HTqPCR - automated high-throughput qPCR analysis

Heidi Dvinge

October 28, 2009

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).
Example data is included 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.

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

The package employs functions from other packages of the Bioconductor project (Gentleman et al., 2004). 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.

Getting help

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

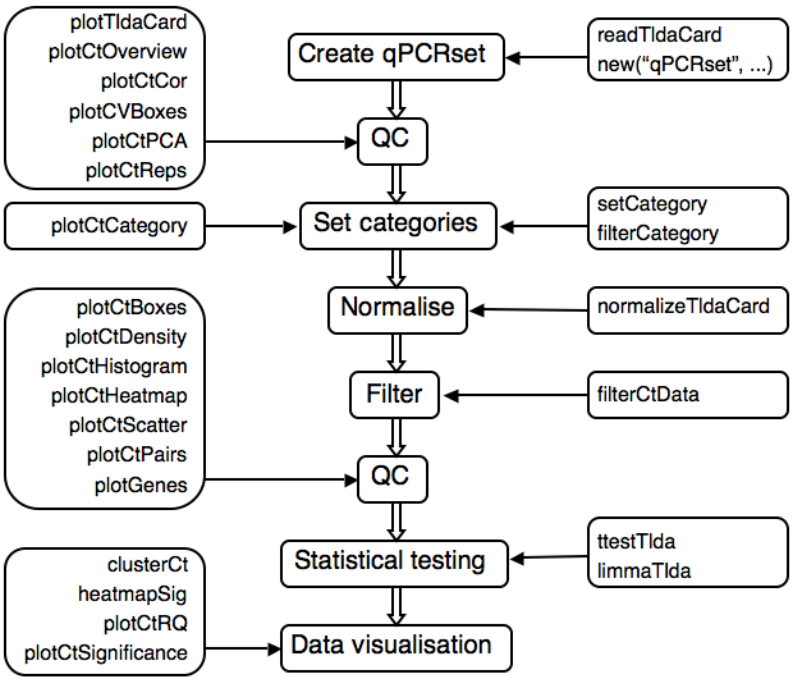

Figure 1: Workflow in *HTqPCR* analysis of qPCR data. Centre column: The main procedural steps in a typical qPCR analysis; Left: examples of visualisation functions; Right: data analysis functions.

2 *qPCRset* objects

The data is stored in an object of class *qPCRset*, which is similar to the *ExpressionSet* from package *Biobase* used for handling microarray data. 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 slots:

`featureNames` Object of class *character* giving the names of the features (genes, miRNAs) used in the experiment.

`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 *factor* representing the different types of features on the card, such as endogenous controls and target genes.

`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.

`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.

`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.

`normalized` Object of class *character* indicating if the data has been normalized, and if so then what method was used. Automatically set when function `normalizeCtData` is called.

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.
Two `qPCRset` test objects are included in the package: one containing raw data, and the other containing processed values.

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

3 Reading in the raw data

3.1 General data format

The 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:                 Endogenous Control, Target
Feature names:                 Gene1 Gene2 Gene3 ...
Feature classes:
Feature categories:         OK, Undetermined
Sample names:                 sample1 sample2 sample3 ...
```

3.2 Sequence Detection Systems format

The qPCR data might come from Sequence Detection Systems (SDS) software, in which case 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
 1        Control        A1        Passed        Sample01        Gene1        Endogenous
 2        Control        A2        Passed        Sample01        Gene2        Target
```

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 `SDS=TRUE` 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.

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 2 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 3 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)
```

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".

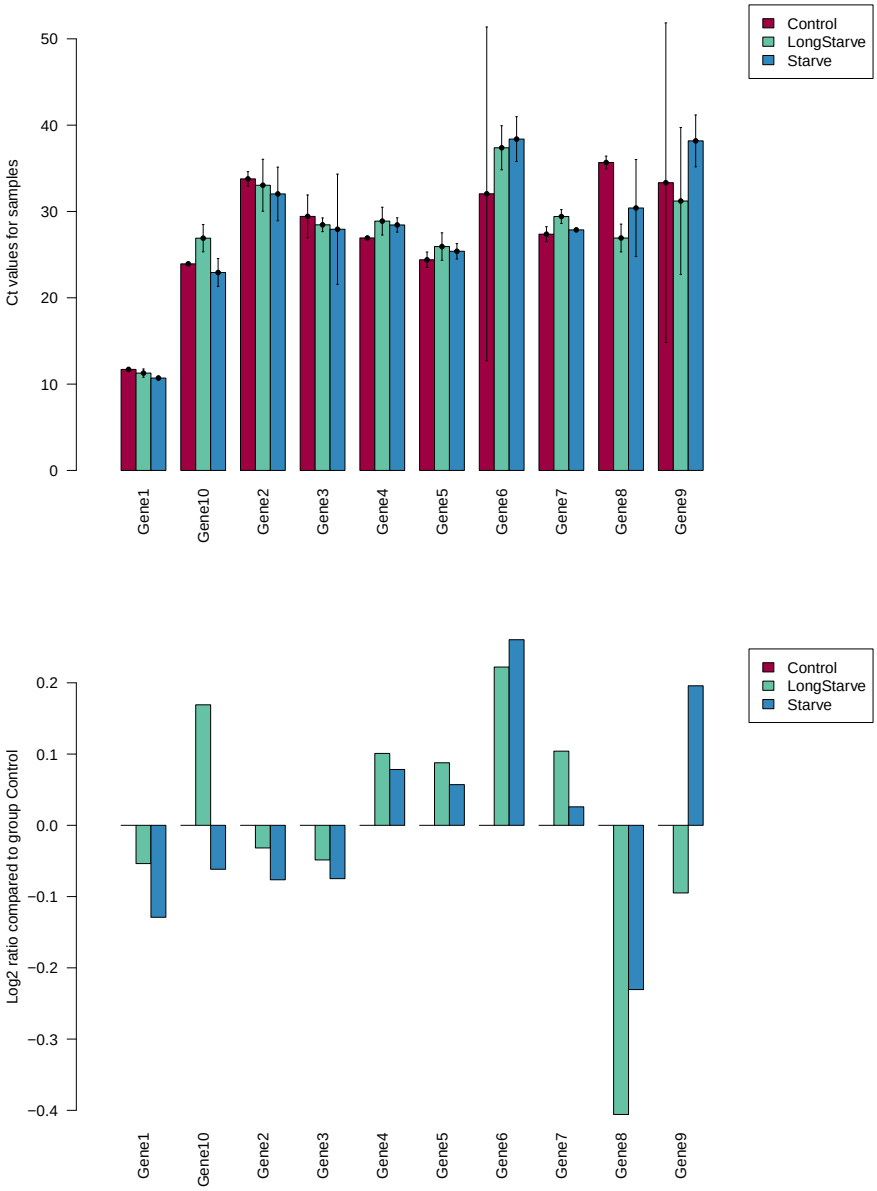

Figure 2: Overview of Ct values for the raw data.

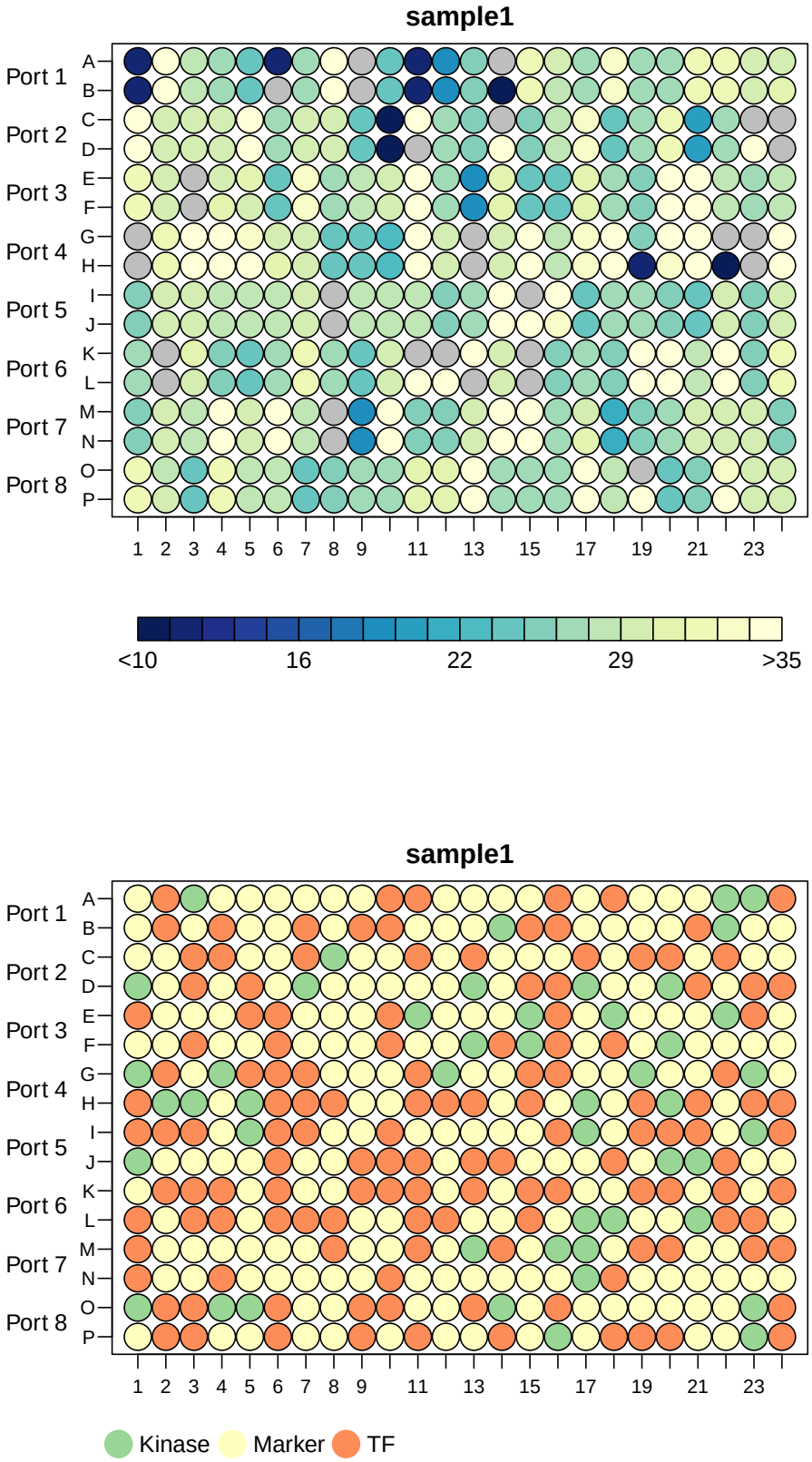

Figure 3: Ct values for the first sample (top), and the location of different feature classes (bottom). Ct values are visualised using colour intensity, and grey circles are features that were marked "undetermined" in the input file.

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".

```
> raw.cat <- setCategory(raw, groups = files$Treatment,
+     quantile = 0.8)
Categories after Ct.max and Ct.min filtering:
             sample1 sample2 sample3 sample4 sample5 sample6
OK               313     264     327     295     296     286
Undetermined      68     119      56      86      86      96
Unreliable         3       1       1       3       2       2
Categories after standard deviation filtering:
             sample1 sample2 sample3 sample4 sample5 sample6
OK               301     254     319     274     277     275
Undetermined      68     119      56      86      86      96
Unreliable        15      11       9      24      21      13
```

A summary plot for the sample categories is depicted in Figure 4. The result can be stratified by `featureType` or `featureClass`, for example to determine whether one class of features performed better or worse than others.

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

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

The results can also be shown per feature rather than averaged across samples (Figure 5).

```
> 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.

6 Normalisation

Four different normalisation methods are currently implemented in *HTqPCR*. Two of these (`scale.rankinvariant` and `deltaCt`) 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.

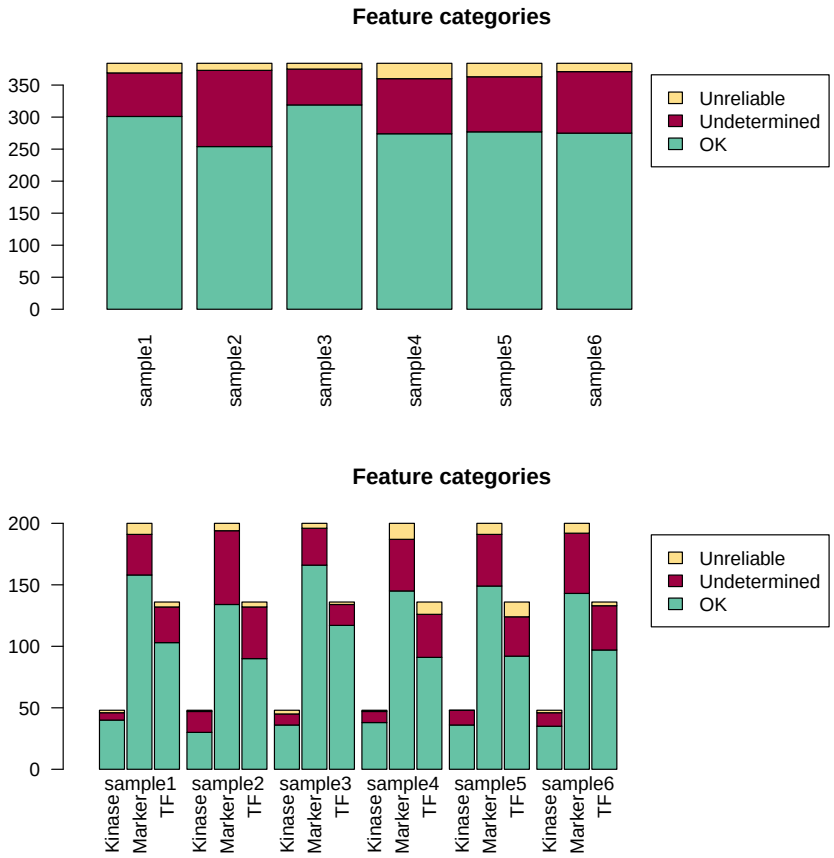

Figure 4: Summary of the categories, either for each sample individually or stratified by feature class.

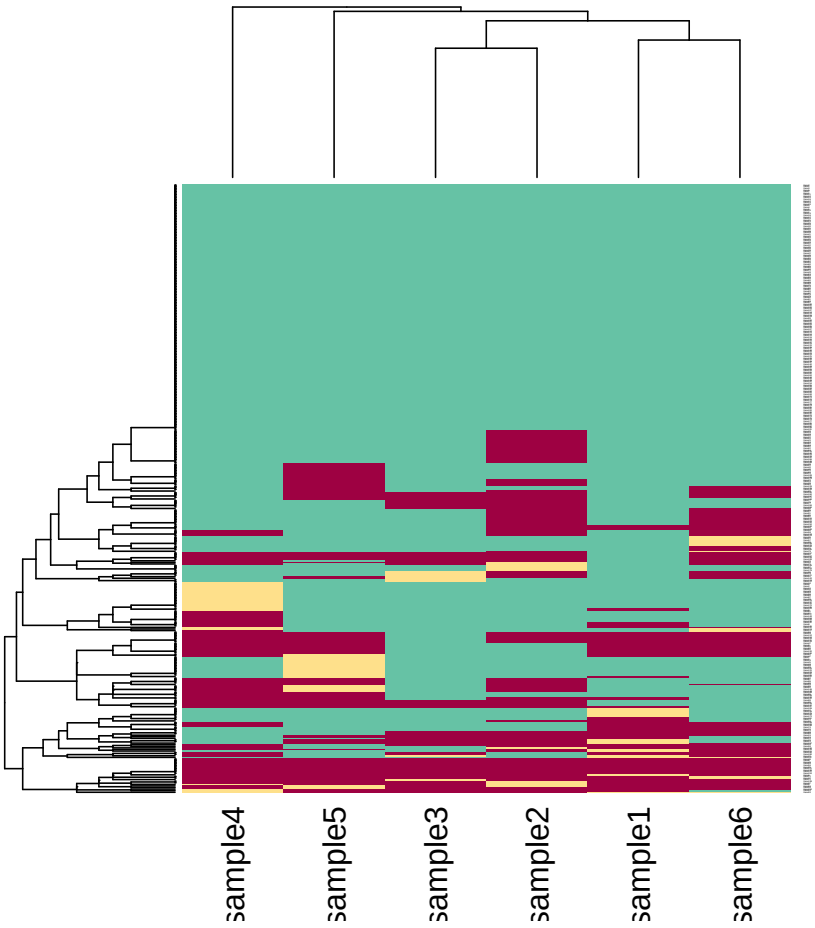

Figure 5: Summary of the categories, clustered across features.

norm.rankinvariant Computes all rank-invariant sets of features between pairwise comparisons of each sample against a reference, such as a pseudo-mean. The rank-invariant 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.

For the rank-invariant normalisation 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: 75 rank invariant genes
        sample2: 33 rank invariant genes
        sample3: 48 rank invariant genes
        sample4: 69 rank invariant genes
        sample5: 18 rank invariant genes
        sample6: 67 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=14.50        Stdev=5.8
        Card 4:         Mean=14.79        Stdev=4.79
        Card 5:         Mean=14.07        Stdev=5.32
        Card 6:         Mean=13.82        Stdev=4.75
```

Comparing the raw and normalised values gives an idea of how much correction has been performed (Figure 6). Note that the scale on the y-axis varies.

7 Filtering and subsettting 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

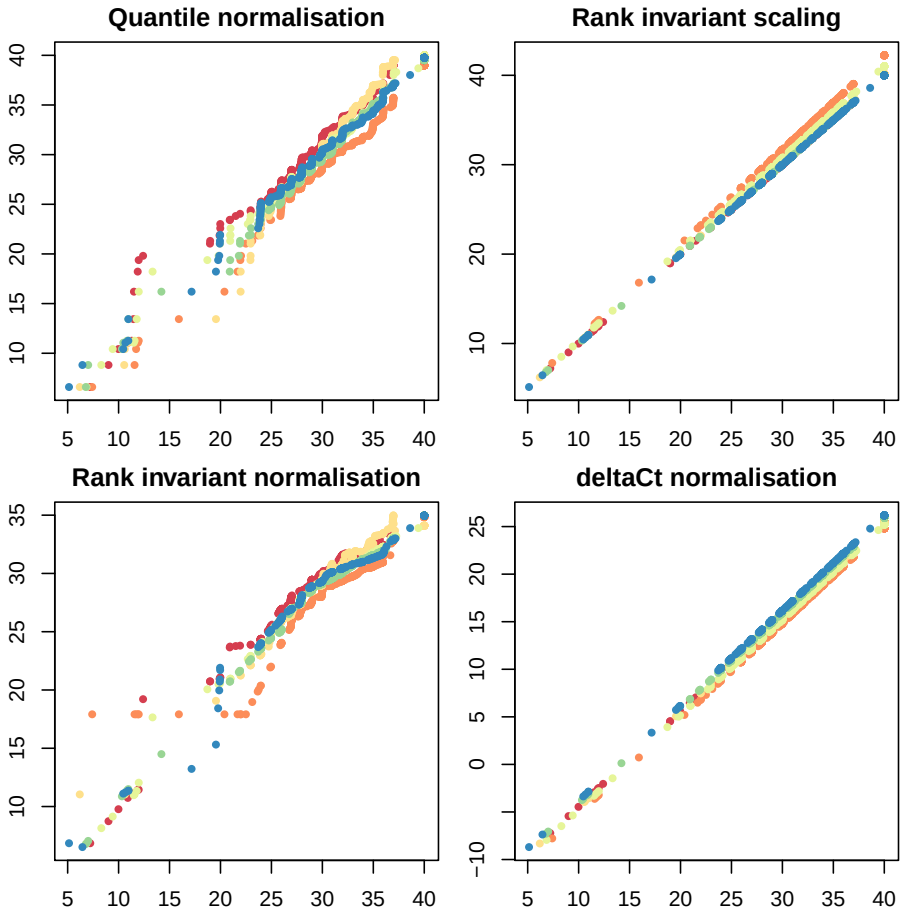

Figure 6: Normalized versus raw data, using a separate colour for each sample. The raw data is plotted along the x-axis and the normalised along y.

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 example:

```
> nr.norm[1:10, ]
An object of class "qPCRset"
Size:  10 features, 6 samples
Feature types:                 Endogenous Control, Target
Feature names:                 Gene1 Gene2 Gene3 ...
Feature classes:         Kinase, Marker, TF
Feature categories:         OK, Unreliable, Undetermined
Sample names:                 sample1 sample2 sample3 ...
> nr.norm[, c(1, 3, 5)]
An object of class "qPCRset"
Size:  384 features, 3 samples
Feature types:                 Endogenous Control, Target
Feature names:                 Gene1 Gene2 Gene3 ...
Feature classes:         Kinase, Marker, TF
Feature categories:         OK, Unreliable, Undetermined
Sample names:                 sample1 sample3 sample5 ...
```

Filtering is done by specifying the components to remove, either by just using a single criteria, or by combining multiple filters:

```
> qFilt <- filterCtData(nr.norm, remove.type = "Endogenous Control")
Removed 4 'Endogenous Control' features based on featureType(q).
> qFilt <- filterCtData(nr.norm, remove.name = c("Gene1",
+     "Gene20", "Gene30"))
Removed 8 'Gene1/Gene20/Gene30' features based on featureNames(q).
> qFilt <- filterCtData(nr.norm, remove.class = "Kinase")
Removed 48 'Kinase' features based on featureClass(q).
> qFilt <- filterCtData(nr.norm, remove.type = c("Endogenous Control"),
+     remove.name = c("Gene1", "Gene20", "Gene30"))
Removed 4 'Endogenous Control' features based on featureType(q).
Removed 4 'Gene1/Gene20/Gene30' features based on featureNames(q).
```

The data can also be adjusted according to feature categories. With `filterCategory` mentioned previously it's possible to replace certain Ct values with NA, but one might want to completely exclude features where a certain number of the Ct values are for example unreliable.

```
> qFilt <- filterCtData(nr.norm, remove.category = "Undetermined")
Removed 64 features with >3 'Undetermined' based on featureCategory(q).
> qFilt <- filterCtData(nr.norm, remove.category = "Undetermined",
+     n.category = 5)
Removed 10 features with >5 'Undetermined' based on featureCategory(q).
```

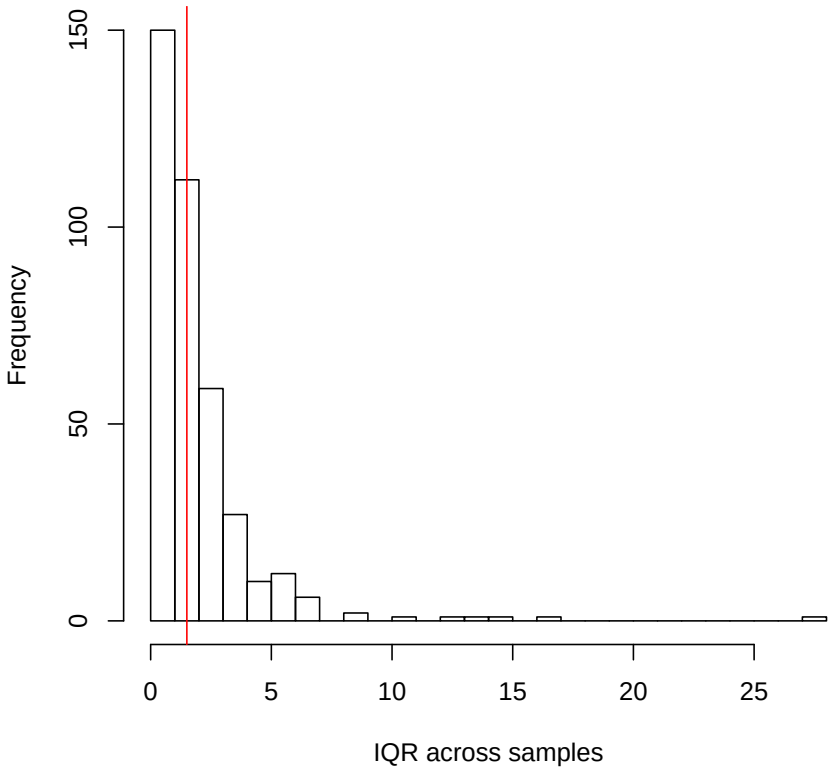

Figure 7: Histogram of the IQR values for all features across the samples, including the cut-off used in the filtering example.

Another typical filtering step would be to remove features showing little or no variation across samples, prior to testing for statistical significance of genes between samples. Features with relatively constant Ct levels are less likely to be differentially expressed, so including them in the downstream analysis would cause some loss of power when adjusting the p-values for multiple testing (the feature-by-feature hypothesis testing). Variation across samples can be assessed using for example the interquartile range (IQR) values for each feature (Figure 7). All features with IQR below a certain threshold can then be filtered out.

```
> qFilt <- filterCtData(nr.norm, remove.IQR = 1.5)
Removed 207 features with IQR <1.5 based on exprs(q).
```

Note that filtering prior to normalisation can affect the outcome of the normalisation procedure. In some cases this might be desirable, for example if a particular feature class are heavily biasing the results so it's preferable to split the `qPCRset` object into smaller data sets. However, in other cases it might for example make it difficult to identify a sufficient number of rank invariant features for the`norm.rankinvariant` and `scale.rankinvariant` methods. Whether to perform filtering, and if so then during what step of the analysis, depends on the genes and biological samples being analysed, as well as the quality of the data. It's therefore advisable to perform a detailed quality assessment and data comparison, as mentioned in the next section.

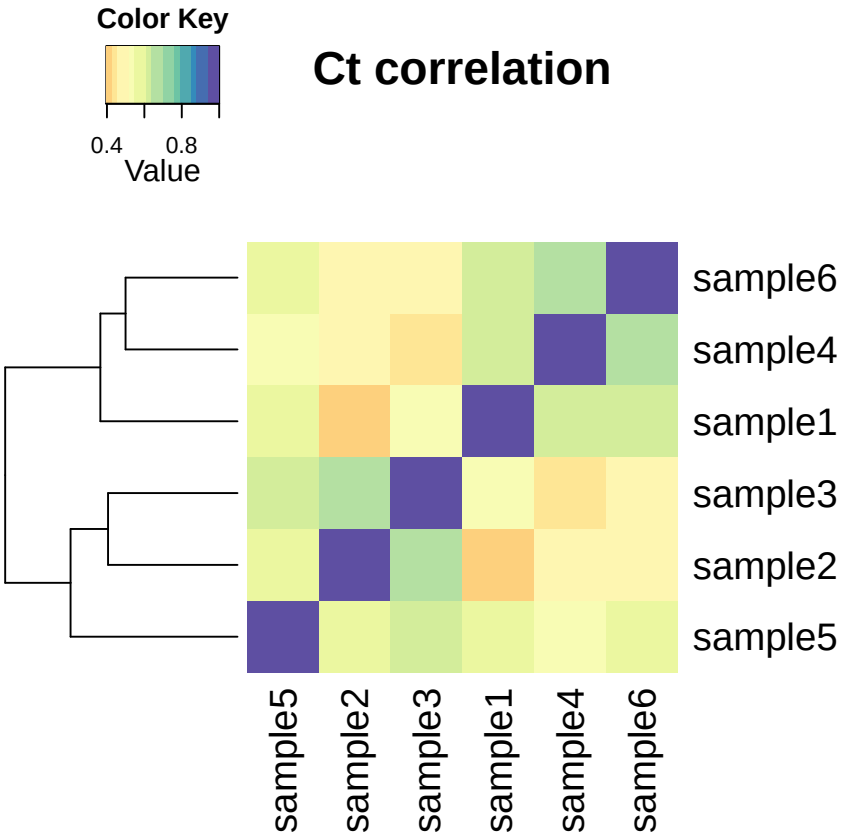

Figure 8: Correlation between raw Ct values.

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 8.

```
> 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.

```
> 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"
```

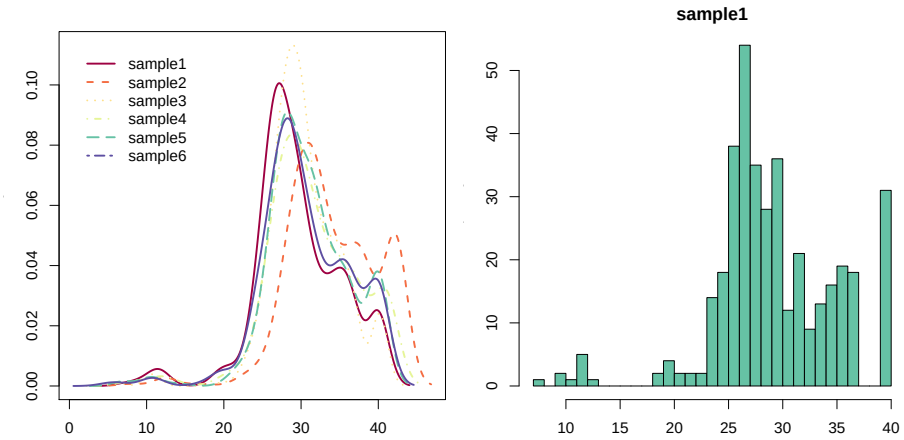

Figure 9: Distribution of Ct values for the individual samples, either using the density of all arrays (left) or a histogram of a single sample (right), after scale rank-invariant normalisation.

```
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 9.

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

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

Plotting the densities of the different normalisation methods lends insight into how they differ (Figure 10).
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. 11).

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

8.3 Comparison of Ct values for two samples

It is often of interest to directly compare Ct values between two samples. In Figure 12, 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)
```

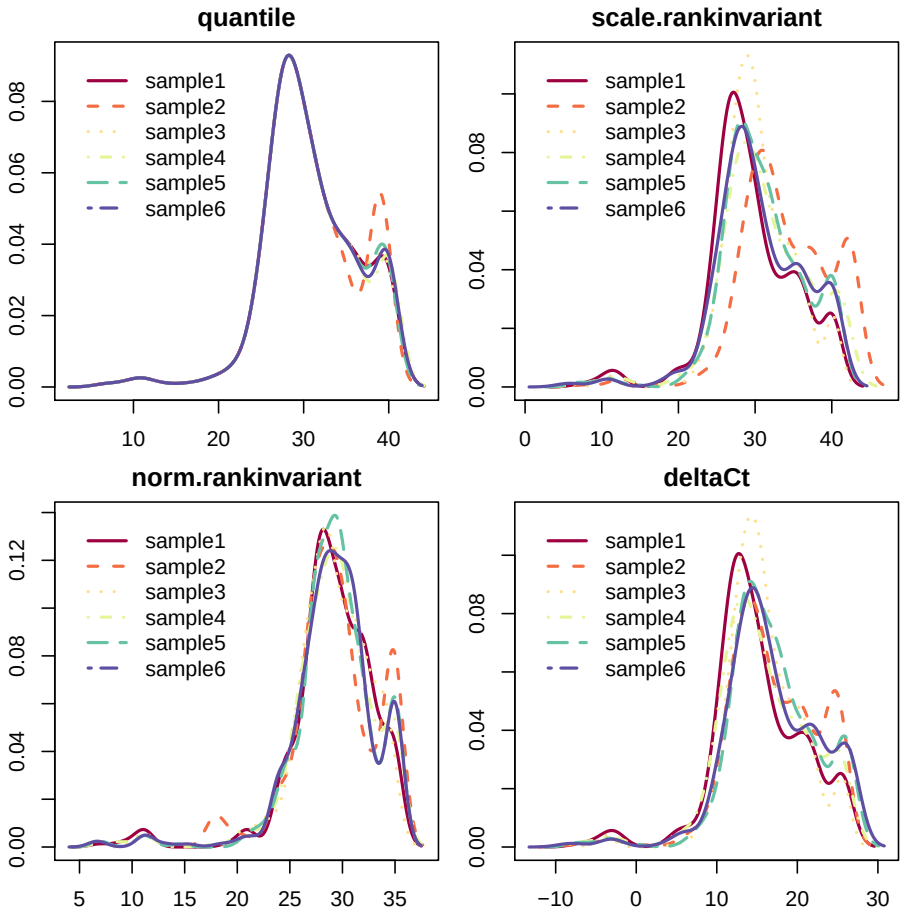

Figure 10: Densities of Ct values for all samples after each of the four normalisation methods. The peak at the high end originates from features with "Undetermined" Ct values, which are assigned the Ct value 40 by default.

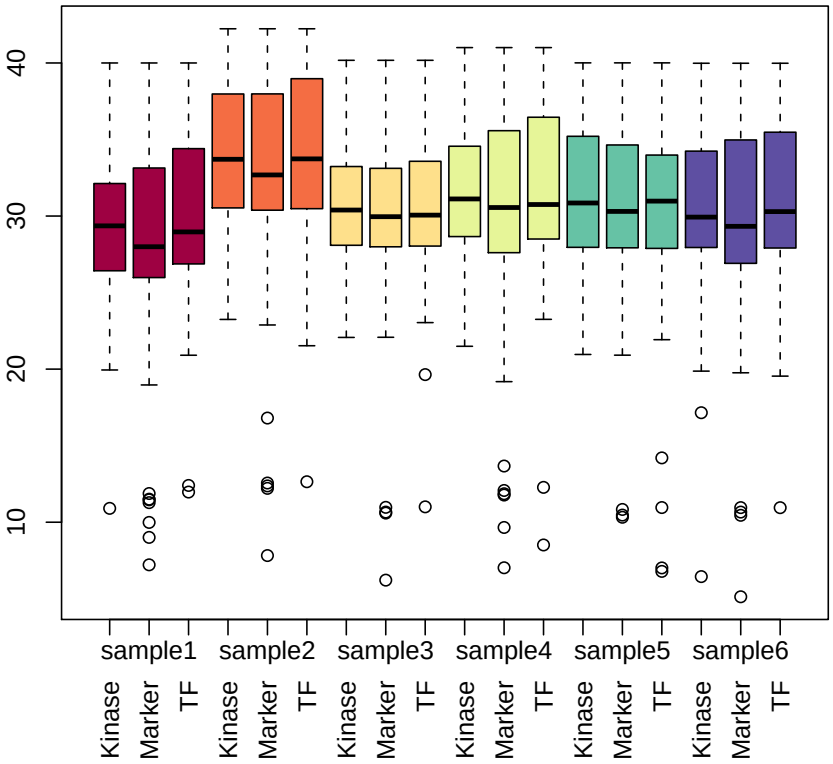

Figure 11: Boxplot of Ct values across all samples, stratified by feature classes.

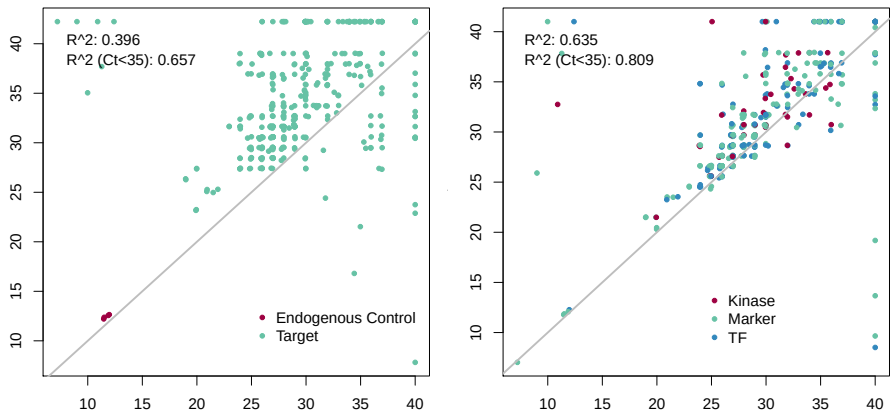

Figure 12: Scatter plot of Ct values in different samples, with points marked either by featureType (left) or featureClass (right) and the diagonal through $x = y$ marked with a grey line.

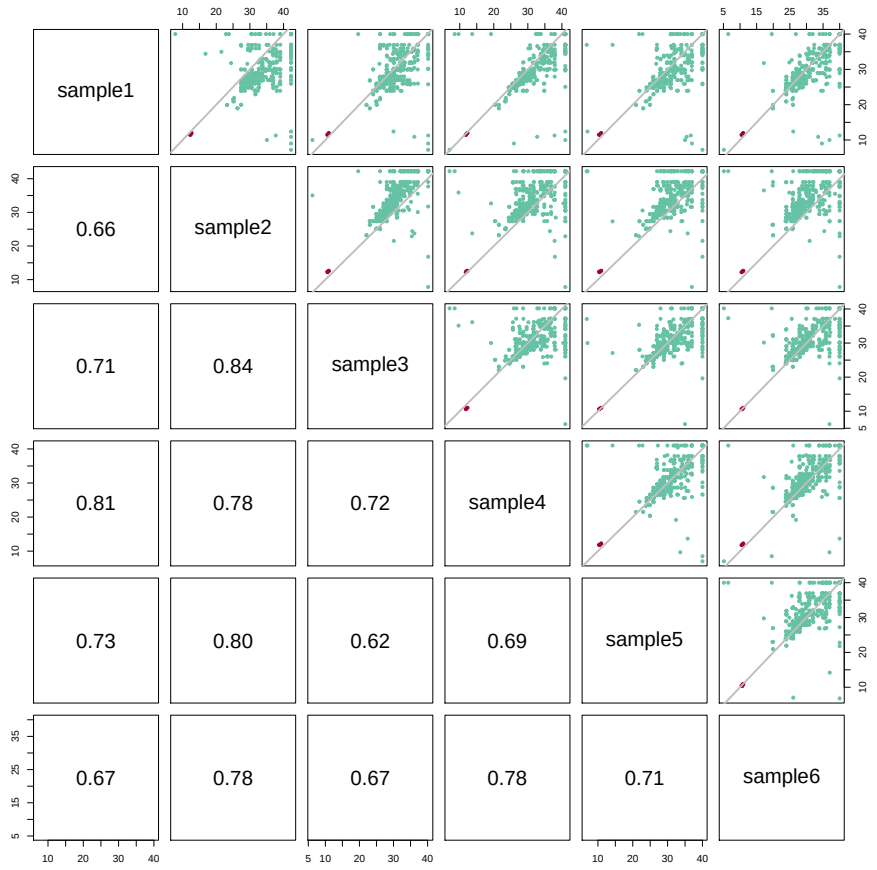

Figure 13: Scatterplot for all pairwise comparisons between samples, with spots marked depending on `featureType`, i.e. whether they represent endogenous controls or targets.

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 13 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 14). 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.

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

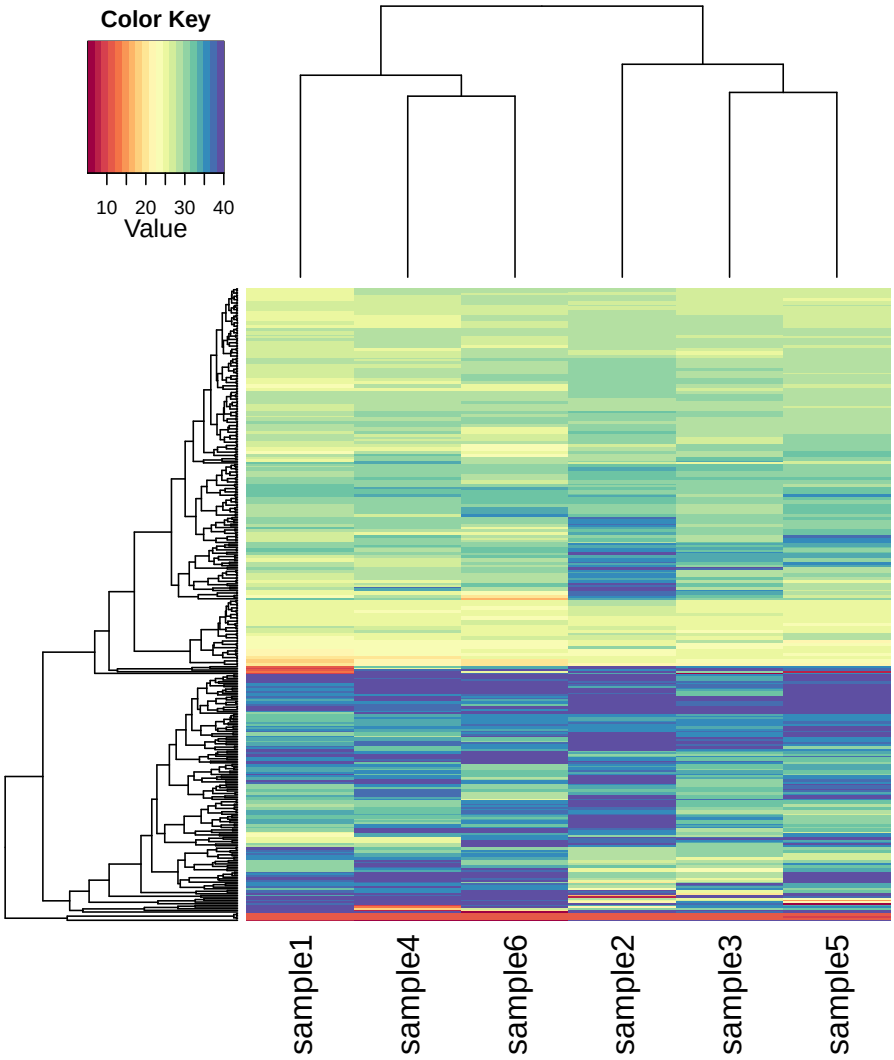

Figure 14: Heatmap for all samples and genes, based on the Euclidean distance between Ct values.

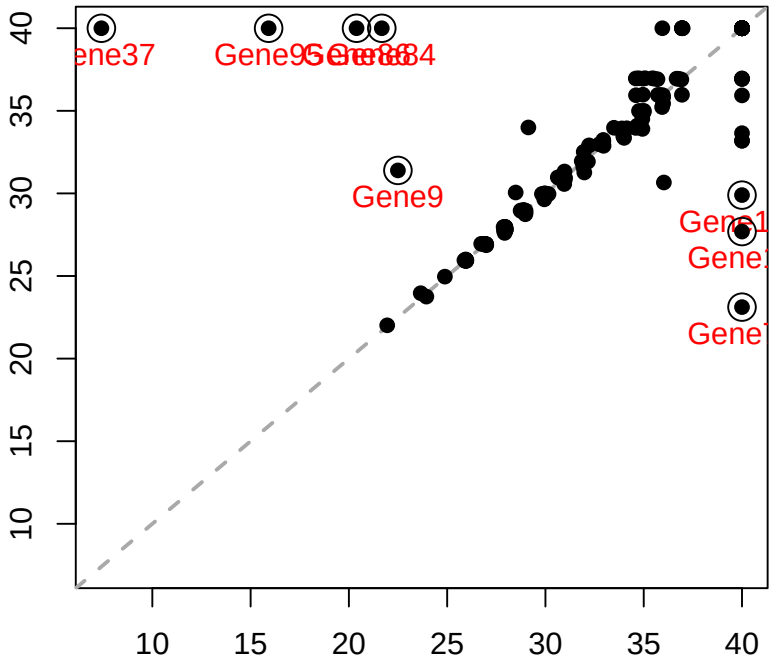

Figure 15: Concordance between duplicated Ct values in sample 2, marking features differing >20% from their mean.

8.6 Comparison of replicated 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 15 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
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.

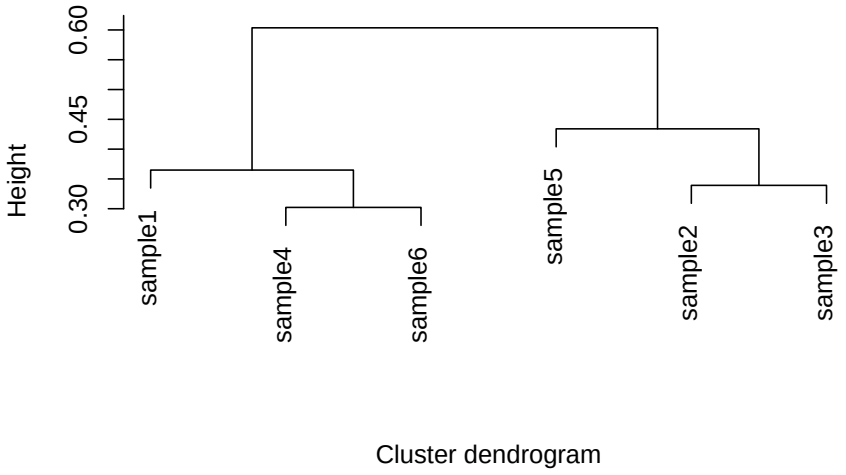

Figure 16: Hierarchical clustering of samples.

9 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 16. In Figure 17 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)

> c6 <- cluster.list[[6]]
> print(c6)
  Gene9  Gene13  Gene46  Gene50  Gene75  Gene93  Gene75 Gene103
     9      14      71      99     148     166     172     200
Gene132 Gene151 Gene151
    277     296     320
> show(sr.norm[c6, ])
An object of class "qPCRset"
Size:  11 features, 6 samples
Feature types:                 Endogenous Control, Target
Feature names:                 Gene9 Gene13 Gene46 ...
Feature classes:         Kinase, Marker, TF
Feature categories:         Undetermined, Unreliable, OK
Sample names:                 sample1 sample2 sample3 ...
```

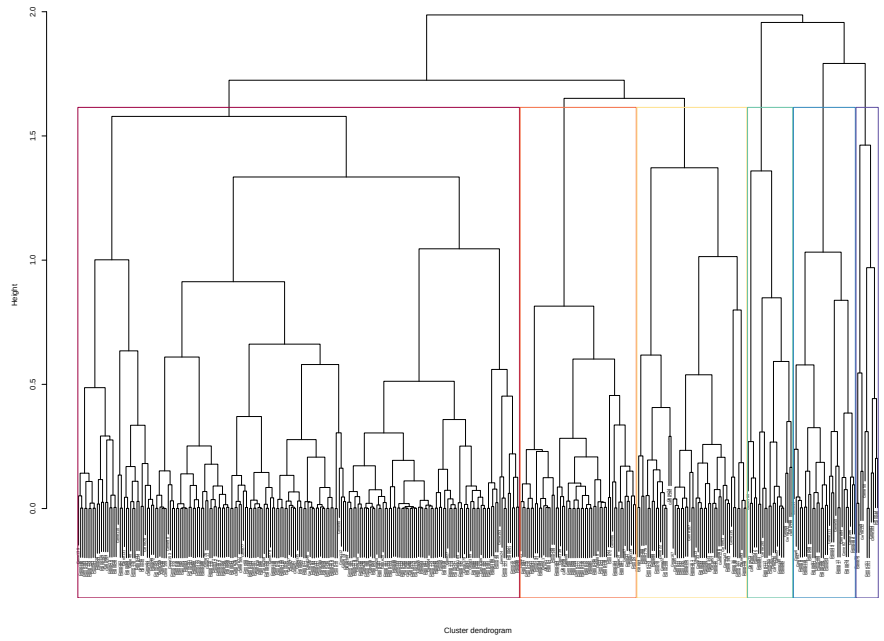

Figure 17: Hierarchical clustering of features, with subclusters marked.

10 Differential expression

In general there are two 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.

limma Using a wrapper for functions from the package *limma* (Smyth, 2005) 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 both 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.

```
> qDE.ttest <- ttestCtData(sr.norm[, 1:4], groups = files$Treatment[1:4],
+     calibrator = "Control")
> qDE.ttest[1:10, ]
```

```
      genes feature.pos     t.test      p.value      ddCt
2    Gene10     A10;B10 -13.705640 9.812948e-06  3.460492
22  Gene118     I23;J23 -10.407281 8.661980e-05  8.982157
36  Gene130     K11;L11   7.644065 3.156334e-04 -9.433460
164  Gene74       G3;H3   7.201124 4.768225e-04 -5.178293
30  Gene125       K6;L6  -7.203614 5.199938e-04  7.052337
33  Gene128       K9;L9 -10.357650 6.826374e-04  5.210558
76  Gene167     M24;N24 -12.360918 7.589634e-04 13.401172
51  Gene144       M1;N1  -7.847655 1.087976e-03  3.582610
167  Gene77       G6;H6  -8.643118 1.397709e-03 11.067840
118  Gene32       C9;D9  -6.332114 1.441690e-03  5.645620
    meanCalibrator meanTarget categoryCalibrator categoryTarget
2         24.23106   27.69156                 OK             OK
22        26.70531   35.68746                 OK             OK
36        39.74414   30.31068                 OK             OK
164       35.40699   30.22869                 OK             OK
30        28.28445   35.33679                 OK             OK
33        25.61542   30.82598                 OK             OK
76        26.25075   39.65192                 OK             OK
51        26.17539   29.75800                 OK             OK
167       29.36973   40.43757                 OK             OK
118       25.73270   31.37832                 OK             OK
```

10.2 Multiple sample types - limma

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.

```
> design <- model.matrix(~0 + files$Treatment)
> colnames(design) <- c("Control", "LongStarve", "Starve")
> print(design)
  Control LongStarve Starve
1       1          0      0
2       0          1      0
3       0          1      0
4       1          0      0
5       0          0      1
6       0          0      1
attr(,"assign")
[1] 1 1 1
attr(,"contrasts")
attr(,"contrasts")$`files$Treatment`
[1] "contr.treatment"
> contrasts <- makeContrasts(LongStarve - Control,
+     LongStarve - Starve, Starve - Control, (Starve +
+         LongStarve)/2 - Control, levels = design)
> colnames(contrasts) <- c("LS-C", "LS-S", "S-C", "bothS-C")
> print(contrasts)
```

```
            Contrasts
Levels       LS-C LS-S S-C bothS-C
  Control      -1    0  -1    -1.0
  LongStarve    1    1   0     0.5
  Starve        0   -1   1     0.5
> sr.norm2 <- sr.norm[order(featureNames(sr.norm)),
+     ]
> qDE.limma <- limmaCtData(sr.norm2, design = design,
+     contrasts = contrasts, ndups = 2, spacing = 1)
```

The result is a list with one component per comparison. Each component is similar to the result from using `ttestCtData`.

```
> class(qDE.limma)
[1] "list"
> names(qDE.limma)
[1] "LS-C"    "LS-S"    "S-C"     "bothS-C" "Summary"
> qDE.limma[["LS-C"]][1:10, ]
      genes feature.pos   t.test      p.value adj.p.value
14   Gene11     A12;B12 7.603024 1.999665e-05 0.003839357
50  Gene142     K23;L23 6.271023 9.883760e-05 0.009488409
3    Gene10     A10;B10 5.076932 5.031756e-04 0.024152428
119  Gene32       C9;D9 5.217172 4.113509e-04 0.024152428
23  Gene118     I23;J23 4.812232 7.417071e-04 0.028481551
133  Gene45     C22;D22 4.195002 1.905657e-03 0.060981013
11  Gene107     I12;J12 3.943239 2.842798e-03 0.068227164
100 Gene188     O21;P21 3.973794 2.706901e-03 0.068227164
31  Gene125       K6;L6 3.689090 4.291790e-03 0.082561735
156  Gene66     E19;F19 3.687908 4.300090e-03 0.082561735
         ddCt meanTarget meanCalibrator categoryTarget
14   6.051791   26.23181       20.18002             OK
50  10.752992   36.48566       25.73267   Undetermined
3    3.460492   27.69156       24.23106             OK
119  5.645620   31.37832       25.73270             OK
23   8.982157   35.68746       26.70531   Undetermined
133  6.108469   34.42031       28.31184             OK
11   4.515573   31.32807       26.81250             OK
100  3.004184   29.29643       26.29224             OK
31   7.052337   35.33679       28.28445             OK
156  3.055333   29.26378       26.20845             OK
    categoryCalibrator
14        Undetermined
50                  OK
3         Undetermined
119                 OK
23                  OK
133                 OK
11                  OK
100                 OK
31                  OK
156                 OK
```

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.

```
> qDE.limma[["Summary"]][21:30, ]
        Contrasts
         LS-C LS-S S-C bothS-C
  Gene116    0    0   0       0
  Gene117    0    0   0       0
  Gene118    1    0   0       0
  Gene119    0    0   0       0
  Gene12     0    0   0       0
  Gene120    0    0   0       0
  Gene121    0    0   0       0
  Gene122    0    0   0       0
  Gene123    0    0   0       0
  Gene124    0    0   0       0
```

11 Displaying the results

The results can be visualised using the generic `plotCtOverview` shown in Figure 2. 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 (figure 18), and another from the first comparison in `limmaCtData` where all genes below a certain p-value are depicted (Figure 19).

```
> plotCtRQ(qDE.ttest, genes = 1:15)
```

```
> plotCtRQ(qDE.limma, p.val = 0.085, transform = "log10",
+     col = "#9E0142")
```

The hatching on the bars indicates whether the target and calibrator Ct samples were unreliable, but this also depend on whether the parameter `stringent=TRUE` or `stringent=FALSE` when testing for differential expression. See the help functions for details (`?ttestCtData` and `?limmaCtData`).

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.

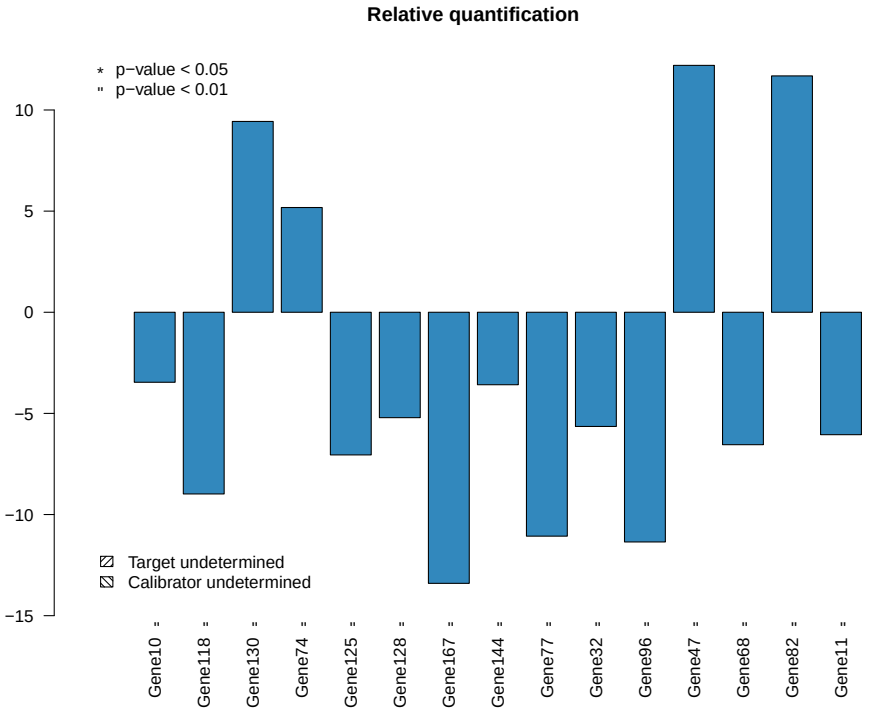

Figure 18: Relative quantification, using the top 15 features from ttestCtData.

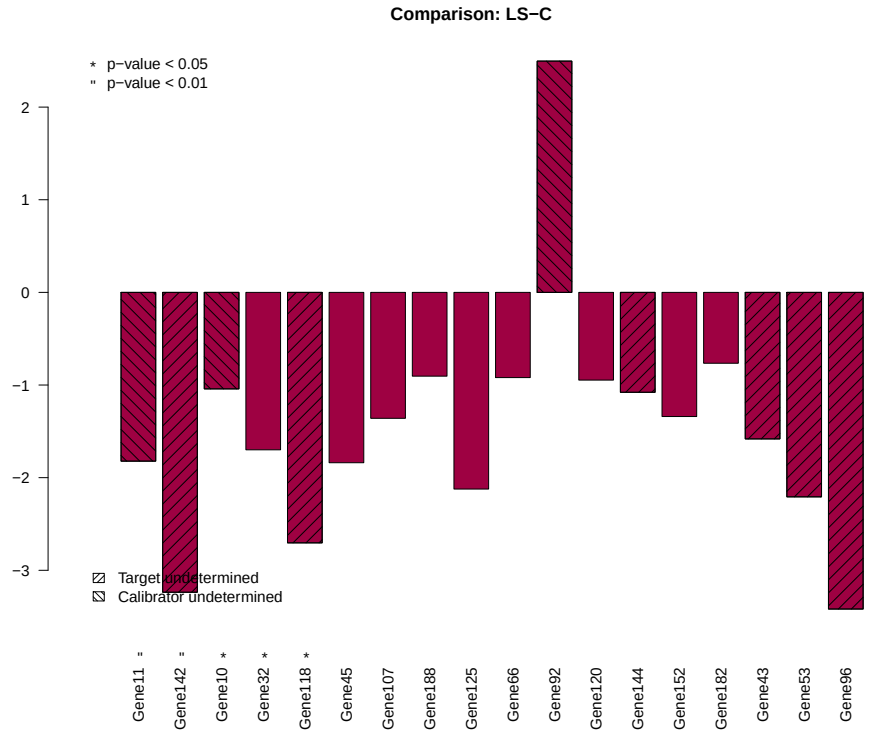

Figure 19: Relative quantification, using all features with p-value<0.085 from limmaCtData.

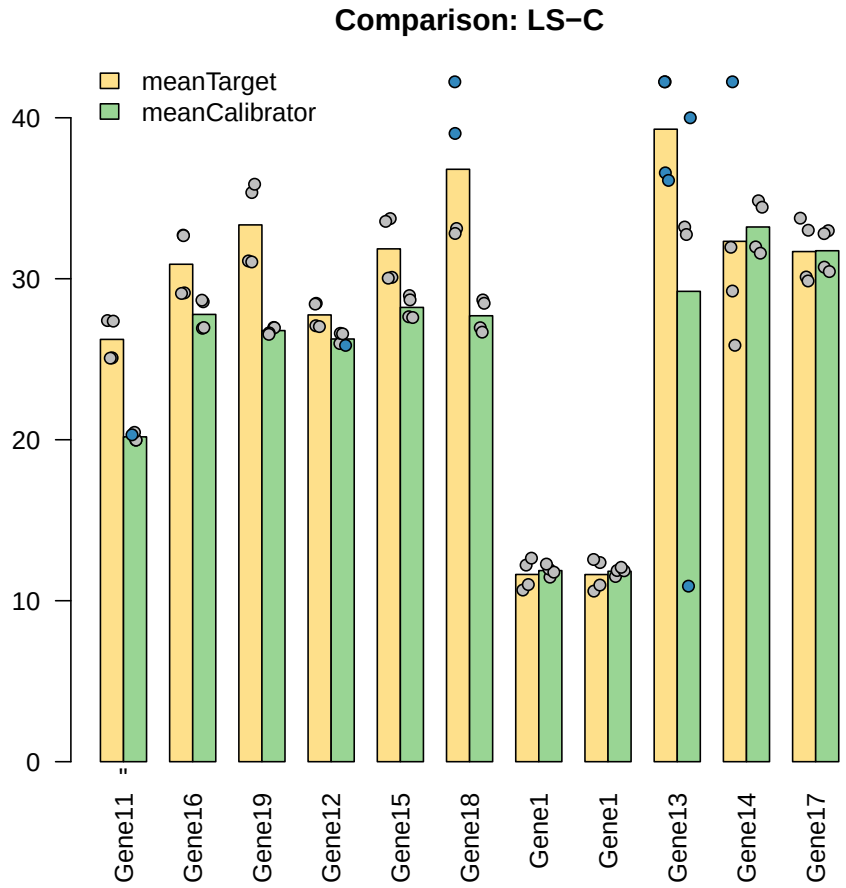

Figure 20: Ten genes from the data set, with the average Ct in two groups plotted along with the individual values. Points marked in blue are "Unreliable" or "Undetermined" whereas grey spots are "OK".

```
> plotCtSignificance(qDE.limma, q = sr.norm, groups = files$Treatment,
+     target = "LongStarve", calibrator = "Control",
+     genes = featureNames(sr.norm)[11:20], un.col = "#3288BD",
+     jitter = 0.2)
```

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 (Figure 21).

```
> heatmapSig(qDE.limma, dist = "euclidean")
```

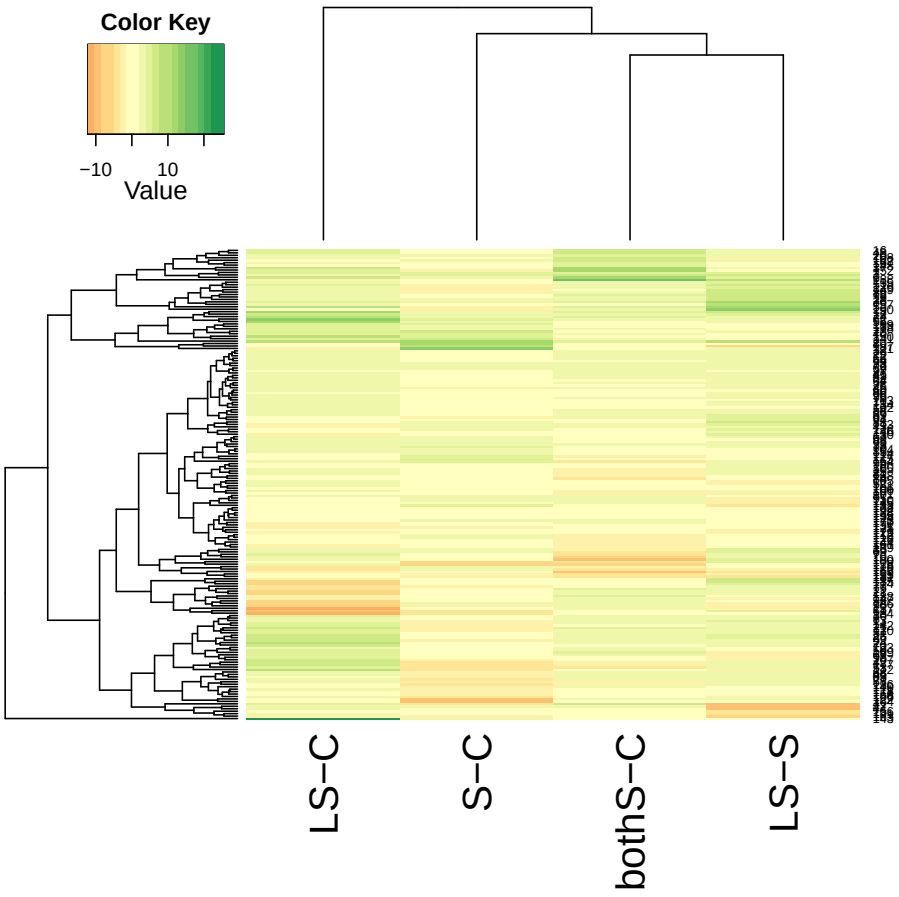

Figure 21: Fold changes across all comparisons, clustered based on Euclidean distance.

12 Concluding remarks

This vignette was generated using:

- R version 2.10.0 Patched (2009-10-27 r50222), `i386-apple-darwin8.11.1`
- Locale: `C/en_US.UTF-8/C/C/C/C`
- Base packages: base, datasets, grDevices, graphics, methods, stats, tools, utils
- Other packages: Biobase 2.6.0, HTqPCR 1.0.0, RColorBrewer 1.0-2, limma 3.2.1, statmod 1.4.1
- Loaded via a namespace (and not attached): affy 1.24.0, affyio 1.14.0, gdata 2.6.1, gplots 2.7.3, gtools 2.6.1, preprocessCore 1.8.0

References

R. C. Gentleman, V. J. Carey, D. M. Bates, B. Bolstad, M. Dettling, S. Dudoit, B. Ellis, L. Gautier, Y. Ge, J. Gentry, K. Hornik, T. Hothorn, W. Huber, S. Iacus, R. Irizarry, F. Leisch, C. Li, M. Maechler, A. J. Rossini, G. Sawitzki, C. Smith, G. Smyth, L. Tierney, J. Y. H. Yang, and J. Zhang. Bioconductor: Open software development for computational biology and bioinformatics. *Genome Biology*, 5:R80, 2004. 1

G. K. Smyth. Limma: linear models for microarray data. In R. Gentleman, V. Carey, S. Dudoit, and W. H. R. Irizarry, editors, *Bioinformatics and Computational Biology Solutions using R and Bioconductor*, pages 397–420. Springer, New York, 2005. 23