Package ‘cellHTS2’

March 13, 2017

Version 2.38.0

Title Analysis of cell-based screens - revised version of cellHTS

Author
Ligia Bras, Wolfgang Huber <whuber@embl.de>, Michael Boutros <m.boutros@dkfz.de>, Gre- goire Pau <gpau@embl.de>, Florian Hahne <florian.hahne@novartis.com>

Maintainer Joseph Barry <joseph.barry@embl.de>

Depends R (>= 2.10), RColorBrewer, Biobase, methods, genefilter, splots, vsn, hwriter, locfit, grid

Suggests ggplot2

Imports prada, GSEABase, Category, stats4

Description This package provides tools for the analysis of high-throughput assays that were performed in microtitre plate formats (including but not limited to 384-well plates). The functionality includes data import and management, normalisation, quality assessment, replicate summarisation and statistical scoring. A webpage that provides a detailed graphical overview over the data and analysis results is produced. In our work, we have applied the package to RNAi screens on fly and human cells, and for screens of yeast libraries. See ?cellHTS2 for a brief introduction.


License Artistic-2.0

URL http://www.dkfz.de/signaling, http://www.ebi.ac.uk/huber

biocViews CellBasedAssays, Preprocessing, Visualization


NeedsCompilation no
R topics documented:

- annotate ................................................. 3
- batch .................................................. 4
- bdgpbiomart ............................................. 5
- Bscore .................................................... 5
- buildCellHTS2 .......................................... 7
- cellHTS-class .......................................... 8
- cellHTS2 ............................................... 11
- configurationAsScreenPlot .............. 14
- configure ............................................. 15
- convertOldCellHTS ......................... 17
- convertWellCoordinates ................. 18
- Data .................................................. 19
- dualCh ................................................. 20
- getDynamicRange ........................... 20
- getEnVisionRawData ................. 22
- getMeasureRepAgreement ............. 23
- getTopTable ........................................... 24
- getZfactor ........................................... 26
- gseaModule ........................................... 28
- imageScreen ......................................... 29
- intensityFiles ....................................... 30
- KcViab .................................................. 31
- KcViabSmall ........................................... 31
- normalizePlates .............................. 32
- oldKcViabSmall ...................................... 35
- oneRowPerId ......................................... 36
- plate .................................................. 36
- plateEffects ........................................... 37
- plotSpatialEffects ....................... 38
- readHTAnalystData ....................... 39
- readPlateList ........................................ 41
- ROC ..................................................... 43
- ROC-class ............................................. 45
- rsa ..................................................... 46
- scoreReplicates ....................................... 47
- scores2calls .......................................... 49
- settings ............................................... 50
- spatialNormalization ..................... 55
- state .................................................. 56
- summarizeChannels ....................... 57
- summarizeReplicates .................... 59
- templateDescriptionFile .................. 60
- updateCellHTS ....................................... 61
- wellAnno .............................................. 62
- write.tabdel .......................................... 62
- writeReport .......................................... 63
- writeTab ............................................... 67

Index ................................................. 68
annotate  

Annotates the reagents (probes) of a cellHTS object

Description
Annotate the reagents (probes) of a cellHTS object. In RNAi-screens, there is a often a 1:1 correspondence between reagents and intended target genes, hence in this software package the term gene ID is used as a synonym.

Usage
## S4 method for signature 'cellHTS'
annotate(object, geneIDFile, path)

Arguments
- object: a cellHTS object.
- geneIDFile: the name of the file with the gene IDs (see details). This argument is just passed on to the read.table function, so any of the valid argument types for read.table are valid here, too. Must contain one row for each well in each plate.
- path: a character of length 1 indicating the path in which to find the gene annotation file (optional).

Details
- geneIDFile: This file is expected to be a tab-delimited file with at least three columns, and column names Plate, Well and GeneID. The contents of Plate are expected to be integer. Further columns are allowed.

Value
An S4 object of class cellHTS, which is obtained by copying object and updating the following slots:
- featureData: the contents of the annotation file are stored here.
- state: the processing status of the cellHTS object is updated to state["annotated"] = TRUE.

Author(s)
Wolfgang Huber, Ligia Bras

References

See Also
readPlateList, configure
**Examples**

```r
datadir <- system.file("KcViabSmall", package = "cellHTS2")
x <- readPlateList("Platelist.txt", path=datadir, name="KcViabSmall")
x <- configure(x, "Description.txt", "Plateconf.txt", "Screenlog.txt", path=datadir)
x <- annotate(x, "GeneIDs_Dm_HFAsubset_1.1.txt", path=datadir)
```

---

**batch**

*Access and replace the batch information of a cellHTS object*

**Description**

This method accesses and replaces the batch data stored in the slot `plateData` of a `cellHTS` object.

**Usage**

```r
batch(object)
batch(object) <- value
```

**Arguments**

- `object`          object of class `cellHTS`.
- `value`           a dataframe of integer values giving the batch number for each plate, and sample.

**Value**

`batch` returns a dataframe containing the contents of slot `plateData$Batch`. This dataframe has dimensions number of plates x number of samples of the `cellHTS` object.

See class `cellHTS` for details.

**Author(s)**

Ligia Bras

**References**


**See Also**

`cellHTS`, `nbatch`
bdgpbiomart

Dataset with annotation of CG identifiers

Description

See the vignette End-to-end analysis of cell-based screens: from raw intensity readings to the annotated hit list, Section Using biomaRt to annotate the target genes online for details. The annotations were obtained on 21 September 2007.

Usage

data(bdgpbiomart)

Format

Dataframe with 21888 rows and 11 columns plate, well, controlStatus, HFAID, GeneID, chromosome_name, start_position, end_position, description, flybasename_gene, go, go_description.

Source

BioMart webinterface to Ensembl.

Examples

data("bdgpbiomart")

Bscore

B score normalization

Description

Correction of plate and spatial effects of data stored in slot assayData of a cellHTS object using the B score method (without variance adjustment of the residuals). Using this method, a two-way median polish is fitted, on a per-plate basis, to account for row and column effects.

Usage

Bscore(object, save.model = FALSE)

Arguments

object a cellHTS object that has already been configured. See details.

save.model a logical value specifying whether the per-plate models should be stored in slots rowcol.effects and overall.effects. See details.
Details

This function is usually not called directly by the user, but from within the `normalizePlates` function. The normalization is performed in a per-plate fashion using the B score method, for each replicate and channel. In the B score method, the residual $r_{ijp}$ of the measurement for row $i$ and column $j$ on the $p$-th plate is obtained by fitting a two-way median polish, in order to account for both row and column effects within the plate:

$$r_{ijp} = y_{ijp} - \hat{y}_{ijp} = y_{ijp} - (\hat{\mu}_p + \hat{R}_{ip} + \hat{C}_{jp})$$

$y_{ijp}$ is the measurement value in row $i$ and column $j$ of plate $p$ (taken from slot `assayData` - only sample wells are considered), and $\hat{y}_{ijp}$ is the corresponding fitted value. This is defined as the sum between the estimated average of the plate ($\hat{\mu}_p$), the estimated systematic offset for row $i$ ($\hat{R}_{ip}$), and the systematic offset for column $j$ ($\hat{C}_{jp}$).

**NOTE:** In the original B score method, as presented by Malo et al., a further step is performed: for each plate $p$, each of the obtained residual values $r_{ijp}$’s are divided by the median absolute deviation of the residuals in plate $p$ (MAD$_p$), resulting in:

$$\frac{r_{ijp}}{MAD_p}$$

The intention of such a further adjustment is to compensate for plate-to-plate variability in dynamic range. In the Bscore function, this step is not automatically performed, but can be done if B score normalization is called using the function `normalizePlates` with arguments `method="Bscore"` and `varianceAdjust="byPlate"`. See the latter function for more details.

If `save.model=TRUE`, the models row and column offsets and overall offsets are stored in the slots `rowcol.effects` and `overall.effects` of object.

Value

An object of class `cellHTS` with B-score normalized data stored in slot `assayData`.

Furthermore, if `save.model=TRUE`, the row and column effects and the overall effects are stored in slots `rowcol.effects` and `overall.effects`, respectively. The latter slots are arrays with the same dimension as `Data(object)`, except the overall.effects slot, which has dimensions `nr Plates x nr Samples x nr Channels`.

After calling this function, the processing status of the `cellHTS` object is updated in the slot `state` to `object@state["normalized"]=TRUE`.

Author(s)

Ligia Bras

References


Build a cellHTS2 object from a data frame containing measurements

**Description**

Builds a cellHTS2 object from a data frame.

**Usage**

```r
buildCellHTS2(xd, measurementNames)
```

**Arguments**

- `xd`: a data frame containing the columns `plate`, `replicate` and `well`, and the measurement columns. The `plate` and `replicate` columns must contain integer values, starting from 1. The `well` column must contain well names formed by one to two capitals letter followed by two digits, e.g. `A12` or `AB01`. Letter ordering is as follows: A, B, C,..., Z, AA, AB, AC,..., AZ, BA, BB,..., ZZ.

- `measurementNames`: an optional character vector containing the measurement names. If missing, the names of the measurement columns in `xd` are used. If NULL, the measurements are not named.

**Details**

The function uses `readPlateList` to build a cellHTS2 object.

**Value**

An object of class `cellHTS`, which extends the class `NChannelSet`. 

---

**Examples**

```r
data("KcViabSmall")
xb <- Bscore(KcViabSmall, save.model = TRUE)
## Calling Bscore function from "normalizePlates" and adding the per-plate variance adjustment step:
xopt <- normalizePlates(KcViabSmall, method="Bscore", varianceAdjust="byPlate", save.model = TRUE)
## Access the slots overall.effects and rowcol.effects
ef1 = plateEffects(xb)
ef2 = plateEffects(xopt)

## double-check
stopifnot(
  all(xb@rowcol.effects==xopt@rowcol.effects, na.rm=TRUE),
  all(xb@overall.effects==xopt@overall.effects, na.rm=TRUE),
  identical(ef1, ef2)
)
```
cellHTS-class

A class for data from cell-based high-throughput assays performed in plate format.

Description

Container for data and experimental meta-data from cell-based high-throughput assays performed in plate format. Typical applications are RNA interference or small molecular compound screens. The class extends the \texttt{NChannelSet} class. Data are from experiments where the same set of reagents (probes) were used. The class can represent data from multi-channel assays.

The data can be thought of as being organised in a two- or three-dimensional array as follows:

- The first dimension corresponds to reagents (e.g. siRNAs, chemical compounds) that were used in the assays. For example, if the screen used 100 plates of 384 wells (24 columns, 16 rows), then the first dimension has size 38,400, and the \texttt{cellHTS} object keeps track of plate ID, row, and column associated with each element. For historic reasons, and because we are using infrastructure that was developed for microarray experiments, the following terms are used synonymously for the elements of the first dimension: reagents, features, probes, genes.

- The second dimension corresponds to assays, including replicates and different experimental conditions (cell type, treatment, genetic background). A potentially confusing terminology is that the data structure that annotates the second dimension is called \texttt{phenoData}. This is because we are using infrastructure (the \texttt{NChannelSet} class) that uses this unfortunate term for this purpose. The software provides methodology for replicate summarization and scoring, however more complicated experimental designs are not directly supported. Multi-purpose tools like \texttt{lmFit} in the \texttt{limma} package should be consulted.

- The (optional) third dimension corresponds to different channels (e.g. different luminescence reporters)

Objects from the Class

Objects can be created by calls of the form \texttt{new("cellHTS", assayData, phenoData, ...)}. See the examples below.

Examples

```r
wells = sprintf("%s%02d", rep(LETTERS[1:8], each=12), 1:12)
xd = expand.grid(plate=1:3, replicate=1:2, well=wells)
xd$cell.number = rnorm(nrow(xd))
xd$cell.size = rnorm(nrow(xd))
x = buildCellHTS2(xd)
```
Slots

plateList: a \texttt{data.frame} containing what was read from input measurement data files plus a column status of type character containing the string "OK" if the data import appeared to have gone well, and the respective error or warning message otherwise.

intensityFiles: a list, where each component contains a copy of the imported input data files. Its length corresponds to the number of rows of \texttt{plateList}.

state: a logical vector of length 4 representing the processing status of the object. It must have the names "configured", "normalized", "scored" and "annotated".

plateConf: a \texttt{data.frame} containing what was read from the configuration file for the experiment (except the first two header rows). It contains at least three columns named Plate, Well and Content. Columns Plate and Well are allowed to contain regular expressions.

descriptLog: a \texttt{data.frame} containing what was read from the screen log file for the experiment, in case it exists. Contains at least three columns, and column names Plate, Well, and Flag. Additional columns are Sample (when there are replicates or more than one sample or condition) and Channel (when there are multiple channels).

screenDesc: a character containing what was read from the description file of the experiment.

rowcol.effects: a 3D array of size Features (i.e. plate size x number of plates) x Samples x Channels containing estimated row and column plate spatial offsets.

overall.effects: a 3D array of size Features x Samples x Channels containing estimated plate overall offsets.

assayData: Object of class \texttt{AssayData}, usually an environment containing matrices of identical size. Each matrix represents a single channel. Columns in each matrix correspond to samples (or replicate), rows to features (probes). Once created, \texttt{cellHTS} manages coordination of samples and channels.

plateData: A list of data frames, with number of rows of each frame equal to the number of plates used in the assay and number of columns equal to the number of samples. Each data frame in the list should contain factorial annotation information relevant for the individual plates, like experimental batches or varying types of micro-titre plates. Currently, this information will be used for between-batch normalization and quality assessment.

phenoData: Object of class \texttt{AnnotatedDataFrame}. Please see the documentation of the phenoData slot of \texttt{NChannelSet} for more details.

It contains information about the screens, and it must have the following columns in its data component: replicate and assay, where replicate is expected to be a vector of integers giving the replicate number, while assay is expected to be a vector of characters giving the name of the biological assay. Both of these vectors should have the same length as the number of Samples.

Once created, \texttt{cellHTS} coordinates selection and subsetting of channels in \texttt{phenoData}.

featureData: Object of class \texttt{AnnotatedDataFrame}, containing information about the reagents: plate, well, column, the well annotation (sample, control, etc.), etc. For a \texttt{cellHTS} object, this slot must contain in its data component at least three mandatory columns named plate, well and controlStatus. Column plate is expected to be a numeric vector giving the plate number (e.g. 1, 2, ...), well should be a vector of characters (alphanumeric characters) giving the well ID within the plate (e.g. A01, B01, H12, etc.). Column controlStatus should be a factor specifying the annotation for each well with possible levels: empty, other, neg, sample, and pos. Other levels besides pos and neg may be employed for controls.

experimentData: Object of class \texttt{MIAME} containing descriptions of the experiment.

annotation: A "character" of length 1, which can be used to specify the name of an annotation package that goes with the reagents used for this experiment.
processingInfo: A list containing information about which normalization and summarization methods have been used.

.__classVersion__: Object of class ` Versions`, containing automatically created information about the class definition, Biobase package version, and other information about the user system at the time the instance was created. See `classVersion` and `updateObject` for examples of use.

**Extends**

Class `NChannelSet`, directly.

**Methods**

Methods with class-specific functionality:

- `name(object)` signature(object="cellHTS"). Obtains the name of the assay stored in the object. This corresponds to the contents of column `assay` of the `phenoData` slot of the `cellHTS` object.

- `name(object) <- value` signature(object = "cellHTS", value = "character") assign the character of length one (value) to the elements in column `assay` of the slot `phenoData` of object.

- `pdim(object)` signature(object = "cellHTS"). Obtain the plate dimension for the data stored in object.

- `nbatch(object)` signature(object = "cellHTS"). Obtain the total number of batches for the data stored in object.

- `compare2cellHTS(x, y)` signature(x = "cellHTS", y = "cellHTS"). Compares two `cellHTS` objects, `x` and `y`, returning TRUE if they are from the same experiment (i.e., if they derive from the same initial `cellHTS` object), or FALSE otherwise.

Methods with functionality derived from class `NChannelSet`: `channel`, `channelNames`, `channelNames<-`, `selectChannels`, `object[features, samples]`, `sampleNames`.

Methods with functionality derived from `eSet`: `annotation`, `assayData`, `assayData<-`, `classVersion<-`, `dim`, `dims`, `experimentData`, `featureData`, `phenoData`, `phenoData<-`, `pubMedIds`, `sampleNames`, `sampleNames<-`, `storageMode`, `varMetadata`, `isCurrent`, `isVersioned`.

Additional methods:

- `initialize` used internally for creating objects
- `show` invoked automatically when the object is displayed to the screen. It prints a summary of the object.
- `state` Access the state slot of a `cellHTS` instance.
- `annotate` Annotate the `cellHTS` object using the `screen annotation file`.
- `configure` Configure the `cellHTS` object using the the `screen description file`, the `screen configuration file` and the `screen log file`.
- `writeTab` Write the contents of `assayData` slot of a `cellHTS` object to a tab-delimited file.
- `ROC` Construct an object of S4 class `ROC`, which represents a receiver-operator-characteristic curve, from the data of the annotated positive and negative controls in a scored `cellHTS` object.

- `meanSdPlot(x)` signature(x = "cellHTS") plots row standard deviations across samples versus row means across samples for data stored in slot `assayData` of a `cellHTS` object. If there are multiple channels, row standard deviations and row means are calculated across samples for each channel separately. Only wells containing "sample" are considered. See `meanSdPlot` for more details about this function.
cellHTS2

Author(s)

Ligia P. Bras <ligia@ebi.ac.uk>, Wolfgang Huber <huber@ebi.ac.uk>

See Also

NChannelSet readPlateList annotate configure writeTab state Data normalizePlates ROC

Examples

```
showClass("cellHTS")
showMethods(class="cellHTS")

## An empty cellHTS
obj <- new("cellHTS")
data("KcViabSmall")
KcViabSmall
state(KcViabSmall)
## Replicate 1 as a cellHTS object
y <- KcViabSmall[,1]
compare2cellHTS(KcViabSmall, y)
data("KcViab")
compare2cellHTS(KcViab, KcViabSmall)
```

---

**cellHTS2 Package Overview**

**Description**

cellHTS2 Package Overview

**Details**

This package provides data structures and algorithms for cell-based high-throughput assays performed in plate format. Typical applications are RNA interference or small molecular compound screens. The most important data class is `cellHTS`, which extends the `NChannelSet` class.

Full help on methods and associated functions is available from within class help pages.

Data sets: `KcViab, KcViabSmall, oldKcViabSmall, dualCh, bdgpbiomart`.

Introductory information is available from vignettes, type `openVignette()`.

Class-specific methods: `annotate, batch, batch<-, compare2cellHTS, configure, Data, Data<-, geneAnno, intensityFiles, name, name<-, nbatch, pdim, plate, plateConf, plateEffects, plateList, position, screenDesc, screenLog, state, well, wellAnno, writeTab, ROC`.

Generic functions: `show, initialize, validObject`.

Other functions: `oneRowPerId, write.tabdel, readPlateList, readHTAnalystData, normalizePlates, Bscore, spatialNormalization, plotSpatialEffects, summarizeChannels, scoreReplicates, summarizeReplicates, imageScreen, configurationAsScreenPlot, getEnVisionRawData, getEnVisionCrosstalkCorrectedData, getTopTable, getMeasureRepAgreement, getDynamicRange, getZfactor, writeReport, convertOldCellHTS, scores2calls, and templateDescriptionFile`.

A full listing of documented topics is available in HTML view by typing `help.start()` and selecting the `cellHTS2` package from the Packages menu or via `library(help="cellHTS2")`. 
Author(s)
Ligia P. Bras

References

See Also
Below, we present a list of ‘high level’ functions in the cellHTS2 package together with a brief description of what they do.

Data import:
- `readPlateList` read a collection of plate reader data files.
- `readHTAnalystData` read input files from a HTanalyser plate reader containing data for a set of plate replicates.
- `getEnVisionRawData` this function can be used as an import function when calling `readPlateList` to read plate result files obtained from EnVision Plate Reader.
- `getEnVisionCrosstalkCorrectedData` this function can be used as an import function when calling `readPlateList` to read plate result files obtained from EnVision Plate Reader.

Screen configuration and annotation:
- `annotate` annotates the reagents (e.g. siRNAs, molecular compounds) of a cellHTS object.
- `configure` annotates the plates and the plate result files of a gcellHTS object.

Accessors:
- `batch` accesses and replaces the batch slot of a cellHTS object.
- `data` accesses and replaces the assayData slot of a cellHTS object. It returns a 3D array with dimensions number of features (product between the number of wells per plates and the number of plates) x number of samples (or replicates) x number of channels.
- `geneAnno` returns the reagent IDs used in the screen (i.e. the contents of fData(object)[,"GeneID"]).
- `intensityFiles` returns a list, where each component contains a copy of the imported input data files.
- `name` obtains the name(s) of the assay, or multiple assays, stored in the object. This corresponds to the contents of column assay of the phenoData slot of the cellHTS object.
- `pdim` obtains the plate dimensions (number of rows and columns) for the data stored in object.
- `plate` plate identifier for each feature (well).
- `plateConf` returns a data.frame that contains what was read from the plate configuration input file (except the first two header rows) during the screen configuration step.
- `plateList` returns a data.frame containing what was read from the plate list file, plus a column status of type character that contains the string "OK" if the data import appeared to have gone well, and the respective error or warning message otherwise.
- `plateEffects` accesses the slots rowcol.effects and overall.effects.
- `position` gives the well number for each feature (well) within each plate.
- `screenDesc` returns an object of class character that contains what was read from the screen description input file during the configuration of the cellHTS object.
- `screenLog` returns a data.frame containing what was read from the screen log input file during the screen configuration step.
**state** This generic function accesses the state of an object derived from the `cellHTS` class.

**well** gives the alphanumeric identifier (e.g. A01, A02, ...) for each well and plate.

**wellAnno** accesses the plate annotation stored in `fData(object)[,"controlStatus"]`.

**Data preprocessing:**

All the following methods work on the data stored in the slot `assayData` of a `cellHTS` object.

**normalizePlates** per-plate data transformation, normalization and variance adjustment.

**Bscore** correction of plate and spatial effects using the B score method (without variance adjustment of the residuals).

**spatialNormalization** correction of spatial effects by fitting a polynomial surface within each plate using local regression (`loess` or `robust local fit`). Uses a second degree polynomial (local quadratic fit). Only wells containing "sample" are considered to fit the models.

**summarizeChannels** combines intensities from a dual-channel assay by applying the function defined in `fun`.

**scoreReplicates** transform per-replicate values into scores.

**summarizeReplicates** summarizes between normalized and scored replicate values, obtaining a single value for each probe.

**scores2calls** applies a sigmoidal transformation to the z-score values stored in a `cellHTS` object mapping them to the range [0,1].

**Miscellaneous:**

**compare2cellHTS** compares two `cellHTS` objects to see whether they derive from the same initial `cellHTS` object.

**convertOldCellHTS** converts an old S3 class `cellHTS` object obtained using `cellHTS` package to new S4 class `cellHTS` object(s) to use with package `cellHTS2`.

**convertWellCoordinates** converts between different ways of specifying well coordinates within a plate. For example, wells can be identified by an alphanumeric character (e.g. "B02" or c("B", "02")) or by an integer value (e.g. 26).

**ROC** creates an object of class `ROC` from a scored `cellHTS` object which can be plotted as a ROC curve.

**getTopTable** generates the hit list from a scored `cellHTS` object and write it to a tab-delimited file.

**getMeasureRepAgreement** calculates the agreement between plate replicates using raw data or normalized data stored in a `cellHTS` object.

**getDynamicRange** calculates per-plate dynamic range of data stored in a `cellHTS` object.

**getZfactor** calculates per-experiment Z’-factor of data stored in a `cellHTS` object. The Z’-factor is a measure that quantifies the separation between the distribution of positive and negative controls.

**plotSpatialEffects** this function plots the per-plate row and column effects estimated by the B score method or by the spatial normalization.

**imageScreen** creates an image plot that gives an overview of the whole set of score values stored in a scored `cellHTS` object.

**writeReport** creates a directory with HTML pages of linked tables and plots documenting the contents of the preprocessing of a `cellHTS` object.

**oneRowPerId** rearranges dataframe entries such that there is exactly one row per ID.

**nbatch** gives the total number of batches in a `cellHTS` object.

**templateDescriptionFile** creates a template description file for an RNAi experiment with default entries compliant with MIAME class and with additional entries specific for a `cellHTS` object.
This function calls the function plotScreen to create a screen plot showing the plate configuration (as defined by the plate configuration file used to configure the cellHTS object – see function configure) used for the RNAi experiment stored in x.

posControls and negControls should be given as vectors of regular expression patterns specifying the name(s) of the positive(s) and negative(s) controls, respectively, as provided in the plate configuration file (and accessed via wellAnno(x)).

By default, if posControls is not given, "pos" will be taken as the annotation name for the wells containing positive controls. Similarly, if negControls is missing, by default "neg" will be considered as the name used to annotate the negative controls. The content of posControls and negControls will be passed to regexpr for pattern matching within the well annotation given in column controlStatus of the featureData slot of the cellHTS object.

Value

Invisibly, a vector with the color map used to display the well annotation in the image plot.
configure

Author(s)

Ligia Bras <ligia@ebi.ac.uk>

References


See Also

plotScreen, writeReport

Examples

data("KcViab")
configurationAsScreenPlot(KcViab)

configure  Configures the plates and plate result files

Description

Annotate the plates and the plate result files of a cellHTS object.

Usage

## S4 method for signature 'cellHTS'
configure(object, descripFile, confFile, logFile, path, descFunArgs=NULL, confFunArgs=NULL, logFunArgs=NULL)

Arguments

object  a cellHTS object.

descripFile  the name of the screen description file (see details). This argument is just passed on to the readLines function, so any of the valid argument types for readLines are valid here, too. Alternatively this can be a function. See details.

confFile  the name of the configuration file (see details). This argument is just passed on to the read.table function, so any of the valid argument types for read.table are valid here, too. Must contain one row for each well and each batch. Alternatively this can be a function. See details.

logFile  optional; the name of the screen log file (see details). This argument is just passed on to the read.table function, so any of the valid argument types for read.table are valid here, too. Alternatively this can be a function. See details.

path  optional; a character of length one indicating the path in which to find the configuration files. It can be useful when the files are located in the same directory, and may be omitted otherwise.

descFunArgs, confFunArgs, logFunArgs  optional; lists of additional arguments that can be passed on if one or more of descripFile, confFile or logFile are functions rather than file names. See details.
Details

The configuration has three components:

\textit{confFile}: This file specifies where the controls are. This file is expected to be a tab-delimited file with two first header rows giving the total number of wells and plates in the screen. The next rows should be in the form of a spreadsheet table with at least three columns named Plate, Well and Content. Columns Plate and Well are allowed to contain regular expressions. Data from wells that are configured as \textit{empty} will be ignored and are set to NA in the data slot \texttt{xraw}. For an example, and for more details, please read the accompanying vignette.

\textit{logFile}: This optional file allows to flag certain measurements as invalid. It is expected to be a tab-delimited file with at least three columns, and column names Plate, Well, and Flag. If there are multiple samples (replicates or conditions), a column called Sample should also be given. If there are multiple channels, a column called Channel must be given. Further columns are allowed.

\textit{descripFile}: The screen description file contains general information about the screen.

Alternatively, any of the three arguments can also be a user-defined function returning data frames similar to those produced by \texttt{read.table} from the respective files. If \texttt{confFile} is a function, it has to return a list, where the first list item is an integer vector of length 2 giving the total number of plates and wells, and the second list item is the \texttt{data.frame} of the actual plate configuration. Additional parameters can be passed on to these functions via the \texttt{descFunArgs}, \texttt{confFunArgs} and \texttt{logFunArgs} arguments. This design allows for instance to import the necessary information directly from a data base rather than using flat files.

Value

An S4 object of class \texttt{cellHTS}, which is obtained by copying \texttt{object} and updating the following slots:

- \texttt{plateConf} a data frame containing what was read from input file \texttt{confFile} (except the first two header rows).
- \texttt{screenLog} a data frame containing what was read from input file \texttt{logFile}.
- \texttt{screenDesc} object of class character containing what was read from input file \texttt{descripFile}.
- \texttt{state} the processing status of the \texttt{cellHTS} object is updated in to \texttt{state["configured"]}=TRUE.
- \texttt{featureData} the column \texttt{controlStatus} is updated taking into account the well annotation given by the plate configuration file.
- \texttt{experimentData} an object of class \texttt{MIAME} containing descriptions of the experiment, constructed from the screen description file.

Author(s)

Wolfgang Huber <huber@ebi.ac.uk>, Ligia Bras <ligia@ebi.ac.uk>

References


See Also

\texttt{readPlateList templateDescriptionFile}
convertOldCellHTS

Examples

datadir <- system.file("KcViabSmall", package = "cellHTS2")
x <- readPlateList("Platelist.txt", name="KcViabSmall", path=datadir)
x <- configure(x, "Description.txt", "Plateconf.txt", "Screenlog.txt", path=datadir)

convertOldCellHTS

Convert an old S3 class cellHTS object to the new S4 class cellHTS object

Description

Convert an old S3 cellHTS object (from the cellHTS package) into one or several S4 cellHTS objects (from the cellHTS2 package).

Usage

convertOldCellHTS(oldObject)

Arguments

oldObject an S3 class cellHTS object obtained using the package cellHTS.

Value

The function returns a list containing one or more cellHTS objects. The element raw contains the unnormalized data from oldObject. Depending on the state of oldObject (on whether it is normalized and scored), the other components of this list can be: normalized, an S4 cellHTS object containing the normalized data, and scored, an S4 class cellHTS object containing the scored data.

Author(s)

Ligia P. Bras <ligia@ebi.ac.uk>

References


See Also

updateCellHTS

Examples

data("oldKcViabSmall")
out <- convertOldCellHTS(oldKcViabSmall)
names(out)
out[["raw"]]

convertWellCoordinates

Converts different well identifiers

Description

For example, "B02" <-> c("B", "02") <-> 26, "AB32" <-> c("AB", "32") <-> 1328.

Usage

convertWellCoordinates(x, pdim, type="384")

Arguments

x
  either: a character vector with alphanumeric well identifiers (e.g. B03 or AB32); or an nx2 character matrix whose first column contains letters and whose second column contains numbers; or an integer vector with position identifiers for wells within a plate (e.g. 27).

pdim
  a vector of length 2 with names nrow and ncol giving the number of rows and columns in a plate. E.g. `c(nrow=32, ncol=48)` for 1536-well plates.

type
  an alternative way of specifying pdim. Supported are the values "24" for c(nrow=4, ncol=6), "96" for c(nrow=8, ncol=12), "384" for c(nrow=16, ncol=24) and "1536" for c(nrow=32, ncol=48).

Value

A list with elements: letnum, with the alphanumeric well identifiers; let.num, with the alphanumeric well identifiers giving as a pair c(letter(s), 2-digits); num, with the integer position of the well within a plate.

Author(s)

Ligia Bras <ligia@ebi.ac.uk> and Wolfgang Huber <huber@ebi.ac.uk>

Examples

pd <- c("nrow"=8L, "ncol"=12L) # 96-well plate
w <- sample(prod(pd), 3L)
wpos <- convertWellCoordinates(w, pd)
wpos
Access and replace the assayData slot of a cellHTS object

Description

This generic function accesses and replaces the data stored in slot assayData of an object of cellHTS class.

Usage

Data(object)
Data(object) <- value

Arguments

object Object derived from class cellHTS.
value a 3D array of dimensions dim(object)[1] (this corresponds to the total number of features: number of wells per plate x number of plates) x dim(object)[2] (this corresponds to the number of samples or replicates) x number of channels.

Value

Data returns a 3D array containing the contents of slot assayData. This array has dimensions number of features (product between the number of wells per plates and the number of plates) x number of samples (or replicates) x number of channels. Depending on the preprocessing status of the cellHTS object, this array corresponds to the raw data, or to normalized data or to scored data.

See class cellHTS for details.

Author(s)

Ligia Bras

References


See Also

cellHTS
dualCh

Description

A sample cellHTS object containing dual channel data

Usage

##cellHTS object, see examples for details

Format

cellHTS object

Examples

data(dualCh)

getDynamicRange

Per-plate dynamic range of a cellHTS object

Description

Calculates per-plate dynamic range of data stored in a cellHTS object.

Usage

getDynamicRange(x,
verbose=interactive(),
definition,
posControls,
negControls)

Arguments

x a configured cellHTS object. See details.
verbose a logical, if TRUE the function reports some of its intermediate progress. The default is the state of interactive().
definition a character string with possible values "ratio" or "difference". See details.
posControls (optional) a list or vector of regular expressions specifying the name of the positive controls. See details.
negControls (optional) a vector of regular expressions specifying the name of the negative controls. See details.
getDynamicRange

Details

x should be an already configured cellHTS object (state(x)["configured"]=TRUE), so that the information about the well annotation of the plates is available.

The per-plate dynamic ranges are calculated for the data stored in slot assayData of x. This can be raw data, normalized data or scored data.

If definition="difference", the dynamic range is calculated as the absolute difference between the arithmetic average on positive and negative controls.

If definition="ratio", the dynamic range is calculated as the ratio between the geometric mean on positive and negative controls.

NOTE: the argument definition should only be set to "ratio" if data are in positive scale!

If definition is missing it is determined based on the scale of the data. By default, if data are in positive scale, definition is set to "ratio", otherwise, it is set to "difference".

posControls and negControls should be given as a vector of regular expression patterns specifying the name of the positive(s) and negative(s) controls, respectively, as provided in the plate configuration file (and accessed via wellAnno(x)). The length of these vectors should be equal to the current number of channels in x (dim(Data(x))[3]). By default, if posControls is not given, pos will be taken as the name for the wells containing positive controls. Similarly, if negControls is missing, by default neg will be considered as the name used to annotated the negative controls. The content of posControls and negControls will be passed to regexpr for pattern matching within the well annotation given in wellAnno(x) (see examples). If no controls are available for a given channel, use "" or NA for that channel. For example, posControls = c("", "(?i)^diap$") means that channel 1 has no positive controls, while diap is the positive control for channel 2.

The arguments posControls and negControls are particularly useful in multi-channel data since the controls might be reporter-specific, or after normalizing multi-channel data.

If there are different positive controls, the dynamic range is calculated between each of the positive controls and the negative controls.

In the case of a two-way assay, where two types of "positive" controls are used in the screen ("activators" and "inhibitors"), posControls should be defined as a list with two components (called act and inh), each of which should be vectors of regular expressions of the same length as the current number of reporters (as explained above). The dynamic range is calculated between each type of positive control (activators or inhibitors) and the negative controls.

Value

The function generates a list with the per-plate dynamic ranges in each channel and each replicate. The average dynamic range between replicates is also given. Each element of this list is an array of dimensions nrPlates x (nrReplicates + 1) x nrChannels, and is named by the positive controls. In the case of a two-way assay, these elements are called activators and inhibitors, while for a one-way assay, the elements have the same name of the positive controls. See Examples section.

Author(s)

Ligia P. Bras <ligia@ebi.ac.uk>

References

getEnVisionRawData

Read a plate file obtain from EnVision Plate Reader

Description
Import functions to read a plate file obtained from EnVision Plate Reader. These functions should be set as the import function of readPlateList through the argument importFun when reading plate result files obtained from EnVision plate reader.

Usage
getEnVisionRawData(f, p)
getEnVisionCrosstalkCorrectedData(f, p)

Arguments
f the name of the result plate file to read.
p capturing the additional path argument, which will actually be ignored. f is expected to be a fully resolved file path.

Details
These functions should not be called directly. Instead, they should be set as the import function of readPlateList through the argument importFun when reading plate result files obtained from an EnVision plate reader.

Value
These functions return a list with two components. The first component should be a 'data.frame' with the following slots: well (a character vector with the well identifier in the plate) and val (the intensity values measured at each well). The second component of this list should be a character vector containing a copy of the imported input data file (such as the output of readLines). It should be suitable to be used as input for writeLines.

Examples
data(KcViabSmall)
## pCtrls <- c("pos")
## nCtrls <- c("neg")
## or for safety reasons (not a problem for the current well annotation, however)
pCtrls <- c("^pos$")
nCtrls <- c("^neg$")
dr <- getDynamicRange(KcViabSmall, definition="ratio", posControls=pCtrls, negControls=nCtrls)
## same as:
## getDynamicRange(KcViabSmall)
	x <- normalizePlates(KcViabSmall, scale="multiplicative", log=TRUE, method="median", varianceAdjust="none")
try(drn <- getDynamicRange(x, definition="ratio"))
try(drn <- getDynamicRange(x, definition="difference")

See Also
configure, writeReport
getMeasureRepAgreement

Author(s)
Ligia Bras <ligia@ebi.ac.uk>

See Also
readPlateList

Examples

plateFile <- system.file("EnVisionExample/XXX_1500.csv", package = "cellHTS2")
onePlate <- getEnVisionRawData(plateFile)

## to get the cross talk corrected data:
onePlate2 <- getEnVisionCrosstalkCorrectedData(plateFile)

getMeasureRepAgreement

*Measures of agreement between plate replicates from a cellHTS object*

Description

Calculate the agreement between plate replicates using raw data or normalized data stored in a cellHTS object. This function calculates the repeatability standard deviation between replicate plates and the correlation coefficient between replicates. If there are more than 2 replicates, the minimum and maximum correlation between replicates is given. These measures are calculated only for sample wells.

Usage

generateRepAgreement(x, corr.method = "spearman")

Arguments

- **x**
  - a configured cellHTS object. See details.
- **corr.method**
  - a character string indicating which correlation coefficient should be computed. Can be either "pearson", "kendall" or "spearman" (default). The correlation is calculated by calling the function cor.

Details

Given an already configured cellHTS object (state(x)["configured"] == TRUE), this function calculates the repeatability standard deviation between replicate plates and the correlation coefficient between plate replicates using only the sample wells. If there are more than 2 replicates, the minimum and maximum correlation value between pairs of replicates are given.

These measures are calculated using the data values stored in slot assayData of the x.

For a given plate p, the repeatability standard deviation is determined as the square root of the average of the squared standard deviations (sr) calculated for each sample well k by considering the measurement of all of the replicates:
RepStDev\_p = \sqrt{\frac{\sum sr^2}{n_k}}

where \( n_k \) is the total number of sample probes for plate \( p \).

**Value**

The function generates a list with elements:

"repStDev": matrix with the calculated repeatability standard deviation between plate replicates. It has dimensions \( nrPlates \times nrChannels \);

"corrCoef" (if the number of replicates equals 2): matrix with the correlation coefficients between plate replicates. It has dimensions: \( nrPlates \times nrChannels \);

"corrCoef.min" (if the number of replicates is greater than 2): matrix with the minimum value of the correlation coefficients between plate replicates. It has dimensions \( nrPlates \times nrChannels \);

"corrCoef.max" (if the number of replicates is greater than 2): matrix with the maximum value of the correlation coefficients between plate replicates. It has dimensions \( nrPlates \times nrChannels \).

**Author(s)**

Ligia Bras <ligia@ebi.ac.uk>

**References**


**See Also**

configure, writeReport

**Examples**

```r
data(KcViabSmall)
repAgree <- getMeasureRepAgreement(KcViabSmall)
x <- normalizePlates(KcViabSmall, scale="multiplicative", log=FALSE, method="median", varianceAdjust="none")
repAgree <- getMeasureRepAgreement(x)

getTopTable(cellHTSlist, file="topTable.txt", verbose=interactive())
```

---

**Description**

Generate the hit list from a scored cellHTS object.

**Usage**

```r
getTopTable(cellHTSlist, file="topTable.txt", verbose=interactive())
```
**Arguments**

- `cellHTSlist`: a list of `cellHTS` objects. See details.
- `file`: the name of the output file. Default is "topTable.txt".
- `verbose`: a logical value, if TRUE, the function reports its progress. Defaults to the state of `interactive()`.

**Details**

Argument `cellHTSlist` should be a list containing at least one component named "scored" which corresponds to a scored `cellHTS` object. Other possible components of `cellHTSlist` can be:

- "raw": a `cellHTS` object containing unpreprocessed data. I.e. `state(cellHTSlist["raw"])["normalized"]=FALSE`.
- "normalized": a `cellHTS` object containing normalized data. I.e. `state(cellHTSlist["normalized"])["normalized"]=TRUE` and `state(cellHTSlist["normalized"])["scored"]=FALSE`.

All of the components of `cellHTSlist` should be `cellHTS` objects containing data from the same experiment, but in different preprocessing stages.

This function generates a data frame that is written to `file`. This data frame and the output file contain the list of scored probes ordered by decreasing score values. They have one row for each well and plate, and contain the following columns (depending on the components of `cellHTSlist`):

- `plate`: plate identifier for each well.
- `position`: gives the position of the well in the plate (ranges from 1 to the total number of wells in the plate).
- `well`: gives the alphanumeric identifier for the wells.
- `score`: content of slot `assayData` of the scored `cellHTS` object given in `cellHTSlist["scored"]`.
- `codewellAnno`: well annotation as given by the plate configuration file.
- `codefinalWellAnno`: gives the final well annotation for the scored values. It combines the information given in the plate configuration file with the values in `assayData` slot of the scored `cellHTS` object, in order to have into account the wells that have been flagged either by the screen log file, or manually by the user during the analysis. These flagged wells appear with the annotation `flagged`.
- `raw_ri_chj`: (if `cellHTSlist["raw"]` is given) contains the raw intensities for replicate i in channel j (content of slot `assayData` of the `cellHTS` object given in `cellHTSlist["raw"]`).
- `median_chj`: (if `cellHTSlist["raw"]` is given) corresponds to the median of raw measurements across replicates in channel j.
- `diff_chj`: (if `cellHTSlist["raw"]` is given and if there are two replicates or samples) gives the difference between replicate (sample) raw measurements in channel j.
- `average_chj`: (if `cellHTSlist["raw"]` is given and if there are more than 2 replicates or samples) corresponds to the average between replicate raw intensities for channel j.
- `raw/PlateMedian_ri_chj`: (if `cellHTSlist["raw"]` is given) this column gives the ratio between each raw measurement and the median intensity in each plate for replicate (or sample) i in channel j. The plate median is determined for the raw intensities using exclusively the wells annotated as `sample`.
- `normalized_ri_chj`: (if `cellHTSlist["normalized"]` is given) gives the normalized intensities for replicate (sample) i in channel j. This corresponds to the content of slot `assayData` of the `cellHTS` object given in `cellHTSlist["normalized"]`.

Additionally, if the `cellHTS` object given in `cellHTSlist["scored"]` is already annotated, the output `topTable` also contains the gene annotation stored in slot `featureData`. 
getZfactor

Per-experiment \( Z' \)-factor of a cellHTS object

Description

Calculates per-experiment \( Z' \)-factor of data stored in a cellHTS object. The \( Z' \)-factor is a measure that quantifies the separation between the distribution of positive and negative controls.

Usage

getZfactor(x, robust=TRUE, verbose=interactive(), posControls, negControls)

Arguments

- **x**: a configured cellHTS object. See details.
- **robust**: a logical, if TRUE the \( Z' \)-factor is calculated using the median and MAD instead of mean and standard deviation, respectively.
- **verbose**: a logical, if TRUE the function reports some of its intermediate progress. The default is the state of interactive().
- **posControls**: (optional) a list or vector of regular expressions specifying the name of the positive controls. See details.
- **negControls**: (optional) a vector of regular expressions specifying the name of the negative controls. See details.
Details

x should be an already configured cellHTS object (state(x)["configured"]=TRUE), so that the information about the well annotation of the plates is available.

The per-experiment Z'-factor values are calculated for the data stored in slot assayData of x.

If robust=TRUE (default), the Z'-factor is calculated using robust estimates of location (median) and spread (mad).

posControls and negControls should be given as a vector of regular expression patterns specifying the name of the positive(s) and negative(s) controls, respectively, as provided in the plate configuration file (and accessed via wellAnno(x)). The length of these vectors should be equal to the current number of channels in x (dim(Data(x))[3]). By default, if posControls is not given, pos will be taken as the name for the wells containing positive controls. Similarly, if negControls is missing, by default neg will be considered as the name used to annotated the negative controls.

The content of posControls and negControls will be passed to regexpr for pattern matching within the well annotation given in wellAnno(x) (see examples). If no controls are available for a given channel, use "" or NA for that channel. For example, posControls = c("", "(?i)^diap$") means that channel 1 has no positive controls, while diap is the positive control for channel 2.

The arguments posControls and negControls are particularly useful in multi-channel data since the controls might be reporter-specific, or after normalizing multi-channel data.

If there are different positive controls, the Z'-factor is calculated between each of the positive controls and the negative controls.

In the case of a two-way assay, where two types of "positive" controls are used in the screen ("activators" and "inhibitors"), posControls should be defined as a list with two components (called act and inh), each of which should be vectors of regular expressions of the same length as the current number of reporters (as explained above). The Z'-factor values are calculated between each type of positive control (activators or inhibitors) and the negative controls.

Value

The function generates a list with the per-experiment Z’-factor values in each channel and each replicate. Each element of this list is a matrix with dimensions nrReplicates x nrChannels, and is named by the positive controls. In the case of a two-way assay, these elements are called activators and inhibitors, while for a one-way assay, the elements have the same name of the positive controls. See examples section.

Author(s)

Ligia P. Bras <ligia@ebi.ac.uk>

References


See Also

configure, writeReport
Examples

data(KcViabSmall)
## pCtrls <- c("pos")
## nCtrls <- c("neg")
## or for safety reasons (not a problem for the current well annotation, however)
pCtrls <- c("^pos$")
nCtrls <- c("^neg$")
zf <- getZfactor(KcViabSmall, robust=TRUE, posControls=pCtrls, negControls=nCtrls)

x <- normalizePlates(KcViabSmall, scale="multiplicative", log=FALSE, method="median", varianceAdjust="none")
zfn <- getZfactor(x)

###

**gseaModule**

*Constructor for an object of class gseaModule*

**Description**

GseaModule objects encapsulate all the information that is necessary to add the results of a gene set enrichment analysis to a cellHTS report. This feature is still experimental.

**Usage**

gseaModule(geneSets, statFuns, scores, annotation)

**Arguments**

geneSets An object of class GeneSetCollection containing the information about gene sets for which to perform the analysis.

statFuns A list of functions to compute per gene set statistics. These will be called by applyByCategory and need to be able to handle two mandatory arguments: x are the scores for the respective category, and y are all scores of the whole assay. This allows for statistics like t.test(x, y).

scores An optional numeric vector of assay scores. This should be extended to also handle numeric matrices for multi-channel assays.

annotation A data.frame with additional annotation for the respective gene sets.

**Details**

The resulting gseaModule object can be supplied as an additional argument to writeReport. This feature is still experimental.

**Value**

An object of class gseaModule.

**Author(s)**

Florian Hahne
imageScreen

Experiment-wide quality control plot of a cellHTS object

Description

Experiment-wide quality control plot of a scored cellHTS object.

Usage

imageScreen(object, ar=3/5, zrange=NULL, map=FALSE, anno=NULL, col=list(posNeg=rev(brewer.pal(11, "RdBu"))[c(1:5, rep(6, 3), 7:11)], pos=brewer.pal(9, "Greys")), nbImageBins=256, nbLegendBins=7)

Arguments

object a cellHTS object that has already been scored (i.e. state(object)['scored']=TRUE).

ar the desired aspect ration for the image plot (i.e. number of columns per number of rows)

zrange the range of values to be mapped into the color scale. If missing, zrange will be set to the range of the score values stored in slot assayData of object.

map a logical value that determines whether an image map should be created using tooltips to indicate the annotation at each position. It only makes sense to set it to TRUE when the function is called from writeReport function, so the default is FALSE.

anno optional input giving the annotation information for the mapping. It should be a vector of the same size as the total number of featured in object. See details.

col a list giving the colors for the plot. The first element posNeg is used if zrange[1]<0 and zrange[2]>0, the second if all values are either positive or negative.

nbImageBins The number of color bins used in the map. Default is 256.

nbLegendBins The number of color bins shown in the legend. Default is 7.

Details

This function creates an image plot that gives an overview of the whole set of score values stored in slot assayData of a scored cellHTS object. When the annotation mapping is performed, by default, anno is set to:

1. (if object is annotated) The content of column named GeneSymbol or named GeneID (if the former is not available) of the featureData slot of object;
2. The position within the plate, if object is not annotated.

Author(s)

Ligia Bras <ligia@ebi.ac.uk>

References

intensityFiles

Retrieve the contents of the input files used to generate a given cellHTS object.

Description

These generic functions access different slots of an object derived from the cellHTS class, which contain the original content of the input files used to create the object.

Usage

intensityFiles(object)
plateList(object)
plateConf(object)
screenLog(object)
screenDesc(object)

Arguments

object an object of class cellHTS.

Value

intensityFiles returns a list, where each component contains a copy of the imported input data files.
plateList returns a data.frame containing what was read from the plate list file, plus a column status of type character that contains the string "OK" if the data import appeared to have gone well, and the respective error or warning message otherwise. See readPlateList for more details.
plateConf returns a data.frame that contains what was read from the plate configuration input file (except the first two header rows) during the screen configuration step. For more details see configure.
screenLog returns a data.frame containing what was read from the screen log input file during the screen configuration step. See configure for more details.
screenDesc returns an object of class character that contains what was read from the screen description input file during the configuration of the cellHTS object. See configure for more details.

See Also

normalizePlates, summarizeChannels, scoreReplicates, summarizeReplicates, Data writeReport

Examples

data(KcViabSmall)
x <- KcViabSmall
x <- normalizePlates(x, scale="multiplicative", log=FALSE, method="median", varianceAdjust="none")
x <- scoreReplicates(x, sign="-", method="zscore")
x <- summarizeReplicates(x, summary="min")
imageScreen(x, zrange=c(-5,5))
**KcViab**

**Author(s)**
Ligia Bras <ligia@ebi.ac.uk>

**See Also**
cellHTS, readPlateList, configure, name

---

**KcViab**

* A sample cellHTS object - *D. melanogaster* genome-wide RNAi screen

---

**Description**
Archived cellHTS object from a genome-wide RNAi screen of cell viability in *Drosophila* Kc167 cells.

**Usage**
##cellHTS object, see examples for details

**Format**
cellHTS object

**References**

**Examples**
data(KcViab)

---

**KcViabSmall**

* A sample cellHTS object - *D. melanogaster* genome-wide RNAi screen

---

**Description**
Archived cellHTS object corresponding to the first three 384-well plates of a genome-wide RNAi screen of cell viability in *Drosophila* Kc167 cells.

**Usage**
##cellHTS object, see examples for details

**Format**
cellHTS object
References


Examples

data(KcViabSmall)

normalizePlates

Per-plate data transformation, normalization and variance adjustment

Description

Plate-by-plate normalization of the raw data stored in slot assayData of a *cellHTS* object. Normalization is performed separately for each plate, replicate and channel. Log2 data transformation can be performed and variance adjustment can be performed in different ways (none, per-plate, per-batch or per-experiment).

Usage

normalizePlates(object, scale="additive", log=FALSE, method="median", varianceAdjust="none", posControls, negControls,...)

Arguments

object a *cellHTS* object that has already been configured. See details.

scale a character specifying the scale that the input data are considered to be on: "additive" scale (default) or "multiplicative". The interpretation of this terminology is that data on an additive scale will be normalised by subtraction of a correction offset, whereas data on a multiplicative scale are normalised by division through a correction factor.

log logical. If TRUE, data will first be log2 transformed. If data are on an additive scale (i.e. if scale is "additive"), then log is only allowed to be FALSE. The default is log=FALSE.

method character specifying the normalization method to use for the per-plate normalization. Allowed values are "median" (the default), "mean", "shorth", "POC", "NPI", "negatives", Bscore and "locfit". See details.

varianceAdjust character specifying the variance adjustment to perform. Allowed values are "none" (the default), code "byPlate", "byBatch" and "byExperiment". See details.

posControls a vector of regular expressions giving the name of the positive control(s). See details.

negControls a vector of regular expressions giving the name of the negative control(s). See details.

Further arguments that get passed on to the function implementing the normalization method chosen by method. Currently, this is only used for Bscore and locfit.
**normalizePlates**

**Details**

The function `normalizePlates` uses the content of the `assayData` slot of object. For dual-channel data, a recommended workflow is (i) to correct for plate effects using the `normalizePlates` function, (ii) combine the two channels using the function `summarizeChannels`, and (iii) finally, if necessary, normalize the summarized intensities calling `normalizePlates` again.

In this function, the normalization is performed in a plate-by-plate fashion, following this workflow:

1. Log transformation of the data (optional)
2. Per-plate normalization
3. Variance adjustment of the plate intensity corrected data (optional)

The argument `scale` defines the scale of the data. If the data are on a multiplicative scale (`scale="multiplicative"`), the data can be log2 transformed by setting `log=TRUE`. This then changes the scale of the data to code "additive".

In the next step of preprocessing, intensities are corrected in a plate-by-plate basis using the chosen normalization method:

- If `method="median"`, plates effects are corrected by the median value across wells that are annotated as sample in `wellAnno(object)`, for each plate and replicate.
- If `method="mean"`, the average in the sample wells is used instead.
- If `method="shorth"`, the midpoint of the shorth of the distribution of values in the wells annotated as sample is used.
- If `method="negatives"`, the median of the negative controls is used.

Depending on the scale of the data prior to normalization, the data are divided by the above defined correction factors (scale: "multiplicative"), or the value is subtracted (scale: "additive").

Further available normalization methods are:

- `method="POC"` (percent of control): for each plate and replicate, each measurement is divided by the average of the measurements on the plate positive controls, and multiplied by 100.
- `method="NPI"` (normalized percent inhibition): each measurement is subtracted from the average of the intensities on the plate positive controls, and this result is divided by the difference between the means of the measurements on the positive and the negative controls.
- `method="Bscore"`: for each plate and replicate, the B-score method, which is based on a 2-way median polish, is applied to remove row and column biases.
- `method="locfit"` (robust local fit regression): for each plate and replicate, spatial effects are removed by fitting a bivariate local polynomial regression (see `spatialNormalization`).

In the final preprocessing step, variance of plate-corrected intensities can be adjusted as follows:

- `varianceAdjust="byPlate"`: per plate normalized intensities are divided by the per-plate median absolute deviations (MAD) in "sample" wells. This is done separately for each replicate and channel;
- `varianceAdjust="byBatch"`: using the content of slot `batch`, plates are split according to assay batches and the individual normalized intensities in each group of plates (batch) are divided by the per-batch of plates MAD values (calculated based on "sample" wells). This is done separately for each replicate and channel;
- `varianceAdjust="byExperiment"`: each normalized measurement is divided by the overall MAD of normalized values in wells containing "sample". This is done separately for each replicate and channel;
By default, no variance adjustment is performed (varianceAdjust="none").

The arguments posControls and negControls are required for applying the normalization methods based on the control measurements that is, when method="POC", or method="NPI", or method="negatives"). posControls and negControls should be vectors of regular expression patterns specifying the name of the positive(s) and negative(s) controls, respectively, as provided in the plate configuration file (and accessed via wellAnno(object)). The length of these vectors should be equal to the current number of channels in object (i.e. to the dim(Data(object))[3]). By default, if posControls is not given, pos will be taken as the name for the wells containing positive controls. Similarly, if negControls is missing, by default neg will be considered as the name used to annotate the negative controls. The content of posControls and negControls will be passed to regexpr for pattern matching within the well annotation given in the featureData slot of object (which can be accessed via wellAnno(object)) (see examples for summarizeChannels). The arguments posControls and negControls are particularly useful in multi-channel data since the controls might be reporter-specific, or after normalizing multi-channel data.

See the Examples section for an example on how this function can be used to apply a robust version of the Z score method, whereby, for each plate and replicate, the per-plate median (computed only from sample wells) is subtracted from the measurements, and the result is divided by the per-plate MAD (only from sample wells).

Value

An object of class cellHTS with the normalized data stored in slot assayData (its previous contents were overridden). The processing status of the object is updated in the slot state to object@state["normalized"] = TRUE. Additional slots of object may be updated if method="Bscore" or method="locfit" are used. Please refer to the help page of the Bscore function and spatialNormalization functions.

Author(s)

Ligia Bras <ligia@ebi.ac.uk>, Wolfgang Huber <huber@ebi.ac.uk>

References


See Also

Bscore, spatialNormalization, summarizeChannels

Examples

data(KcViabSmall)
  # per-plate median scaling of intensities
  x1 <- normalizePlates(KcViabSmall, scale="multiplicative", log=FALSE, method="median", varianceAdjust="none")
  # per-plate median subtraction of log2 transformed intensities
  x2 <- normalizePlates(KcViabSmall, scale="multiplicative", log=TRUE, method="median", varianceAdjust="none")
  ## Not run:
  x3 <- normalizePlates(KcViabSmall, scale="multiplicative", log=TRUE, method="Bscore", varianceAdjust="none")
  ## End(Not run)

  ## robust Z score method (plate intensities are subtracted by the per-plate median on sample wells and divided by per-plate MAD on sample wells):
  xZ <- normalizePlates(KcViabSmall, scale="additive", log=FALSE, method="median", varianceAdjust="byPlate"")
## an example to illustrate the use of slot 'batch':
## Not run:
try(xnorm <- normalizePlates(KcViabSmall, scale="multiplicative", method="median", varianceAdjust="byBatch")

# It doesn't work because we need to have slot 'batch'!
# For example, we will suppose that a different lot of reagents was used for plate 1:
pp <- plate(KcViabSmall)
fData(KcViabSmall)$"reagent" <- "lot B"
fData(KcViabSmall)$"reagent"[pp==1] <- "lot A"
fvarMetadata(KcViabSmall)["reagent",] <- "Lot of reagent used"

bb <- as.factor(fData(KcViabSmall)$"reagent")
batch(KcViabSmall) <- array(as.integer(bb), dim=dim(Data(KcViabSmall)))
## check number of batches:
bbatch(KcViabSmall)
x1 <- normalizePlates(KcViabSmall, scale="multiplicative", log = FALSE, method="median", varianceAdjust="byBatch")

## End(Not run)

---

**oldKcViabSmall**

A sample S3 class cellHTS object - D. melanogaster genome-wide RNAi screen

### Description

Archived S3 class cellHTS object corresponding to the first three 384-well plates of a genome-wide RNAi screen of cell viability in *Drosophila* Kc167 cells. This data set was assembled and stored using cellHTS package and is used to show how to convert an old S3 class cellHTS object to the new S4 data class associated with cellHTS2 package.

### Usage

```r
##cellHTS object, see examples for details
```

### Format

S3 class cellHTS object as detailed in package cellHTS.

### References


### Examples

```r
data(oldKcViabSmall)
```
oneRowPerId

Rearrange dataframe entries such that there is exactly one row per ID.

Description

Rearrange dataframe entries such that there is exactly one row per ID. The IDs are taken from the argument ids and are matched against the first column of x. If an ID is missing in x[,1], a row with NA values is inserted. If an ID occurs multiple times in x[,1], rows are collapsed into characters of comma-separated values.

Usage

oneRowPerId(x, ids)

Arguments

x   dataframe.
ids  character vector.

Value

A dataframe whose rows correspond 1:1 to ids.

Author(s)

W. Huber <huber@ebi.ac.uk>, Ligia Pedroso Bras <ligia@ebi.ac.uk>

Examples

x = data.frame(ids=I(c("a", "a", "c")), val=11:13)
oneRowPerId(x, letters[1:3])

plate

Retrieve information related with the format of a RNAi experiment

Description

These generic functions retrieves information related with the format of RNAi experiment conducted in multi-plate format and stored in an object of cellHTS class.

Usage

pdim(object)
plate(object)
well(object)
position(object)

Arguments

object  an object of class cellHTS.
Value

pdim returns a vector of length 2 containing integer values that correspond to the number of rows and columns in a plate. This corresponds to the plate format used in the screen (for example, 96-well or 384-well plates).

plate returns a vector of integers with the same length as the product between the number of wells per plate and the number of plates. This gives the plate number for each well in the assay. Corresonds to fData(object)[,"plate"].

well returns a character vector containing the wells identifiers (for example "A01", "H06"). Its length equals the product between the number of wells per plate and the number of plates. This vector corresponds to fData(object)[,"well"].

position gives a numeric vector containing every well positions (1, 2, 3, ...). Its length is equal to the product between the number of wells per plate and the number of plates.

Author(s)

Ligia Bras <ligia@ebi.ac.uk>

See Also

cellHTS

plateEffects(object)

Arguments

object Object derived from class cellHTS that has been normalized by a spatial normalization method. See details.

Value

plateEffects returns a list with two elements: rowcol and overall.

plateEffects["rowcol"] corresponds to the contents of slot rowcol.effects of object. This is a 3D array with the same dimensions as Data(object): nr of Features x nr of Samples x nr of Channels of the current cellHTS object.

plateEffects["overall"] contains the data stored in slot overall.effects of object. This is a 3D array with dimensions nr Plates x nr Samples x nr Channels of object. Slot overall.effects is only estimated when B score method is used to normalize the data.

Note that these 2 slots accessed by this function were stored after preprocessing the data either using Bscore method or local regression fit (see normalizePlates for details), by setting the option save.model=TRUE.
Author(s)
Ligia Bras <ligia@ebi.ac.uk>

See Also

- `cellHTS`
- `normalizePlates`
- `Bscore`
- `spatialNormalization`

---

**plotSpatialEffects**

*Plate plot with the row and column offsets estimated by the a spatial normalization method*

**Description**

The function plots the per-plate row and column effects estimated by the B score method or by the spatial normalization.

**Usage**

```r
plotSpatialEffects(object, channel = 1, plates)
```

**Arguments**

- `object` a `cellHTS` object that has been normalized using the B score method or other spatial normalization (see details).
- `channel` a numeric value giving the channel of `object` to plot.
- `plates` a vector of integers giving the plate numbers to plot. If missing, the function considers all of the plates.

**Details**

The function plots the *plate plots* displaying the row and column offsets (stored in slot `rowcol.effects` of the `cellHTS` object) within the plates in `plates`, and for channel `channel`, as determined by the B score method or spatial normalization. Before plotting the spatial offsets, the values within the chosen channel (argument `channel`) are transformed in order to be confined in the range $[0,1]$, as follows:

$$y' = \frac{(y - \min(y))}{\max(y) - \min(y)}$$

Here, $y'$ are the transformed values, and $y$ the estimated spatial effects. The maximum and the minimum values are calculated using all the values in `plateEffects(object)$rowcol[,channel]`.

**Author(s)**
Ligia P. Bras <ligia@ebi.ac.uk>

**See Also**

- `plotPlate`
- `Bscore`
- `spatialNormalization`
- `normalizePlates`
- `summarizeChannels`
Examples

data(KcViabSmall)
x <- normalizePlates(KcViabSmall, scale="multiplicative", log=TRUE, method="Bscore", save.model = TRUE)
## see plate plots with the row and column estimated offsets for plates 1 and 3:
plotSpatialEffects(x, plates=c(1,3))

readHTAnalystData

---

**Description**

Reads input files (specified by the argument `filenames`) containing replicate data for the same set of assay plates and obtained using an HTanalyser. The number of replicates corresponds to the number of given files (`length(filenames)`), while the total number of plates should be given by the argument `nrPlates`.

**Usage**

```r
readHTAnalystData(filenames, path=dirname(filenames), name, nrPlates, verbose=interactive())
```

**Arguments**

- `filenames`: vector of characters giving the name(s) of the input file(s) obtained in a HTanalyser plate reader and containing the set of measurements for each set of replicate plates (see details).
- `path`: can be either a character vector with the same length as `filenames` or a character of length 1 indicating the path in which to find the input file(s) `filenames`. By default, it can extract the path from `filenames`.
- `name`: a character of length 1 with the experiment name.
- `nrPlates`: an integer value indicating the number of plates in the input file(s).
- `verbose`: a logical value, if TRUE, the function reports some of its intermediate progress. Defaults to the state of `interactive()`.

**Details**

This function reads input files obtained in a HTanalyser plate reader for a multi-plate format screening experiment. Data for the same set of replicate measurements of all plates are expected to come in a single input file. So, the argument `filenames` should be a character vector specifying the name of the input files for each replicate. Each of these files is expected to contain data for a total of `nrPlates` assay plates. It contains meta-experimental data together with plate measurements in a matrix-like format. The same type of format is expected for each of the `nrPlates` contained in each input file indicated in `filenames`. The input files should be suitable to be used as input for `readLines`.

The following metadata fields are expected to be repeated along the input file(s) for each assay plate:

- Microplate format, indicates the plate format used in the screen. Should be the same across plates.
- Barcode, this field is used to uniquely identify each assay plate.
readHTAnalystData

- Method ID, indicates the method used to perform the assay. Should be the same for every plate.
- Data, indicates the type of data and should be the same for every plate.
- Units, indicates the units of the readings. Should be the same for every plate.
- Display format, indicating the numeric format used to display the data measurements. Should be the same along plates.

This function expects that the next line after the meta field "Display format" for a given plate contains the column numbers (1:ncol) which are then followed by the matrix of measurements in each well. Each entry of the data matrix corresponds to a position in the assay plate with coordinates (row, col), except the first column, which gives the row ID, as upper letters (A, B, ...).

Value

An object of class cellHTS, which extends the class NChannelSet. After calling this function, the content of the following slots is as follows:

- assayData: an object of class AssayData containing the imported measurement data. Each matrix represents a single channel, and each sample (replicate) corresponds to a column. Thus, the total number of rows in each matrix corresponds to the product between the number of wells per plate and the number of assay plates.
- phenoData: the argument name is stored in its column assay.
- featureData: the information about the plate and well identifiers for each plate measurement are stored in columns plate and well of this slot.
- intensityFiles: a list, where each component contains a copy of the measurement data of a given plate, replicate and channel. Its length corresponds to the number of rows of plateList.
- plateList: a data.frame containing the columns "Filename", "Plate", "Replicate", "Channel" and "status". Each row of this slot makes the correspondence between a given component (name given in column "Filename") in the list stored in slot intensityFiles (i.e. plate) and its respective plate, replicate and channel number. Thus, this data.frame contains as many rows as the product between the total number of plates, replicates and channels. The last column named status is of type character and contains the string "OK" indicated the success status of the data import.

Author(s)

W. Huber <huber@ebi.ac.uk>, Ligia Bras <ligia@ebi.ac.uk>

References


See Also

To read a collection of plate result files with measurements of a single plate and replicate, see readPlateList.
Examples

dadir <- system.file("KcCellTiter", package = "cellHTS2")
x <- readHTAnalystData(filenames = c("KcCellTiter0607.txt", "KcCellTiter0707.txt"), name="KcData", path=dadir)
x <- configure(x, confFile = "Plateconf.txt", logFile="Screenlog.txt", descripFile = "Description.txt", path = datadir)

readPlateList

Read a collection of plate reader data files

Description

Reads a collection of plate reader data files into a cellHTS object. The names of the files, plus additional information (plate number, repeat number, assay/treatment/condition) is expected to be stored in a tab-delimited table specified by the argument filename. Alternatively, the data can be provided directly from non file-based sources, e.g. a data base (see details).

Usage

readPlateList(filename, path=NA, name="anonymous", importFun, dec=".", verbose=interactive(), ...)

Arguments

filename

the name of the plate list file (see details). This argument is just passed on to the read.table function, so any of the valid argument types for the file argument of read.table are valid here, too. Alternatively, a user-defined function which is supposed to create a table-like R object of the plate list. Additional arguments to readPlateList in ... will be passed on to this function.

path

a character of length 1 indicating the path in which to find the plate reader files. If the importFun argument is supplied, the value of path will not be automatically prepended to the file names. This has to be explicitly dealt with in the importFun function. See details.

name

a character of length 1 with the experiment name.

importFun

a function to read the data files. The default function works for a certain file format, such as that of the example files provided with this package. If your plate reader software produces files with a different format or if you want to import data from a non file-based source, the import function needs to be adapted. See details and examples.

dec

Optional argument that is passed to importFun, and can be used to accommodate for data files that use characters different from . to represent the decimal point (e.g., the comma ,).

verbose

a logical value, if TRUE, the function reports some of its intermediate progress.

... additional arguments that are being passed on to filename if it is a function.
The plate list is expected to be a tab-delimited file with at least three columns named `Filename`, `Plate`, and `Replicate`. The contents of the columns `Plate` and `Replicate` are expected to be integers. `Filename` should be a vector of file names of the respective raw data files. If the `path` argument is supplied and `importFun` is missing, its value will automatically be prepended to the file names. The optional `Batch` column can be used to supply batch information for an experiment, e.g., when a reagent has been changed or the experiment has been run over several days.

Further columns are allowed, and can be used to denote, for example, different variants of the assay, treatments, incubation times, conditions, etc.

Alternatively, the value of `filename` can be a user-defined function which creates a `data.frame` similar to the one described before. This is for instance useful if the plate list information is directly imported from a data base. In order to allow for non-elementary data types, the output of the function can also be a named list, where each element has to be a vector of equal length. The aforementioned type restrictions still apply. The function will be called with all additional ... arguments, which allows to pass in additional information like experiment identifiers or data base queries.

`importFun` can be used to define a custom function to import data files. The `importFun` function should accept as its first argument names from the `Filename` column of the plate list (which in principle do not need to be individual files, they could also be handles for database entries, queries, or pointers into relevant parts of a file). As its second argument, the function should accept the value of the `path` argument, and the user needs to explicitly prepend this to the file names if needed. It should return a list with two components:

- The first component should be a `data.frame` with the following columns
  - `well`, a character vector with the well identifier in the plate.
  - `val`, the intensity values measured in that well.
  and with as many rows as there are wells in the plate.

- The second component should be a character vector containing a copy of the imported input data file (such as the output of `readLines`). It should be suitable to be used as input for `writeLines`, since it will be used to reproduce the intensity files that are linked in the HTML quality reports generated by `writeReport`.

For example, to import plate data files from EnVision plate reader, set `importFun=getEnVisionRawData` or `importFun=getEnvisionCrosstalkCorrectedData`. See function `getEnVisionRawData`.

Direct data base import without the need for any flat files at all could for instance be achieved by:

- Providing a user-defined function as the `filename` argument and an experiment identifier as an additional ... argument. The function would have to query the data base for the plate information using this identifier and return a table as described above, where the `Filename` column contains identifiers needed to fetch the respective raw data for a single plate in a subsequent query. Alternatively, this could be a data base handle, or the query itself.

- Providing a second user-defined function as the `importFun` argument, which takes the value of the `Filename` column for a single plate and retrieves the respective raw data from the data base.

Value

An object of class `cellHTS`, which extends the class `NChannelSet`.

After calling this function, the content of the following slots is as follows:
assayData an object of class AssayData containing the imported measurement data. Each matrix represents a single channel, and each run corresponds to a column. Thus, the total number of rows in each matrix corresponds to the product between the number of wells per plate and the number of assay plates.

phenoData information about the runs, inferred from the plateList file: which replicate, which variant of the assay, treatment, incubation times etc.

featureData the information about the plate and well identifiers for each plate measurement are stored in columns plate and well of this slot.

plateList a data.frame containing what was read from input file x, plus a column status of type character: it contains the string "OK" if the data import appeared to have gone well, and the respective error or warning message otherwise.

intensityFiles a list, where each component contains a copy of the imported input data files. Its length corresponds to the number of rows of plateList.

Author(s)
W. Huber <huber@ebi.ac.uk>, Ligia Bras <ligia@ebi.ac.uk>, Florian Hahne <florian.hahne@novartis.com>

References

See Also
To read input files obtained in a HTanalyser plate reader, see readHTAnalystData. To build a cellHTS2 object from a data frame, see buildCellHTS2.

Examples

datadir <- system.file("KcViabSmall", package = "cellHTS2")
x <- readPlateList("Platelist.txt", "KcViabSmall", path=datadir)

## To read data files obtained from an EnVision plate reader:
datadir <- system.file("EnVisionExample", package = "cellHTS2")
x <- readPlateList("platelist.txt", name="EnVisionEx", importFun=getEnVisionRawData, path=datadir)

## to get the cross talk corrected data:
y <- readPlateList("platelist.txt", name="EnVisionEx", importFun=getEnVisionCrosstalkCorrectedData, path=datadir)

---

ROC Creates an object of class "ROC" which can be plotted as a ROC curve

Description
The function ROC construct an object of S4 class ROC, which represents a receiver-operator-characteristic curve, from the data of the annotated positive and negative controls in a scored cellHTS object.
Usage

```r
## S4 method for signature 'cellHTS'
ROC(object, positives, negatives)
## S4 method for signature 'ROC,missing'
plot(x, col="darkblue", type="l", main = "ROC curve", ...)
## S4 method for signature 'ROC'
lines(x, ...)
```

Arguments

- `object`: a `cellHTS` object which replicate data have already been scored and summarized (see details).
- `positives`: a list or vector of regular expressions specifying the name of the positive control(s). See the details for the argument `posControls` of `writeReport` function. The default is `"^pos$"`.
- `negatives`: a vector of regular expressions specifying the name of the negative control(s). See the details for the argument `negControls` of `writeReport` function. The default is `"^neg$"`.
- `x`: a `ROC` object obtained using function `ROC`.
- `col`: the graphical parameter for color; see `par` for details.
- `type`: the graphical parameter giving the type of plot desired; see `par` for details.
- `main`: the graphical parameter giving the desired title of plot; see `par` for details.
- `...`: other graphical parameters as in `par` may be also passed as arguments.

Details

The `cellHTS` object `object` must be already scored (`state(object)["scored"]=TRUE`), and selection proceeds from large to small values of this single per-probe score. Furthermore, `object` is expected to contain positive and negative controls annotated in the column `controlStatus` of the `featureData` slot - which can be accessed via `wellAnno(object)`. The arguments `positives` and `negatives` should be given as regular expression patterns specifying the name of the positive(s) and negative(s) controls, respectively. By default, if `positives` is not given, `pos` will be taken as the name for the wells containing positive controls. Similarly, if `negatives` is missing, by default `neg` will be considered as the name used to annotate the negative controls. The content of `posControls` and `negControls` are passed to `regexpr` for pattern matching within the well annotation (see examples for `summarizeChannels`). If the assay is a two-way experiment, `positives` should be a list with components `act` and `inh`, specifying the name of the activators, and inhibitors, respectively. In this case, the ROC curve is constructed based on the absolute values of `Data(object)`.

Value

An S4 object of class `ROC`. There are methods `show`, `plot` and `lines`.

Author(s)

Ligia P. Bras <ligia@ebi.ac.uk>
ROC-class

Class that contain data that can be plotted as a ROC curve.

Description

Container for data that represent a receiver-operator-characteristic curve, and that were generated from the data of the annotated positive and negative controls in a scored cellHTS object.

Creating Objects

\[
\text{new("ROC")}
\]

ROC(object, positives, negatives) with object being an cellHTS instance.

Slots

- name: a character of length 1 with the name of the experiment from which the ROC object derives.
- assayType: a character of length 1 with the type of screening assay. Possible values are: "one-way assay" and "two-way assay".
- TP: a vector of integers of length 1000.
- FP: a vector of integers of length 1000.
- posNames: a character vector with the name of the positive controls.
- negNames: a character vector with the name of the negative controls.

Methods

- show: Print a summary of the object.
- plot: Plot the ROC curve corresponding to the object.
- lines: Line plot of the ROC object.

Author(s)

Ligia Bras <ligia@ebi.ac.uk>, Wolfgang Huber <huber@ebi.ac.uk>

See Also

ROC
Examples

showClass("ROC")
showMethods(class="ROC")

## Not run:
data(KcViabSmall)
x <- normalizePlates(KcViabSmall, scale="multiplicative", log=FALSE, method="median", varianceAdjust="none")
x <- scoreReplicates(x, sign="-", method="zscore")
x <- summarizeReplicates(x, summary="mean")
y <- ROC(x)
plot(y)
lines(y, col="green")
show(y)

## End(Not run)

rsa

Perform RSA ranking on the screening results.

Description

The RSA method ranks the resulting hit list of a screening experiment, taking into account the
design of the screening library (i.e., multiple probes targeting the same effector molecule).

Usage

rsa(x, geneColumn="GeneID", lowerBound=0, upperBound=1, reverse=FALSE, drop=FALSE)

Arguments

x

type: Object derived from class cellHTS.

geneColumn

type: The name of the well annotation column to be used for the grouping of effector

types: molecules and probes.

lowerBound

type: The lower boundary parameter for the RSA algorithm.

upperBound

type: The upper boundary parameter for the RSA algorithm.

reverse

type: Boolean. Reverse the ranking.

drop

type: Boolean. Drop all probes from the analysis for which no effector molecule is
defined.

Details

The input argument x has to be a cellHTS2 object which has been scored, summarized and anno-
tated. For details on the RSA algorithm please see the publication referenced below.

Value

A data.frame with the following columns:

Value of argument geneColumn:
the target molecule identifier.

Plate:
the plate identifier.
**scoreReplicates**

- **Well**: the well identifier.
- **Score**: the probe score in the screen.
- **RSA Rank**: the computed RSA rank.
- **Score Rank**: the rank based on a simple cutoff scheme.
- **PValue**: the computed RSA $p$-value.
- **RSA Hit**: the RSA hit flag.
- **#HitWell**: the number of probes counted as positive RSA hits for a given target molecule.
- **#TotalWell**: the total number of probes for a given target molecule.
- **%HitWell**: the percentage of positive hits for a given molecule.

**Author(s)**

Florian Hahne <florian.hahne@novartis.com>

**References**

Renate Koenig, Chih-yuan Chiang, Buu P Tu, S Frank Yan, Paul D DeJesus, Angelica Romero, Tobias Bergauer, Anthony Orth, Ute Krueger, Yingyao Zhou & Sumit K Chanda: A probability-based approach for the analysis of large-scale RNAi screens

*NATURE METHODS | VOL.4 NO.10 | OCTOBER 2007 | 847*

**See Also**

cellHTS

**Examples**

```r
data(KcViabSmall)
KcViabSmall <- scoreReplicates(KcViabSmall, sign="-", method="zscore")
KcViabSmall <- summarizeReplicates(KcViabSmall, summary="mean")
ranking <- rsa(KcViabSmall)
head(ranking)
```

**scoreReplicates**

_Scores normalized replicate values given in a cellHTS object_

**Description**

This function scores the normalized replicate values stored in slot assayData of a cellHTS object. Current available options are to take the z-score value or the per-replicate normalized percent inhibition (NPI). Data are stored in slot assayData overridding its current content.

**Usage**

```r
scoreReplicates(object, sign="+", method="zscore", ...)
```
Arguments

object an object of class cellHTS that has already been normalized.

sign a character string, either "+" (default) or "-", which corresponds to multiplying the data by +1 or -1, respectively, after applying the scoring method specified by argument method. See details.

method a character string indicating which method to use to score the replicate measurements. Available options are "none", "zscore" (default), "NPI". See details.

... additional parameters required by some of the methods chosen in method.

Details

This function scores the normalized values given in the slot assayData of object. Current available scoring methods are:

- method="none", no scoring is applied.
- method="zscore" (robust z-scores), for each replicate, this is calculated by subtracting the overall median from each measurement and dividing the result by the overall mad. These are estimated for each replicate by considering the distribution of intensities (over all plates) in the wells whose content is annotated as sample.
- method="NPI" (normalized percent inhibition applied in a per-replicate basis, i.e. using the overall mean of positive and negative controls across all plates of a given replicate), for each replicate, this method consists of subtracting each measurement from the average of the intensities on the positive controls (taken across all plates), and this result is then divided by the difference between the averages of the measurements on the positive and the negative controls (taken across all plates). If this method is chosen, one may need to provide further arguments to scoreReplicates, namely, arguments posControls and negControls. These arguments should be vectors of regular expression patterns specifying the name of the positive(s) and negative(s) controls, respectively, as provided in the plate configuration file. The length of these vectors should match the current number of channels in object (i.e. dim(Data(object))[3]). By default, if posControls or negControls are not given, pos and neg will be taken as the name for the wells containing positive or negative controls. The content of posControls and negControls is passed to regexpr for pattern matching within the well annotation given in the featureData slot of object (which can be accessed via wellAnno(object)) (see examples for summarizeChannels).

After replicate scoring using the chosen method, the value given in sign ("+") or "-" is used to set the sign of the calculated scores. For example, with a sign="-", a strong decrease in the signal will be represented by a positive score, whereas setting sign="+", such a phenotype will be represented by a negative score. This option can be set to calculate the results to the commonly used convention.

Value

A cellHTS object with its slot assayData replaced with the scored values (same dimension).

Important: Note that the processing state "scored" of the cellHTS object is only updated to TRUE after summarizing the replicates, which is the next preprocessing step (see summarizeReplicates).

Author(s)

W. Huber <huber@ebi.ac.uk>, Ligia Braz <ligia@ebi.ac.uk>
scores2calls

References

See Also
normalizePlates, summarizeChannels, summarizeReplicates.

Examples

data(KcViabSmall)
x <- normalizePlates(KcViabSmall, scale="multiplicative", method="median", varianceAdjust="none")
x <- scoreReplicates(x, sign="-", method="zscore")
x <- summarizeReplicates(x, summary="min") # conservative approach

scores2calls

Sigmoidal transformation of the score values stored in a cellHTS object obtaining the call values for each probe.

Description
Apply a sigmoidal transformation with parameters z0 and lambda to the summarized scored values stored in a cellHTS object. The obtained results are called calls and are stored in slot assayData, overriding its current content.

Currently this function is implemented only for single-color data.

Usage
scores2calls(x, z0, lambda)

Arguments
x an object of class cellHTS containing replicate data that have already been scored and summarized (see details).
z0 a numeric value giving the centre of the sigmoidal transformation. See details.
lambda a numeric value (>0) that corresponds to the parameter lambda of the sigmoidal transformation. This value should be >0, but usually it makes more sense to use a value >=1. See details.

Details
This function applies a sigmoidal transformation with parameters z0 and lambda to the single per-probe score values stored in a cellHTS object. The obtained results are called calls. The transformation is given by:

\[
\frac{1}{1 + e^{-(\lambda \times (z - z_0))}}
\]

where \( z \) are the score values, \( z_0 \) is the centre of the sigmoidal transformation, and the \( \lambda \) is a parameter that controls the smoothness of the transformation. The higher is \( \lambda \), more steeper is the transition from lower to higher values. \( \lambda \) should be > 0, but usually it makes more sense to use a value >=1.
This transformation maps the score values to the interval \([0,1]\), and is intended to expand the scale of scores with intermediate values and shrink the ones showing extreme values, therefore making the difference between intermediate phenotypes larger.

**Value**

The `cellHTS` object with the call values stored in slot `assayData`. This is an object of class `assayData` corresponding to a single matrix of dimensions `Features x 1`.

**Author(s)**

W. Huber <huber@ebi.ac.uk>, Ligia Braz <ligia@ebi.ac.uk>

**References**


**See Also**

`normalizePlates, summarizeChannels, scoreReplicates, summarizeReplicates, imageScreen`.

**Examples**

```r
data(KcViabSmall)
x <- normalizePlates(KcViabSmall, scale="multiplicative", method="median", varianceAdjust="none")
x <- scoreReplicates(x, sign="-", method="zscore")
x <- summarizeReplicates(x, summary="min")
xc <- scores2calls(x, z0=1.5, lambda=2)
plot(Data(x), Data(xc), col="blue", xlab="z-scores", ylab="calls", main=expression(1/(1+e^{-lambda *(z-z[0])})))
if(require(splots)) {
  sp = split(Data(xc), plate(xc))
  plotScreen(sp, zrange=c(0,1), fill=c("white", "red"), na.fill="yellow", main="Calls", ncol=3L)
}
```

---

**settings**

*cellHTS2 HTML report settings*

**Description**

Functions to control the output of `writeReport` through session-wide or call-specific settings.

**Usage**

```r
setSettings(x)
getSettings()
```

**Arguments**

- `x` A named list of settings. See details for supported values.
Details

The `writeReport` function produces a complete audit trail of the analysis in the form of an HTML report. The content of this report is highly customizable though session-wide and also though call-specific settings. The former are supposed to be set using the `setSettings` function by providing a named nested list of values. The latter can be set by passing a similar list on to `writeHTML` as the optional `settings` argument. The current values for all available settings can be queried using the `getSettings` function.

Similar to the structure of the HTML report, the available settings are broken up into subsections, and the names of these subsection have to be matched by the names in the nested list structure:

**plateList** The settings for all the plots in the plate list section of the report. There are several sub-section:

- **correlation** The settings for the correlation plots:
  - `size` The width in inches of the pdf device holding the plot. The default value is 7.5.
  - `fontSize` The point size of the font for all the text in the pdf version of the plot. The default value is 14.
  - `font` The font used for all the text in the pdf version of the plot. The default is "Helvetica".
  - `thumbFactor` The factor by which the thumbnail png version of the plot is smaller compared to the high-resolution pdf version. The default value is 1.5.
  - `thumbFontSize` The point size of the font for all the text in the thumbnail png version of the plot. The default value is 12.

- **maplot** The settings for the correlation plots:
  - `size` The width in inches of the pdf device holding the plot. The default value is 7.5.
  - `fontSize` The point size of the font for all the text in the pdf version of the plot. The default value is 14.
  - `font` The font used for all the text in the pdf version of the plot. The default is "Helvetica".
  - `thumbFactor` The factor by which the thumbnail png version of the plot is smaller compared to the high-resolution pdf version. The default value is 1.5.
  - `thumbFontSize` The point size of the font for all the text in the thumbnail png version of the plot. The default value is 12.

- **histograms** The settings for the histogram plots:
  - `size` The width in inches of the pdf device holding the plot. The default value is 8.
  - `fontSize` The point size of the font for all the text in the pdf version of the plot. The default value is 14.
  - `font` The font used for all the text in the pdf version of the plot. The default is "Helvetica".
  - `thumbFactor` The factor by which the thumbnail png version of the plot is smaller compared to the high-resolution pdf version. The default value is 2.
  - `thumbFontSize` The point size of the font for all the text in the thumbnail png version of the plot. The default value is 10.
  - `type` The type of plot produced here. One in `histogram` or `density`. Both plot types have pros and cons: histograms can be misleading because of bin size artefacts, whereas density plots hide the sample size information.

- **reproducibility** The settings for the reproducibility plate plots:
  - `size` The width in inches of the pdf device holding the plot. The default value is 8.
  - `fontSize` The point size of the font for all the text in the pdf version of the plot. The default value is 12.
  - `font` The font used for all the text in the pdf version of the plot. The default is "Helvetica".
thumbFactor The factor by which the thumbnail png version of the plot is smaller compared to the high-resolution pdf version. The default value is 1.3.

thumbFontSize The point size of the font for all the text in the thumbnail png version of the plot. The default value is 10.

col The color range that gets passed on to plotPlate as argument col. The default value is brewer.pal(9, "YlOrRd").

range The data range that gets passed on to plotPlate as argument xrange. The default value is function(x) c(0, quantile(x, 0.95, na.rm=TRUE)).

include A logical indicating whether to create the plate plot or not. The default value is FALSE.

map A logical indicating whether to tooltips containing the plate annotation for each well or not. The default value is FALSE.

average The settings for the reproducibility plate plots:

size The width in inches of the pdf device holding the plot. The default value is 8.

fontSize The point size of the font for all the text in the pdf version of the plot. The default value is 12.

font The font used for all the text in the pdf version of the plot. The default is "Helvetica".

thumbFactor The factor by which the thumbnail png version of the plot is smaller compared to the high-resolution pdf version. The default value is 1.3.

thumbFontSize The point size of the font for all the text in the thumbnail png version of the plot. The default value is 10.

col The color range that gets passed on to plotPlate as argument col. The default value is brewer.pal(9, "YlOrRd").

range The data range that gets passed on to plotPlate as argument xrange. The default value is function(x) c(0, quantile(x, 0.95, na.rm=TRUE)).

include A logical indicating whether to create the plate plot or not. The default value is FALSE.

map A logical indicating whether to tooltips containing the plate annotation for each well or not. The default value is FALSE.

intensities The settings for the raw data plate plots:

size The width in inches of the pdf device holding the plot. The default value is 8.

fontSize The point size of the font for all the text in the pdf version of the plot. The default value is 12.

font The font used for all the text in the pdf version of the plot. The default is "Helvetica".

thumbFactor The factor by which the thumbnail png version of the plot is smaller compared to the high-resolution pdf version. The default value is 1.6.

thumbFontSize The point size of the font for all the text in the thumbnail png version of the plot. The default value is 10.

col The color range that gets passed on to plotPlate as argument col. The default value is rev(brewer.pal(9, "RdBu")).

range The data range that gets passed on to plotPlate as argument xrange. The default value is function(x) quantile(x, c(0.025, 0.975), na.rm = TRUE). A useful alternative setting here would be function(x) c(-1,1) * max(abs(x), na.rm=TRUE), which forces a value of 0 to be white.

include A logical indicating whether to create the plate plot or not. The default value is FALSE.

map A logical indicating whether to tooltips containing the plate annotation for each well or not. The default value is FALSE.
settings

plateConfiguration This controls settings for the plate configuration part of the report. Available settings are:

- **size** The width in inches of the pdf device holding the plot. The default value is 14.
- **fontSize** The point size of the font for all the text in the pdf version of the plot. The default value is 12.
- **font** The font used for all the text in the pdf version of the plot. The default is "Helvetica".
- **thumbFactor** The factor by which the thumbnail png version of the plot is smaller compared to the high-resolution pdf version. The default value is 2.
- **thumbFontSize** The point size of the font for all the text in the thumbnail png version of the plot. The default value is 9.
- **include** A logical indicating whether to create the plate configuration plot or not. The default value is TRUE.

plateSummaries This controls settings for the plate summary plots in the report. There are two sub-sections:

- **boxplot size** The width in inches of the pdf device holding the plot. The default value is 7.5.
  - **fontSize** The point size of the font for all the text in the pdf version of the plot. The default value is 12.
  - **font** The font used for all the text in the pdf version of the plot. The default is "Helvetica".
  - **thumbFactor** The factor by which the thumbnail png version of the plot is smaller compared to the high-resolution pdf version. The default value is 1.5.
  - **thumbFontSize** The point size of the font for all the text in the thumbnail png version of the plot. The default value is 11.
  - **col** The colors used to fill the boxes. A vector of length two, where the first item specifies the color for the raw data panel and the second item specifies the color for the normalized data panel. The default value is c("pink", "lightblue").

- **controls size** The width in inches of the pdf device holding the plot. The default value is 7.5.
  - **fontSize** The point size of the font for all the text in the pdf version of the plot. The default value is 12.
  - **font** The font used for all the text in the pdf version of the plot. The default is "Helvetica".
  - **thumbFactor** The factor by which the thumbnail png version of the plot is smaller compared to the high-resolution pdf version. The default value is 1.5.
  - **thumbFontSize** The point size of the font for all the text in the thumbnail png version of the plot. The default value is 11.

screenSummary The settings for all the plots in the screen summary section of the report. There are several sub-sections:

- **scores** The settings for summary screen image plots:
  - **size** The width in inches of the pdf device holding the plot. The default value is 7.
  - **fontSize** The point size of the font for all the text in the pdf version of the plot. The default value is 10.
  - **font** The font used for all the text in the pdf version of the plot. The default is "Helvetica".
  - **thumbFactor** The factor by which the thumbnail png version of the plot is smaller compared to the high-resolution pdf version. The default value is 1.
  - **thumbFontSize** The point size of the font for all the text in the thumbnail png version of the plot. The default value is 9.
  - **col** The color range used to map the data values into. This has to be a named list of length two, with the first item posNeg being the color range used when there are both positive and negative values, and the second item being the color range
for positive values only. One usually wants the former to be centered on white to blend into the background, and the former to start from white. The default value is `list(posNeg=rev(brewer.pal(11, "RdBu"))[c(1:5, rep(6,3), 7:11)], pos=brewer.pal(9, "Greys"))`.

**aspect** The aspect ratio of the plot. The default value is 1.

**annotation** An alternative character vector of annotation mappings. See `imageScreen` for details.

**map** A logical indicating whether to tooltips containing the plate annotation for each well or not. The default value is `FALSE`.

**range** The range of values into which the colors will be mapped. A numeric of length 2.

**nbImageBins** The number of color bins used in the map. Default is 256.

**nbLegendBins** The number of color bins shown in the legend. Default is 7.

**qqplot** The settings for the Normal Q-Q plots:

- **size** The width in inches of the pdf device holding the plot. The default value is 7.
- **fontSize** The point size of the font for all the text in the pdf version of the plot. The default value is 10.
- **font** The font used for all the text in the pdf version of the plot. The default is "Helvetica".
- **thumbFactor** The factor by which the thumbnail png version of the plot is smaller compared to the high-resolution pdf version. The default value is 1.
- **thumbFontSize** The point size of the font for all the text in the thumbnail png version of the plot. The default value is 9.

**distribution** The settings for the density distribution plot:

- **size** The width in inches of the pdf device holding the plot. The default value is 7.
- **fontSize** The point size of the font for all the text in the pdf version of the plot. The default value is 10.
- **font** The font used for all the text in the pdf version of the plot. The default is "Helvetica".
- **thumbFactor** The factor by which the thumbnail png version of the plot is smaller compared to the high-resolution pdf version. The default value is 1.
- **thumbFontSize** The point size of the font for all the text in the thumbnail png version of the plot. The default value is 9.

**screenResults** The settings for the screen results panel:

- **keepFieldPattern** A regular expression indicating which column names to keep. Default is "^plate$|well$|score$|wellAnno$|finalWellAnno$|raw_normalized_|GeneID|GeneSymbol".
- **htmlMaxItems** The maximal number of cells in the output table to produce an HTML table. Default is 20000.
- **htmlLinks** A data frame containing a `plate` column, a `well` column, followed by the columns containing the HTML hyper links to add to the output table. Default is `NULL`, no links.

**controls** The settings for the annotation of control wells and other type of wells. Currently these are only two item:

- **col** A named vector of colors to be used across all plots for the annotation of the well types. Allowed values are: sample, neg, controls, other, empty, flagged, act, inh and pos. Additional user-defined well types that don’t fall in any of these groups will be assigned a color from the palette defined in `otherCol`.
- **col** A color palette from which colors are drawn for well types not defined in `col`.

**controls** Global report settings:

- **ppi** A numeric indicating the resolution of the screen. This parameter is used to generate PNG images. Large numbers produce large PNG images. Default is 72.
The current settings for `getSettings`. `setSettings` is called for its side effect of setting session-wide settings.

**Author(s)**
Florian Hahne

**See Also**
`writeReport`

**Examples**
```
oset <- getSettings()
oset

setSettings(list(plateConfiguration=list(size=2),
list(plateList=list(intensities=list(include=FALSE)))))

getSettings()

setSettings(oset)
```

---

**spatialNormalization  Spatial normalization**

**Description**
Adjust spatial plate effects. This function works on the data stored in the slot `assayData` of a `cellHTS` object by fitting a bivariate function within each plate using local regression (robust local fit) with second degree polynomials. Only wells containing "sample" are considered for the parameter fitting, but adjusted data for all wells are returned.

**Usage**
```
spatialNormalization(object, save.model=FALSE, ...)
```

**Arguments**
- `object` a `cellHTS` object that has already been configured.
- `...` Parameters that get passed on to the `lp` function of `locfit`. Most relevant are `nn` and `h`.
- `save.model` a logical value specifying whether the values of the fitted adjustment functions should be returned in the slot `rowcol.effects` of the returned object.
Details

This function is typically not called directly, but rather indirectly from `normalizePlates` function. The normalization is performed separately for each replicate and channel.

Value

An object of class `cellHTS` with normalized data stored in slot `assayData`. Furthermore, if `save.model=TRUE`, it will contain a slot `rowcol.effects`, a 3D array with the same dimension as `Data(object)`.

Moreover, the processing status of the `cellHTS` object is updated in the slot `state` to `object@state["normalized"]`=TRUE.

Author(s)

Ligia Bras <ligia@ebi.ac.uk>

See Also

`medpolish, locfit, plotSpatialEffects, normalizePlates, summarizeChannels, plateEffects`

Examples

data(KcViabSmall)
x <- KcViabSmall
xs <- spatialNormalization(x, save.model = TRUE, h=3)

## Calling spatialNormalization function from "normalizePlates":
xopt <- normalizePlates(x, varianceAdjust="none", save.model = TRUE)
all(xs@rowcol.effects == xopt@rowcol.effects, na.rm=TRUE)
**summarizeChannels**

**See Also**

*cellHTS*

---

**Description**

Combines plate intensities (raw or already corrected in a per-plate fashion) from multi-channel data stored in slot `assayData` of a *cellHTS* instance by applying the function defined in `fun`.

**Usage**

```r
summarizeChannels(object, 
    fun = function(r1, r2, thresh=-Inf) ifelse(r1>thresh, r2/r1, as.numeric(NA)))
```

**Arguments**

- `object` an object of class *cellHTS* that has been configured. See details.
- `fun` a user-defined function for the multi channel summarization. `fun` takes as many numeric vectors as there are channels, names `r1`, `r2`, etc., and returns a single numeric vector of the same length. The default is to take the ratio between the second and first channels, with a threshold on `r1` shown above in the Usage section that should be set by the user.

**Details**

For each plate and replicate of a multi-color experiment, the function defined in `fun` is applied to relate the intensity values in the respective channels of the *cellHTS* object. The default is to take the ratio between the second and first channels, with a threshold on `r1` (see the Usage section). This threshold should be adjusted by the user according to the data. For an example, see the Examples section. This function uses the content of slot `assayData` of `object` and can be applied either to raw data or after per-plate correction of the intensity values in each channel using function `normalizePlates`. This choice depends on channel summarization method that one intends to apply (i.e., the function given by argument `fun`).

**Value**

An object of class *cellHTS* with the summarized multi-channel intensities stored in slot `assayData`. This is an object of class `assayData` containing one matrix with the summarized channel data (dimensions nrFeatures x nrSamples).

**Author(s)**

Ligia Bras <ligia@ebi.ac.uk>, Wolfgang Huber <huber@ebi.ac.uk>

**References**

summarizeChannels

See Also
normalizePlates, scoreReplicates, summarizeReplicates.

Examples

data(dualCh)
x <- dualCh
table(wellAnno(x))

## Define the controls for the different channels:
negControls=vector("character", length=dim(Data(x))[3])

## channel 1 - gene A
## case-insensitive and match the empty string at the beginning and end of a line (to distinguish between "geneA" and "geneAB", for example, although this is not a problem for the well annotation in this example)
negControls[1]="(?i)^geneA$"

## channel 2 - gene A and geneB
negControls[2]="(?i)^geneA$|^geneB$"

posControls = vector("character", length=dim(Data(x))[3])

## channel 1 - no controls
## channel 2 - geneC and geneD
posControls[2]="(?i)^geneC$|^geneD\$"

## Not run:
writeReport(cellHTSlist=list("raw"=x), map=TRUE, plotPlateArgs=TRUE, posControls=posControls, negControls=negControls)

## In this example, we first normalize each channel separately by
## plate median scaling (no variance adjustment), since we need to make the measurements
## comparable across plates for the next step of channel summarization:
xn = normalizePlates(x, scale="multiplicative", log=FALSE, method="median", varianceAdjust="none")

## Then, we define a low intensity threshold for the measurements in the constitutive channel R1,
## which will be set to the 5% quantile of the overall plate median corrected intensities in R1,
## and take the ratio R2/R1.
xn = summarizeChannels(xn, fun = function(r1, r2, thresh=quantile(r1, probs=0.05, na.rm=TRUE)) ifelse(r1>thresh, r2/r1, as.numeric(NA)))

## After channel summarization, we take the log2 and apply plate median normalization,
## and opt to not adjust the variance:
xn = normalizePlates(xn, scale="multiplicative", log=TRUE, method="median", varianceAdjust="none")

## Define the controls for the normalized and summarized intensities (only one channel):
negControls = vector("character", length=dim(Data(xn))[3])

## For the single channel, the negative controls are geneA and geneB
negControls[1]="(?i)^geneA$|^geneB\$"
posControls = vector("character", length=dim(Data(xn))[3])

## For the single channel, the negative controls are geneC and geneD
posControls[1]="(?i)^geneC$|^geneD\$"

## Not run:
writeReport(cellHTSlist=list("raw"=x, "normalized"=xn), force=TRUE, map=TRUE, plotPlateArgs=list(xrange=...

## Another option could be to just take the log2 of the ratio between R2 and R1 raw intensities:
xn1 = summarizeChannels(x, fun = function(r1, r2) log2(r2/r1))
summarizeReplicates

summarizeReplicates

Summary between scored replicate values given in a cellHTS object to obtain a single value for each probe

Description

This function summarizes the replicate values stored in slot assayData of a cellHTS object and calculates a single score for each probe. Data are stored in slot assayData overriding its current content.

This function is implemented for single- and multi-channel data.

Usage

summarizeReplicates(object, summary = "min")

Arguments

object an object of class cellHTS that has already been normalized and scored (see details).

summary a character string indicating how to summarize between replicate measurements. One of "min" (default), "mean", "median", "max", "rms", "closestToZero", or "FurthestFromZero" can be used (see details).

Details

A single value per probe is calculated by summarizing between scored replicates stored in the slot assayData of object. The summary is performed as follows:

- If summary = "mean", the average of replicate values is considered;
- If summary = "median", the median of replicate values is considered;
- If summary = "max", the maximum of replicate intensities is taken;
- If summary = "min", the minimum is considered, instead;
- If summary = "rms", the square root of the mean squared value of the replicates (root mean square) is taken as a summary function;
- If summary = "closestToZero", the value closest to zero is taken as a summary (useful when both sides of the distribution of z-score values are of interest);
- If summary = "furthestFromZero", the value furthest from zero is taken as a summary (useful when both sides of the distribution of z-score values are of interest).

Value

The cellHTS object with the summarized scored values stored in slot assayData. This is an object of class assayData corresponding to matrices of dimensions Features x 1 for each channel. Moreover, the processing status of the cellHTS object is updated in the slot state to object@state["scored"] = TRUE.

Author(s)

W. Huber <huber@ebi.ac.uk>, Ligia Bras <ligia@ebi.ac.uk>
**References**


**See Also**

normalizePlates, summarizeChannels, scoreReplicates, imageScreen.

**Examples**

```r
data(KcViabSmall)
# normalize
x <- normalizePlates(KcViabSmall, scale="multiplicative", method="median", varianceAdjust="none")
# score the replicates
x <- scoreReplicates(x, sign="-", method="zscore")
# summarize the replicates (conservative approach: take the minimum value between replicates)
x <- summarizeReplicates(x, summary="min")
```

---

**templateDescriptionFile**

*Creates a template description file for an RNAi experiment*

**Description**

This function creates a template description file for an RNAi experiment with default entries compliant with MIAME class and with additional entries specific for a cellHTS object.

**Usage**

```r
templateDescriptionFile(filename="Description.txt", path, force=FALSE)
```

**Arguments**

- `filename` 
  the name of the output file. Default is "Description.txt".
- `path` 
  a character of length 1 indicating the path in which to create the screen description file. By default, it can extract the path from filename.
- `force` 
  a logical value, determines the behaviour of the function if file filename exists. If `force` is TRUE, the function overwrites filename, otherwise it casts an error.

**Details**

This function can be called to generate a template file for the RNAi experiment. This file contains the fields that are compliant with the MIAME class and also additional entries specific for the cellHTS class, which should be edited and completed by the user.

This file, which we call *Screen description file*, is required to configure the cellHTS object via function configure. It is intended to contain general information about the screen, such as its title, its goal, when and how it was performed, which organism, which library, type of assay, references, and any other information that is pertinent to the biological interpretation of the experiments.

**Value**

The function returns a character with the full path and name of the file that was created.
updateCellHTS

Author(s)
Ligia Bras <ligia@ebi.ac.uk>

See Also
configure

Examples
```r
out <- templateDescriptionFile("Description.txt", path=tempdir())
out
readLines(out)
```

updateCellHTS

Update old serialized cellHTS objects.

Description
During the development of the cellHTS2 package, the definition of the cellHTS object has changed. This function can be used to update old serialized cellHTS objects.

Usage
```r
updateCellHTS(object)
```

Arguments

- `object`  The cellHTS object to update.

Value
An updated cellHTS object.

Author(s)
Florian Hahne

See Also
convertOldCellHTS

Examples
```r
data(KcViabSmall)
updateCellHTS(KcViabSmall)
```
wellAnno  Access the annotation from a cellHTS object.

Description

These generic functions access the annotation data stored in the featureData slot of an object of class cellHTS.

Usage

wellAnno(object)
geneAnno(object)

Arguments

object Object derived from class cellHTS.

Value

wellAnno returns a factor of length equal to the total number of features (number of plates x number of wells per plate) indicating the contents of the wells. Corresponds to fData(object)[, “controlStatus”].
geneAnno returns a vector of the same length as the number of features in object (number of plates x number of wells per plate) containing the gene IDs used in the screen. This corresponds to the contents of fData(object)[, “GeneID”].

See cellHTS class for details.

Author(s)

Ligia Bras <ligia@ebi.ac.uk>

References


See Also

cellHTS

write.tabdel  Wrapper to function 'write.table' used to write data to a tab-delimited file

Description

Wrapper for the function write.table to write data to a tab-delimited file.

Usage

write.tabdel(...)
writeReport

Arguments

... arguments that get passed on to the function write.table.

Details

A trivial function, which we have included for convenience.

Value

The name of the file that was written.

Author(s)

Ligia Bras <ligia@ebi.ac.uk>

See Also

write.table

Examples

data(KcViabSmall)
x <- KcViabSmall
## determine the ratio between each well and the plate median
xraw <- Data(x)
y <- array(as.numeric(NA), dim=dim(xraw))
nrWell <- prod(pdim(x))

for(p in 1:max(plate(x))) {
  ind <- (1:nrWell)+nrWell*(p-1)
  samples <- (wellAnno(x)[ind]=="sample")
  y[ind,] <- apply(xraw[ind, ,drop=FALSE], 2:3, function(w) w/median(w[samples], na.rm=TRUE))
} y <- signif(y, 4)
out <- y[,1]
out <- cbind(geneAnno(x), wellAnno(x), out)
colnames(out) <- c("GeneID", "wellAnno", sprintf("Well/Median_r%d_ch%d", rep(1:dim(y)[2], dim(y)[3]), rep(1:dim(y)[3], each=dim(y)[2])))
write.tabdelt(out, file = tempfile())

writeReport

Create a directory with HTML pages of linked tables and plots documenting the contents of a cellHTS experiment

Description

Creates a directory with HTML pages of linked tables and plots documenting the contents of the preprocessing of a cellHTS object.
writeReport(
  raw,
  normalized=NULL,
  scored=NULL,
  cellHTSlist=NULL,
  outdir,
  force=FALSE,
  map=FALSE,
  plotPlateArgs=NULL,
  imageScreenArgs=NULL,
  posControls,
  negControls,
  mainScriptFile=NA,
  gseaModule=NULL,
  settings=list())

Arguments

raw         the initial raw cellHTS object. See details.
normalized  a normalized cellHTS object. See details.
scored      a scored cellHTS object. See details.
cellHTSlist a list of cellHTS objects. See details. Note: this argument is deprecated. Please use the separate arguments raw, normalized and scored instead.
outdir      a character of length 1 with the name of a directory where to write the report HTML file and images. If the directory does not exist, it is created. If it exists and is not empty, then the behaviour depends on the value of force. If outdir is missing, it is set to file.path(getwd(), name(cellHTSlist[["raw"]])). i.e. a directory with the name of the cellHTS object(s) in the current working path.
force       a logical value, determines the behaviour of the function if outdir exists and is not empty. If force is TRUE, the function overwrites (removes and recreates) outdir, otherwise it casts an error.
map         a logical value indicating whether tooltips with the annotation should be added to the plate plots and image screen. Default value is FALSE. NOTE: This argument is deprecated and will go away in the next release. Please see settings to learn how to control the output of writeReport.
plotPlateArgs either a list with parameters for the plate plots of the per plate quality report pages, or a logical scalar with values FALSE or TRUE. If FALSE or NULL, the plate plots are omitted, this option is here because the production of the plate plots takes a long time. See details. NOTE: This argument is deprecated and will go away in the next release. Please see settings to learn how to control the output of writeReport.
isimageScreenArgs a list with parameters for the function imageScreen. See details. NOTE: This argument is deprecated and will go away in the next release. Please see settings to learn how to control the output of writeReport.
posControls   a list or vector of regular expressions specifying the name of the positive controls. See details.
negControls  a vector of regular expressions specifying the name of the negative controls. See details.

mainScriptFile  The path to the R script generating the current report. We strongly advice to store this script in the compendium for future reference.

gseaModule  Add the output of a gene set enrichment analysis to the report. This is totally experimental at this time.

settings  A named list of settings controlling the output of writeReport. Please see settings for details.

Details

The function has to be called with the mandatory argument raw corresponding to an unnormalized cellHTS object (i.e. state(cellHTSlist["raw"])["normalized"] = FALSE). Additional optional arguments are:

- "normalized": a cellHTS object containing normalized data (i.e. state(cellHTSlist["normalized"])["normalized"] and state(cellHTSlist["normalized"])["scored"] = FALSE).
- "scored": a cellHTS object containing data scored data (i.e. state(cellHTSlist["scored"])["scored"] = TRUE).

If this component is available, then cellHTSlist["normalized"] should also be given.

All of the above arguments have to be cellHTS objects containing data from the same experiment, but in different preprocessing stages.

The cellHTS argument is deprecated and should no be used anymore.

The following elements are recognized for plotPlateArgs and passed on to plotPlate: sdcol, the color scheme for the standard deviation plate plot, sdrange, the sd range to which the colors are mapped, xcol, the color scheme for the intensity plate plot, xrange, the intensity range to which the colors are mapped. If an element is not specified, default values are used. Both sdrange and xrange can also be provided as functions, which take the values to be plotted by platePlot as a single argument and has to return a numeric vector of length 2. See its documentation for details.

The following elements are recognized for imageScreenArgs and passed on to imageScreen: ar, aspect ratio, zrange, range, anno, gene annotation for the image map (if map = TRUE).

From now on, all settings controlling the output of writeReport should either be provided through the settings argument, or as session-wide parameters set using setSettings. Please see settings for details.

posControls and negControls should be given as a vector of regular expression patterns specifying the name of the positive(s) and negative(s) controls, respectively, as provided in the plate configuration file (and accessed via wellAnno(objects)).

If the cellHTS object containing normalized data was provided as argument norm, the length of posControls and negControls should be equal to the number of channels in this cellHTS object (dim(Data(cellHTSlist["normalized"]))[3]). Otherwise, the length of these vectors should be equal to the number of channels in the unpreprocessed cellHTS object (i.e., dim(Data(cellHTSlist["raw"]))[3]).

By default, if posControls is not given, "pos" will be taken as the name for the wells containing positive controls. Similarly, if negControls is missing, by default "neg" will be considered as the name used to annotate the negative controls. The content of posControls and negControls will be passed to regexpr for pattern matching within the well annotation given in column controlStatus of the featureData slot of the cellHTS object. If no controls are available for a given channel, use "" or NA for that channel. For example, posControls = c("", "(?i)^diap$") means that channel 1 has no positive controls, while "diap" is the positive control for channel 2.

The arguments posControls and negControls are particularly useful in multi-channel data since the controls might be reporter-specific, or after normalizing multi-channel data.
writeReport

In the case of a two-way assay, where two types of "positive" controls are used in the screen ("activators" and "inhibitors"), `posControls` should be defined as a list with two components (called `act` and `inh`), each of which should be vectors of regular expressions of the same length as the current number of reporters (as explained above).

By default, tooltips doing the mapping between the probe annotation and the plate wells are not added to the plate plots and to the overall screen plot. If any of the `cellHTS` objects in `cellHTSlist` is annotated, the probe annotation uses the information contained either in column `GeneSymbol` or column `GeneID` (if the former is missing) of the `featureData` slot of the annotated `cellHTS` object. Otherwise, the mapping simply uses the well identifiers.

Value

The function is called for its side-effect. It returns a character with the full path and name of the report index file, this is an HTML file which can be read by a web browser.

Author(s)

Florian Hahne <florian.hahne@novartis.com>, Ligia P. Bras <ligia@ebi.ac.uk>, Wolfgang Huber <huber@ebi.ac.uk>, Gabor Bakos

References


See Also

`plotPlate`, `imageScreen`

Examples

data(KcViabSmall)
pCtrls <- c("pos")
nCtrls <- c("neg")
## Not run:
## or for safety reasons (not a problem for the current well annotation, however)
pCtrls <- c("pos$")
nCtrls <- c("neg$")
writeReport(raw=KcViabSmall, posControls=pCtrls, negControls=nCtrls)
## same as
## writeReport(raw=KcViabSmall)
xn <- normalizePlates(KcViabSmall, scale="multiplicative", log=FALSE, method="median")
xsc <- scoreReplicates(xn, sign="-", method="zscore")
xsc <- summarizeReplicates(xsc, summary="min")
## to turn on the tooltips in the plate plots and in the image screen plot:
writeReport(raw=KcViabSmall, normalized=xn, scored=xsc, force=TRUE, map=TRUE, plotPlateArgs = TRUE, imageScreenArgs=list(zrange=c(-4,4)))
## End(Not run)
writeTab

Write the data from a cellHTS object to a tab-delimited file

Description

Write the data from a cellHTS object to a tab-delimited file.

Usage

```r
## S4 method for signature 'cellHTS'
writeTab(object, file=paste(name(object), "txt", sep="."))
```

Arguments

- `object`: a cellHTS object.
- `file`: the name of the output file.

Details

This function is a wrapper for function `write.table` to write the contents of assayData slot of a cellHTS object to a tab-delimited file. If the object is already annotated, the probe information (`fData(object)@GeneID`) is also added.

Value

The name of the file that was written.

Author(s)

Wolfgang Huber <huber@ebi.ac.uk>, Ligia P. Bras <ligia@ebi.ac.uk>

See Also
cellHTS

Examples

```r
data(KcViabSmall)
writeTab(KcViabSmall, file=tempfile())
```
Index

*Topic classes
  cellHTS-class, 8
  ROC-class, 45

*Topic datasets
  bdgpbiomart, 5
dualCh, 20
KcViab, 31
KcViabSmall, 31
oldKcViabSmall, 35

*Topic manip
  batch, 4
  Bscore, 5
  Data, 19
getDynamicRange, 20
getEnVisionRawData, 22
getMeasureRepAgreement, 23
getTopTable, 24
getZfactor, 26
imageScreen, 29
intensityFiles, 30
normalizePlates, 32
oneRowPerId, 36
plate, 36
plateEffects, 37
plotSpatialEffects, 38
readHTAnalystData, 39
readPlateList, 41
rsa, 46
scoreReplicates, 47
scores2calls, 49
spatialNormalization, 55
state, 56
summarizeChannels, 57
summarizeReplicates, 59
templateDescriptionFile, 60
wellAnno, 62
write.tabdel, 62
writeReport, 63
writeTab, 67

*Topic package
  cellHTS2, 11

*Topic univar
  ROC, 43

annotate, 3, 11, 12
annotate, cellHTS-method (annotate), 3
AnnotatedDataFrame, 9
applyByCategory, 28
AssayData, 9, 40, 43

batch, 4, 11, 12
batch, cellHTS-method (batch), 4
batch<-(batch), 4
batch<-, cellHTS, data.frame-method (batch), 4
bdgpbiomart, 5, 11
Bscore, 5, 11, 13, 32, 34, 38
buildCellHTS2, 7, 43
cellHTS, 3–7, 9–17, 19–21, 23–27, 29–32, 34,
36–38, 40, 42–49, 55–57, 59, 60,
62–65, 67
cellHTS (cellHTS-class), 8
cellHTS-class, 8
cellHTS2, 11
cellHTS2-package (cellHTS2), 11
class, cellHTS (cellHTS-class), 8
classVersion, 10
cor, 23
cor, cellHTS-method (cor), 15
convertOldCellHTS, 11, 13, 17, 61
convertWellCoordinates, 13, 18

Data, 11, 12, 19, 30
Data, cellHTS-method (Data), 19
data.frame, 9, 26
Data<-(Data), 19
Data<-, cellHTS, array-method (Data), 19
dualCh, 11, 20
initialize, cellHTS-method
initialize
intensityFiles, cellHTS-method
imageScreen
gseaModule
getZfactor
interactive()
getSettings
getTopTable
getMeasureRepAgreement
getEnVisionRawData
getEnVisionCrosstalkCorrectedData
getEnVisionCrosstalkCorrectedData
(getEnVisionRawData), 22
getEnVisionRawData, 11, 12, 22, 42
getMeasureRepAgreement, 11, 13, 23
getSettings (settings), 50
getTopTable, 11, 13, 24
getzFactor, 11, 13, 26
gseaModule, 28
imageScreen, 11, 13, 29, 50, 54, 60, 64–66
initialize, 11
initialize, cellHTS-method
(cellHTS-class), 8
intensityFiles, 11, 12, 30
intensityFiles, cellHTS-method
(intensityFiles), 30
interactive(), 14, 20, 25, 26, 39

KcViab, 11, 31
KcViabSmall, 11, 31

length, GeneSet-method (gseaModule), 28
lines, ROC-method (ROC), 43
lmFit, 8
locfit, 32, 56
locfit.robust, 7
loess, 7, 13
log2, 32, 33
lp, 55

mad, 48
meanSdPlot, 10, 14
meanSdPlot (cellHTS-class), 8
meanSdPlot, cellHTS-method
(cellHTS-class), 8
medpolish, 7, 56
MIAME, 9, 16, 60

name<-, cellHTS, character-method
(cellHTS-class), 8
nbatch, 4, 11, 13
nbatch (cellHTS-class), 8
nbatch, cellHTS-method (cellHTS-class), 8
NChannelSet, 7–11, 40, 42
normalizePlates, 6, 7, 11, 13, 30, 32, 37, 38, 49, 50, 56–58, 60

oldKcViabSmall, 11, 35
oneRowPerId, 11, 13, 36

par, 44
pdim, 11, 12
pdim (plate), 36
pdim, cellHTS-method (plate), 36
plate, 11, 12, 36
plate, cellHTS-method (plate), 36
plateConf, 11, 12
plateConf (intensityFiles), 30
plateConf, cellHTS-method
(intensityFiles), 30
plateEffects, 7, 11, 12, 37, 56
plateEffects, cellHTS-method
(plateEffects), 37
plateList, 11, 12
plateList (intensityFiles), 30
plateList, cellHTS-method
(intensityFiles), 30

plot, ROC, missing-method (ROC), 43
plotPlate, 38, 52, 65, 66
plotScreen, 14, 15
plotSpatialEffects, 7, 11, 13, 38, 56
position, 11, 12
position (plate), 36
position, cellHTS-method (plate), 36

read.table, 3, 15, 41
readHTAnalystData, 11, 12, 39, 43
readLines, 15, 22, 39, 42
readPlateList, 3, 8, 11, 12, 16, 22, 23, 30, 31, 40, 41

repro, 14, 21, 27, 34, 44, 48, 65
ROC, 10, 11, 13, 43, 44, 45
ROC, cellHTS-method (ROC), 43
ROC-class, 45
rsa, 46

scoreReplicates, 11, 13, 30, 47, 50, 58, 60
scores2calls, 11, 13, 49
screenDesc, 11, 12
screenDesc (intensityFiles), 30
screenDesc, cellHTS-method
(intensityFiles), 30
screenLog, 11, 12
screenLog(intensityFiles), 30
screenLog, cellHTS-method (intensityFiles), 30
setSettings, 65
setSettings(settings), 50
settings, 50, 64, 65
shorth, 33
show, 11
show, cellHTS-method (cellHTS-class), 8
show, ROC-method (ROC-class), 45
spatialNormalization, 11, 13, 33, 34, 38, 55
state, 11, 13, 56
state, cellHTS-method (state), 56
summarizeChannels, 7, 11, 13, 30, 33, 34, 38, 44, 48–50, 56, 57, 60
summarizeReplicates, 11, 13, 30, 48–50, 58, 59
templateDescriptionFile, 11, 13, 16, 60
two-way median polish, 6
updateCellHTS, 17, 61
updateObject, 10
validObject, 11
Versions, 10
well, 11, 13
well (plate), 36
well, cellHTS-method (plate), 36
wellAnno, 11, 13, 62
wellAnno, cellHTS-method (wellAnno), 62
write.tabdel, 11, 14, 62
write.table, 14, 62, 63, 67
writeLines, 22, 42
writeReport, 11, 13, 15, 22, 24, 27, 28, 30, 42, 50, 51, 55, 63
writeTab, 11, 14, 67
writeTab, cellHTS-method (writeTab), 67