HTqPCR - high-throughput qPCR analysis in R and Bioconductor

Heidi Dvinge

May 30, 2026

1 Introduction

The package *HTqPCR* is designed for the analysis of cycle threshold (Ct) values from quantitative real-time PCR data. The main areas of functionality comprise data import, quality assessment, normalisation, data visualisation, and testing for statistical significance in Ct values between different features (genes, miRNAs).
The example data used throughout this vignette is from from TaqMan Low Density Arrays (TLDA), a proprietary format of Applied Biosystems, Inc. However, most functions can be applied to any kind of qPCR data, regardless of the nature of the experimental samples and whether genes or miRNAs were measured. Section 13 gives some examples of how to handle other types of input data, including output formats from other qPCR assay vendors (e.g. Roche Applied Science and Bio-Rad) and data from non-well based microfluidic systems (e.g. BioMark from Fluidigm Corporation).

```
> library("HTqPCR")
```

The package employs functions from other packages of the Bioconductor project. Dependencies include *Biobase*, *RColorBrewer*, *limma*, *statmod*, *affy* and *gplots*.

Examples from the vignette

This vignette was developed in Sweave, so the embedded R code was compiled when the PDF was generated, and its output produced the results and plots that appear throughout the document. The following commands will extract all of the code from this file:

```
> all.R.commands <- system.file("doc", "HTqPCR.Rnw",
+     package = "HTqPCR")
> Stangle(all.R.commands)
```

This will create a file called HTqPCR.R in your current working directory, and this file can then either be sourced directly, or the commands run individually.

General workflow

The main functions and their use are outlined in Figure 1. Note that the QC plotting functions can be used both before and after normalisation, in order to examine the quality of the data or look for particular trends.
For the full set of available functions type:

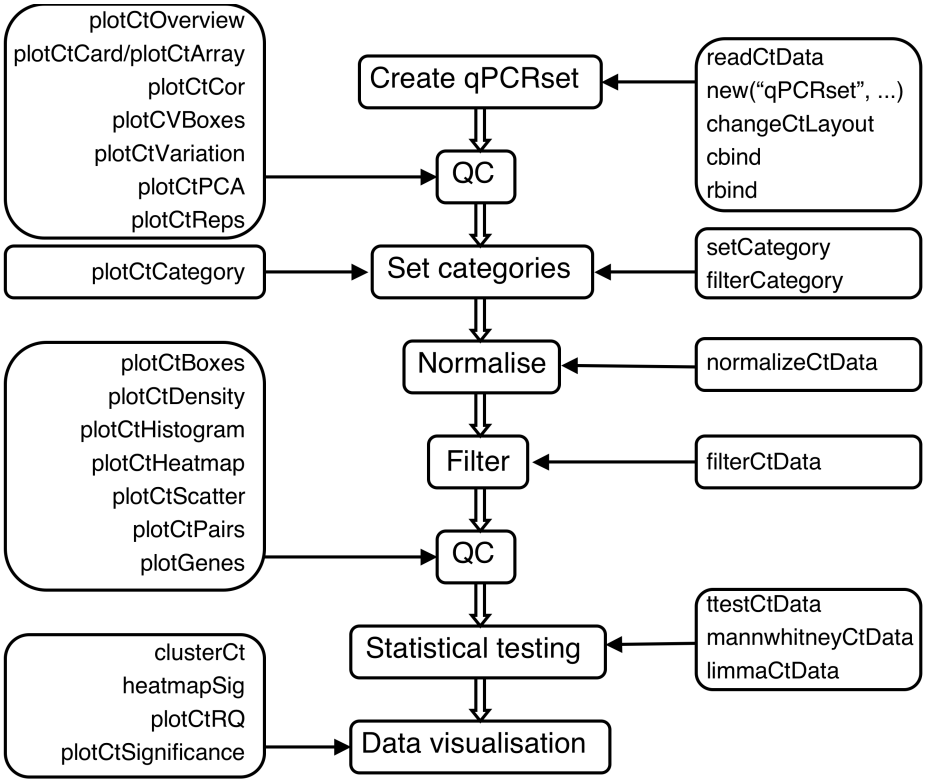


Figure 1: Workflow in *HTqPCR* analysis of qPCR data. Centre column: The main procedural steps in a typical qPCR analysis; Left: examples of visualisation functions; Right: data analysis functions.

```
> ls("package:HTqPCR")
 [1] "changeCtLayout"     "clusterCt"
 [3] "exprs"              "exprs<-"
 [5] "featureCategory"    "featureCategory<-"
 [7] "featureClass"       "featureClass<-"
 [9] "featureNames"       "featureNames<-"
[11] "featurePos"         "featurePos<-"
[13] "featureType"        "featureType<-"
[15] "filterCategory"     "filterCtData"
[17] "flag"               "flag<-"
[19] "getCt"              "getCtHistory"
[21] "heatmapSig"         "limmaCtData"
[23] "mannwhitneyCtData"  "n.samples"
[25] "n.wells"            "normalizeCtData"
[27] "plotCtArray"        "plotCtBoxes"
[29] "plotCtCard"         "plotCtCategory"
[31] "plotCtCor"          "plotCtDensity"
[33] "plotCtHeatmap"      "plotCtHistogram"
[35] "plotCtLines"        "plotCtOverview"
[37] "plotCtPairs"        "plotCtPCA"
[39] "plotCtReps"         "plotCtRQ"
[41] "plotCtScatter"      "plotCtSignificance"
[43] "plotCtVariation"    "plotCVBoxes"
[45] "readCtData"         "sampleNames"
[47] "sampleNames<-"      "setCategory"
[49] "setCt<-"            "setCtHistory<-"
[51] "show"               "summary"
[53] "ttestCtData"
```

Getting help

Please send questions about *HTqPCR* to the Bioconductor mailing list.
See http://www.bioconductor.org/docs/mailList.html

2 *qPCRset* objects

The data is stored in an object of class *qPCRset*, which inherits from the *eSet* class from package *Biobase*. *eSet* was originally designed for handling microarray data, but can deal with any kind of data where the same property (e.g. qPCR of genes) is measured across a range of samples.
Two `qPCRset` test objects are included in the package: one containing raw data, and the other containing processed values. An example is shown in figure 2, along with some of the functions that can typically be used for manipulating *qPCRset* objetcs.

```
> data(qPCRraw)
> data(qPCRpros)
> class(qPCRraw)
[1] "qPCRset"
attr(,"package")
[1] ".GlobalEnv"
```

The format is the same for raw and normalized data, and depending on how much information is available about the input data, the object can contain the following information:

`featureNames` Object of class *character* giving the names of the features (genes, miRNAs) used in the experiment. This is a column in the *featureData* of the *qPCRset* object (see below).

`sampleNames` Object of class *character* containing the sample names of the individual experiments.

`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. These are typically set during the calculation of Ct values, and indicate whether the results are flagged as e.g. "Passed" or "Flagged".

`featureType` Object of class *character* representing the different types of features on the card, such as endogenous controls and target genes. This is a column in the *featureData* of the *qPCRset* object.

`featurePos` Object of class *character* representing the location "well" of a gene in case TLDA cards are used, or some other method containing a defined spatial layout of features. Like `featureType` and `featureName`, `featurePos` is found within the *featureData*.

`featureClass` Object of class *factor* with some meta-data about the genes, for example if it is a transcription factor, kinase, marker for different types of cancers or similar. This is typically set by the user, and will be located within the *featureData*.

`featureCategory` Object of class *data.frame* representing the quality of the measurement for each Ct value, such as "OK", "Undetermined" or "Unreliable". These can be set using the function `setCategories` depending on a number of parameters, such as how the Ct values are flagged, upper and lower limits of Ct values and variations between technical and biological replicates of the same feature.

`history` Object of class *data.frame* indicating what operations have been performed on the *qPCRset* object, and what the parameters were. Automatically set when any of the functions on the upper right hand side of figure 1 are called (`readCtData, setCategory, filterCategory, normalizeCtData, filterCtData, changeCtLayout, rbind, cbind`).

Generally, information can be handled in the *qPCRset* using the same kind of functions as for *ExpressionSet*, such as `exprs`, `featureNames` and `featureCategory` for extracting the data, and `exprs<-`, `featureNames<-` and `featureCategory<-` for replacing or modifying values. The use of `exprs` might not be intuitive to users who are not used to dealing with microarray data, and hence *ExpressionSet*. The functions `getCt` and `setCt<-` that perform the same operations as `exprs` and `exprs<-` are therefore

also included. For the sake of consistency, `exprs` will be used throughout this vignette for accessing the Ct values, but it can be replaced by `getCt` in all examples.
The overall structure of `qPCRset` is inherited from `eSet`, as shown in the example below. This is a flexible format, which allows the used to add additional information about for example the experimental protocol. Information about the samples is contained within the `phenoData` slot, and details can be accessed of modified using `pData`. Likewise, for the individual features (mRNA, miRNAs) are available in the `featureData` slot, and can be accessed of modified using `fData`. See e.g. `AnnotatedDataFrame` for details.

```
> slotNames(qPCRraw)
[1] "CtHistory"         "assayData"         "phenoData"
[4] "featureData"       "experimentData"    "annotation"
[7] "protocolData"      ".__classVersion__"
> phenoData(qPCRraw)
An object of class 'AnnotatedDataFrame'
  sampleNames: sample1 sample2 ... sample6 (6 total)
  varLabels: sample
  varMetadata: labelDescription
> pData(qPCRraw)
        sample
sample1      1
sample2      2
sample3      3
sample4      4
sample5      5
sample6      6
> pData(qPCRraw) <- data.frame(Genotype = rep(c("A",
+     "B"), each = 3), Replicate = rep(1:3, 2))
> pData(qPCRraw)
  Genotype Replicate
1        A         1
2        A         2
3        A         3
4        B         1
5        B         2
6        B         3
> featureData(qPCRraw)
An object of class 'AnnotatedDataFrame'
  featureNames: 1 2 ... 384 (384 total)
  varLabels: featureNames featureType featurePos
    featureClass
  varMetadata: labelDescription
> head(fData(qPCRraw))
```

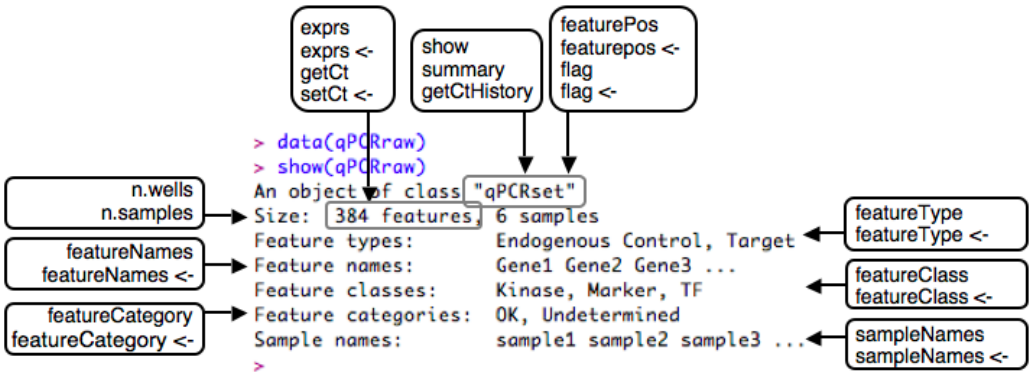


Figure 2: An example of a *qPCRset* object, and some of the functions that can be used to display and/or alter different aspects of the object, i.e. the accessor and replacement functions.

```
  featureNames        featureType featurePos featureClass
1        Gene1 Endogenous Control         A1       Kinase
2        Gene2             Target         A2       Marker
3        Gene3             Target         A3       Kinase
4        Gene4             Target         A4           TF
5        Gene5             Target         A5       Marker
6        Gene6             Target         A6       Marker
```

`qPCRset` objects can also be combined or reformatted in various ways (see section 12).

3 Reading in the raw data

The standard input consists of tab-delimited text files containing the Ct values for a range of genes. Additional information, such as type of gene (e.g. target, endogenous control) or groupings of genes into separate classes (e.g. markers, kinases) can also be read in, or supplied later.
The package comes with example input files (from Applied Biosystem's TLDA cards), along with a text file listing sample file names and biological conditions.

```
> path <- system.file("exData", package = "HTqPCR")
> head(read.delim(file.path(path, "files.txt")))
```

```
         File  Treatment
1 sample1.txt    Control
2 sample2.txt LongStarve
3 sample3.txt LongStarve
4 sample4.txt    Control
5 sample5.txt     Starve
6 sample6.txt     Starve
```

The data consist of 192 features represented twice on the array and labelled "Gene1", "Gene2", etc. There are three different conditions, "Control", "Starve" and "LongStarve", each having 2 replicates.
The input data consists of tab-delimited text files (one per sample); however, the format is likely to vary depending on the specific platform on which the data were obtained (e.g., TLDA cards, 96-well plates, or some other format). The only requirement is that columns containing the Ct values and feature names are present.

```
> files <- read.delim(file.path(path, "files.txt"))
> raw <- readCtData(files = files$File, path = path)
```

The *qPCRset* object looks like:

```
> show(raw)
```

```
An object of class "qPCRset"
Size:  384 features, 6 samples
Feature types:
Feature names:                 Gene1 Gene2 Gene3 ...
Feature classes:
Feature categories:         OK, Undetermined
Sample names:                 sample1 sample2 sample3 ...
```

NB: This section only deals with data presented in general data format. For notes regarding other types of input data, see section 13. This section also briefly deals with other types of qPCR results besides Ct data, notably the Cp values reported by the LightCycler System from Roche.

4 Data visualisation

4.1 Overview of Ct values across all groups

To get a general overview of the data the (average) Ct values for a set of features across all samples or different condition groups can be displayed. In principle, all features in a sample might be chosen, but to make it less cluttered Figure 3 displays only the first 10 features. The top plot was made using just the Ct values, and shows the 95% confidence interval across replicates within and between samples. The bottom plot represents the same values but relative to a chosen calibrator sample, here the "Control". Confidence intervals can also be added to the relative plot, in which case these will be calculated for all values compared to the average of the calibrator sample per gene.

```
> g <- featureNames(raw)[1:10]
> plotCtOverview(raw, genes = g, xlim = c(0, 50), groups = files$Treatment,
+     conf.int = TRUE, ylim = c(0, 55))
```

```
> plotCtOverview(raw, genes = g, xlim = c(0, 50), groups = files$Treatment,
+     calibrator = "Control")
```

4.2 Spatial layout

When the features are organised in a particular spatial pattern, such as the 96- or 384-well plates, it is possible to plot the Ct values or other characteristics of the features using this layout. Figure 4 shows an example of the Ct values, as well as the location of different classes of features (using random examples here), across all the wells of a TLDA microfluidic card.

```
> plotCtCard(raw, col.range = c(10, 35), well.size = 2.6)
```

```
> featureClass(raw) <- factor(c("Marker", "TF", "Kinase")[sample(c(1,
+     1, 2, 2, 1, 3), 384, replace = TRUE)])
> plotCtCard(raw, plot = "class", well.size = 2.6)
```

4.3 Comparison of duplicated features within samples

When a sample contains duplicate measurements for some or all features, the Ct values of these duplicates can be plotted against each other to measure accordance between duplicates. In Figure 5 the duplicates in sample2 are plotted against each other, and those where the Ct values differ more than 20% from the average of a given feature are marked.

```
> plotCtReps(qPCRraw, card = 2, percent = 20)
Replicates differing > 20% on card 2:
             rep1     rep2
Gene135 40.000000 29.90044
Gene14  40.000000 27.69185
```

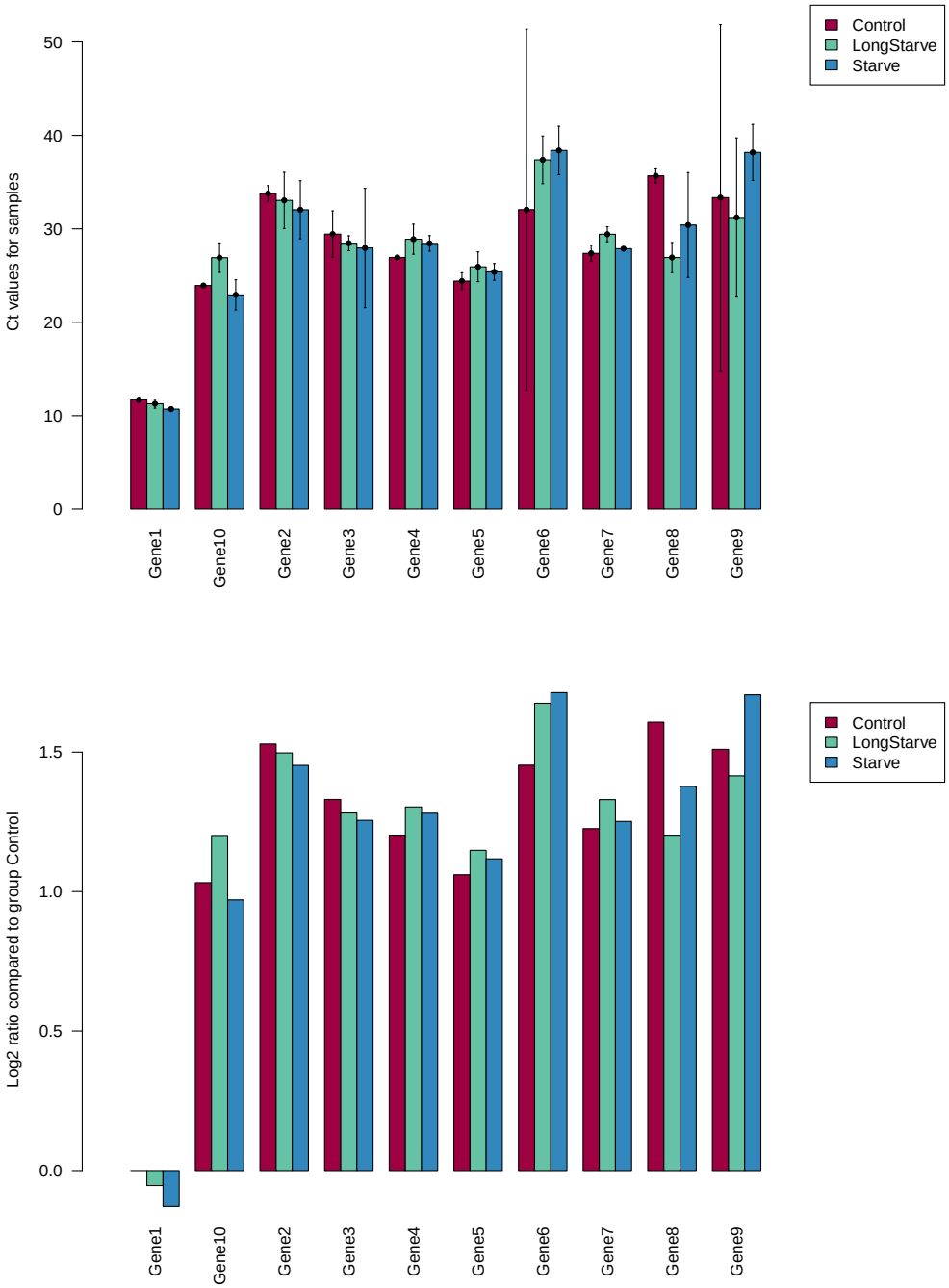
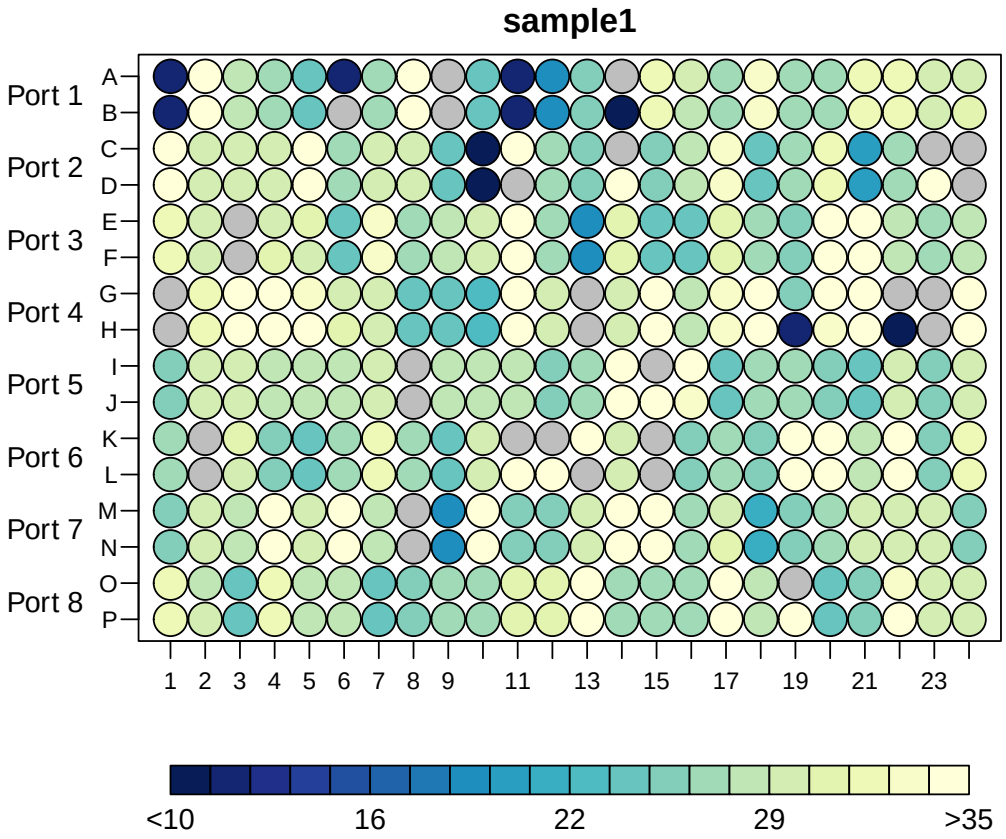
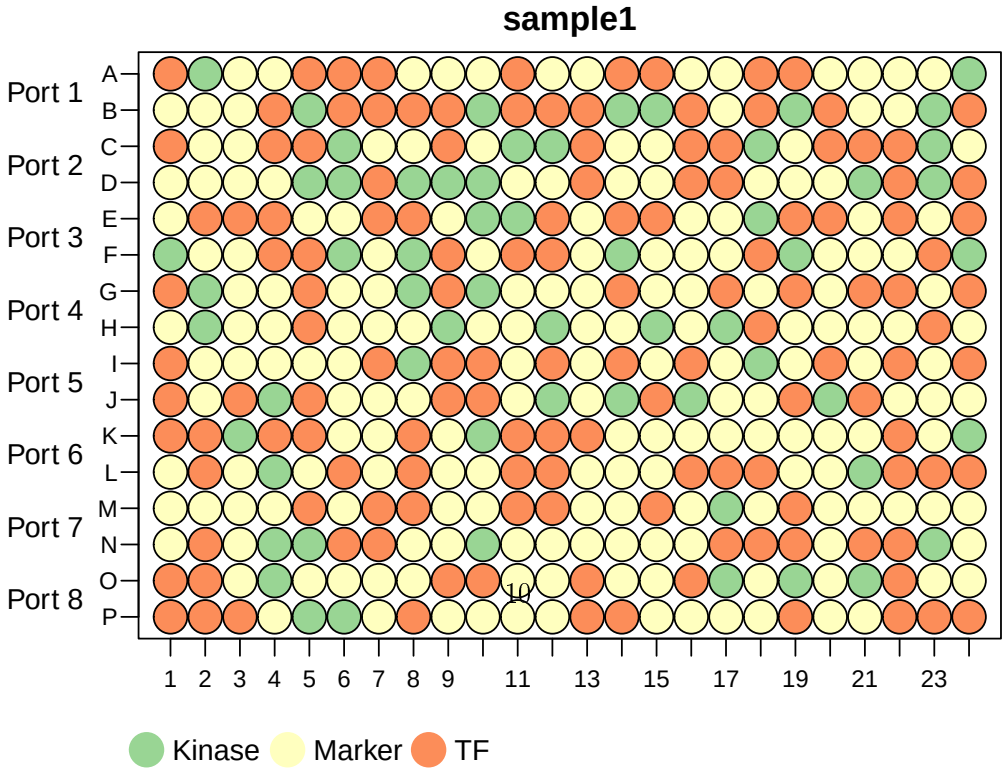


Figure 3: Overview of Ct values for the raw data.


sample1
Port 1
Port 2
Port 3
Port 4
Port 5
Port 6
Port 7
Port 8
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
1 2 3 4 5 6 7 8 9 11 13 15 17 19 21 23
<10
16
22
29
>35


sample1
Port 1
Port 2
Port 3
Port 4
Port 5
Port 6
Port 7
Port 8
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
1 2 3 4 5 6 7 8 9 11 13 15 17 19 21 23
Kinase
Marker
TF

```
Gene37   7.408248 40.00000
Gene73  40.000000 23.11949
Gene84  21.673946 40.00000
Gene86  20.389658 40.00000
Gene9   22.494440 31.39404
Gene95  15.916160 40.00000
```

Differences will often arise due to one of the duplicates marked as "Undetermined", thus contributing to an artificially high Ct value, but other known cases exist as well.

4.4 Variation within and across samples

In some cases more than two replicates are present, either within each qPCR card (feature replicates) or across cards (replicated samples). Assessing the variation within replicates can indicate whether some samples or individual features are less reliable, or if perhaps an entire qPCR card shows high variation between replicate features and needs to be discarded.
`plotCtVariation` generates a boxplot with all the variation values, either across genes or with each samples. That way the general distribution of variation or standard deviation values can be compared quickly (Figure 6). In this example, the variation across samples doesn't differ much.

```
> raw.mix <- raw
> plotCtVariation(raw.mix, variation = "sd", log = TRUE,
+     main = "SD of replicated features", col = "lightgrey")
```

If it looks like there's an unacceptable (or interesting) difference in the variation, this can be further investigated using the parameter `type="detail"`. This will generate multiple sub-plots, containing a single scatterplot of variation versus mean for each gene or sample (Figure 6). That way individual outliers can be identified, or whole samples removed by examining the resulting variation in more detail.

```
> raw.variation <- plotCtVariation(raw.mix, type = "detail",
+     add.featurenames = TRUE, pch = " ", cex = 1.2)


> names(raw.variation)
[1] "Var"  "Mean"
> head(raw.variation[["Var"]][, 1:4])
  Group.1      sample1      sample2      sample3
1   Gene1 6.603053e-02 3.367143e-02 4.340261e-02
2  Gene10 6.087458e-05 2.864660e-04 1.869050e-03
3 Gene100 1.232561e-05 7.789010e-04 4.452426e-04
4 Gene101 5.859356e-03 8.787857e-02 7.988004e-06
5 Gene102 6.571738e-03 7.440192e-02 2.768647e-03
6 Gene103 0.000000e+00 2.544768e-06 7.899120e-04
> head(raw.variation[["Mean"]][, 1:4])
```

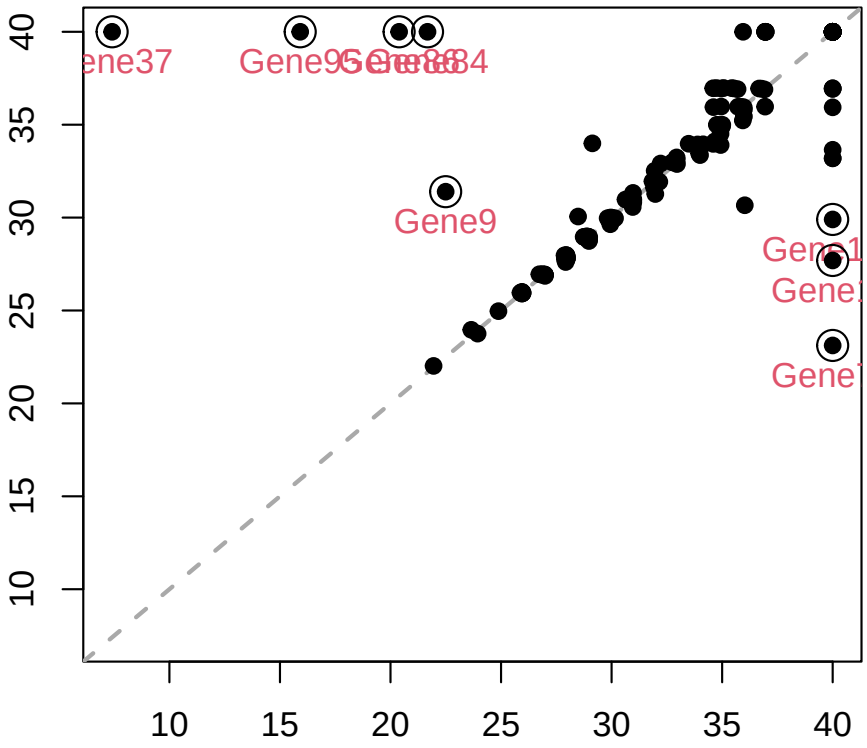


Figure 5: Concordance between duplicated Ct values in sample 2, marking features differing >20% from their mean.

```
  Group.1  sample1  sample2  sample3
1   Gene1 11.70414 11.78938 10.76248
2  Gene10 23.93387 25.92385 27.88952
3 Gene100 27.93282 31.92726 30.92547
4 Gene101 27.88928 34.71769 28.93203
5 Gene102 28.91345 30.76310 28.90544
6 Gene103 40.00000 30.92895 29.90892
> apply(raw.variation[["Var"]][, 3:7], 2, summary)
             sample2      sample3      sample4      sample5
Min.    0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
1st Qu. 3.673657e-04 7.036888e-04 3.073682e-04 3.466089e-04
Median  4.805137e-03 5.065715e-03 6.370044e-03 6.630107e-03
Mean    8.666670e+00 4.440271e+00 7.155378e+00 6.887166e+00
3rd Qu. 6.221935e-02 3.446367e-02 3.648247e-02 3.863380e-02
Max.    5.311111e+02 5.715739e+02 4.675274e+02 3.328081e+02
             sample6
Min.    0.000000e+00
1st Qu. 5.975255e-04
Median  6.346025e-03
Mean    5.628291e+00
3rd Qu. 3.671908e-02
Max.    5.627668e+02
> colSums(raw.variation[["Var"]][, 3:7] > 20)
sample2 sample3 sample4 sample5 sample6
     11       3       7       6       8
```

In the example in this section a lot of features from sample 6 have intra-replicate variation above an arbitrary threshold selected based on Figure 6, and the mean and median values are much higher than for the remaining samples. Sample 1 is excluded due to the page width.
Variation across Ct values is discussed further in the following section regarding filtering.

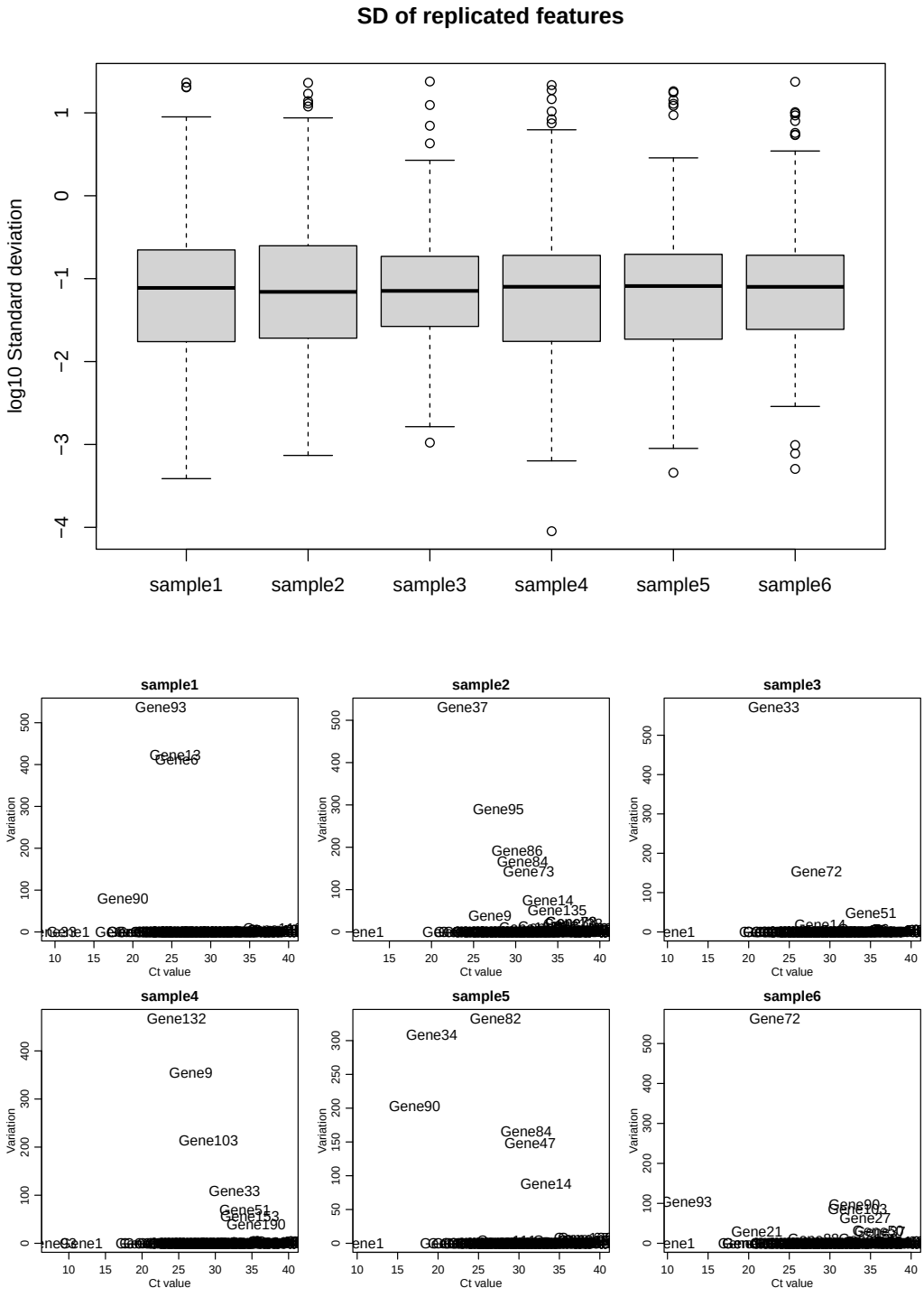


Figure 6: (Top) summary of standard deviation between replicated features within each of the six samples. (Bottom) Variation versus mean for replicated features.

5 Feature categories and filtering

Each Ct values in HTqPCR has an associated feature category. This is an important component to indicate the reliability of the qPCR data. Aside from the "OK" indicator, there are two other categories: "Undetermined" is used to flag Ct values above a user-selected threshold, and "Unreliable" indicates Ct values that are either so low as to be estimated by the user to be problematic, or that arise from deviation between individual Ct values across replicates. By default, only Ct values labelled as "undetermined" in the input data files are placed into the "Undetermined" category, and the rest are classified as "OK". However, either before or after normalisation these categories can be altered depending on various criteria.

Range of Ct values Some Ct values might be too high or low to be considered a reliable measure of gene expression in the sample, and should therefore not be marked as "OK".

Flags Depending on the qPCR input the values might have associated flags, such as "Passed" or "Failed", which are used for assigning categories.

Biological and technical replicates If features are present multiple times within each sample, or if samples are repeated in the form of technical or biological replicates, then these values can be compared. Ct values lying outside a user-selected confidence interval (90% by default) will be marked as "Unreliable".

A summary plot for the sample categories is depicted in Figure 7. The result can be stratified by `featureType` or `featureClass`, for example to determine whether one class of features performed better or worse than others.

```
> raw.cat <- raw
> plotCtCategory(raw.cat)
```

```
> plotCtCategory(raw.cat, stratify = "class")
```

The results can also be shown per feature rather than averaged across samples (Figure 8).

```
> plotCtCategory(raw.cat, by.feature = TRUE, cexRow = 0.1)
```

If one doesn't want to include unreliable or undetermined data in part of the analysis, these Ct values can be set to NA using `filterCategory`. However, the presence of NAs could make the tests for differential expression less robust. When testing for differential expression the result will come with an associated category ("OK" or "Unreliable") that can instead be used to assess the quality of the results. For the final results both "Undetermined" and "Unreliable" are pooled together as being "Unreliable". However, the label for each feature can either be set according to whether half or more of the samples are unreliable, or whether only a single non-"OK" category is present, depending on the level of stringency the user wishes to enforce.

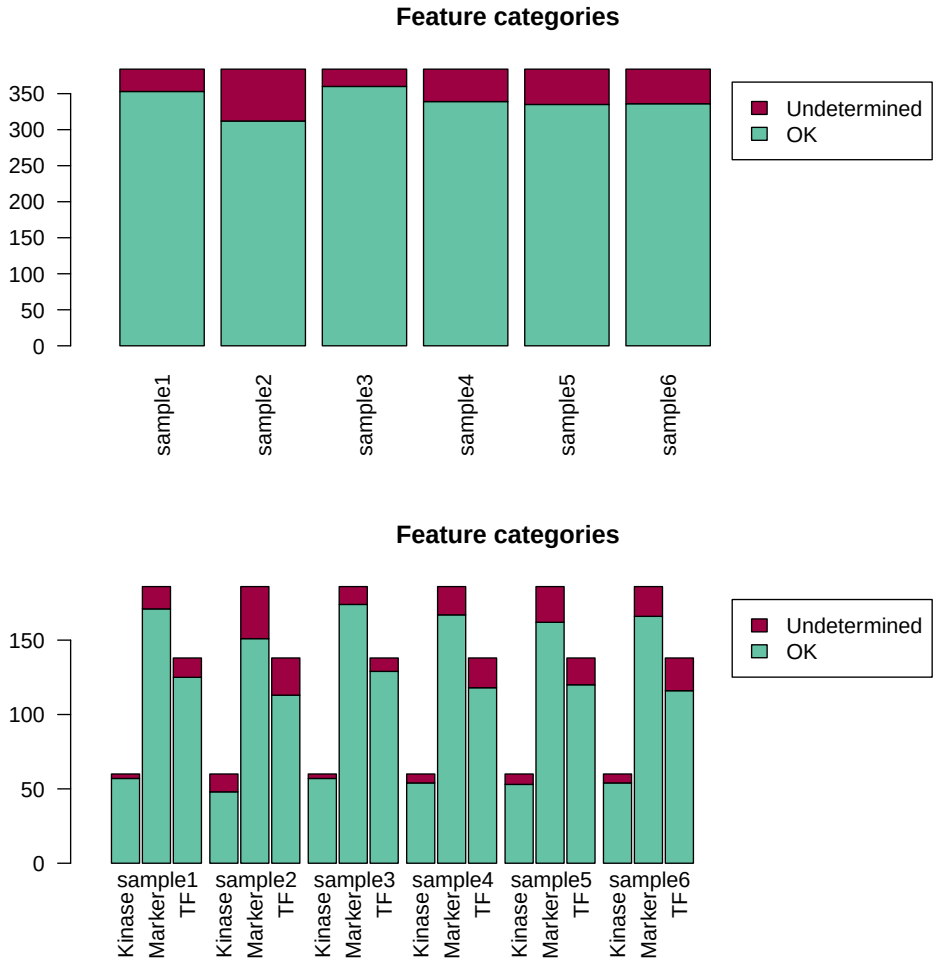


Figure 7: Summary of the categories, either for each sample individually or stratified by feature class.

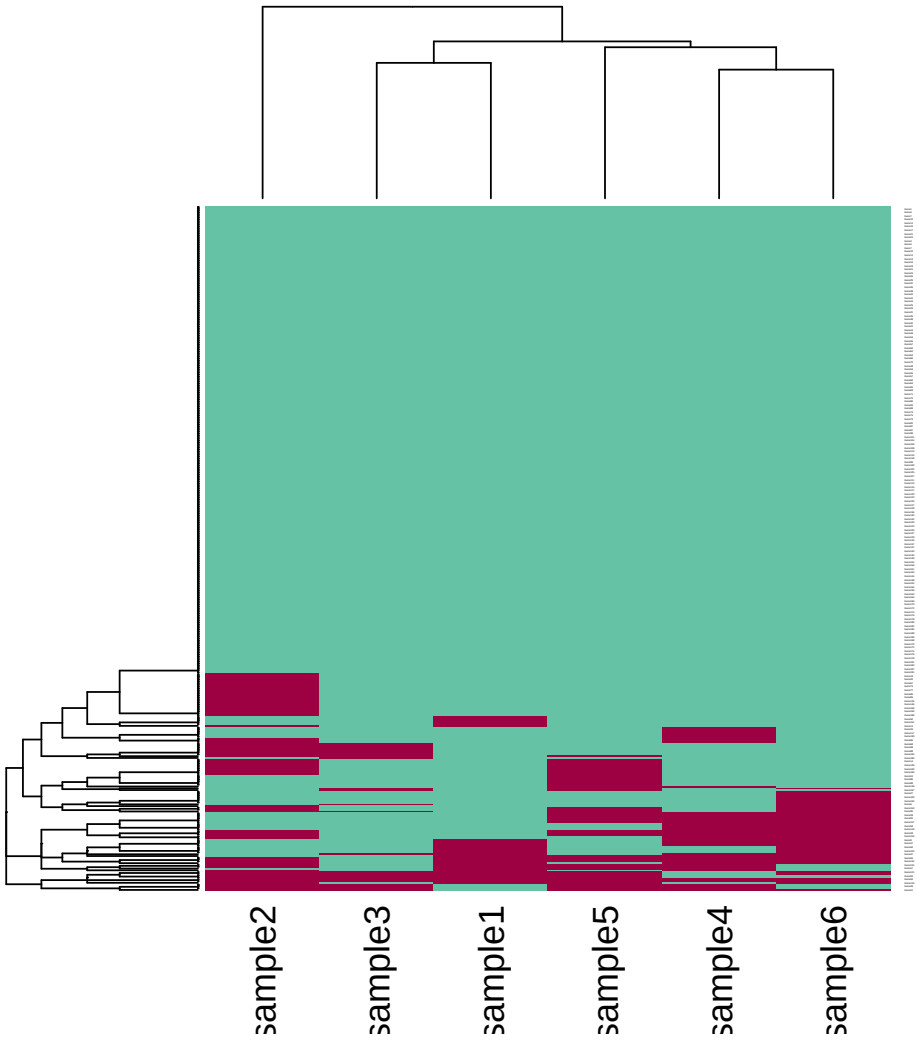


Figure 8: Summary of the categories, clustered across features.

6 Normalisation

Five different normalisation methods are currently implemented in *HTqPCR*. Three of these (`scale.rankinvariant`, `deltaCt` and `geometric.mean`) will scale each individual sample by a given value, whereas the remaining two will change the distribution of Ct values.

quantile Will make the distribution of Ct values more or less identical across samples.

norm.rankinvariant Computes all rank-invariant sets of features between pairwise com- parisons of each sample against a reference, such as a pseudo-mean. The rank-invari- ant features are used as a reference for generating a smoothing curve, which is then applied to the entire sample.

scale.rankinvariant Also computes the pairwise rank-invariant features, but then takes only the features found in a certain number of samples, and used the average Ct value of those as a scaling factor for correcting all Ct values.

deltaCt Calculates the standard deltaCt values, i.e. subtracts the mean of the chosen controls from all other values in the feature set.

geometric.mean Calculates the average Ct value for each sample, and scales all Ct values according to the ratio of these mean Ct values across samples. There are some indications that this is beneficial for e.g. miRNA studies.

For the rank-invariant normalisation and geometric mean methods, Ct values above a given threshold can be excluded from the calculation of a scaling factor or normalisation curve. This is useful so that a high proportion of "Undetermined" Ct values (assigned a value of 40 by default) in a given sample doesn't bias the normalisation of the remaining features.
In the example dataset, Gene1 and Gene60 correspond to 18S RNA and GADPH, and are used as endogenous controls. Normalisation methods can be run as follows:

```
> q.norm <- normalizeCtData(raw.cat, norm = "quantile")
> sr.norm <- normalizeCtData(raw.cat, norm = "scale.rank")
Scaling Ct values
        Using rank invariant genes: Gene1 Gene29
        Scaling factors: 1.00 1.06 1.00 1.03 1.00 1.00
> nr.norm <- normalizeCtData(raw.cat, norm = "norm.rank")
Normalizing Ct values
        Using rank invariant genes:
        sample1: 71 rank invariant genes
        sample2: 43 rank invariant genes
        sample3: 41 rank invariant genes
        sample4: 76 rank invariant genes
        sample5: 22 rank invariant genes
        sample6: 62 rank invariant genes
> d.norm <- normalizeCtData(raw.cat, norm = "deltaCt",
+     deltaCt.genes = c("Gene1", "Gene60"))
```

```
Calculating deltaCt values
        Using control gene(s): Gene1 Gene60
        Card 1:         Mean=14.45        Stdev=4.25
        Card 2:         Mean=15.19        Stdev=5.27
        Card 3:         Mean=14.5        Stdev=5.79
        Card 4:         Mean=14.79        Stdev=4.79
        Card 5:         Mean=14.07        Stdev=5.32
        Card 6:         Mean=13.82        Stdev=4.75
> g.norm <- normalizeCtData(raw.cat, norm = "geometric.mean")
Scaling Ct values
        Using geometric mean within each sample
        Scaling factors: 1.00 1.06 1.05 1.02 1.04 1.02
```

Comparing the raw and normalised values gives an idea of how much correction has been performed (Figure 9), as shown below for the `q.norm` object. Note that the scale on the y-axis varies.

```
> plot(exprs(raw), exprs(q.norm), pch = 20, main = "Quantile normalisation",
+     col = rep(brewer.pal(6, "Spectral"), each = 384))
```

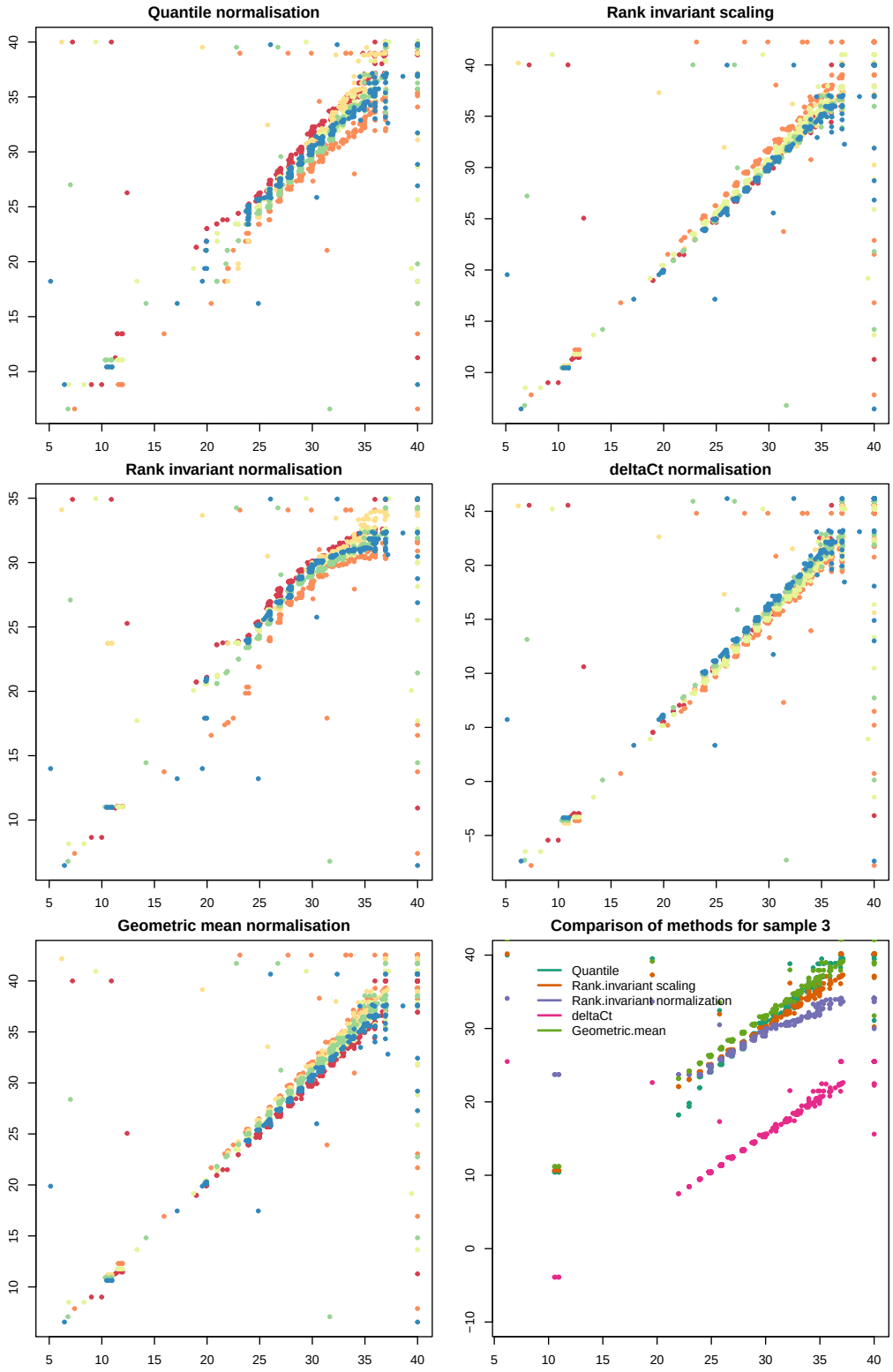


Figure 9: Normalized versus raw data, using a separate colour for each sample. The raw data is plotted along the x-axis and the normalised along y. The last plot is a comparison between normalization methods for the third sample, still with the raw Ct values along the x-axis.

7 Filtering and subsetting the data

At any point during the analysis it's possible to filter out both individual features or groups of features that are either deemed to be of low quality, or not of interest for a particular aspect of the analysis. This can be done using any of the feature characteristics that are included in the `featureNames`, `featureType`, `featureClass` and/or `featureCategory` slots of the data object. Likewise, the `qPCRset` object can be turned into smaller subsets, for example if only a particular class of features are to be used, or some samples should be excluded.
Simple subsetting can be done using the standard `[,]` notation of R, for both rows (genes) and columns (samples).

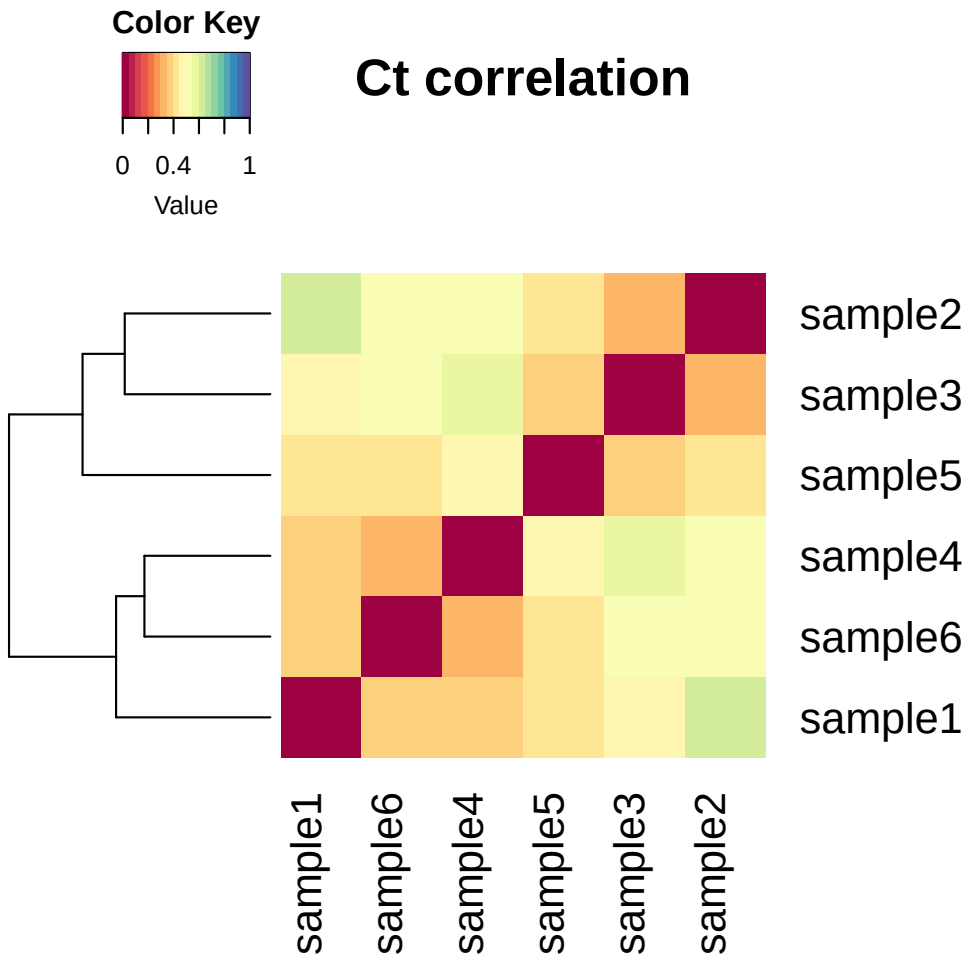


Figure 10: Correlation between raw Ct values.

8 Quality assessment

8.1 Correlation between samples

The overall correlation between different samples can be displayed visually, such as shown for the raw data in Figure 10. Per default, 1 minus the correlation is plotted.

```
> plotCtCor(raw, main = "Ct correlation")
```

8.2 Distribution of Ct values

It may be of interest to examine the general distribution of data both before and after normalisation. A simple summary of the data can be obtained using `summary` as shown below.

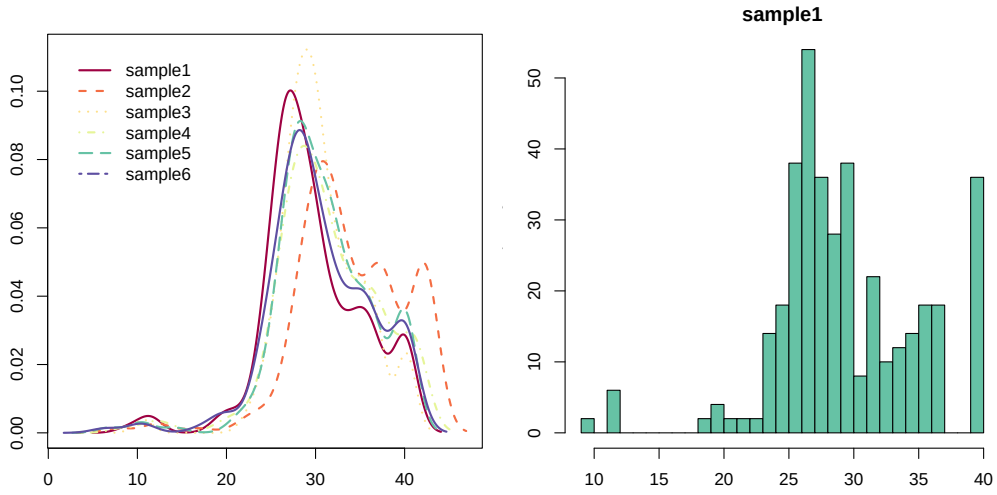


Figure 11: Distribution of Ct values for the individual samples, either using the density of all arrays (left) or a histogram of a single sample (right), after scale rank-invariant normalisation.

```
> summary(raw)
        sample1  sample2  sample3 sample4  sample5  sample6
Min.    " 7.218" " 7.408" " 6.19" " 6.853" " 6.787" " 5.133"
1st Qu. "26.738" "28.855" "27.90" "26.964" "27.913" "27.514"
Median  "28.937" "30.994" "29.92" "29.943" "30.778" "29.931"
Mean    "29.542" "32.190" "30.35" "30.590" "30.995" "30.663"
3rd Qu. "33.323" "35.985" "32.98" "34.694" "34.702" "35.046"
Max.    "40.000" "40.000" "40.00" "40.000" "40.000" "40.000"
```

However, figures are often more informative. To that end, the range of Ct values can be illustrated using histograms or with the density distribution, as shown in Figure 11.

```
> plotCtDensity(sr.norm)
```

```
> plotCtHistogram(sr.norm)
```

Plotting the densities of the different normalisation methods lends insight into how they differ (Figure 12).
Ct values can also be displayed in boxplots, either with one box per sample or stratified by different attributes of the features, such as `featureClass` or `featureType` (Fig. 13).

```
> plotCtBoxes(sr.norm, stratify = "class")
```

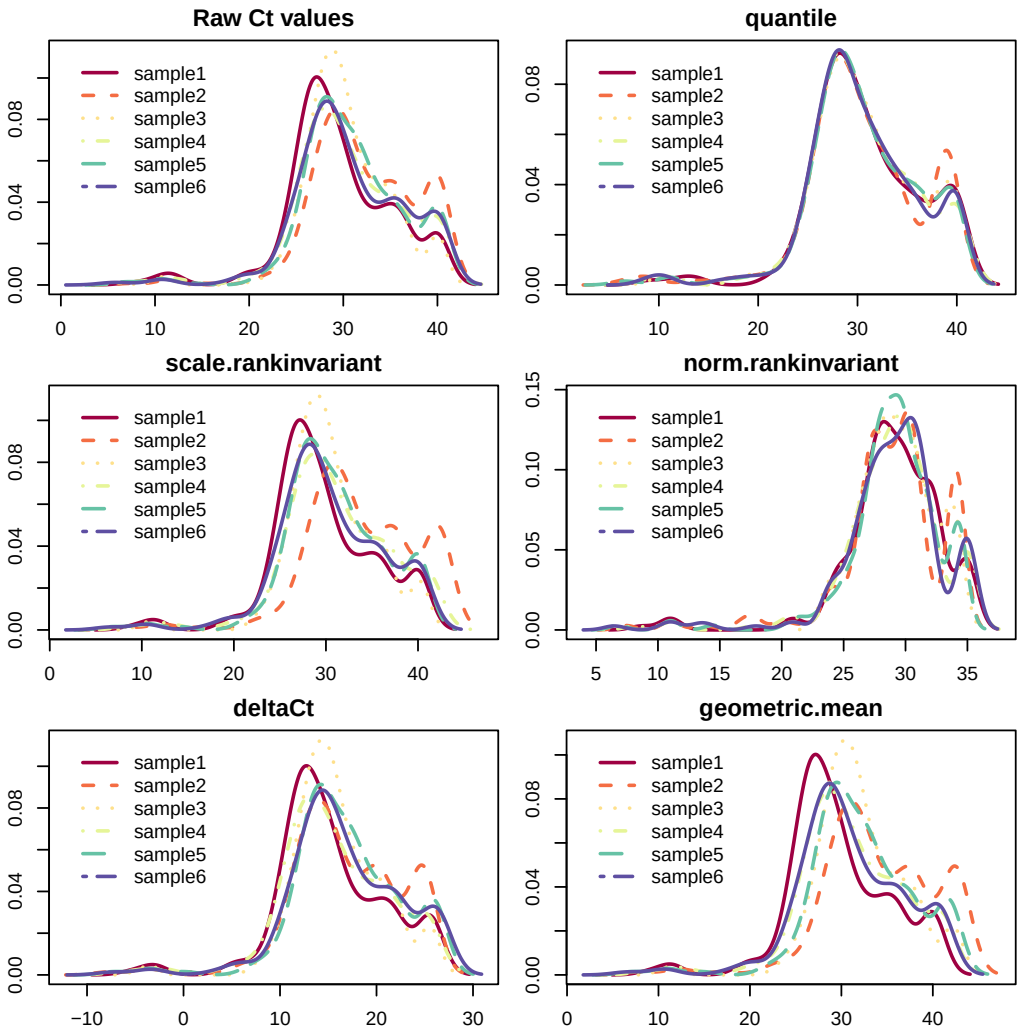


Figure 12: Densities of Ct values for all samples before and after each of the normalisation methods. The peak at the high end originates from features with "Undetermined" Ct values, which are assigned the Ct value 40 by default.

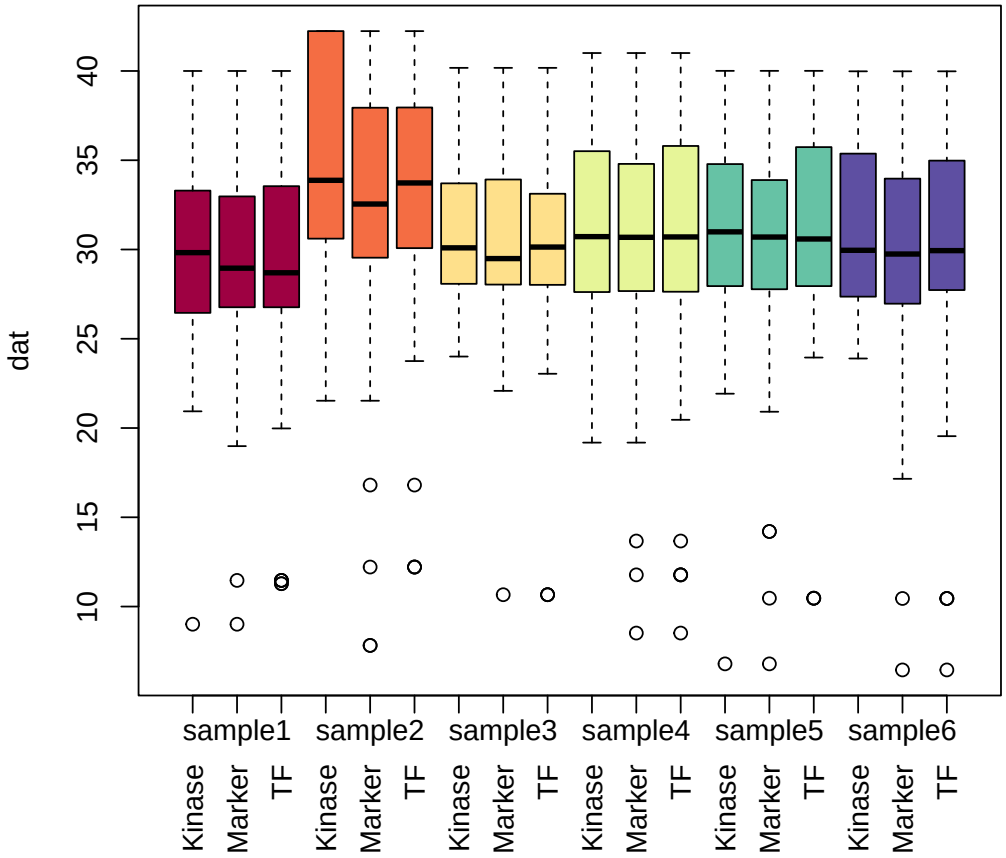


Figure 13: Boxplot of Ct values across all samples, stratified by feature classes.

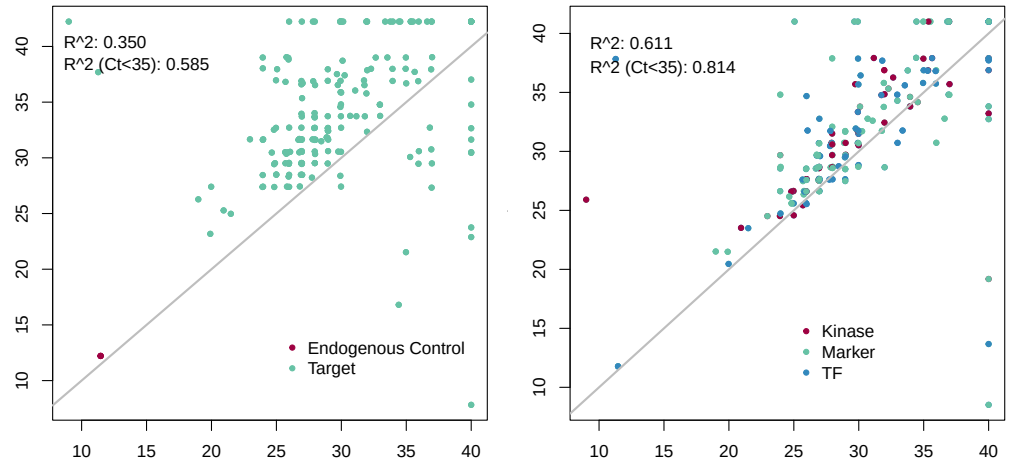


Figure 14: Scatter plot of Ct values in different samples, with points marked either by featureType (left) or featureClass (right) and the diagonal through $x = y$ marked with a grey line.

8.3 Comparison of Ct values for two samples

It is often of interest to directly compare Ct values between two samples. In Figure 14, two examples are shown for the rank-invariant normalised data: one for different biological samples, and one for replicates.

```
> plotCtScatter(sr.norm, cards = c(1, 2), col = "type",
+     diag = TRUE)

> plotCtScatter(sr.norm, cards = c(1, 4), col = "class",
+     diag = TRUE)
```

8.4 Scatter across all samples

It is also possible to generate a scatterplot of Ct values between more than the two samples shown above. In Figure 15 all pairwise comparisons are shown, along with their correlation when all Ct values <35 are removed.

```
> plotCtPairs(sr.norm, col = "type", diag = TRUE)
```

8.5 Ct heatmaps

Heatmaps provide a convenient way to visualise clustering of features and samples at the same time, and show the levels of Ct values (Figure 16). The heatmaps can be based on either Pearson correlation coefficients or Euclidean distance clustering. Euclidean-based heatmaps will focus on the magnitude of Ct values, whereas Pearson clusters the samples based on similarities between the Ct profiles.

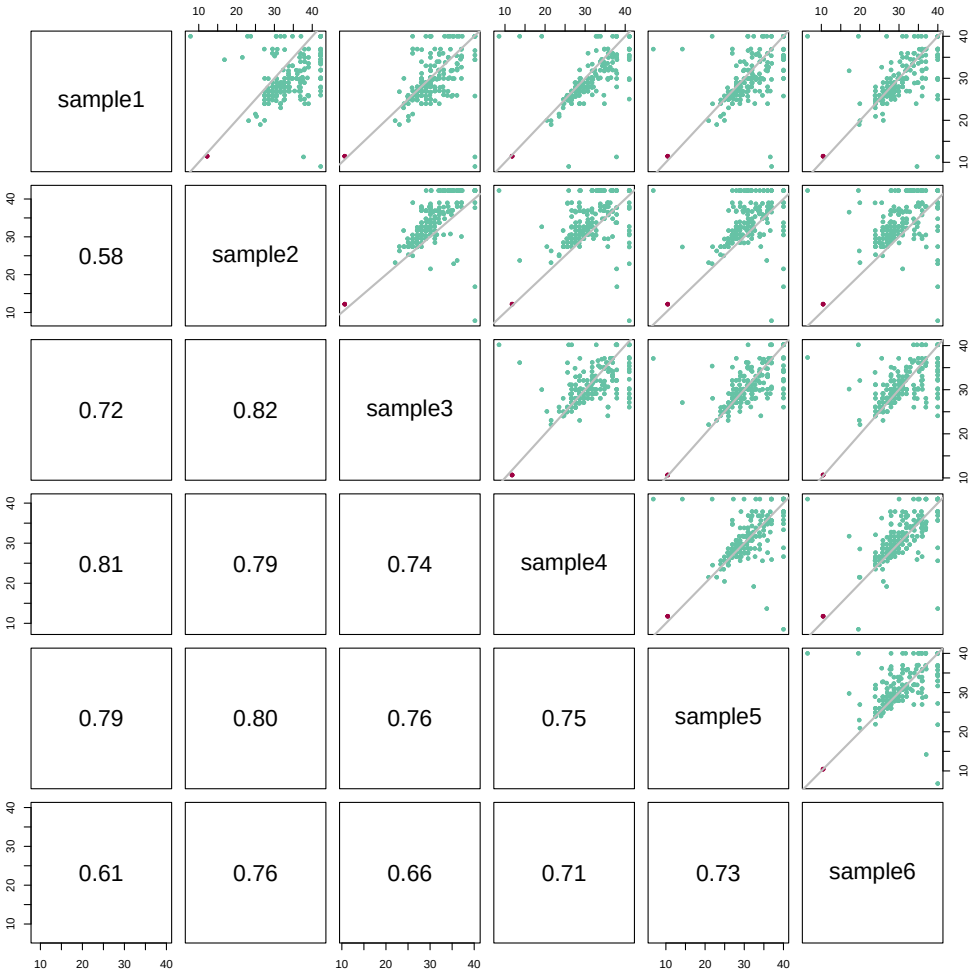


Figure 15: Scatterplot for all pairwise comparisons between samples, with spots marked depending on `featureType`, i.e. whether they represent endogenous controls or targets.

```
> plotCtHeatmap(raw, gene.names = "", dist = "euclidean")
```

8.6 Coefficients of variation

The coefficients of variation (CV) can be calculated for each feature across all samples. Stratifying the CV values by `featureType` or `featureClass` can help to determine whether one class of features is more variable than another (Figure 17). For the example data feature classes have been assigned randomly, and the CVs are therefore similar, whereas for the feature types there's a clear difference between controls and targets.

```
> plotCVBoxes(qPCRraw, stratify = "class")
> plotCVBoxes(qPCRraw, stratify = "type")
```

9 Clustering

At the moment there are two default methods present in *HTqPCR* for clustering; hierarchical clustering and principal components analysis (PCA).

9.1 Hierarchical clustering

Both features and samples can be subjected to hierarchical clustering using either Euclidean or Pearson correlation distances, to display similarities and differences within groups of data. Individual subclusters can be selected, either using pre-defined criteria such as number of clusters, or interactively by the user. The content of each cluster is then saved to a list, to allow these features to be extracted from the full data set if desired.
An example of a clustering of samples is shown in Figure 18. In Figure 19 these data are clustered by features, and the main subclusters are marked.

```
> clusterCt(sr.norm, type = "samples")
```

```
> cluster.list <- clusterCt(sr.norm, type = "genes",
+     n.cluster = 6, cex = 0.5)
```

9.2 Principal components analysis

PCA is performed across the selected features and samples (observations and variables), and can be visualized either in a biplot, or showing just the clustering of the samples (Figure 20).

```
> plotCtPCA(qPCRraw)
> plotCtPCA(qPCRraw, features = FALSE)
```

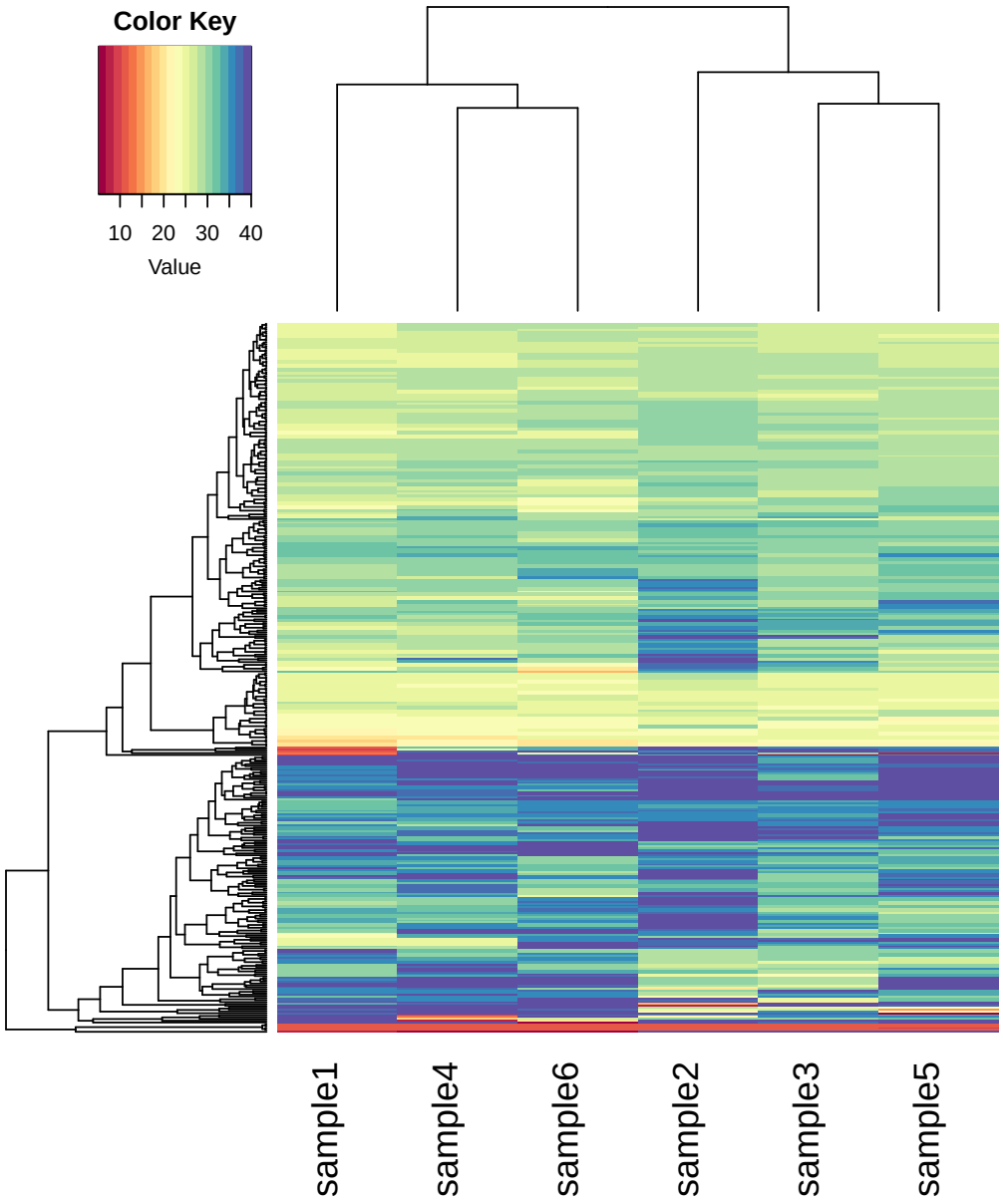


Figure 16: Heatmap for all samples and genes, based on the Euclidean distance between Ct values.

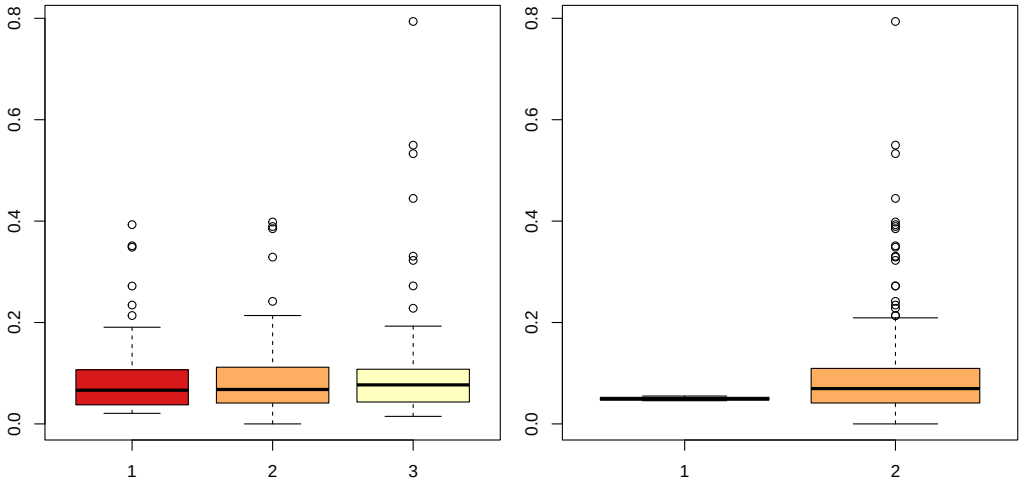


Figure 17: Coefficients of variation for each feature across all samples.

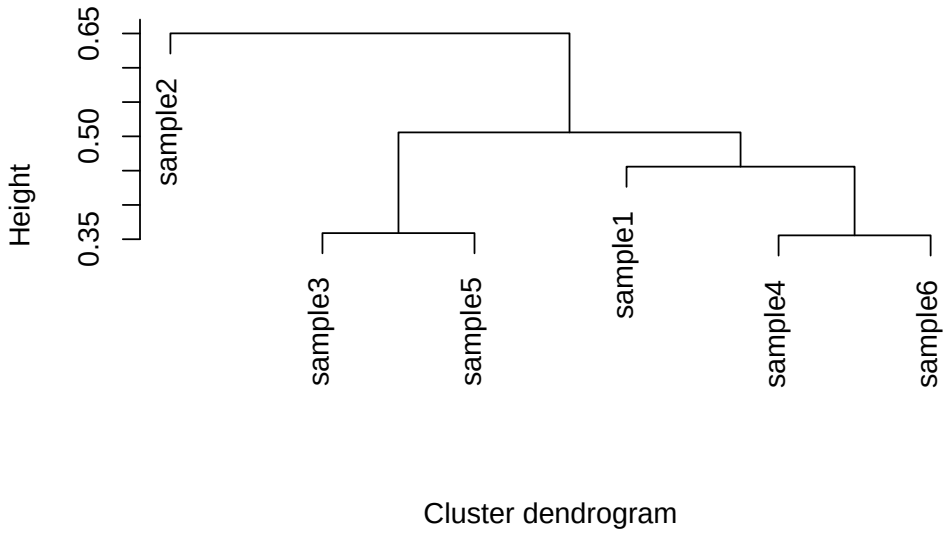


Figure 18: Hierarchical clustering of samples.

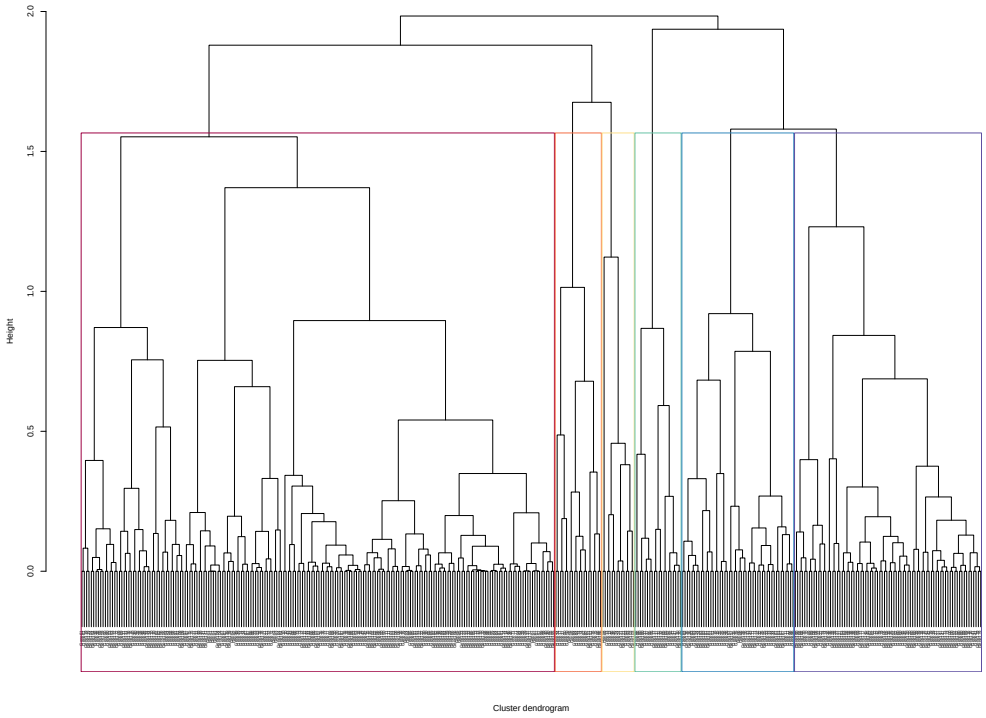


Figure 19: Hierarchical clustering of features, with subclusters marked.

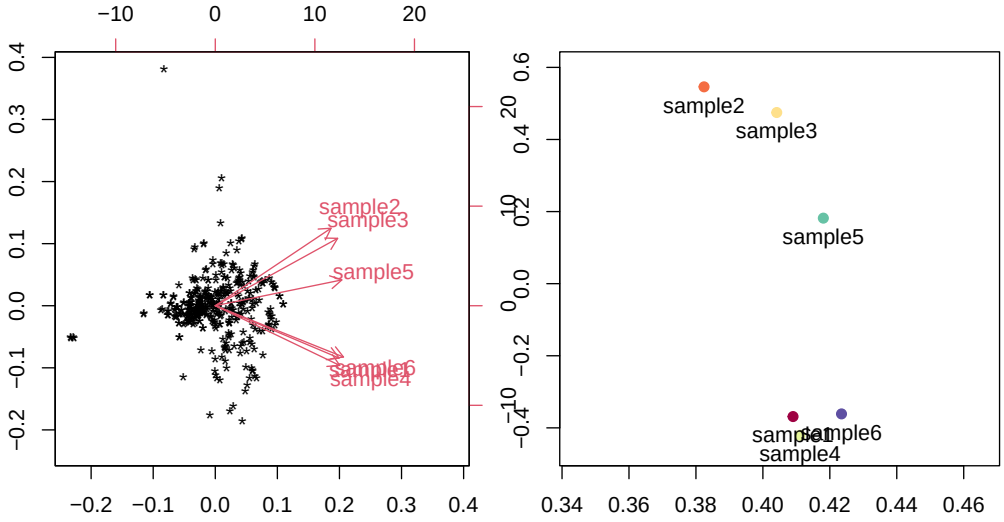


Figure 20: Left: a biplot including all features, with samples represented by vectors. Right: the same plot, including only the samples.

10 Differential expression

At this stage multiple filterings might have been performed on the data set(s). To remind yourself about those, you can use the `getCtHistory` function on the *qPCRset* object.

```
> getCtHistory(sr.norm)
                                                history
1       readCtData(files = files$File, path = path)
2 normalizeCtData(q = raw.cat, norm = "scale.rank")
```

In general there are three approaches in *HTqPCR* for testing the significance of differences in Ct values between samples.

t-test Performing a standard t-test between two sample groups. This function will incorporate information about replicates to calculate t and p-values. This is a fairly simple approach, typically used when comparing a single treatment and control sample, and multiple pair-wise tests can be carried out one-by-one by the user.

Mann-Whitney This is a non-parametric test, also known as a two sample Wilcoxon test. Similar to the t-test, multiple pair-wise tests will have to be carried out one by one if more than two types of samples are present. This is a rank-based test that doesn't make any assumptions about the population distribution of Ct values.

limma Using a wrapper for functions from the package *limma*to calculate more sophisticated t and p-values for any number of groups present across the experiment. This is more flexible in terms of what types of comparisons can be made, but the users need to familiarise themselves with the *limma* conventions for specifying what contrasts are of interest.

Examples of how to use each of these are given in the next sections. In all cases the output is similar; a data frame containing the test statistics for each feature, along with fold change and information about whether the Ct values are "OK" or "Unreliable". This result can be written to a file using standard functions such as `write.table`.

10.1 Two sample types - t-test

This section shows how to compare two samples, e.g. the control and long starvation samples from the example data. A subset of the `qPCRset` data is created to encompass only these samples, and a t-test is then performed. See '?ttestCtData' for examples.

10.2 Two sample types - Mann-Whitney

When only two samples are of interest, these can also be compared using a Mann-Whitney test. As is the section above, the function is demonstrated using the control and long starvation samples from the example data. A subset of the `qPCRset` data is created to encompass only these samples, and a Mann-Whitney test is performed. See '?mannwhitneyCtData' for examples.

10.3 Multiple sample types - limma

Methods taken from *limma* can be used to compare either two or multiple sample types. In this example all three types of treatment are compared, as well as the control against both starvation samples combined. The data is sorted by feature names, to make easier use of replicated features. See '?limmaCtData' for examples.
The result is a list with one component per comparison. Each component is similar to the result from using `ttestCtData`.
Furthermore, there is a "Summary" component at the end where each feature is denoted with -1, 0 or 1 to respectively indicate down-regulation, no change, or up-regulation in each of the comparisons.

11 Displaying the results

The results can be visualised using the generic `plotCtOverview` shown in Figure 3. However, *HTqPCR* also contains more specialised functions, for example to include information about whether differences are significant or not.

11.1 Fold changes: Relative quantification

The relative Ct levels between two groups can be plotted with the function `plotCtRQ`. Below are two examples: one the result of `ttestCtData` where the top 15 genes are selected and another from the first comparison in `limmaCtData` where all genes below a certain p-value are depicted.

11.2 Fold changes: Detailed visualisation

In some cases it will be beneficial to more closely examine individual Ct data points from the fold changes, partly to look at the data dispersion, and partly to determine which of these values are in the "OK" versus "Unreliable"/"Undetermined" category. The function `plotCtSignificance` will take the result of `ttestCtData` or `limmaCtData`, along with the input data to these functions, and display a combined barplot showing the individual data points and marking those comparisons with a significant p-value.

11.3 Heatmap across comparisons

When multiple conditions are compared with `limmaCtData`, the fold changes from all comparisons can be compared to see if features cluster together in groups.

12 Manipulating qPCRset objects

Depending on the design of the qPCR card and how the data is to be analysed, it will sometimes be necessary to manipulate the **qPCRset** objects in different ways, such as by combining several objects or altering the layout of features x samples.

12.1 Multiple samples present on each plate

The result from each qPCR run of a given card typically gets presented together, such as in a file with 384 lines, one per feature, for 384 well plates. However, some cards may contain multiple samples, such as commercial cards that are designed to be loaded with two separate samples and then include 192 individual features.
Per default, **readCtData** reads each card into a *qPCRset* object as consisting of a single sample, and hence one column in the Ct data matrix. When this is not the case, the data can subsequently be split into the correct features x samples (rows x columns) dimensions using the function **changeCtLayout**. The parameter **sample.order** is a vector, that for each feature in the *qPCRset* indicates what sample it actually belongs to, and reformats all the information in the *qPCRset* (**exprs**, **featureCategory**, **flag** etc.) accordingly. The actual biological samples are likely to differ on each card, so **sample.order** merely indicates the *location* of different samples among the features present in the input data.

```
> sample2.order <- rep(c("subSampleA", "subSampleB"),
+     each = 192)
> sample4.order <- rep(c("subA", "subB", "subC", "subD"),
+     each = 96)
> qPCRnew2 <- changeCtLayout(sr.norm, sample.order = sample2.order)
> show(qPCRnew2)
An object of class "qPCRset"
Size:  192 features, 12 samples
Feature types:
Feature names:                 Gene1 Gene2 Gene3 ...
Feature classes:
Feature categories:         OK, Undetermined
Sample names:                 subSampleA:sample1 subSampleA:sample2 subSampleA:sample3 ...
> qPCRnew4 <- changeCtLayout(sr.norm, sample.order = sample4.order)
> show(qPCRnew4)
An object of class "qPCRset"
Size:  96 features, 24 samples
Feature types:
Feature names:                 Gene1 Gene2 Gene3 ...
Feature classes:
Feature categories:         OK, Undetermined
Sample names:                 subA:sample1 subA:sample2 subA:sample3 ...
```

As with the other functions that manipulate the Ct data, the operation is stored in the **history** slot of the *qPCRset* for future reference.

```
> getCtHistory(qPCRnew4)
                                                    history
1                          Manually created qPCRset object.
2 changeCtLayout(q = sr.norm, sample.order = sample4.order)
```

12.2 Combining multiple qPCRset objects

In some cases it might be desirable to merge multiple qPCRset objects, that have been read into R or processed individually. The *HTqPCR* package contains two functions for combining multiple `qPCRset` objects into one, by either adding columns (samples) or rows (features). This can be done for either identical samples across multiple different cards (such as a 384 well plate), or if more samples have been run on cards with the same layout.

`cbind` combines data assuming that all experiments have been carried out on identical cards, i.e. that `featureNames, featureType, featurePos` and `featureClass` is identical across all the qPCRset objects. The number of features in each object much be identical, but number of samples can vary.

`rbind` combines data assuming that the same samples have been analysed using different qPCR cards. The number of samples in each object must be identical, but the number of features can vary.

Both these functions should be used with some care; consider e.g. whether to normalize before or after joining the samples, and what method to use.
In the examples here objects with different normalisation are combined, although in a real study the `qPCRset` objects would typically contain different data.
As with other functions where the `qPCRset` object is being manipulated, the information is stored in the `history` slot.

13 How to handle different input data

General information about how to read qPCR data into a *qPCRset* object is presented in section 3. Below, some more specific cases are illustrated. Functions specifically designed for such qPCR platforms are still on a test stage in *HTqPCR*, and will be expanded (/corrected/modified) depending on demand.

13.1 Sequence Detection Systems format

The qPCR data might come from Sequence Detection Systems (SDS) software. The is supplied with most instruments from Applied Biosystems, but the software can also be used for assays from other vendors, such as the miRCURY LNA Universal RT microRNA PCR system from Exiqon.
For SDS output, each file has a header containing some generic information about the initial Ct detection. This header varies in length depending on how many files were analysed simultaneously, and an example is shown below.

```
> path <- system.file("exData", package = "HTqPCR")
> cat(paste(readLines(file.path(path, "SDS_sample.txt"),
+     n = 19), "\n"))
SDS 2.3 RQ Results        1.2
 Filename        Testscreen analys all.sdm
 Assay Type        RQ Study
 EmbeddedFile        FileA
 Run DateTime        Fri May 15 17:10:28 BST 2009
 Operator
 ThermalCycleParams
 EmbeddedFile        FileB
 Run DateTime        Sat May 16 10:36:09 BST 2009
 Operator
 ThermalCycleParams
 EmbeddedFile        FileC
 Run DateTime        Sun May 17 13:21:05 BST 2009
 Operator
 ThermalCycleParams

 #         Plate        Pos        Flag        Sample        Detector        Task        Ct
 1        Control        A1        Passed        Sample01        Gene1        Endogenous Contr
 2        Control        A2        Passed        Sample01        Gene2        Target        33
```

Only the first 7 columns are shown, since the file shown here contains >30 columns (of which many are empty). All columns for the first 20 lines can be seen in an R terminal with the command:

```
> readLines(file.path(path, "SDS_sample.txt"), n = 20)
```

For these files the parameter `for="SDS"` can be set in `readCtData`. The first 100 lines of each file will be scanned, and all lines preceding the actual data will be skipped (in this case 17), even when the length of the header varies between files.

```
> path <- system.file("exData", package = "HTqPCR")
> raw <- readCtData(files = "SDS_sample.txt", path = path,
+     format = "SDS")
> show(raw)
An object of class "qPCRset"
Size:  384 features, 1 samples
Feature types:
Feature names:                 Gene1 Gene2 Gene3 ...
Feature classes:
Feature categories:         OK, Undetermined
Sample names:                 SDS_sample ...
```

13.2 LightCycler format

Some qPCR systems, such as the LightCycler from Roche, don't provide the results as Ct values, but instead as crossing points (Cp). Ct values are measured at the exponential phase of amplification by drawing a line parallel to the x-axis of the real-time fluorescence intensity curve (fit point method), whereas Cp (second derivative method) calculates the fractional cycle where the second derivative of the real-time fluorescence intensity curve reaches the maximum value.
As long as all samples and features are quantified using the same method, it shouldn't matter whether Ct or Cp values are being used to test for significant differences between samples. The analysis of e.g. LightCycler Cp data can therefore proceed as outlined in this vignette. One thing to bear in mind though, is that for Ct values a value of 40 generally means NA, and above 35 is considered unreliable, however these numbers will be different for Cp. When using the filtering methods to set results as being "OK", "Unreliable" and "Undetermined", the parameters in e.g. `setCategory(..., Ct.max = 35, Ct.min = 10, ...)` and `readCtData(..., na.value, ...)` will need to be adjusted accordingly.
HTqPCR contains example data from the LightCycler 480 Real-Time PCR System, however not all wells were used during that particular experiment.

```
> path <- system.file("exData", package = "HTqPCR")
> raw <- readCtData(files = "LightCycler_sample.txt",
+     path = path, format = "LightCycler")
> show(raw)
An object of class "qPCRset"
Size:  384 features, 1 samples
Feature types:
Feature names:                  Sample 1 Sample 2 Sample 3 ...
Feature classes:
Feature categories:         OK
Sample names:                 LightCycler_sample ...
```

13.3 CFX format

The CFX Connect Real-Time PCR Detection System from Bio-Rad laboratories is another qPCR system based on microtitre plates, such as the CFX96 and CFX384 Touch. The output values from the software are "Cq", the quantification cycle values. This is the cycle number where the fluorescence increases above a given threshold, i.e. equivalent to Ct values. The example file included into *HTqPCR* contains a number of empty wells. Per default, these are excluded from the output file, and hence *n.features* has to be set accordingly.

```
> path <- system.file("exData", package = "HTqPCR")
> raw <- readCtData(files = "CFX_sample.txt", path = path,
+     format = "CFX", n.features = 330)
> show(raw)
An object of class "qPCRset"
Size:  330 features, 1 samples
Feature types:
Feature names:                 miR-101 miR-101 miR-116 ...
Feature classes:
Feature categories:         OK
Sample names:                 CFX_sample ...
```

The file is expected to contain comma-separated values, rather than tab-separated, although the file ending can be either of .txt and .csv. In some (mainly older versions) of CFX files, "," is the character used for decimal points rather than ".", so the Cq values can be for example "25,3" instead of "25.3". If this is the case, the parameters *dec=","* needs to be added to *readCtData*.

13.4 BioMark format

In addition to multi-well microtitre plates, other assay formats have also emerged for performing high-throughput qPCR analysis. These include for example the 48.48 and 96.96 BioMark HD System from Fluidigm Corporation. For the 48.48 assay, 48 individual samples are loaded onto a plate along with qPCR primers for 48 genes. Using microfluidic channels all possible sample $\times$ primer reactions are assayed in a combinatorial manner using 2,304 individual reaction chambers, to generate e.g. 48 real-time curves for each of 48 samples. Results are then reported in a single file with 2304 rows of data. This file completely determines the order in which the samples are being read in, i.e. from row 1 onwards, regardless of how the samples are usually loaded onto each specific assay type.
HTqPCR includes a comma-separated file containing example data from a BioMark 48.48 array. The data can be read in two way. Setting n.features=2304 will read in all the information and create a `qPCRset` object with dimensions 2304x1. Setting n.data=48 and n.features=48 will however automatically convert this into a 48x48 `qPCRset`. The latter is typically what will be required for doing statistical tests of differences between samples. However, in some cases when results are combined across multiple arrays it may be advantageous to keep these arrays as separate columns in the `qPCRset` object initially, in case it's necessary to perform an array-based normalisation.

```
> exPath <- system.file("exData", package = "HTqPCR")
> raw1 <- readCtData(files = "BioMark_sample.csv",
```

```
+     path = exPath, format = "BioMark", n.features = 48,
+     n.data = 48)
> dim(raw1)
Features  Samples
      48       48
> raw2 <- readCtData(files = "BioMark_sample.csv",
+     path = exPath, format = "BioMark", n.features = 48 *
+         48, n.data = 1)
> dim(raw2)
Features  Samples
    2304        1
```

If array-specific effects are likely to be present, a useful normalisation strategy might be to combine the data into a `qPCRset` object containing one column for each array, and 48x48 or 96x96 samples. That way overall differences between the arrays can be removed, and afterwards the data can be split with `changeCtLayout` to generate one column per individual sample rather than one column per array.
The column "Call" in the sample file contains information about the result of the qPCR reaction. Per default, a call of "Pass" is translated into "OK" in the *featureCategory*, and "Fail" as "Undetermined". The assay readouts are Ct values, which can be analysis using `HTqPCR` in a similar fashion to Ct values from other technologies, both regarding the statistical analysis and data visualisation. The function `plotCtArray` can be used for displaying the layout of the qPCR results instead of `plotCtCard` (Figure 21).

```
> plotCtArray(raw1)
```

13.5 OpenArray format

Like the BioMark output from Fluidigm, files from the OpenArray Real-Time PCR System from Applied Biosystems contains multiple samples per plate, currently up to 48. As mentioned in the section on BioMark, this can be used either in a `qPCRset` object with one column per plate, or with one column per sample. A comma-separated example file is included in *HTqPCR*, where a plate with 5076 qPCR reactions contains 846 features measured across 6 separate samples:
This file completely determines the order in which the samples are being read in, i.e. from row 1 onwards, regardless of how the samples are usually loaded onto each specific assay type.

```
> exPath <- system.file("exData", package = "HTqPCR")
> raw1 <- readCtData(files = "OpenArray_sample.csv",
+     path = exPath, format = "OpenArray", n.features = 846,
+     n.data = 6)
> dim(raw1)
Features  Samples
     846        6
> raw2 <- readCtData(files = "OpenArray_sample.csv",
+     path = exPath, format = "OpenArray", n.features = 846 *
```

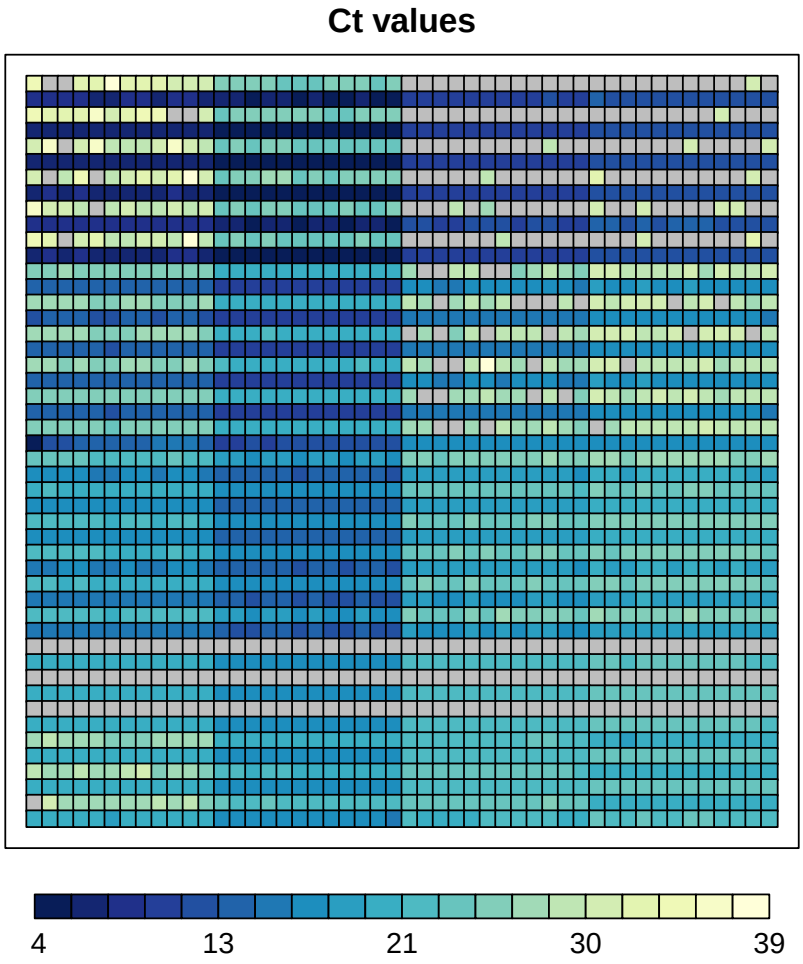


Figure 21: Ct values for a test Fluidigm 48.48 Array. 48 samples are loaded into rows, and 48 PCR primers into columns, resulting in 2,304 combinatorial qPCR reactions. Four individual genes were added in 12 replicates each into the 48 columns to assess technical variability. Grey corresponds to "NA".

```
+         6, n.data = 1)
> dim(raw2)
Features  Samples
    5076        1
```

The column "ThroughHole.Outlier" in the sample file indicates the quality of the qPCR measurement. Per default, if the Ct value is an outlier it is translated into having *featureCategory* "Unreliable", otherwise it's "OK".

13.6 Additional devices

More assays for high-throughput real-time qPCR systems are constantly emerging. In case of systems based on standard microtitre plates, these can still be imported by *HTqPCR*, even if they are not specifically included into (readCtData). The results need to be converted into tab-separated text, and read using *format="plain"*. Some assays are more esoteric, or contain multiple samples on each plates, which need to be re-formatted into a matrix in *R*. Below, a BioMark file from Fluidigm is used as an example of how to do this. This file can be read automatically by setting *format="BioMark"*, but it's here used to illustrate how the same thing can be done manually, and be transformed into a `qPCRset` object in one of two ways.
First, `qPCRset` can be constructed indirectly, by reading the data into a data frame, creating a 48x48 matrix manually, and generating a new `qPCRset`. The file was inspected first, and turned out to have an 11 line header plus a header line, which have to be skipped when reading the data into *HTqPCR*.
Alternatively, the data can be read directly into a `qPCRset` in a 1x2304 format using `readCtData`, and then re-formatted into 48x48 using `changeCtLayout`. This will automatically read in additional information such as feature names, positions and categories (Failed/Passed) if available.
Information from multiple microfluidic devices can then subsequently be combined using `cbind` and `rbind`.

14 Concluding remarks

This vignette was generated using:

- R version 4.6.0 (2026-04-24), `x86_64-pc-linux-gnu`
- Locale: `LC_CTYPE=en_US.UTF-8`, `LC_NUMERIC=C`, `LC_TIME=en_US.UTF-8`, `LC_COLLATE=en_US.UTF-8`, `LC_MONETARY=en_US.UTF-8`, `LC_MESSAGES=en_US.UTF-8`, `LC_PAPER=en_US.UTF-8`, `LC_NAME=C`, `LC_ADDRESS=C`, `LC_TELEPHONE=C`, `LC_MEASUREMENT=en_US.UTF-8`, `LC_IDENTIFICATION=C`
- Time zone: `Etc/UTC`
- TZcode source: `system (glibc)`
- Running under: `Ubuntu 24.04.4 LTS`
- Matrix products: default
- BLAS: `/usr/lib/x86_64-linux-gnu/openblas-pthread/libblas.so.3`
- LAPACK: `/usr/lib/x86_64-linux-gnu/openblas-pthread/libopenblasp-r0.3.26.so` ; LAPACK version3.12.0
- Base packages: base, datasets, graphics, grDevices, methods, stats, utils
- Other packages: Biobase 2.73.1, BiocGenerics 0.59.6, generics 0.1.4, HTqPCR 1.67.0, limma 3.69.1, RColorBrewer 1.1-3
- Loaded via a namespace (and not attached): affy 1.91.0, affyio 1.83.0, BiocManager 1.30.27, bitops 1.0-9, buildtools 1.0.0, caTools 1.18.3, cli 3.6.6, compiler 4.6.0, evaluate 1.0.5, gplots 3.3.0, gtools 3.9.5, KernSmooth 2.23-26, knitr 1.51, maketools 1.3.2, preprocessCore 1.75.0, rlang 1.2.0, statmod 1.5.2, stats4 4.6.0, sys 3.4.3, tools 4.6.0, xfun 0.57