HTqPCR

April 19, 2010

`clusterCt` *Clustering of qPCR Ct values*

Description

Hierarchical clustering of samples or genes from high-throuhgput qPCR experiments, such as the TaqMan Low Density Array platform. Individual clusters can be selected, and the features within them listed in the given order.

Usage

```
clusterCt(q, main = NULL, type = "genes", dist = "pearson", xlab = "Cluster dend
```

Arguments

`q`	object of class qPCRset.
`main`	character string, plot title.
`type`	character string, either "genes" (default) or "samples", indicating what is to be clustered.
`dist`	character string, specifying whether to use "pearson" correlation (default) or "euclidean" distance for the clustering.
`xlab`	character string, label for the x-axis.
`n.cluster`	integer, the number of cluster to divide the dendrogram into. See details.
`h.cluster`	numeric, the height at which to cut the dendrogram into clusters. See details.
`select.cluster`	logical, whether to select clusters interactively. See details.
`...`	any other arguments will be passed to the `plot` function.

Details

This function may be used to cluster the Ct values and present the result as a dendrogram.

The `n.cluster` and `h.cluster` parameters are from the `rect.hclust` function and can be used to divide the dendrogram into subclusters based on either number of clusters or height of branch, drawing boxes around subclusters. The members of each cluster can be returned (see value). If `n.cluster` is specified `h.cluster` will be ignored.

If `select.cluster` is chosen individual subclusters can be selected and marked by a box by clicking on their highest comment branch with the (first) mouse buttom. Multiple clusters can be selected until any mouse button other than the first is pressed, and the function can be used in conjunction with either `n.cluster` or `h.cluster`. The members of each cluster will likewise be returned, in the order they were selected.

Value

A plot is created on the current graphics device. If any subclusters are marked, these will be returned invisibly in a list, with one component for each subcluster. The individual slots in the list contain the names og the genes, and their position in the original input data (row number).

Author(s)

Heidi Dvinge

See Also

`hclust`, `dist`, `rect.hclust`, `identify.hclust`

Examples

```
# Load example data
data(qPCRraw)
# Clustering samples
clusterCt(qPCRraw, type="samples")
clusterCt(qPCRraw, type="samples", dist="euclidean")
# Clustering genes
clusterCt(qPCRraw, type="genes", cex=0.5)
clusterCt(qPCRraw, type="genes", h.cluster=1.5, cex=0.5)
cluster.list <- clusterCt(qPCRraw, type="genes", n.cluster=6, cex=0.5)
cluster.list[[1]]
```

`filterCategory` *Filter Ct values based on their feature categories.*

Description

Ct values corresponding to selected feature categories will be replaced by NA. Generally, the feature categories indicate how reliable the values are.

Usage

```
filterCategory(q, na.categories = c("Unreliable", "Undetermined"))
```

Arguments

`q`	a qPCRset object.
`na.categories`	character vector, with the name(s) of the feature categories where Ct values will be considered NA.

Value

A qPCRset object like the input, but with the selected Ct values replaced by NAs

Author(s)

Heidi Dvinge

See Also

`setCategory` for adjusting the categories.

Examples

```
data(qPCRraw)
qPCRraw2 <- setCategory(qPCRraw, groups=NULL)
x <- filterCategory(qPCRraw2)
summary(qPCRraw)
summary(x)
```

`filterCtData` *Filter out features (genes) from qPCR data.*

Description

This function is for filtering Ct data from high-throughput qPCR platforms like the TaqMan Low Density Arrays. This can for example be done prior to analysing the statistical significance of the data, to remove genes where the results are of low quality, or that are not of interest to the analysis in question.

Usage

```
filterCtData(q, remove.type, remove.name, remove.class, remove.category, n.categ
```

Arguments

`q`	object of class qPCRset.
`remove.type`	character vector, the feature type(s) to be removed from the data object.
`remove.name`	character vector, the feature name(s) to be removed from the data object.
`remove.class`	character vector, the feature class(es) to be removed from the data object.
`remove.category`	character vector, the features categories(s) to be assessed across samples.
`n.category`	numeric, all features with more than this number of `remove.category` across samples are removed.
`remove.IQR`	numeric, all features with an interquartile range (IQR) below this limit across samples will be removed.
`verbose`	boolean, should some information be printed to the prompt.

Details

This function may be used to exclude individual or small groups of features that are irrelevant to a given analysis. However, it can also be used on a more general basis, to for example split the data into separate `qPCRset` objects based on features with different characteristics, such as groups of markers or other gene classes present in `featureClass`.

`remove.IQR` can be used to exclude features that show only little variation across the samples. These are unlikely to be differentially expressed, so including them in downstream analysis such as `limmaCtData` or `ttestCtData` would result in a slight loss of power caused by the adjustment of p-values required due to multiple testing across all features.

Value

An object of class qPCRset like the input, but with the required features removed.

Note

After removing features the function `plotCtCard` will no longer work, since the number of features is now smaller than the card dimensions.

When using `remove.category` or `remove.IQR` and there are replicated features present on the array, it might no longer be possible to use the `ndups` parameter of `limmaCtData`, since the number of replicates isn't identical for each feature.

Filtering can be performed either before or after normalization, but in some cases normalization might be affected by this, for example if many features are removed, making it difficult to identify rank-invariant genes.

Author(s)

Heidi Dvinge

Examples

```
# Load some example data
data(qPCRpros)
show(qPCRpros)
# Filter based on different feature type
qFilt <- filterCtData(qPCRpros, remove.type=c("Endogenous Control"))
# Filter based on feature type and name
qFilt <- filterCtData(qPCRpros, remove.type=c("Endogenous Control"), remove.name=c("Gene1
# Filter based on feature class
qFilt <- filterCtData(qPCRpros, remove.class="Kinase")
# Filter based on feature categories, using two different cut-offs
qFilt <- filterCtData(qPCRpros, remove.category="Undetermined")
qFilt <- filterCtData(qPCRpros, remove.category="Undetermined", n.category=5)
# Remove features without much variation across samples
iqr <- apply(exprs(qPCRpros), 1, IQR, na.rm=TRUE)
hist(iqr, n=20)
qFilt <- filterCtData(qPCRpros, remove.IQR=2)
```

`heatmapSig` *Heatmap of deltadeltaCt values from qPCR data.*

Description

Heatmap and clustering of deltadeltaCt values from different sample comparisons using qPCR data.

Usage

```
heatmapSig(qDE, comparison = "all", col, zero.center = TRUE, mar, dist = "pearsc
```

Arguments

`qDE`	data.frame or list, as created by `ttestCtData` or `limmaCtData`.
`comparison`	integers or the names of the comparisons to include in the plot. Deaults to all results in the `qDE` data, but a minimum of two is required.
`col`	colour scheme to use for the plot.
`zero.center`	logical, should the colour scale be centered around 0.
`mar`	vector of length two, the bottom and right side margins mof the heatmap.
`dist`	character string, either "pearson" (default) or "euclidean" indicating what type of distance is used for the clustering.
`...`	further arguments passed to `heatmap.2`.

Details

This function can be useful if multiple conditions are compared, for detecting features with similar behaviour in comparisons, and look at the general level of up and down regulation.

Value

A plot if produced in the current graphics device.

Author(s)

Heidi Dvinge

See Also

`heatmap.2` for modifying the plot, and `ttestCtData` or `limmaCtData` for generating the data used for the plotting.

Examples

```
# Load example preprocessed data
data(qPCRpros)
samples <- read.delim(file.path(system.file("exData", package="HTqPCR"), "files.txt"))
# Define design and contrasts
design <- model.matrix(~0+samples$Treatment)
colnames(design) <- c("Control", "LongStarve", "Starve")
contrasts <- makeContrasts(LongStarve-Control, LongStarve-Starve, Starve-Control, levels=
# Reorder data to get the genes in consecutive rows
```

```
temp <- qPCRpros[order(featureNames(qPCRpros)),]
# The actual test
qDE <- limmaCtData(temp, design=design, contrasts=contrasts, ndups=2, spacing=1)
# Plotting the heatmap
heatmapSig(qDE)
heatmapSig(qDE, dist="euclidean")
```

`HTqPCR-package` *Analysis of High-Throughput qPCR data (HTqPCR)*

Description

This package is for analysing high-throughput qPCR data. Focus is on data from Taqman Low Density Arrays, but any kind of qPCR performed across several samples is applicable. Cycle threshold (Ct) data from different cards (samples) is read in, normalised, processed and the genes are tested for differential expression across different samples. Results are visualised in various ways.

Details

Package:	HTqPCR
Type:	Package
Version:	1.0
Date:	2009-07-03
License:	Artistic
LazyLoad:	yes
Depends:	methods

Author(s)

Maintainer: Heidi Dvinge <heidi@ebi.ac.uk> Maintainer: Paul Bertone <bertone@ebi.ac.uk>

Examples

```
# Locate example data and create qPCRset object
exPath <- system.file("exData", package="HTqPCR")
exFiles <- read.delim(file.path(exPath, "files.txt"))
raw <- readCtData(files=exFiles$File, path=exPath)
# Preprocess
raw.cats <- setCategory(raw, groups=exFiles$Treatment, plot=FALSE)
norm <- normalizeCtData(raw.cats, norm="scale.rank")
# Various plots
plotCtDensity(norm)
plotCtBoxes(norm)
plotCtOverview(norm, groups=exFiles$Treatment, genes=featureNames(raw)[1:10], calibrator=
plotCtCor(norm)
plotCtScatter(norm, cards=c(1,4), col="type")
# Define design and contrasts for testing differential expression
design <- model.matrix(~0+exFiles$Treatment)
colnames(design) <- c("Control", "LongStarve", "Starve")
```

```
contrasts <- makeContrasts(LongStarve-Control, LongStarve-Starve, Starve-Control, levels=
# Reorder by featureNames (2 replicates of each feature) and the actual test
norm2 <- norm[order(featureNames(norm)),]
diff.exp <- limmaCtData(norm2, design=design, contrasts=contrasts, ndups=2, spacing=1)
# Some of the results
names(diff.exp)
diff.exp[["LongStarve - Control"]][1:10,]
diff.exp[["Summary"]][1:10,]
# Some plots of results
plotCtRQ(diff.exp, genes=1:10)
plotCtSignificance(qDE=diff.exp, q=norm2, groups=exFiles$Treatment, calibrator="Control",
plotCtSignificance(qDE=diff.exp, q=norm2, comparison="LongStarve - Starve", groups=exFile
```

`limmaCtData` *Differentially expressed features with qPCR: limma*

Description

Function for detecting differentially expressed genes from high-throughput qPCR Ct values, based on the framework from the `limma` package. Multiple comparisons can be performed, and across more than two groups of samples.

Usage

```
limmaCtData(q, design = NULL, contrasts, sort = TRUE, stringent = TRUE, ndups =
```

Arguments

`q`	object of class qPCRset.
`design`	matrix, design of the experiment rows corresponding to cards and columns to coefficients to be estimated. See details.
`contrasts`	matrix, with columns containing contrasts. See details
`sort`	boolean, should the output be sorted by adjusted p-values.
`stringent`	boolean, for flagging results as "Undetermined". See details.
`ndups`	integer, the number of times each feature is present on the card.
`spacing`	integer, the spacing between duplicate spots, spacing=1 for consecutive spots
`...`	any other arguments are passed to `lmFit`, `contrasts.fit`, `eBayes` or `decideTests`.

Details

This function is a wrapper for the functions `lmFit`, `contrasts.fit` (if a contrast matrix is supplied) and `eBayes` from the limma package. See the help pages for these functions for more information about setting up the design and contrast matrices.

All results are assigned to a category, either "OK" or "Unreliable" depending on the input Ct values. If `stringent=TRUE` any unreliable or undetermined measurements among technical and biological replicates will result in the final result being "Undetermined". For `stringent=FALSE` the result will be "OK" unless at least half of the Ct values for a given gene are unreliable/undetermined.

Note that when there are replicated features in the samples, each feature is assumed to be present the same number of times, and with regular spacing between replicates. Reordering the sample by `featureNames` and setting `spacing=1` is recommendable.

Value

A list of data.frames, one for each column in `design`, or for each comparison in `contrasts` if this matrix is supplied. Each component of the list contains the result of the given comparisons, with one row per gene and has the columns:

`genes`	Feature IDs.
`feature.pos`	The unique feature IDs from `featurePos` of the `q` object. Useful if replicates are not collapsed, in which case there might be several features with identical names.
`t.test`	The result of the t-test.
`p.value`	The corresponding p.values.
`adj.p.value`	P-values after correcting for multiple testing using the Benjamini-Holm method.
`ddCt`	deltadeltaCt values, the log fold changes.
`meanTest`	The average Ct across the test samples for the given comparison.
`meanReference`	The average Ct across the reference samples for the given comparison.
`categoryTest`	The category of the Ct values ("OK", "Undetermined") across the test samples for the given comparison.
`categoryReference`	The category of the Ct values ("OK", "Undetermined") across the reference samples for the given comparison.

Also, the last item in the list is called "Summary", and it's the result of calling `decideTests` from limma on the fitted data. This is a data frame with one row per feature and one column per comparison, with downregulation, no change and upregulation marked by -1, 0 and 1.

Author(s)

Heidi Dvinge

References

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

See Also

`lmFit`, `contrasts.fit` and `ebayes` for more information about the underlying limma functions. `plotCtRQ`, `heatmapSig` and `plotCtSignificance` can be used for visualising the results.

Examples

```
# Load example preprocessed data
data(qPCRpros)
samples <- read.delim(file.path(system.file("exData", package="HTqPCR"), "files.txt"))
# Define design and contrasts
design <- model.matrix(~0+samples$Treatment)
colnames(design) <- c("Control", "LongStarve", "Starve")
contrasts <- makeContrasts(LongStarve-Control, LongStarve-Starve, Starve-Control, levels=
```

```
# The actual test
diff.exp <- limmaCtData(qPCRpros, design=design, contrasts=contrasts)
# Some of the results
diff.exp[["LongStarve - Control"]][1:10,]
# Example with duplicate genes on card.
# Reorder data to get the genes in consecutive rows
temp <- qPCRpros[order(featureNames(qPCRpros)),]
diff.exp <- limmaCtData(temp, design=design, contrasts=contrasts, ndups=2, spacing=1)
# Some of the results
names(diff.exp)
diff.exp[["LongStarve - Control"]][1:10,]
diff.exp[["Summary"]][1:10,]
```

`normalizeCtData` *Normalization of Ct values from qPCR data.*

Description

This function is for normalizing Ct data from high-throughput qPCR platforms like the TaqMan Low Density Arrays. Normalization can be either within or across different samples.

Usage

```
normalizeCtData(q, norm = "deltaCt", deltaCt.genes = NULL, scale.rank.samples, r
```

Arguments

`q`	object of class qPCRset.
`norm`	character string with partial match allowed, the normalisation method to use. "deltaCt" (default) , "scale.rankinvariant", "norm.rankinvariant" and "quantile" are implemented. See details.
`deltaCt.genes`	character vector, the gene(s) to use for deltaCt normalization. Must correspond to some of the `featureNames` in `q` or `NULL`, in which case the endogenous controls from `featureType` are used.
`scale.rank.samples`	integer, for the "scale.rankinvariant" method, how many samples should a feature be rank invariant across to be included. Defaults to number of samples-1.
`rank.type`	string, the reference sample for the rank invariant normalisation. Either "pseudo.median" or "pseudo.mean" for using the median or mean across samples as a pseudo-reference sample.
`Ct.max`	numeric, Ct values above this will be ignored when identifying rank invariant genes.
`verbose`	boolean, should some informaiton be printed to the prompt.

Details

"quantile" will make the expression distributions across all cards more or less identical. "deltaCt" calculates the standard deltaCt values, i.e. subtracts the mean of the chosen controls from all other values on the array. "scale.rankinvariant" sorts features from each sample based on Ct values, and identifies a set of features that remain rank invariant, i.e. whose ordering is constant. The average of these rank invariant features is then used to scale the Ct values on each array individually. "norm.rankinvariant" also identifies rank invariant features between each sample and a reference, and then uses these features to generate a normalisation curve individually for each sample by smoothing.

For the rank invariant methods it can make a significant difference whether high Ct values, such as "40" or something else being used for undetermined Ct values is removed during the normalisation using the Ct.max parameter. "norm.rankinvariant" also depends on having enough rank invariant genes for generating a robust smoothing curve.

"quantile" is base on `normalizeQuantiles` from `limma`, and the rank invariant normalisations implement methods from `normalize.invariantset` in package `affy`.

The distribution of Ct values before/after normalisation can be assessed with the function `plotCtDensity`.

Value

An object of class qPCRset like the input.

Author(s)

Heidi Dvinge

See Also

`normalize.invariantset` for the rank invariant normalisations, `normalizequantiles` and `plotCtDensity`

Examples

```
# Load example data
data(qPCRraw)
# Perform different normalisations
dnorm <- normalizeCtData(qPCRraw, norm="deltaCt", deltaCt.genes="Gene1")
qnorm <- normalizeCtData(qPCRraw, norm="quantile")
nrnorm <- normalizeCtData(qPCRraw, norm="norm.rankinvariant")
srnorm <- normalizeCtData(qPCRraw, norm="scale.rankinvariant")
# Normalized versus raw data
cols <- rep(brewer.pal(6, "Spectral"), each=384)
plot(exprs(qPCRraw), exprs(dnorm), pch=20, col=cols, main="dCt normalization")
plot(exprs(qPCRraw), exprs(qnorm), pch=20, col=cols,  main="Quantile normalization")
plot(exprs(qPCRraw), exprs(nrnorm), pch=20, col=cols,  main="Housekeeping genes normaliza
plot(exprs(qPCRraw), exprs(srnorm), pch=20, col=cols,  main="Rank invariant normalizatio
# With or without removing high Ct values
nrnorm <- normalizeCtData(qPCRraw, norm="norm.rankinvariant")
nrnorm2 <- normalizeCtData(qPCRraw, norm="norm.rankinvariant", Ct.max=40)
plot(exprs(nrnorm), exprs(nrnorm2), pch=20, col=cols, xlab="Ct.max = 35", ylab="Ct.max =
# Distribution of the normalised data
par(mfrow=c(2,2), mar=c(3,3,2,1))
plotCtDensity(dnorm, main="deltaCt")
plotCtDensity(qnorm, main="quantile")
plotCtDensity(srnorm, main="scale.rankinvariant")
```

```
plotCtDensity(nrnorm, main="norm.rankinvariant")
```

`plotCtBoxes` *Boxplots for qPCR Ct values.*

Description

Function for making boxplots of Ct values from high-throughput qPCR data. The boxes can be made either using all values on each card, or stratified by different feature information.

Usage

```
plotCtBoxes(q, cards = TRUE, xlab = "", col, main = NULL, names, stratify = "typ
```

Arguments

`q`	object of class qPCRset.
`cards`	vector, the numbers of the cards to plot. Defaults to TRUE = all cards.
`xlab`	character string, label for the x-axis.
`col`	vector of colours to use, defaults to different colour for each card.
`main`	character string, plot title.
`names`	vector, names to plot under the boxes. Defaults to sample names.
`stratify`	character, specifying what to stratify the Ct values by. NULL, the default means no stratification, "type" is the feature types of the qPCRset, and "class" the feature class.
`mar`	vector, the size of the margins. See `par` for details.
`...`	any other arguments will be passed to the `boxplot` or `par`function.

Details

For the stratified plots all boxes with Ct values from the same card are plotted in identical colours. "type" and "class" are automatically extracted from the qPCRset using `featureType` and `featureClass`.

Value

A plot is created on the current graphics device.

Author(s)

Heidi Dvinge

See Also

`boxplot`

Examples

```
# Loading the data
data(qPCRraw)
# Make plot with all samples or just a few
plotCtBoxes(qPCRraw, stratify=NULL)
plotCtBoxes(qPCRraw, cards=c(1,4))
plotCtBoxes(qPCRraw, stratify="class")
```

`plotCtCard` *Image plot of qPCR Ct values from a card format*

Description

Function for plotting high-throughput qPCR Ct values from a platform with a defined spatial layout, such as TaqMan Low Density Assay cards. The location of Ct values in the plot corresponds to the position of each well on the card.

Usage

```
plotCtCard(q, card = 1, plot = "Ct", main, nrow = 16, ncol = 24, col, col.range,
```

Arguments

`q`	object of class qPCRset.
`card`	integer, the sample number to plot.
`plot`	character string among "Ct", "flag", "type", "class") indicating what type of plot to produce. See Details for a longer description.
`main`	character string, the title of the plot. Per deault this is the sample name corresponding to card.
`nrow`	integer, the numer of rows on the card (16 for a standard 384 well format).
`ncol`	integer, the numer of columns on the card (24 for a standard 384 well format).
`col`	vector of colors of the same length as the number of different groups for the categorical data, or the name of a colour scheme for the continuous data.
`col.range`	vector, the range of colours to use.
`na.col`	the colour used for well with NA (undetermined) Ct values.
`na.value`	numeric, if NA has been replaced by an (arbitrary) high Ct value in the data.
`legend.cols`	integer, how many columns should the legend text be split into (defaults to number of labels).
`well.size`	numeric, for adjusting the size of the wells on the card.
`zero.center`	logical, should the colours be shifted to be zero-centered.
`unR`	logical, should wells from the category "Unreliable" be crossed out.
`unD`	logical, should wells from the category "Undetermined" be crossed out.
`...`	any other arguments will be passed to the `plot` and `points` functions.

Details

This function may be used to plot the values of any well-specific information, such as the raw or normalized Ct values, or categorical data such as flag, gene class etc. The image follows the layout of an actual HTqPCR card.

If `unR=TRUE` these will wells will be crossed out using a diagonal cross (X), whereas `unD=TRUE` will be marked with a horisontal/vertical cross.

Value

A plot is created on the current graphics device.

Author(s)

Heidi Dvinge

See Also

`image`

Examples

```
# Load some example data
data(qPCRraw)
# Plot Ct values from first card
plotCtCard(qPCRraw)
plotCtCard(qPCRraw, card=2, col.range=c(10,35))
plotCtCard(qPCRraw, unR=TRUE, unD=TRUE)
# Other examples
plotCtCard(qPCRraw, plot="class")
plotCtCard(qPCRraw, plot="type")
plotCtCard(qPCRraw, plot="flag")
```

`plotCtCategory` *Summarising the feature categories for Ct values.*

Description

This function will provide a summary of the `featureCategory` for a qPCRset. Focus can either be on categories across samples, or across features.

Usage

```
plotCtCategory(q, cards = TRUE, by.feature = FALSE, stratify, col, xlim, main, .
```

Arguments

`q`	object of class qPCRset.
`cards`	integers, the number of the cards (samples) to plot.
`by.feature`	logical, should the categories be summarised for features rather than samples. See details.
`stratify`	character string, either "type" or "class" indicating if the categories should be stratified by `featureType` or `featureClass` of `q`. Ignored if `by.features` is TRUE.
`col`	vector with the colours to use for the categories. Default is green for "OK", yellow for "Unreliable" and red for "Undetermined".
`xlim`	vector, the limits of the x-axis. If `by.feature` is FALSE, this can be used to adjust the size of the barplot to fit in the colour legend.
`main`	character string, the title of the plot.
`...`	further arguments passed to `barplot` or `heatmap`.

Details

This function is for generating two different types of plot. If `by.feature=FALSE` the number of each `featureCategory` will be counted for each card, and a barplot is made. If however `by.feature=TRUE`, then the categories for each feature across the selected cards will be clustered in a heatmap.

For `by.feature=TRUE` the plot can be modified extensively using calls to the underlying `heatmap` function, such as setting `cexRow` to adjust the size of row labels.

Value

A figure is produced on the current graphics device.

Author(s)

Heidi Dvinge

See Also

`setCategory`, and `heatmap` for the underlying plotting function.

Examples

```
# Load example preprocessed data
data(qPCRpros)
# Plot categories for samples
plotCtCategory(qPCRpros)
plotCtCategory(qPCRpros, cards=1:3, stratify="class")
# Categories for features
plotCtCategory(qPCRpros, by.feature=TRUE)
```

`plotCtCor` *Correlation between Ct values from qPCR data*

Description

Function for plotting the correlation based on Ct values between samples containing high-throughput qPCR data.

Usage

```
plotCtCor(q, col, col.range = c(0, 1), main, mar, ...)
```

Arguments

`q`	object of class qPCRset.
`col`	vector of colours to use, defaults to a spectrum from red to blue/purple.
`col.range`	vector, the range of colours to use.
`main`	character string, plot title.
`mar`	vector, the size of the borrom and right hand side margins.
`...`	any other arguments will be passed to the `heatmap.2` function.

Details

This function may be used to cluster the samples based on Ct values and present the result in a heatmap. Per default the colours are a rainbow scale from 0 to 1.

A standard heatmap is drawn, but this can be modified extensively using the arguments available in the `heatmap.2` function.

Value

A plot is created on the current graphics device.

Author(s)

Heidi Dvinge

See Also

`heatmap.2`

Examples

```
data(qPCRraw)
plotCtCor(qPCRraw)
plotCtCor(qPCRraw, col.range=c(0.5,0.8))
```

`plotCtDensity` *Distribution plot for qPCR Ct values.*

Description

Function for plotting the density distribution of Ct values from high-throughput qPCR data.

Usage

```
plotCtDensity(q, cards = TRUE, xlab = "Ct", ylab = "Density", col, main = NULL,
```

Arguments

`q`	object of class qPCRset.
`cards`	vector, the numbers of the cards to plot. Defaults to TRUE = all cards.
`xlab`	character string, label for the x-axis.
`ylab`	character string, label for the y-axis.
`col`	vector of colours to use, defaults to different colour for each card.
`main`	character string, plot title.
`legend`	logical, whether to include a colour legend or not.
`lwd`	numeric, the width of the lines.
`...`	any other arguments will be passed to the `matplot` function.

Details

The distribution of Ct values in the qPCRset `q` is calculated using `density`.

Value

A plot is created on the current graphics device.

Author(s)

Heidi Dvinge

See Also

`matplot`, `density`

Examples

```
# Loading the data
data(qPCRraw)
# Make plot with all samples or just a few
plotCtDensity(qPCRraw)
plotCtDensity(qPCRraw, cards=c(1,4))
```

`plotCtHeatmap` *Heatmap of qPCR Ct values.*

Description

Function for drawing a heatmap of Ct values from high-throughput qPCR experiments such as using TaqMan Low Density Arrays.

Usage

```
plotCtHeatmap(q, main = NULL, col, col.range, dist = "pearson", zero.center, mar
```

Arguments

`q`	object of class qPCRset.
`main`	character string, plot title.
`col`	the colours to use. See details.
`col.range`	vector, the range of colours to use.
`dist`	character string, specifying whether to use "pearson" correlation (default) or "euclidean" distance for the clustering.
`zero.center`	logical, should the colours be shifted to be zero-centered. See details.
`mar`	vector, the size of the borrom and right hand side margins.
`gene.names`	character vector, names to replace the genes (rows) with. See details.
`sample.names`	character vector, names to replace the samples (columns) with. See details.
`...`	any other arguments will be passed to the `heatmap.2` function.

Details

This function may be used to cluster the raw or normalized Ct values, and present the result in a heatmap.

The color range is used to represent the range of values for the statistic. If `col==NULL` the colour will be set to a spectrum from red to blue/purple, unless there are negative values in which case it goes red-yellow-green to reflect up and down regulation of genes. If `zero.center=NULL` then zero.center will automatically be set to TRUE to make the colour scale symmetric around 0.

Especially gene names will often not be readable in a standard size plotting device, and might therefore be removed. If `gene.names` or `sample.names` is set to a single character (such as "" for no naming), then this character will be repeated for all rows or columns.

A standard heatmap is drawn, but this can be modified extensively using the arguments available in the `heatmap.2` function.

Value

A plot is created on the current graphics device.

Author(s)

Heidi Dvinge

See Also

`heatmap.2`

Examples

```
# Load example data
data(qPCRraw)
# Some standard heatmaps
plotCtHeatmap(qPCRraw, gene.names="")
plotCtHeatmap(qPCRraw, gene.names="", dist="euclidean", col.range=c(10,35))
plotCtHeatmap(qPCRraw, gene.names="", dist="euclidean", col=colorRampPalette(rev(brewer.p
```

`plotCtHistogram` *Histrogram of Ct values from qPCR experiments.*

Description

The distribution of Ct values for a selected qPCR sample is shown in a histogram.

Usage

```
plotCtHistogram(q, card = 1, xlab = "Ct", col, main, n = 30, ...)
```

Arguments

`q`	an object of class qPCRset.
`card`	integer, the number of the card (sample) to plot.
`xlab`	character string, the label for the x-axis.
`col`	integer or character, the colour for the histogram.
`main`	character string, the plot title. Default is the name of the sample.
`n`	integer, number of bins to divide the Ct values into.
`...`	any other arguments are passed to `hist`.

Value

A figure is generated in the current graphics device.

Author(s)

Heidi Dvinge

See Also

`plotCtDensity` or `plotCtBoxes` for including multiple samples in the same plot.

Examples

```
# Load example data
data(qPCRraw)
# Create the plots
plotCtHistogram(qPCRraw, card=2)
plotCtHistogram(qPCRraw, card=3, n=50, col="blue")
```

`plotCtOverview` *Overview plot of qPCR Ct values across multiple conditions.*

Description

Function for high-throughput qPCR data, for showing the average Ct values for features in a barplot, either for individual samples or averaged across biological or technical groups. If Ct values are shown, error bars can be included, or the Ct values can be displayed relative to a calibrator sample.

Usage

```
plotCtOverview(q, cards = TRUE, genes, groups, calibrator, replicates = TRUE, c
```

Arguments

`q`	object of class qPCRset.
`cards`	integer, the cards (samples) to use. Defaults to all.
`genes`	vector selecting the features to show. See Details.
`groups`	vector with groups to average the samples across. If missing all the samples are displayed individually.
`calibrator`	the value in `groups` to use as calibrator sample. See Details.
`replicates`	logical, if should values from replicated features in each sample be collapsed or kept separate.
`col`	colours to use for each sample or group.
`conf.int`	logical, should the 95 percent confidence interval be shown. See Details.
`legend`	logical, should a legend be included in the plot.
`...`	further arguments passed to `barplot`.

Details

If a calibrator is chosen all values will be displayed relative to this, i.e. as Ct(sample)-Ct(calibrator). If there is no calibrator, the full Ct values are shown, including 95% confidence interval if selected. For confidence intervals when there is a calibrator, it's the variation across Ct(sample)-average(Ct(calibrator)) that is shown.

When setting `replicates=TRUE` it is often better to specify `genes` by name rather than selecting for example the first 10 features using 1:10. This literally only takes the first 10 rows of the data, although some of these features might be replicated elsewhere in the data.

Value

A figure is produced in the current graphics device.

Author(s)

Heidi Dvinge

Examples

```
# Load example data
data(qPCRraw)
exPath <- system.file("exData", package="HTqPCR")
samples <- read.delim(file.path(exPath, "files.txt"))
# Show all samples for the first 10 genes
g <- featureNames(qPCRraw)[1:10]
plotCtOverview(qPCRraw, genes=g, xlim=c(0,90))
plotCtOverview(qPCRraw, genes=g, xlim=c(0,50), groups=samples$Treatment)
plotCtOverview(qPCRraw, genes=g, xlim=c(0,60), groups=samples$Treatment, conf.int=TRUE, y
# Relative to a calibrator sample
plotCtOverview(qPCRraw, genes=g, groups=samples$Treatment, calibrator="Control")
plotCtOverview(qPCRraw, genes=g, groups=samples$Treatment, calibrator="Control", conf.int
plotCtOverview(qPCRraw, genes=g, groups=samples$Treatment, calibrator="LongStarve")
```

`plotCtPairs` *Pairwise scatterplot of multiple sets of Ct values from qPCR data.*

Description

Produces a plot of high-throughput qPCR Ct values from N number of samples plotted pairwise against each other in an N by N plot. The Ct values will be in the upper triangle, and the correlation between samples in the lower. Features can be marked based on for example feature class or type.

Usage

```
plotCtPairs(q, cards = TRUE, lower.panel = panel.Ct.cor, upper.panel = panel.Ct.
```

Arguments

`q`	object of class qPCRset.
`cards`	vector, the cards to plot against each other.
`lower.panel`	function, to use for plotting the lower triangle.
`upper.panel`	function, to use for plotting the upper triangle.
`Ct.max`	numeric, Ct values above this limit will be excluded when calculating the correlation.
`col`	vector with the colour(s) to use for the points, or a character string ("type" or "class") indicating whether points should be coloured according to `featureType` or `featureClass` of `q`.
`pch`	integer or single character, which plotting symbol to use for the points.
`cex.cor`	numeric, the expansion factor for the text in `panel.Ct.cor`.
`cex.pch`	numeric, the expansion factor for the points in `panel.Ct.scatter`.
`diag`	logical, should the diagonal line y=x be plotted.
`...`	any other arguments are passed to the panel functon or `pairs`.

Value

A figure is generated in the current graphics device.

Author(s)

Heidi Dvinge

See Also

`pairs` or `plotCtScatter` for plotting just two samples.

Examples

```
# Load example data
data(qPCRraw)
# Various types of plot
plotCtPairs(qPCRraw, cards=1:4)
plotCtPairs(qPCRraw, col="black")
plotCtPairs(qPCRraw, Ct.max=40)
```

`plotCtPCA` *PCA for qPCR Ct values.*

Description

Perform and plot a principal component analysis for high-throughput qPCR data from any platform, for doing clustering.

Usage

```
plotCtPCA(q, s.names, g.names)
```

Arguments

`q`	a matrix or an object of class qPCRset containing Ct values.
`s.names`	character vector, names of samples. See details.
`g.names`	character vector, names of genes. See details.

Details

Per default the sample names from the qPCRset are used, howeer the gene names are replaced by "*" to avoid cluttering the plot.

Value

A plot is created on the current graphics device.

Note

This is still a work in progress, and the function is not particularly sophisticated.

Author(s)

Heidi Dvinge

See Also

`prcomp`

Examples

```
# Load example data
data(qPCRraw)
# Plot
plotCtPCA(qPCRraw)
```

`plotCtReps` *Scatter plot of features analysed twice during each qPCR experiment.*

Description

In high-throughput qPCR data some features may be present twice on each card (sample). This function will make a scatter plot of one replicate versus the other for each sample individually, as well as mark genes with very deviating replicate values.

Usage

```
plotCtReps(q, card = 1, percent = 20, verbose = TRUE, col = 1, ...)
```

Arguments

`q`	object of class qPCRset.
`card`	integer, the sample number to plot.
`percent`	numeric, features with replicate values differ more than this percentage from their average will be marked on the plot.
`verbose`	logical, should the deviating genes and their Ct values be printed to the terminal.
`col`	integer or character; the colour of the points in the scatter plot.
`...`	any other arguments are passed to `plot`.

Details

This function will look through the data in the qPCRset, find all genes with are presented twice on the array, and plot the Ct values of these replicated genes against each other. Whether a genes goes to the x or y-axis depends on the first occurrence of the gene names.

All genes where abs(rep1-rep2) > percent/100*replicate mean will be marked by an open circle, and the gene names written in red letters.

Value

An plot is created on the current graphics device. Also, a data.frame with the names and values of deviating genes is returned invisibly.

Author(s)

Heidi Dvinge

See Also

`plot`, and `par` for the plotting parameters.

Examples

```
# Load example data
data(qPCRraw)
# Plot replicates
plotCtReps(qPCRraw, card=1, percent=30)
plotCtReps(qPCRraw, card=2, percent=10)
reps <- plotCtReps(qPCRraw, card=2, percent=20)
reps
```

`plotCtRQ` *Plot the relative quantification of Ct values from qPCR experiments.*

Description

Function for plotting the relative quantification (RQ) between two groups of data, whose Ct valuse have been tested for significant differential expression.

Usage

```
plotCtRQ(qDE, comparison = 1, genes, transform = "log2", p.val = 0.1, mark.sig =
```

Arguments

`qDE`	list or data.frame, the result from `ttestCtData` or `limmaCtData`.
`comparison`	integer or character string, indicating which component to use if `qDE` is a list.
`genes`	numeric or character vector, selected genes to make the plot for.
`transform`	character string, how should the data be displayed. Options are "none", "log2" or "log10". See details
`p.val`	numeric between 0 and 1, if `genes` is not supplied all given with (adjusted) p-value below this threshold will be included.
`mark.sig`	logical, should significant features be marked.
`p.sig`	numeric, the cut-off for significant p-values that will be marked by *.
`p.very.sig`	numeric, the cut-off for very significant p-values that will be marked by ".
`mark.un`	logical, should data with unreliable target or calibrator samples be marked. See details.
`un.tar`	colour to use for the undetermined targets. See details.
`un.cal`	colour to use for the undetermined calibrators. See details.
`col`	vector, colours to use for the bars.
`legend`	logical, should a legend be included in the barplot.
`xlim`	vector of length 2, the limits on the x-axis. Mainly used for moving the legend to the left of bars.
`mar`	vector with 4 values, the size of the margins. See `par` for more info.
`main`	character string, the image title. Default to the name of the chosen comparison.
`...`	any other arguments will be passed to the `barplot` function.

Details

The relative quantification is calculated as RQ=2^-ddCT, where ddCT is the deltadeltaCt value.

If `mark.un=TRUE`, those bars where either the calibrator or target sample measurements were undetermined are marked using diagonal lines. Whether either of these are called undetermined (includes unreliable values) or not depends on all the input Ct values in `ttestCtData` or `limmaCtData`, and whether `stringent=TRUE` was used in these functions.

Value

A plot is created on the current graphics device.

Author(s)

Heidi Dvinge

See Also

`ttestCtData` and `limmaCtData` for testing the Ct data for differential expression.

Examples

```
# Load example data and calculate differential expression
data(qPCRpros)
qDE <- ttestCtData(qPCRpros[,1:4], groups=factor(c("A", "B", "B", "A")), calibrator="B")
# Plotting the top 10 results or first 10 genes
plotCtRQ(qDE, genes=1:10)
plotCtRQ(qDE, genes=featureNames(qPCRpros)[1:10])
# Plot all results with p-value below 0.08
plotCtRQ(qDE, p.val=0.08, transform="none")
plotCtRQ(qDE, p.val=0.08, transform="log10")
```

`plotCtScatter` *Scatterplot of two sets of Ct values from qPCR data.*

Description

Produces a plot of Ct values from two samples plotted against each other. Features can be marked based on for example feature class or type.

Usage

```
plotCtScatter(q, cards = c(1, 2), col = "class", pch = 20, diag = FALSE, cor = T
```

Arguments

`q`	object of class qPCRset.
`cards`	vector, the two cards to plot against each other.
`col`	vector with the colour(s) to use for the points, or a character string ("type" or "class") indicating whether points should be coloured according to `featureType` or `featureClass` of `q`.

`pch`	integer, the point type to use for the plot.
`diag`	logical, should the diagonal line y=x be plotted.
`cor`	logical, should information about the correlation between the two samples be included in the plot. The correlation is calculated both with and without removing Ct values above Ct.max.
`Ct.max`	numeric, all Ct values above this will be removed for calculating one of the correlations.
`legend`	logical, if `col` is either "type" or "class", should a colour legend for these be included.
`...`	any other arguments are passed to `plot`.

Value

A figure is generated in the current graphics device.

Author(s)

Heidi Dvinge

Examples

```
# Load example data
data(qPCRraw)
# Various types of plot
plotCtScatter(qPCRraw, cards=c(1,2))
plotCtScatter(qPCRraw, cards=c(1,4), col="type")
plotCtScatter(qPCRraw, cards=c(1,4), col="black", cor=FALSE, diag=TRUE)
```

`plotCtSignificance` *Barplot with Ct values between genes from qPCR.*

Description

Function for producing a barplot of the Ct values from high-throughput qPCR samples. A comparison is made between two groups which have been tested for differential expression, and all individual Ct values are shown, to identify potential outliers.

Usage

```
plotCtSignificance(qDE, q, comparison = 1, genes, p.val = 0.1, groups, calibrat
```

Arguments

`qDE`	list or data.frame, the result from `ttestCtData` or `limmaCtData`.
`q`	the qPCRset data that was used for testing for differential expression.
`comparison`	integer or character string, indicating which component to use if `x` is a list.
`genes`	numeric or character vector, selected genes to make the plot for.
`p.val`	numeric between 0 and 1, if `genes` is not supplied all given with (adjusted) p-value below this threshold will be included.

`groups`	vector, the groups of all the samples in `q`.
`calibrator`	character string, which of the `groups` is the calibrator.
`target`	character string, which of the `groups` is the target.
`p.sig`	numeric, the cut-off for significant p-values that will be marked by *.
`p.very.sig`	numeric, the cut-off for very significant p-values that will be marked by ".
`mark.sig`	logical, should significant features be marked.
`col`	vector, colours to use for the two sets of bars, one per sample type.
`un.col`	integer or character string, the colour to use for all Ct values that are "Unreliable" or "Undetermined".
`point.col`	integer or character string, the colour to use for all other Ct values.
`legend`	logical, should a legend be included int eh barplot.
`mar`	vector with 4 values, the size of the margins. See `par` for more info.
`main`	character string, the image title. Default to the name of the chosen comparison.
`jitter`	numeric, between 0 and 1. If Ct values are very similar, the individual points might lie on top of each other in the bars. This adds a jittering factor along the x-axis. If 0 the points will all be aligned.
`...`	any other arguments will be passed to the `barplot` function.

Details

This function will make a barplot with the average Ct values for the test and reference samples for the selected genes. All the individual Ct values are plotted on top of the bars though, and the "Unreliable" or "Undetermined" ones are marked, to do a visual assesment of the impact of non-valid measurements on the average.

It's up to the user to specify the correct `calibrator` and `target` for the given comparison; no checking is done.

Value

A plot is created on the current graphics device.

Author(s)

Heidi Dvinge

See Also

`barplot` and `plotCtRQ` or `plotCtOverview` for a plot of the relative quantification between samples.

Examples

```
# Load example data and calculate differential expression
data(qPCRpros)
grp <- factor(c("A", "B", "B", "A"))
qDE <- ttestCtData(qPCRpros[,1:4], groups=grp, calibrator="B")
# Plot
plotCtSignificance(qDE, q=qPCRpros, groups=grp, target="A", calibrator="B", genes=1:10, 
plotCtSignificance(qDE, q=qPCRpros, groups=grp, target="A", calibrator="B", genes=feature
plotCtSignificance(qDE, q=qPCRpros, groups=grp, target="A", calibrator="B", p.val=0.001, 
```

`plotCVBoxes` *Boxplots of CV for qPCR Ct values.*

Description

Function that will calculate the coefficients of variation across selected qPCR data, and plot the results in a boxplot.

Usage

```
plotCVBoxes(q, cards = TRUE, xlab = "", ylab = "CV", col = brewer.pal(5, "Spectr
```

Arguments

`q`	object of class qPCRset.
`cards`	vector, the numbers of the cards to plot. Defaults to TRUE = all cards.
`xlab`	character string, label for the x-axis.
`ylab`	character string, label for the y-axis.
`col`	vector of colours to use.
`main`	character string, plot title.
`stratify`	character, specifying what to stratify the Ct values by. NULL, the default means no stratification, "type" is the feature types of the qPCRset, and "class" the feature class.
`...`	any other arguments will be passed to the `boxplot` function.

Details

The CV is calculated across all the selected cards based on each well position, without taking possibly replicated genes on the cards into consideration. "type" and "class" are automaticalle extracted from the qPCRset using `featureType` and `featureClass`.

Value

A plot is created on the current graphics device. The CV valus are returned invisibly.

Author(s)

Heidi Dvinge

See Also

`boxplot`

Examples

```
# Load example data
data(qPCRraw)
# Make plot with all samples or just a few
plotCVBoxes(qPCRraw)
plotCVBoxes(qPCRraw, cards=c(1,4))
plotCVBoxes(qPCRraw, stratify="class")
x <- plotCVBoxes(qPCRraw, stratify="type")
x[1:10]
```

`plotCtLines` *Plotting Ct values from qPCR across multiple samples.*

Description

This function is for displaying a set of features from a qPCRset across multiple samples, such as a timeseries or different treatments. Values for each feature are connected by lines, and the can be averaged across groups rather than shown for individual smaples.

Usage

```
plotCtLines(q, genes, groups, col = brewer.pal(10, "Spectral"), xlab = "Sample",
```

Arguments

`q`	object of class qPCRset.
`genes`	numeric or character vector, selected genes to make the plot for.
`groups`	vector, the different groups that the samples in `q` belong to. See details.
`col`	vector, colours to use for the lines.
`xlab`	character string, label for the x-axis.
`ylab`	character string, label for the y-axis.
`legend`	logical, whether to include a colour legend or not.
`lwd`	numeric, the width of the lines.
`lty`	vector, line types to use. See `par` or `lines` for details.
`pch`	vector, if `groups` is set, the point types that will be used for each feature in `genes`.
`xlim`	vector of length two, the limits for the x-axis. Mainly used for adjusting the position of the legend.
`...`	any other arguments will be passed to the `matplot` function.

Details

The default plot shows the Ct values across all samples in `q`, with lines connecting the samples. However, if `groups` is set the Ct values will be averaged within groups. Lines connect these averages, but the individual values are shown with different point types, as chosen in `pch`.

Value

A plot is created on the current graphics device.

Author(s)

Heidi Dvinge

See Also

`matplot`

Examples

```
# Load some example data
data(qPCRraw)
samples <- exFiles <- read.delim(file.path(system.file("exData", package="HTqPCR"), "file
# Draw dfferent plots
plotCtLines(qPCRraw, genes=1:10)
plotCtLines(qPCRraw, genes=1:10, groups=samples$Treatment, xlim=c(0,3))
plotCtLines(qPCRraw, genes=1:10, col=as.numeric(featureType(qPCRraw)[1:10]))
```

`qPCRpros` *Example processed qPCR data*

Description

Processed version of the raw data in qPCRraw, to be used as example data in the HTqPCR package. The data has been processed with `setCategory` to mark the feature categories, and with `normalizaHTqPCRCard` using rank invariant normalisation.

Usage

```
data(qPCRpros)
```

Format

The format is: Formal class 'qPCRset' [package ".GlobalEnv"] with 9 slots ..@ featureNames : chr [1:384] "Gene1" "Gene2" "Gene3" "Gene4" ... ..@ sampleNames : chr [1:6] "sample1" "sample2" "sample3" "sample4" ... ..@ exprs : num [1:384, 1:6] 11.5 33.9 28 26.9 25 ... .- attr(*, "dimnames")=List of 2 .. .. ..$: chr [1:384] "Gene1" "Gene2" "Gene3" "Gene4" ... .$: chr [1:6] "sample1" "sample2" "sample3" "sample4" ... ..@ flag :'data.frame': 384 obs. of 6 variables: .. ..$ V1: chr [1:384] "Passed" "Passed" "Passed" "Passed" ... .$ V2: chr [1:384] "Passed" "Passed" "Passed" "Passed" ... .$ V3: chr [1:384] "Passed" "Passed" "Passed" "Passed" ... .$ V4: chr [1:384] "Flagged" "Flagged" "Passed" "Passed" ... .$ V5: chr [1:384] "Passed" "Passed" "Passed" "Passed" ... .$ V6: chr [1:384] "Passed" "Passed" "Passed" "Passed" ... ..@ featureType : Factor w/ 2 levels "Endogenous Control",..: 1 2 2 2 2 2 2 2 2 2 ... ..@ featurePos : chr [1:384] "A1" "A2" "A3" "A4" ... ..@ featureClass : Factor w/ 3 levels "Kinase","Marker",..: 3 3 2 1 2 3 1 3 3 3 ... ..@ featureCategory:'data.frame': 384 obs. of 6 variables: .. ..$ X1: chr [1:384] "Unreliable" "OK" "OK" "OK" ... .$ X2: chr [1:384] "Unreliable" "Undetermined" "OK" "OK" ... .$ X3: chr [1:384] "Unreliable" "OK" "OK" "OK" ... .$ X4: chr [1:384] "OK" "OK" "OK" "OK" ... .$ X5: chr [1:384] "Unreliable" "Undetermined" "OK" "OK" ... .$ X6: chr [1:384] "OK" "OK" "OK" "OK" ... ..@ normalized : chr "rankinvariant"

Examples

```
data(qPCRpros)
```

`qPCRraw` *Example raw qPCR data.*

Description

Six qPCR samples, performed on the TaqMan Low Density Arrays from Applied Biosystem. Each sample contains 384 PCR reactions, and there are 3 different samples with 2 replicates each. To be used as example data in the HTqPCR package.

Usage

```
data(qPCRraw)
```

Format

An object of class qPCRset. The format is: Formal class 'qPCRset' [package ".GlobalEnv"] with 9 slots ..@ featureNames : chr [1:384] "Gene1" "Gene2" "Gene3" "Gene4" ... ..@ sampleNames : chr [1:6] "sample1" "sample2" "sample3" "sample4" ... ..@ exprs : num [1:384, 1:6] 11.5 33.9 28 26.9 25 ... .- attr(*, "dimnames")=List of 2 .. .. ..$: chr [1:384] "Gene1" "Gene2" "Gene3" "Gene4" ... .$: chr [1:6] "sample1" "sample2" "sample3" "sample4" ... ..@ flag :'data.frame': 384 obs. of 6 variables: .. ..$ V1: chr [1:384] "Passed" "Passed" "Passed" "Passed" ... .$ V2: chr [1:384] "Passed" "Passed" "Passed" "Passed" ... .$ V3: chr [1:384] "Passed" "Passed" "Passed" "Passed" ... .$ V4: chr [1:384] "Flagged" "Flagged" "Passed" "Passed" ... .$ V5: chr [1:384] "Passed" "Passed" "Passed" "Passed" ... .$ V6: chr [1:384] "Passed" "Passed" "Passed" "Passed" ... ..@ featureType : Factor w/ 2 levels "Endogenous Control",..: 1 2 2 2 2 2 2 2 2 2 ... ..@ featurePos : chr [1:384] "A1" "A2" "A3" "A4" ... ..@ featureClass : Factor w/ 3 levels "Kinase","Marker",..: 3 3 2 1 2 3 1 3 3 3 ... ..@ featureCategory:'data.frame': 384 obs. of 6 variables: .. ..$ X1: chr [1:384] "OK" "OK" "OK" "OK" ... .$ X2: chr [1:384] "OK" "OK" "OK" "OK" ... .$ X3: chr [1:384] "OK" "OK" "OK" "OK" ... .$ X4: chr [1:384] "OK" "OK" "OK" "OK" ... .$ X5: chr [1:384] "OK" "OK" "OK" "OK" ... .$ X6: chr [1:384] "OK" "OK" "OK" "OK" ... ..@ normalized : chr "none"

Examples

```
data(qPCRraw)
```

`qPCRset-class` *Class "qPCRset"*

Description

This is a class for containing the raw or normalized data, such as Ct values and some related quality information, from TaqMan Low Density Arrays or any other type of (high-throughput) qPCR data. It is similar to `eSet` for microarray data.

Objects from the Class

Objects can be created by calls of the form `new("qPCRset", ...)` or using `readCtData`.

Slots

- `featureNames`: Object of class `"character"` giving the names of the features, such as genes or miRNAs, in the samples.
- `sampleNames`: Object of class `"character"` containing the sample names.
- `exprs`: Object of class `"matrix"` containing the Ct values.
- `flag`: Object of class `"data.frame"` containing the flag for each Ct value, as supplied by the input files.
- `featureType`: Object of class `"factor"` representing the different types of features on the card, such as controls and target genes.
- `featurePos`: Object of class `"character"` representing the location "well" of a gene on the card. If data does not come from a card format, the positions will be given consecutive names.
- `featureClass`: Object of class `"factor"` with some meta-data about the genes, for example if it is a marker, transcription factor or similar.
- `featureCategory`: Object of class `"data.frame"` representing the quality of the measurement for each Ct value, such as "OK", or "Unreliable" if the Ct value is considered too migh.
- `normalized`: Object of class `"character"` indicating if the data has been normalized, and if so then what method was used.

Methods

- **[** `signature(x = "qPCRset")`: Subsets by genes or samples.
- **exprs** `signature(object = "qPCRset")`: Extracts the Ct matrix.
- **exprs<-** `signature(object = "qPCRset", value = "matrix")`: Replaces the Ct matrix.
- **featureNames** `signature(object = "qPCRset")`: Extracts the features (gene names) on the card.
- **featureNames<-** `signature(object = "qPCRset", value = "character")`: Replaces the features (gene names) on the card.
- **sampleNames** `signature(object = "qPCRset")`: Extracts the sample names.
- **sampleNames<-** `signature(object = "qPCRset", value = "character")`: Replaces the sample names.
- **featureType** `signature(object = "qPCRset")`: Extracts the feature type for each gene.
- **featureType<-** `signature(object = "qPCRset", value = "factor")`: Replaces the feature type for each gene.
- **featurePos** `signature(object = "qPCRset")`: Extracts the position of each feature (gene) on the card.
- **featurePos<-** `signature(object = "qPCRset", value = "character")`: Replaces the position of each feature (gene) on the card.
- **featureClass** `signature(object = "qPCRset")`: Extracts the feature class for each gene.
- **featureClass<-** `signature(object = "qPCRset", value = "factor")`: Replaces the feature class for each gene.
- **featureCategory** `signature(object = "qPCRset")`: Extracts the category of each Ct value.
- **featureCategory<-** `signature(object = "qPCRset", value = "data.frame")`: Replaces the category of each Ct value.

- **n.wells** `signature(object = "qPCRset")`: Extracts information about the number of wells on the card.
- **n.samples** `signature(object = "qPCRset")`: Extracts information about the number of samples in the set.
- **normalized** `signature(object = "qPCRset")`: Extracts information about normalization.
- **normalized** `signature(object = "qPCRset")`: Extracts information about normalization.
- **show** `signature(object = "qPCRset")`: Displays som abbreviated information about the data object.
- **summary** `signature(object = "qPCRset")`: Displays a summary of the Ct values from each sample.

Author(s)

Heidi Dvinge

Examples

```
data(qPCRraw)
show(qPCRraw)
showClass("qPCRset")
str(qPCRraw)
```

`readCtData` *Reading Ct values from qPCR experiments data into a qPCRset*

Description

This function will read tab separated text files with Ct values and feature meta-data from high-throughput qPCR experiments into a qPCRset containing all the relevant information.

Usage

```
readCtData(files, path = NULL, n.features = 384, flag = 4, feature = 6, type = 7
```

Arguments

`files`	character vector witht he names of the files to be read.
`path`	character string with the path to the folder containing the data files.
`n.features`	integer, number of features present in each file.
`flag`	integer indicating the number of column containing information about the flags. See Details.
`feature`	integer indicating the number of column containing information about the individual features (typically gene names).
`type`	integer indicating the number of column containing information about the type of each feature. See Details.
`position`	integer indicating the number of column containing information about the position of features on the card. See Details.

`Ct`	integer indicating the number of column containing information about the Ct values.
`header`	logical, does the file contain a header row or not.
`SDS`	logical, is the data in the output format from Sequence Detection Systems (SDS) Software. See Details.
`samples`	character vector with names for each sample. Per default the file names are used.
`na.value`	integer, a Ct value that will be assigned to all undetermined/NA wells.
`...`	any other arguments are passed to `read.table`.

Details

This is the main data input function for the HTqPCR package for analysing qPCR data. It extracts the threshold cycle, Ct value, of each well on the card, as well as information about the quality (e.g.~passed/failed) of the wells. The function is tuned for data from TaqMan Low Density Array cards, but can be used for any kind of qPCR data.

`featureNames`, `featureType` and `featurePos` will be extracted from the first file. If `flag`, `type` or `position` er set to NULL, this means that this information is not available in the file. `flag` will then be set to "Passed", `type` to "Target" and `position` to "feature1", "feature2", ... etc until the end of the file. Especially `position` might not be available in case the data does not come from a card format, but it is required in subsequent functions in order to disambiguate between features in case some features are present multiple times.

If the data was analysed using SDS Software it may contain a variable length header specifying parameters for files that were analysed at the same time. If `SDS=TRUE` then `readCtData` will scan through the first 100 lines of each file, and skip all lines until (and including) the line beginning with "#", which is the header. The end of the file might also contain some plate ID information, but only the number of lines specified in `n.features` will be read.

Value

A "`qPCRset`" object.

Warnings

The files are all assumed to belong to the same design, i.e.~have the same features (genes) in them and in identical order.

Author(s)

Heidi Dvinge

See Also

`read.delim` for further information about reading in data, and "`qPCRset`" for a definition of the resulting object.

Examples

```
# Locate example data and create qPCRset object
exPath <- system.file("exData", package="HTqPCR")
exFiles <- read.delim(file.path(exPath, "files.txt"))
raw <- readCtData(files=exFiles$File, path=exPath)
# Example of adding missing information (random data in this case)
featureClass(raw) <- factor(rep(c("A", "B", "C"), each=384/3))
```

`setCategory` *Assign categories to Ct values from qPCR data.*

Description

Data in qPCRset objects will have feature categories ("Unreliable", "Undtermined") assigned to them based on different Ct criteria.

Usage

```
setCategory(q, Ct.max = 35, Ct.min = 10, replicates = TRUE, quantile = 0.9, grou
```

Arguments

`q`	qPCRset object.
`Ct.max`	numeric, the maximum tolerated Ct value. Everything above this will be "Undetermined".
`Ct.min`	numeric, the minimum tolerated Ct value. Everything below this will be "Unreliable".
`replicates`	logical, should Ct values from genes replicated within each sample be collapsed for the standard deviation.
`quantile`	numeric from 0 to 1, the quantile interval accepted for standard deviations. `NULL` means that variation between replicates is not used for setting the categories. See details.
`groups`	vector, grouping of cards, for example biological or technical replicates
`flag`	logical, should categories also be set to "Unreliable" according to the content of `flag(q)`.
`flag.out`	character vector, if `flag=TRUE`, what are the flag(s) to be set as "Unreliable".
`verbose`	logical, should a summary about category counts per sample be printed to the prompt.
`plot`	logical, should some plots of the standard deviations be created.
`...`	any other arguments are passed to `plot`.

Details

Categories can be assigned to the `featureCategory` of the qPCRset using either just simple criteria (max/min of Ct values or `flag` of `q`) or by looking at the standard deviation of Ct values across biological and technical replicates for each gene.

When looking at replicates, the standard deviation and mean are calculated and a normal distribution following these parameters is generated. Individual Ct values that are outside the interval set by `quantile` are set as "Unreliable". So if e.g. `quantile=90` the values outside the top 5% and lower 5% of the normal distribution with the given mean and standard deviation are removed.

"Undetermined" has priority over "Unreliable", so if a value is outside `quantile` but also above `Ct.max` it will be "Undetermined".

NB: When setting categories based on replicates, the Ct values are assumed to follow a normal distribution. This might not be the case if the number of samples within each group is small, and there are no replicates on the genes within each sample.

If the number of replicates vary significantly between biological groups, this will influence the thresholds used for determining the range of "OK" Ct values.

Value

If `plot=TRUE` one figure per sample grous is returned to the current graphics device. A qPCRset with the new feature categories is returned invisibly.

Note

It's adviced to try several different values for `quantile`, depending on the input data set. Using the function `PlotCtCategory(..., by.feature=FALSE)` or `plotCtCategory(..., by.feature=TRUE`
might help assess the result of different `quantile` choices.

Author(s)

Heidi Dvinge

See Also

`filterCategory`, `plotCtCategory`

Examples

```
# Load example data
data(qPCRraw)
exFiles <- read.delim(file.path(system.file("exData", package="HTqPCR"), "files.txt"))
 # Set categories in various ways
setCategory(qPCRraw, flag=FALSE, quantile=NULL)
setCategory(qPCRraw[,1:4], groups=exFiles$Treatment[1:4], plot=TRUE)
setCategory(qPCRraw[,1:4], groups=exFiles$Treatment[1:4], plot=TRUE, quantile=0.80)
x <- setCategory(qPCRraw, groups=exFiles$Treatment, verbose=FALSE, quantile=0.80)
# Plot the categories
plotCtCategory(x)
```

`ttestCtData` *Differentially expressed features with qPCR: t-test*

Description

Function for calculating t-test and p-values across two groups for the features present in high-throughput qPCR data, such as from TaqMan Low Density Arrays.

Usage

```
ttestCtData(q, groups = NULL, calibrator, alternative = "two.sided", paired = FA
```

Arguments

`q`	qPCRset object.
`groups`	factor, assigning each sample to one of two groups.
`calibrator`	which of the two groups is to be considered as the reference and not the test? Defaults to the first group in `groups`.
`alternative`	character string (first letter is enough), specifying the alternative hypothesis, "two.sided" (default), "greater" or "less".

`paired`	logical, should a paired t-test be used.
`replicates`	logical, if replicated genes are present on the array, the statistics will be calculated for all the replicates combined, rather than the individual wells.
`sort`	boolean, should the output be sorted by p-values.
`stringent`	boolean, for flagging results as "Undetermined". See details.
`...`	any other arguments will be passed to the `t.test` function.

Details

Once the Ct values have been normalised, differential expression can be calculated. This function deals with just the simple case, where there are two types of samples to compare. For more complicated studies, see `limmaCtData`.

All results are assigned to a category, either "OK" or "Undetermined" depending on the input Ct values. If `stringent=TRUE` any unreliable or undetermined measurements among technical and biological replicates will result in the final result being "Undetermined". For `stringent=FALSE` the result will be "OK" unless at least half of the Ct values for a given gene are unreliable/undetermined.

Value

A data.frame containing the following information:

`genes`	The names of the features on the card.
`feature.pos`	The `featurePos` of the genes. If replicated genes are used, the feature positions will be concatenated together.
`t.test`	The value of the t-test.
`p.value`	The corresponding p-value.
`meanCalibrator`	The average expression level of each gene in the calibrator sample(s).
`meanTarget)`	The average expression level of each gene in the target sample(s).
`categoryCalibrator`	The category of the Ct values ("OK", "Undetermine") across the calibrator.
`categoryTarget`	Ditto for the target.

Author(s)

Heidi Dvinge

See Also

`t.test`, `limmaCtData`. `plotCtRQ` and `plotCtSignificance` can be used for visualising the results.

Examples

```
# Load example preprocessed data
data(qPCRpros)
# Test between two groups, collapsing replicated features
diff.exp <- ttestCtData(qPCRpros[,1:4], groups=factor(c("A", "B", "B", "A")), calibrator=
diff.exp[1:10,]
# The same test, taking replicated features individually
diff.exp <- ttestCtData(qPCRpros[,1:4], groups=factor(c("A", "B", "B", "A")), calibrator=
```

Index