r
ppset.ttest <- function(ppset, covariate, pmcorrect.fun = pmcorrect.pmonly, ...) {
  probes <- do.call("pmcorrect.fun", list(ppset))
  my.ttest <- function(x) {
    y <- split(x, get(covariate))
    t.test(y[[1]], y[[2]])$p.value
  }
  r <- apply(probes, 1, my.ttest)
  return(r)
}
```

#this takes a long time - and rowttests is a good alternative
## eg: rt = rowttests(exprs(Dilution), Dilution$liver)
## Not run:
  data(Dilution)
all.test <- ppsetApply(Dilution, ppset.ttest, covariate="liver")
## End(Not run)

probeMatch-methods

Methods for accessing perfect matches and mismatches

Description
Methods for perfect matches and mismatches probes

Methods
object = AffyBatch All the perfect match (pm) or mismatch (mm) probes on the arrays the object represents are returned.
object = ProbeSet The pm or mm of the object are returned.

probeNames-methods

Methods for accessing the Probe Names

Description
Methods for accessing Probe Names

Methods
object = Cdf an accessor function for the name slot.
object = probeNames returns the probe names associated with the rownames of the intensity matrices one gets with the pm and mm methods.

ProbeSet-class

Class ProbeSet

Description
A simple class that contains the PM and MM data for a probe set from one or more samples.

Objects from the Class
Objects can be created by applying the method probeset to instances of AffyBatch.

Slots
id: Object of class "character" containing the probeset ID.
pm: Object of class "matrix" containing the PM intensities. Columns represent samples and rows the different probes.
mm: Object of class "matrix" containing the MM intensities.
Methods

**colnames** signature(x = "ProbeSet"): the column names of the pm matrices which are the sample names

**express.summary.stat** signature(x = "ProbeSet", pmcorrect = "character", summary = "character"): applies a summary statistic to the probe set.

**sampleNames** signature(object = "ProbeSet"): the column names of the pm matrices which are the sample names.

Note

More details are contained in the vignette.

See Also

`probeset`, `AffyBatch-class`

Examples

```r
if (require(affydata)) {
  data(Dilution)
  ps <- probeset(Dilution, geneNames(Dilution)[1:2])
  names(ps)
  print(ps[[1]])
}
```

Description

A class to handle progress bars in text mode.

Objects from the Class

Objects can be created by calls of the form `new("ProgressBarText", steps)`.

Slots

- **steps**: Object of class "integer". The total number of steps the progress bar should represent.
- **barsteps**: Object of class "integer". The size of the progress bar.
- **internals**: Object of class "environment". For internal use.

Methods

**close** signature(con = "ProgressBarText"): Terminate the progress bar (i.e. print what needs to be printed). Note that closing the instance will ensure the progress bar is plotted to its end.

**initialize** signature(.Object = "ProgressBarText"): initialize a instance.

**open** signature(con = "ProgressBarText"): Open a progress bar (i.e. print things). In the case open is called on a progress bar that was 'progress', the progress bar is resumed (this might be useful when one wishes to insert text output while there is a progress bar running).

**updateMe** signature(object = "ProgressBarText"): Update the progress bar (see examples).
Author(s)
Laurent

Examples

```r
f <- function(x, header = TRUE) {
  pbt <- new("ProgressBarText", length(x), barsteps = as.integer(20))

  open(pbt, header = header)

  for (i in x) {
    Sys.sleep(i)
    updateMe(pbt)
  }
  close(pbt)
}

## if too fast on your machine, change the number
x <- runif(15)

f(x)
f(x, header = FALSE)

## 'cost' of the progress bar:
g <- function(x) {
  z <- 1
  for (i in 1:x) {
    z <- z + 1
  }
}
h <- function(x) {
  pbt <- new("ProgressBarText", as.integer(x), barsteps = as.integer(20))
  open(pbt)
  for (i in 1:x) {
    updateMe(pbt)
  }
  close(pbt)
}

system.time(g(10000))

system.time(h(10000))
```

---

**read.affybatch**

**Read CEL files into an AffyBatch**

**Description**

Read CEL files into an Affybatch.

**Usage**

```r
read.affybatch(..., filenames = character(0),
                phenoData = new("AnnotatedDataFrame"),
```
description = NULL,
notes = "",
compress = getOption("BioC")$affy$compress.cel,
rm.mask = FALSE, rm.outliers = FALSE, rm.extra = FALSE,
verbose = FALSE, sd=FALSE, cdfname = NULL)

ReadAffy(..., filenames=character(0),
widget=getOption("BioC")$affy$use.widgets,
compress=getOption("BioC")$affy$compress.cel,
celfile.path=NULL,
sampleNames=NULL,
phenoData=NULL,
description=NULL,
notes="",
rm.mask=FALSE, rm.outliers=FALSE, rm.extra=FALSE,
verbose=FALSE, sd=FALSE, cdfname = NULL)

Arguments

... file names separated by comma.
filenames file names in a character vector.
phenoData an AnnotatedDataFrame object, a character of length one, or a data.frame.
description a MIAME object.
notes notes.
compress are the CEL files compressed?
rm.mask should the spots marked as 'MASKS' set to NA?
rm.outliers should the spots marked as 'OUTLIERS' set to NA?
rm.extra if TRUE, then overrides what is in rm.mask and rm.outliers.
verbose verbosity flag.
widget a logical specifying if widgets should be used.
celfile.path a character denoting the path ReadAffy should look for cel files.
sampleNames a character vector of sample names to be used in the AffyBatch.
sd should the standard deviation values in the CEL file be read in? Since these are typically not used default is not to read them in. This also save lots of memory.
cdfname used to specify the name of an alternative cdf package. If set to NULL, then the usual cdf package based on Affymetrix’s mappings will be used.

Details

ReadAffy is a wrapper for read.affybatch that permits the user to read in phenoData, MIAME information, and CEL files using widgets. One can also define files where to read phenoData and MIAME information.

If the function is called with no arguments ReadAffy() all the CEL files in the working directory are read and put into an AffyBatch. However, the arguments give the user great flexibility.

If phenoData is a character vector of length 1, the function read.AnnotatedDataFrame is called to read a file of that name and produce the AnnotationDataFrame object with the sample metadata. If phenoData is a data.frame, it is converted to an AnnotatedDataFrame. If it is NULL and widget=FALSE (widget=TRUE is not currently supported), then a default object of class AnnotatedDataFrame
is created, whose pData is a data.frame with rownames being the names of the CEL files, and with one column sample with an integer index.

AllButCelsForReadAffy is an internal function that gets called by ReadAffy. It gets all the information except the cel intensities.

description is read using read.MIAME. If a character is given, then it tries to read the file with that name to obtain a MIAME instance. If left NULL but widget=TRUE, then widgets are used. If left NULL and widget=FALSE, then an empty instance of MIAME is created.

**Value**

An AffyBatch object.

**Author(s)**

Ben Bolstad <bmb@bmbolstad.com> (read.affybatch), Laurent Gautier, and Rafael A. Irizarry (ReadAffy)

**See Also**

AffyBatch

**Examples**

```r
if(require(affydata)){
  celpath <- system.file("celfiles", package="affydata")
  fns <- list.celfiles(path=celpath,full.names=TRUE)

  cat("Reading files:
  paste(fns,collapse="\n"),"\n")
  ##read a binary celfile
  abatch <- ReadAffy(filenames=fns[1])
  ##read a text celfile
  abatch <- ReadAffy(filenames=fns[2])
  ##read all files in that dir
  abatch <- ReadAffy(celfile.path=celpath)
}
```

---

**read.probematrix**  
Read CEL file data into PM or MM matrices

**Description**

Read CEL data into matrices.

**Usage**

```r
read.probematrix(..., filenames = character(0),
  phenoData = new("AnnotatedDataFrame"),
  description = NULL,
  notes = "",
  compress = getOption("BioC")$affy$compress.cel,
  rm.mask = FALSE, rm.outliers = FALSE, rm.extra = FALSE,
  verbose = FALSE, which = "pm", cdfname = NULL)
```
Arguments

... file names separated by comma.
filenames file names in a character vector.
phenoData a AnnotatedDataFrame object.
description a MIAME object.
notes notes.
compress are the CEL files compressed?
rm.mask should the spots marked as 'MASKS' set to NA?
rm.outliers should the spots marked as 'OUTLIERS' set to NA?
rm.extra if TRUE, overrides what is in rm.mask and rm.outliers.
verbose verbosity flag.
which should be either "pm", "mm" or "both".
cdfname Used to specify the name of an alternative cdf package. If set to NULL, the usual cdf package based on Affymetrix’s mappings will be used.

Value

A list of one or two matrices. Each matrix is either PM or MM data. No AffyBatch is created.

Author(s)

Ben Bolstad <bmb@bmbolstad.com>

See Also

AffyBatch, read.affybatch

rma

Robust Multi-Array Average expression measure

Description

This function converts an AffyBatch object into an ExpressionSet object using the robust multi-array average (RMA) expression measure.

Usage

rma(object, subset=NULL, verbose=TRUE, destructive=TRUE, normalize=TRUE, background=TRUE, bgversion=2, ...)
Arguments

object an AffyBatch object.
subset a character vector with the the names of the probesets to be used in expression calculation.
verbose logical value. If TRUE, it writes out some messages indicating progress. If FALSE nothing should be printed.
destructive logical value. If TRUE, works on the PM matrix in place as much as possible, good for large datasets.
normalize logical value. If TRUE, normalize data using quantile normalization.
background logical value. If TRUE, background correct using RMA background correction.
bgversion integer value indicating which RMA background to use: 1: use background similar to pure R rma background given in affy version 1.0-1.0.2, 2: use background similar to pure R rma background given in affy version 1.1 and above
... further arguments to be passed (not currently implemented - stub for future use).

Details

This function computes the RMA (Robust Multichip Average) expression measure described in Irizarry et al Biostatistics (2003).

Note that this expression measure is given to you in log base 2 scale. This differs from most of the other expression measure methods.

Please note that the default background adjustment method was changed during the lead up to the Bioconductor 1.2 release. This means that this function and expresso should give results that directly agree.

Value

An ExpressionSet

Author(s)

Ben Bolstad <bmb@bmbolstad.com>

References


See Also

expresso
Examples

```r
if (require(affydata)) {
  data(Dilution)
  eset <- rma(Dilution)
}
```

---

**SpikeIn**

*SpikeIn Experiment Data: ProbeSet Example*

**Description**

This **ProbeSet** represents part of SpikeIn experiment data set.

**Usage**

```r
data(SpikeIn)
```

**Format**

*SpikeIn* is **ProbeSet** containing the $PM$ and $MM$ intensities for a gene spiked in at different concentrations (given in the vector `colnames(pm(SpikeIn))`) in 12 different arrays.

**Source**

This comes from an experiments where 11 different cRNA fragments have been added to the hybridization mixture of the GeneChip arrays at different pM concentrations. The 11 control cRNAs were BioB-5, BioB-M, BioB-3, BioC-5, BioC-3, BioDn-5 (all *E. coli*), CreX-5, CreX-3 (phage P1), and DapX-5, DapX-M, DapX-3 (*B. subtilis*) The cRNA were chosen to match the target sequence for each of the Affymetrix control probe sets. For example, for DapX (a *B. subtilis* gene), the 5’, middle and 3’ target sequences (identified by DapX-5, DapX-M, DapX-3) were each synthesized separately and spiked-in at a specific concentration. Thus, for example, DapX-3 target sequence may be added to the total hybridization solution of 200 micro-liters to give a final concentration of 0.5 pM.

For this example we have the $PM$ and $MM$ for BioB-5 obtained from the arrays where it was spiked in at 0.0, 0.5, 0.75, 1, 1.5, 2, 3, 5, 12.5, 25, 50, and 150 pM.


---

**summary**

*Probe Set Summarizing Functions*

**Description**

These were used with the function `express`, which is no longer part of the package. Some are still used by the `generateExprVal` functions, but you should avoid using them directly.

**See Also**

`expresso`
**Tukey biweight**

*One-step Tukey's biweight*

**Description**

One-step Tukey's biweight on a matrix.

**Usage**

```r
tukey.biweight(x, c = 5, epsilon = 1e-04)
```

**Arguments**

- `x`: a matrix.
- `c`: tuning constant (see details).
- `epsilon`: fuzzy value to avoid division by zero (see details).

**Details**

The details can be found in the given reference.

**Value**

a vector of values (one value per column in the input matrix).

**References**


**See Also**

`pmcorrect.mas` and `generateExprVal.method.mas`

---

**Whatcdf**

*Find which CDF corresponds*

**Description**

Find which kind of CDF corresponds to a CEL file.

**Usage**

```r
whatcdf(filename, compress = getOption("BioC")$affy$compress.cel)
```

**Arguments**

- `filename`: a `.CEL` file name.
- `compress`: logical (file compressed or not).
xy2indices

Details

Information concerning the corresponding CDF file seems to be found in CEL files. This allows us to try to link CDF information automatically.

Value

A character with the name of the CDF.

See Also

getInfoInAffyFile, read.celfile

xy2indices (Functions to convert indices to x/y (and reverse))

Description

Functions to convert indices to x/y (and reverse)

Usage

xy2indices(x, y, nc = NULL, cel = NULL, abatch = NULL, cdf = NULL, xy.offset = NULL)
indices2xy(i, nc = NULL, cel = NULL, abatch = NULL, cdf = NULL, xy.offset = NULL)

Arguments

- **x**: A numeric vector of X (column) position(s) for the probes.
- **y**: A numeric vector of Y (row) position(s) for the probes.
- **i**: A numeric vector of indices in the AffyBatch for the probes.
- **nc**: total number of columns on the chip. It is usually better to specify either the cdf or abatch arguments rather than the number of columns.
- **cel**: a corresponding object of class Cel. This has been deprecated. Use abatch or cdf instead.
- **abatch**: a corresponding object of class AffyBatch.
- **cdf**: character - the name of the corresponding cdf package.
- **xy.offset**: an eventual offset for the XY coordinates. See Details.

Details

The Affymetrix scanner reads data from a GeneChip by row, and exports those data to a CEL file. When we read in the CEL file data to an AffyBatch object, we store data for each GeneChip as a single column in a matrix of probe-wise intensity values.

The CDF files that Affymetrix make available for various GeneChips map individual probes to probesets based on their (x,y) coordinates on the GeneChip. Note that these coordinates are zero-based, and (x,y) is the same as (column, row). In other words, the x coordinate indicates the horizontal location of the probe, and the y coordinate indicates the vertical location of the probe. By convention, (0,0) is the coordinate location for the top left position, and (ncol-1, nrow-1) is the coordinate location of the lower right position.
For most users, the mapping of probes to probeset is handled internally by various functions (rma, espresso, etc), and in general usage it is never necessary for a user to convert probe index position in an `AffyBatch` to the corresponding (x,y) coordinates on the GeneChip. These functions are only useful for those who wish to know more about the internal workings of the Affymetrix GeneChip.

The parameter `xy.offset` is there for compatibility. For historical reasons, the xy-coordinates for the features on Affymetrix GeneChips were decided to start at 1 (one) rather than 0 (zero). One can set the offset to 1 or to 0. Unless the you \_really\_ know what you are doing, it is advisable to let it at the default value `NULL`. This way the package-wide option `xy.offset` is always used.

**Value**

A vector of indices or a two-columns matrix of Xs and Ys.

**Warning**

Even if one really knows what is going on, playing with the parameter `xy.offset` could be risky. Changing the package-wide option `xy.offset` appears much more sane.

**Author(s)**

L.

**See Also**

`indexProbes`

**Examples**

```r
if (require(affydata)) {
  data(Dilution)
  pm.i <- indexProbes(Dilution, which="pm", genenames="AFFX-BioC-5_at")[[1]]
  mm.i <- indexProbes(Dilution, which="mm", genenames="AFFX-BioC-5_at")[[1]]

  pm.i.xy <- indices2xy(pm.i, abatch = Dilution)
  mm.i.xy <- indices2xy(mm.i, abatch = Dilution)

  ## and back to indices
  i.pm <- xy2indices(pm.i.xy[,1], pm.i.xy[,2], cdf = "hgu95av2cdf")
  i.mm <- xy2indices(mm.i.xy[,1], mm.i.xy[,2], cdf = "hgu95av2cdf")

  identical(pm.i, as.integer(i.pm))
  identical(mm.i, as.integer(i.mm))

  image(Dilution[1], transf=log2)
  ## plot the pm in red
  plotLocation(pm.i.xy, col="red")
  plotLocation(mm.i.xy, col="blue")
}
```
Index

*Topic **aplot**
  plotLocation, 44
*Topic **character**
  cleancdfname, 14
  list.celfiles, 26
*Topic **classes**
  AffyBatch-class, 6
  ProbeSet-class, 47
  ProgressBarText-class, 48
*Topic **datasets**
  cdfenv.example, 12
  SpikeIn, 54
*Topic **hplot**
  AffyRNAdeg, 8
  barplot.ProbeSet, 10
  MAplot, 27
  mva.pairs, 32
  pairs.AffyBatch, 41
  plot.ProbeSet, 42
  plotDensity, 43
*Topic **interface**
  expressoWidget, 16
*Topic **manip**
  .setAffyOptions, 3
  affy-options, 4
  affy.scalevalue.exprSet, 5
  AffyRNAdeg, 8
  bg.adjust, 11
  bg.correct, 11
  expresso, 15
  fit.li.wong, 18
  generateExprSet-method, 20
  generateExprVal, 21
  generateExprVal.method.avgdiff, 22
  generateExprVal.method.playerout, 23
  justRNA, 24
  masS, 28
  masScalls, 29
  merge.AffyBatch, 31
  normalize-methods, 33
  normalize.constant, 34
  normalize.contrasts, 35
  normalize.invariantset, 35
  normalize.qspline, 37
  normalize.quantiles, 39
  normalize.quantiles.robust, 40
  pmcorrect, 45
  ppssetApply, 46
  read.affybatch, 49
  read.probematrix, 51
  rma, 52
  summary, 54
  tukey.biweight, 55
  whatcdf, 55
  xy2indices, 56
*Topic **math**
  hlog, 24
*Topic **methods**
  debug.affy123, 14
  probeMatch-methods, 47
  probeNames-methods, 47
*Topic **models**
  fit.li.wong, 18
*Topic **smooth**
  normalize.loess, 36
*Topic **utilities**
  cdfFromBioC, 13
  .setAffyOptions, 3
  [,,AffyBatch-method (AffyBatch-class), 6
  [<=-,AffyBatch-method (AffyBatch-class), 6
  [[,AffyBatch-method (AffyBatch-class), 6
  $:.AffyBatch (AffyBatch-class), 6
  affy-deprecated, 4
  affy-options, 4
  affy.scalevalue.exprSet, 5, 16, 28
  AffyBatch, 11, 12, 15, 16, 20, 21, 28, 31,
  33–37, 39–41, 43, 44, 51–53, 56
  AffyBatch (AffyBatch-class), 6
  AffyBatch, ANY (AffyBatch-class), 6
  AffyBatch-class, 6
  AffyRNAdeg, 8
  AllButCelsForReadAffy (read.affybatch), 49
  AnnotatedDataFrame, 25, 26, 50, 52
avdiff (summary), 54
barplot, 10
barplot, ProbeSet-method (ProbeSet-class), 47
barplot, ProbeSet, 10
g. adjust, 11, 12
g. correct, 11
g. correct, AffyBatch, character-method (AffyBatch-class), 6
g. correct.rma, 11
bg.adjust, 11, 12
bg.correct, 11
bg.correct, AffyBatch, character-method (AffyBatch-class), 6
bg.correct.AffyBatch, character-method
bg.correct, AffyBatch, ANY-method (AffyBatch-class), 6
bg.mc(), 11
boxplot, 7
boxplot, AffyBatch-method (AffyBatch-class), 6
cdfenv.example, 12
cdffromBioC, 13
cdffromEnvironment (dfffromBioC), 13
cdffromLibPath (dfffromBioC), 13
cdfName, (AffyBatch-class), 6
cdfName, AffyBatch-method (AffyBatch-class), 6
ccheckValidFilenames (AffyBatch-class), 6
cleanCdfName, 14
close, ProgressBarText-method (ProgressBarText-class), 48
col, AffyBatch-method (AffyBatch-class), 6
colnames, ProbeSet-method (ProbeSet-class), 47
computeExprSet, 16
computeExprSet (generateExprSet-method), 20
computeExprSet, AffyBatch, character-method (AffyBatch-class), 6
counter, 57
clearCdfName, 14
close, ProgressBarText-method (ProgressBarText-class), 48
col, AffyBatch-method (AffyBatch-class), 6
colnames, ProbeSet-method (ProbeSet-class), 47
computeExprSet, 16
computeExprSet (generateExprSet-method), 20
computeExprSet, AffyBatch, character-method (AffyBatch-class), 6
counter, 57
clearCdfName, 14
close, ProgressBarText-method (ProgressBarText-class), 48
col, AffyBatch-method (AffyBatch-class), 6
colnames, ProbeSet-method (ProbeSet-class), 47
computeExprSet, 16
computeExprSet (generateExprSet-method), 20
computeExprSet, AffyBatch, character-method (AffyBatch-class), 6
counter, 57
clearCdfName, 14
close, ProgressBarText-method (ProgressBarText-class), 48
col, AffyBatch-method (AffyBatch-class), 6
colnames, ProbeSet-method (ProbeSet-class), 47
computeExprSet, 16
computeExprSet (generateExprSet-method), 20
computeExprSet, AffyBatch, character-method (AffyBatch-class), 6
concentrations (SpikeIn), 54
debug.affy123, 14
dim, AffyBatch-method (AffyBatch-class), 6
eSet, 6, 8
eXpress.summary.stat (generateExprVal), 21
eXpress.summary.stat, ProbeSet, character-method (ProbeSet-class), 47
eXpress.summary.stat-methods (generateExprVal), 21
express.summary.stat.methods
indexProbes(AffyBatch-class), 6
indexProbes,AffyBatch,character-method
(AffyBatch-class), 6
indexProbes,AffyBatch,missing-method
(AffyBatch-class), 6
indexProbes,AffyBatch-method
(AffyBatch-class), 6
indices2xy (xy2indices), 56
initialize,AffyBatch-method
(AffyBatch-class), 6
initialize,ProgressBarText-method
(ProgressBarText-class), 48
intensity(AffyBatch-class), 6
intensity,AffyBatch-method
(AffyBatch-class), 6
intensity<-(AffyBatch-class), 6
intensity<-,AffyBatch-method
(AffyBatch-class), 6
just.rma (justRMA), 24
justRMA, 24
length,AffyBatch-method
(AffyBatch-class), 6
li.wong, 20
li.wong (fit.li.wong), 18
list.celfiles, 26
list.files, 27
loess, 27, 32, 35, 37
loess.normalize (affy-deprecated), 4
ma.plot (Maplot), 27
maffy.normalize, 35
maffy.normalize (affy-deprecated), 4
mapCdfName (cleancdfname), 14
Maplot, 27
MAplot,AffyBatch-method (Maplot), 27
mas5, 28
mas5.detection (mas5calls), 29
mas5calls, 29
mas5calls,AffyBatch-method (mas5calls), 29
mas5calls,ProbeSet-method (mas5calls), 29
mas5calls.AffyBatch (mas5calls), 29
mas5calls.ProbeSet (mas5calls), 29
Mbox (Maplot), 27
Mbox,AffyBatch-method (Maplot), 27
medianpolish (summary), 54
merge.AffyBatch, 8, 31
MIAME, 25, 50, 52
mm (probeMatch-methods), 47
mm,AffyBatch-method (AffyBatch-class), 6
mm,ProbeSet-method (ProbeSet-class), 47
mm<- (probeMatch-methods), 47
mm<-,AffyBatch,ANY-method
(AffyBatch-class), 6
mm<-,ProbeSet,matrix-method
(ProbeSet-class), 47
mmindex (AffyBatch-class), 6
mmindex,AffyBatch-method
(AffyBatch-class), 6
multiloess (affy-deprecated), 4
mva.pairs, 28, 32
normalize, 7, 33, 36, 37, 40, 41
normalize,AffyBatch-method
(normalize-methods), 33
normalize-methods, 33
normalize.AffyBatch
(normalize-methods), 33
normalize.AffyBatch.constant
(normalize.constant), 34
normalize.AffyBatch.contrasts
(normalize.contrasts), 35
normalize.AffyBatch.invariantset
(normalize.invariantset), 35
normalize.AffyBatch.loess
(normalize.loess), 36
normalize.AffyBatch.qspline
(normalize.qspline), 37
normalize.AffyBatch.quantiles
(normalize.quantiles), 39
normalize.AffyBatch.quantiles.robust
(normalize.quantiles.robust), 40
normalize.constant, 34
normalize.contrasts, 35
normalize.invariantset, 35
normalize.loess, 4, 36
normalize.methods (normalize-methods), 33
normalize.methods,AffyBatch-method
(normalize-methods), 33
normalize.qspline, 37
normalize.quantiles, 39, 41
normalize.quantiles.robust, 40
open,ProgressBarText-method
(ProgressBarText-class), 48
optim, 23
pairs, 32, 41, 42
pairs.AffyBatch, 8, 41
par, 42
playerout.costfunction

(plot, 9, 43)

plot.ProbeSet, 42

plotAffyRNAdeg (AffyRNAdeg), 8

plotDensity, 7, 43

plotLocation, 44

pm (probeMatch-methods), 47

pm, AffyBatch-method (AffyBatch-class), 6

pm, ProbeSet-method (ProbeSet-class), 47

pm<- (probeMatch-methods), 47

pm<-, AffyBatch, ANY-method (AffyBatch-class), 6

pm<-, ProbeSet, matrix-method (ProbeSet-class), 47

pmcorrect, 45

pmcorrect.mas, 55

pmcorrect.methods (normalize-methods), 33

pmindex (AffyBatch-class), 6

pmindex, AffyBatch-method (AffyBatch-class), 6

ppset.ttest (ppsetApply), 46

ppsetApply, 46

probeMatch (probeMatch-methods), 47

probeMatch-methods, 47

probeNames (probeNames-methods), 47

probeNames, AffyBatch-method (AffyBatch-class), 6

probeNames-methods, 47

probeNames<- (probeNames-methods), 47

probes (AffyBatch-class), 6

probes, AffyBatch-method (AffyBatch-class), 6

ProbeSet, 7, 22, 42, 45, 54

probeset, 47, 48

probeset (AffyBatch-class), 6

probeset, AffyBatch-method (AffyBatch-class), 6

ProbeSet-class, 47

ProgressBarText-class, 48

qspline-normalize (normalize.qspline), 37

read.affybatch, 6, 26, 49, 52

read.AnnotatedDataFrame, 26, 50

read.MIAME, 26, 51

read.probeMatrix, 51

ReadAffy, 6

ReadAffy (read.affybatch), 49

rma, 26, 39, 52

tukey.biweight, 55

tukeybiweight (summary), 54

updateMe (ProgressBarText-class), 48

updateMe, ProgressBarText-method (ProgressBarText-class), 48

updateObject, AffyBatch-method (AffyBatch-class), 6

whatcdf, 55

xy2indices, 56