PCRedux Package - An Overview

Stefan Rödiger, Michał Burdukiewicz, Andrej-Nikolai Spiess

2017-11-20

Contents

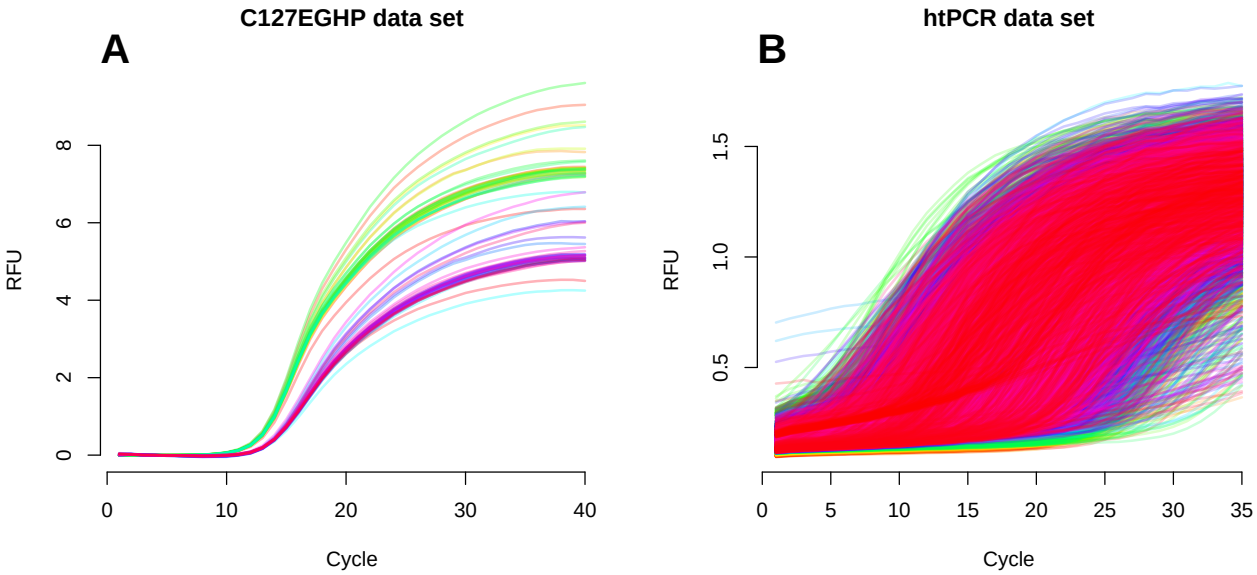

Figure 1: Sample data from A) the **C127EGHP** data set with 64 amplification curves (*chipPCR* package, (Rödiger, Burdukiewicz, and Schierack 2015)) and B) from the **htPCR** data set with 8858 amplification curves (*qpcR* package, (Ritz and Spiess 2008)).

1 Introduction to the Analysis of Sigmoid Shaped Curves – or the Analysis of Amplification Curve Data from Quantitative real-time PCR Experiments

`PCRedux` is an R package designed to perform analysis of sigmoid curves. A sigmoid function results a S-shaped curve. For example, the equation Equation 1 produces such curvature.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{1}$$

These non-linear functions are real-valued and can be differentiated (first derivative maximum, with one local minimum and one local maximum).

This kind of curve is common in many biological assays such as quantitative real-time PCR (qPCR) experiments. Applications include human diagnostics and forensics (Martins et al. 2015, Sauer, Reinke, and Courts (2016)). qPCRs are performed in thermo-cyclers. There are many manufactures of such systems, which offer such technologies as commercial products. Moreover, thermo-cyclers are produced in house as part of scientific projects. An example is the VideoScan-technology (Rödiger et al. 2013).

A challenge for the user is, to make sense from the vast amount of data produced. In particular, the reproducible and objective analysis of the amplification curve data exposes challenges to inexperienced users. Even among peers is not uncommon that they rate (classify) results differently.

The analysis of sigmoid data (e.g., quantitative PCR) it is a manageable task if the volume of data is low, or dedicated software is available for the analysis thereof. An example such a scenario (low number of amplification curves) is shown in Figure 1A. All 65 curves are sigmoid. In contrast, the example in Figure 1B is not manageable with a reasonable effort by simple visual inspection. The data originate from a high-throughput experiment an encompass in total 8858 amplification curves. Similarly, a manual analysis of the data is time-consuming and prone to errors. Specialized software that can distinguish the amplification curves automatically is needed.

There are several open source and closed source software tools, which can be used for the analysis of qPCR data. The software packages deal for example with

* missing values,
* noise,
* inter run calibration,
* normalization,
* quantification cycle estimation,
* amplification efficiency estimation,
* data exchange,
* relative gene expression analysis

and other tasks (Pabinger et al. 2014, Rödiger, Burdukiewicz, and Schierack (2015), Ritz and Spiess (2008), Rödiger et al. (2015), Lefever et al. (2009), Jan M. Ruijter, Lefever, et al. (2015), Rödiger et al. (2017), Perkins et al. (2012), Pabinger et al. (2009), Neve et al. (2014), Feuer et al. (2015), McCall et al. (2014), Ruijter et al. (2013), Jan M. Ruijter, Ruiz Villalba, et al. (2015), Dvinge and Bertone (2009), Ronde et al. (2017), Mallona, Weiss, and Egea-Cortines (2011), Mallona et al. (2017)). Many of them are implemented in the R statistical computing language. This list is far from complete. Dedicated literature is available from peer-reviewed publications and textbooks.

Others and we have shown that amplification curve analysis is not a simple task. For example selected qPCR systems (device and/or detection chemistry) cause periodicity in the amplification curve data (Spiess et al. 2016). Or the commonly employed smoothing of data might case alterations to the raw data that affects both the estimation of the template material (Cq value) and the amplification efficiency (Spiess et al. 2015).

At this level all software assumes that the amplification resemble a sigmoid curve shape (ideal positive amplification reaction) or a flat low line (ideal negative amplification reaction). For example, Ritz and Spiess (2008) published an R package that contains functions to fit several multi-parameter models. This includes the five-parameter Richardson function, which is often used for the analysis of qPCR data.

Most software packages do not take into account if an amplification curve fulfills a certain criterion. For example, they do not try to make a classification if an amplification curve is positive (good quality) or negative (bad quality). This is a needed information for some algorithms. For example, the *linreg* method by Ruijter et al. (2009) requires a decision, if an amplification curve is positive or negative.

To illustrate this problem we give and example for the analysis of amplification curves of the `htPCR` data set in the section about the function `decision_modus()`.

In summary, researchers solved many challenges that were daunting the users of qPCR devices in the past. In particular, algorithms for the processing of the positive amplification curves are widely available. A bottleneck of qPCR data analysis is the lack of classifier for amplification curve data. Classifier herein refer to a vector of features that can be used to distinguish the amplification curves by their shape only.

One reason for this is the lack of features that are known for amplification curve data. Only few features for amplification curves are described in the literature. For example, tools described by us are the `amptester()` and the `amptester.gui()` functions, which are part of the `chipPCR` package ((Rödiger, Burdukiewicz, and Schierack 2015)). The **qpcR** package (Ritz and Spiess 2008) contains an amplification curve test via the `modlist()` function. The parameter check = "uni2" offers an analytical approach, as part of a method for the kinetic outlier detection. It checks for a sigmoid structure of the amplification curve. Then it tests for the location of the first derivative maximum and the second derivative maximum. However, multi-parameter functions, like sigmoid models, fit "successful" in most cases including noise and give false positive results.

2 Aims of the `PCRedux` Package

The `PCRedux` package contains function and data sets for machine learning on quantitative PCR data in R. There are numerous software packages for R which can be used for the analysis of quantitative PCR (qPCR)

data (Rödiger et al. 2015, Pabinger et al. (2014)).

This vignette covers important features of the package and should be used as an addendum to the manual.

Reproducible research is an important foundation of research. Both, the technical and the experimental aspects need to be performed under principles that follow good practice of reproducible research. Several authors addressed the matter (Huggett, O'Grady, and Bustin 2014, Bustin (2014), Bustin (2017), Rödiger et al. (2015), Rödiger et al. (2017), Wilson et al. (2016)). During our literature studies we found barely material, which can be used for further studies. Therefore, we included data set of amplification curves, human rated amplification curves (negative, ambiguous, positive) and examples in this package.

3 Functions of the PCRedux Package

The function described following are aimed for experimental studies. It is important to note that the features proposed herein emerged during a critical reasoning process. The aim of the package is to propose a set of features, functions and data for an independent research.

3.1 autocorrelation_test() - A Function to Detect Positive Amplification Curves

Autocorrelation analysis is a technique that is used in the field of time series analysis. It can be used to reveal regularly occurring patterns in one-dimensional data (Spiess et al. 2016). The autocorrelation measures the correlation of a signal $f(t)$ with itself shifted by some time delay $f(t-\tau)$.

The `autocorrelation_test()` function coercers the amplification curve data to an object of the class "zoo" (*zoo* package) as indexed totally ordered observations. Next follows the computation of a lagged version of the amplification curve data. The shifting the amplification curve data is based back by a given number of observations (default $\tau = 3$). Then follows a significance test for correlation between paired samples (amplification curve data & lagged amplification curve data). The hypothesis is that the paired sample of positive amplification curves has a significant correlation (`stats::cor.test`, significance level is 0.01) in contrast to negative amplification curves (noise). The application of the `autocorrelation_test()` function is shown in the following example.

```
# Test for autocorrelation in amplification curve data
# Load the libraries magrittr for pipes and qpcR for the data
library(magrittr)
library(qpcR)
library(PCRedux)
# Test for autocorrelation in the testdat data set
res_ac <- sapply(2L:ncol(testdat), function(i) {
    autocorrelation_test(testdat[, i])
}
)

# Plot curve data as overview
# Define the colors for the amplification curves
colors <- rainbow(ncol(testdat)-1, alpha=0.3)
# Names of samples
samples <- colnames(testdat)[-1]
layout(matrix(c(1,2,1,3), 2, 2, byrow = TRUE))
matplot(testdat[, 1], testdat[, -1], xlab="Cycle", ylab="RFU",
        main="testdat data set", type="l", lty=1, col=colors, lwd=2)
legend("topleft", samples, pch=19, col=colors, ncol=2, bty="n")
```

```
mtext("A", cex = 2, side = 3, adj = 0, font = 2)

# Curves rated by a human after analysis of the overview. 1 = positive,
# 0 = negative
human_rating <- c(1,1,0,0,1,1,0,0,
                  1,1,0,0,1,1,0,0,
                  1,1,0,0,1,1,0,0)

# Convert the n.s. (not significant) in 0 and others to 1.
# Combine the results of the aromatic autocorrelation_test as variable "ac",
# the human rated values as variable "hr" in a new data frame (res_ac_hr).
res_ac_hr <- data.frame(ac=ifelse(res_ac=="n.s.", 0, 1),
                        hr=human_rating) %>% as.matrix
                        res_performeR <- performeR(res_ac_hr[, "ac"], res_ac_hr[, "hr"])

                        # Add ratings by human and autocorrelation_test to the plot
                        par(las=2)
                        plot(1:nrow(res_ac_hr), res_ac_hr[, "hr"], xlab="Sample",
                  ylab="Decisions", xaxt="n", yaxt="n", pch=19)
                        axis(2, at=c(0,1), labels=c("negative", "positive"), las=2)
                        axis(1, at=1:nrow(res_ac_hr), labels=colnames(testdat)[-1], las=2)
                        points(1:nrow(res_ac_hr), res_ac_hr[, "ac"], pch=1, cex=2,
                   col="red")
                        legend("topleft", c("Human", "autocorrelation_test"), pch=c(19,1),
                               bty="n", col=c("black","red"))
                        mtext("B", cex = 2, side = 3, adj = 0, font = 2, las=0)

                        barplot(as.matrix(res_performeR[, c(1:10,12)]), yaxt="n", ylab="",
                                main="Performance of autocorrelation_test")
                        axis(2, at=c(0,1), labels=c("0", "1"), las=2)
                        mtext("C", cex = 2, side = 3, adj = 0, font = 2, las=0)
```

As shown in this example, the `autocorrelation_test()` function is able to distinguish between positive and negative amplification curves. Negative amplification curve were in all cases non-significant. In contrast, the coefficients of correlation for positive amplification curves ranged between 0.955 and 0.963 at a significance level of 0.01 and a lag of 3.

3.2 `decision_modus()` - A Function to Get a Decision (Modus) from a Vector of Classes

Several approaches for machine learning exist. It is a very rich topic. One of them is the supervised learning, where the aim is to infer a function from labeled training data. Labeled training data are for example created by a human. As described in the sections before, amplification curves can be labeled as negative, ambiguous or positive for example (Figure 4) were taken from the `htPCR` data set (compare Figure 1B).

Provided that different users are given the task to rate the amplification curves, it is likely that the ambiguous amplification curve (P06.W47) in Figure 4 will get another rating (ambiguos → positive) from an inexperienced user. We performed exactly this kind of experiment with different users. For this task the amplification curve data were analyzed as described in Rödiger, Burdukiewicz, and Schierack (2015) (supplementary information).

The following table gives the first five and the last five rows for the rating of the amplification curves for the `htPCR` data set (in total 8858 amplification curves). All curves were rated randomly three times. All raw data a contained in the CSV file `decision_res_htPCR.csv` as part of the `PCRedux` package.

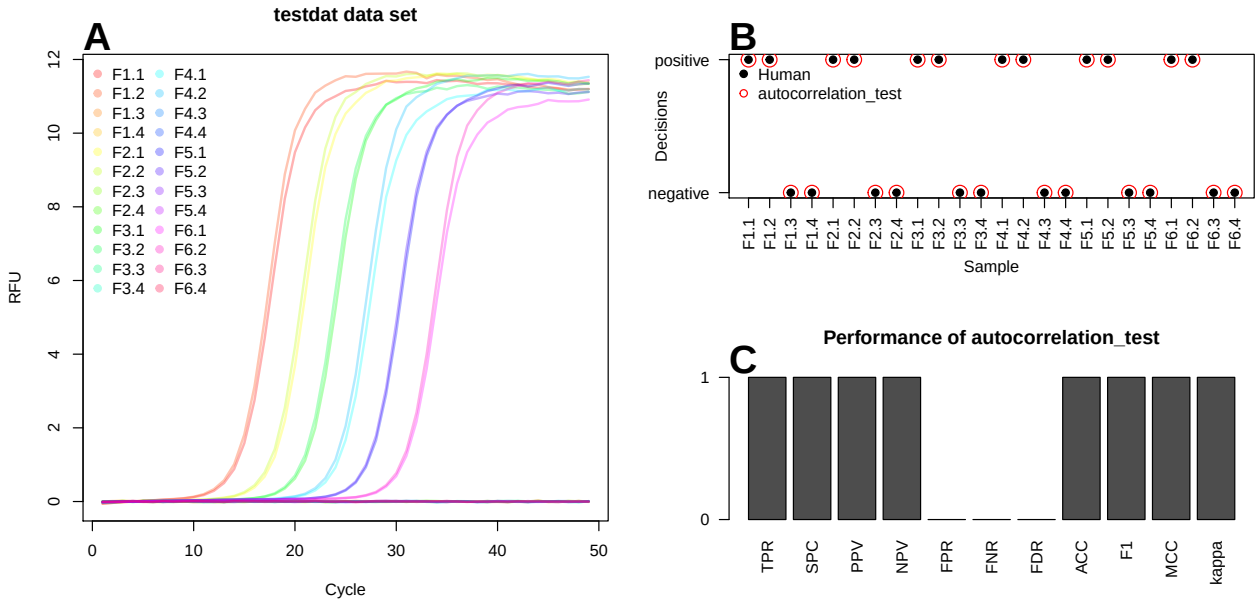

Figure 2: Autocorrelation analysis for amplification curves of the **testdat** data set (*qpcR* package, (Ritz and Spiess 2008)). A) All amplification curves of the **testdat** data set. B) Positive curves and negative curves as determined by the `autocorrelation_test()` and a human. C) Performance analysis by the `performeR()` function (see Section 3.8 for details).

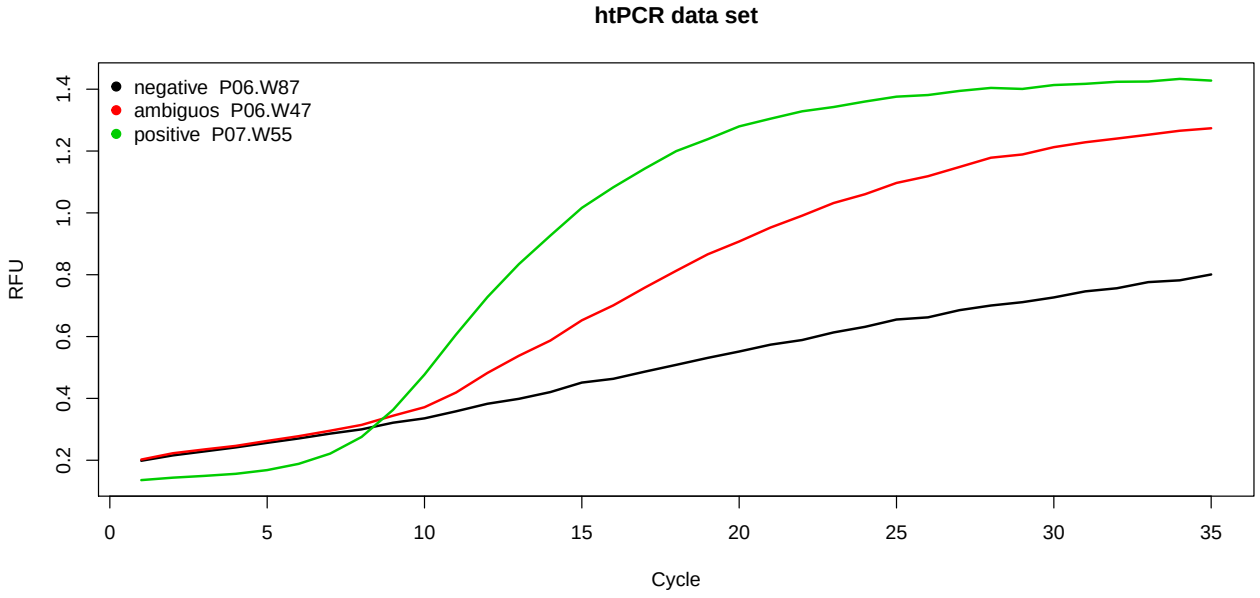

Figure 3: Sample data from the **htPCR** data set (*qpcR* package, (Ritz and Spiess 2008)) with negative (black), ambiguous (red) or positive (green) amplification curves. The negative amplification curve has no sigmoid curvature, just a strong positive trend. The ambiguous amplification curve is not a perfect sigmoid curve, exhibits a positive slope in the background phase (cylce 1 $ ightarrow$ 5) and a low amplitude. In contrast, the positive amplification curve has a sigmoid curvature, starting with a background phase (cylce 1 $ ightarrow$ 5), an exponential phase (cylce 6 $ ightarrow$ 25) and a plateau phase (cylce 26 $ ightarrow$ 35).

```
library(PCRedux)
suppressMessages(library(data.table))
library(magrittr)
filename <- system.file("decision_res_htPCR.csv", package="PCRedux")
decision_res_htPCR <- fread(filename) %>% as.data.frame()
head(decision_res_htPCR)
```

```
##    sample test.result.1 test.result.2 test.result.3 conformity
## 1 P01.W01             y             a             a      FALSE
## 2 P01.W02             a             y             a      FALSE
## 3 P01.W03             y             a             a      FALSE
## 4 P01.W04             y             a             a      FALSE
## 5 P01.W05             y             a             a      FALSE
## 6 P01.W06             a             a             a       TRUE
```

```
tail(decision_res_htPCR)
```

```
##       sample test.result.1 test.result.2 test.result.3 conformity
## 8853 P95.W91             a             a             a       TRUE
## 8854 P95.W92             a             a             a       TRUE
## 8855 P95.W93             a             a             a       TRUE
## 8856 P95.W94             y             y             y       TRUE
## 8857 P95.W95             y             a             a      FALSE
## 8858 P95.W96             y             a             a      FALSE
```

As seen in the example the rating for the same amplification curves vary (e.g., row 1 "P01.W01"), while others are in agreement (e.g., row 8856 "P95.W94"). `decision_modus()` is a function that can be used to find the most frequent (modus) decision in such a data set. The classes were defined by the user (e.g., "a", "n", "y" -> "ambiguous", "negative", "positive"). This function is useful if large collections of varying decision (e.g., "a", "a", "a", "n", "n") need to be condensed to a single decision (3 x "a", 2 x "n" → "a").

We used the `decision_modus()` function on the `decision_res_htPCR.csv` data set with all the ratings (column 2 to 4) and determined the modus of each line.

```
# Use decision_modus to go rowise through all human ratings

dec <- lapply(1L:nrow(decision_res_htPCR), function(i) {
decision_modus(decision_res_htPCR[i, 2:4])
}) %>% unlist

names(dec) <- decision_res_htPCR[, 1]

# Show statistic of the decisions
summary(dec)
```

```
##    a    y    n
## 2839 5670  349
```

Another usage of the `decision_modus()` function is by setting the `max_freq` parameter to `FALSE`. This option gives the counts of all decisions.

```
# Decisions for sample P01.W06
decision_modus(decision_res_htPCR[which(decision_res_htPCR[["sample"]] == "P01.W06"),
               2:4], max_freq=FALSE)
```

```
##   variable freq
## 1        a    3
```

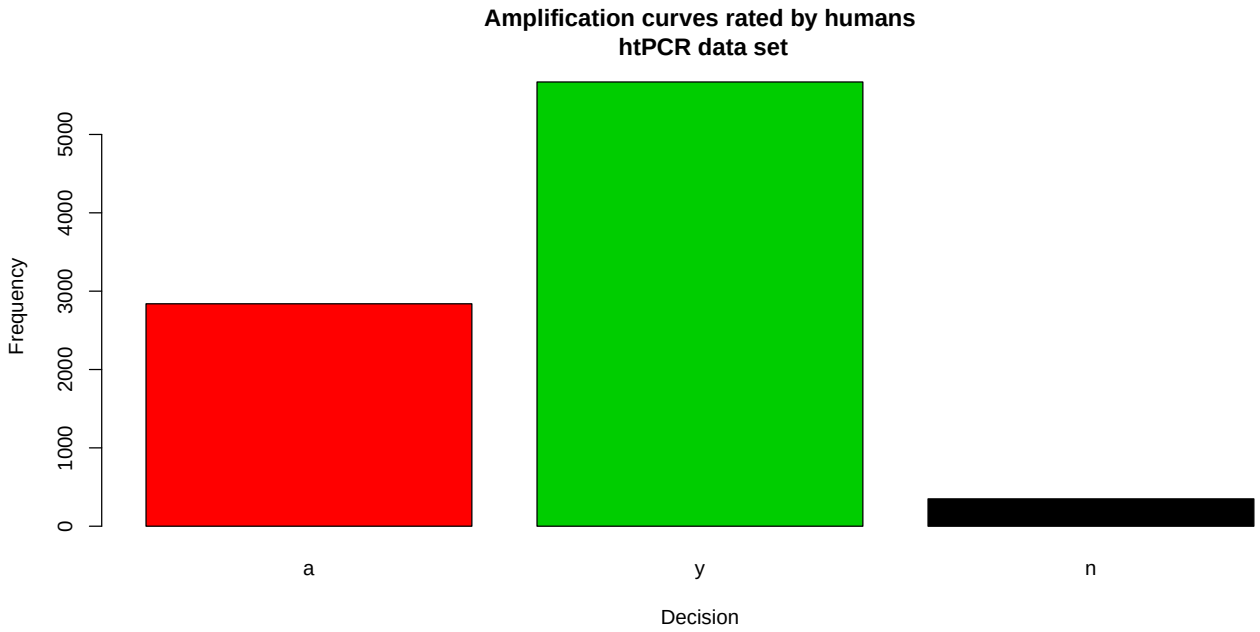

Figure 4: Summary of all sample data from the **htPCR** data set (*qpcR* package, (Ritz and Spiess 2008)) with negative (black), ambiguous (red) or positive (green) amplification curves.

```
# Decisions for sample P01.W13
decision_modus(decision_res_htPCR[which(decision_res_htPCR[["sample"]] == "P01.W13"),
               2:4], max_freq=FALSE)
```

```
##   variable freq
## 1        y    2
## 2        a    1
```

```
# Load the human rated data sets decision_res_htPCR.csv and the reevaluation
# decision_res_htPCR_reevaluation.csv thereof.
# Load the decision_res_htPCR.csv data set
filename <- system.file("decision_res_htPCR.csv", package="PCRedux")
decision_res_htPCR <- as.data.frame(fread(filename))

# Load the decision_res_htPCR_reevaluation.csv data set
filename <- system.file("decision_res_htPCR_reevaluation.csv", package="PCRedux")
decision_res_htPCR_reevaluation <- as.data.frame(fread(filename))

# Show the first five elements of both data sets
head(decision_res_htPCR)
```

```
##    sample test.result.1 test.result.2 test.result.3 conformity
## 1 P01.W01             y             a             a      FALSE
## 2 P01.W02             a             y             a      FALSE
## 3 P01.W03             y             a             a      FALSE
## 4 P01.W04             y             a             a      FALSE
## 5 P01.W05             y             a             a      FALSE
## 6 P01.W06             a             a             a       TRUE
```

```
head(decision_res_htPCR_reevaluation)
```

```
##   htPCR_reevaluate test.result.1 test.result.2 test.result.3 conformity
## 1          P01.W01             y             a             a      FALSE
## 2          P01.W02             a             a             a       TRUE
## 3          P01.W03             n             a             n      FALSE
## 4          P01.W04             a             a             a       TRUE
## 5          P01.W05             a             a             a       TRUE
## 6          P01.W08             a             a             a       TRUE
```

```
# Renanme the columns to give them the same pattern
pattern <- c("sample", "test.result.1", "test.result.2", "test.result.3", "conformity")
colnames(decision_res_htPCR_reevaluation) <- pattern
colnames(decision_res_htPCR) <- pattern

# Merge the two data sets based on the sample name (amplification curve).
decision_res_htPCR_merged <- merge(decision_res_htPCR[, -5],
                                   decision_res_htPCR_reevaluation[, -5],
                                   by="sample")

# Show the first five elements of the merged data sets
head(decision_res_htPCR_merged)
```

```
##    sample test.result.1.x test.result.2.x test.result.3.x test.result.1.y
## 1 P01.W01               y               a               a               y
## 2 P01.W02               a               y               a               a
## 3 P01.W03               y               a               a               n
## 4 P01.W04               y               a               a               a
## 5 P01.W05               y               a               a               a
## 6 P01.W08               y               a               a               a
##   test.result.2.y test.result.3.y
## 1               a               a
## 2               a               a
## 3               a               n
## 4               a               a
## 5               a               a
## 6               a               a
```

Plot everything.

```
par(mfrow=c(1,3))


data <- decision_res_htPCR_merged
decision_plot <- function(data, name="", rating_columns=2:4){
    dec <- lapply(1L:nrow(data), function(i) {
        decision_modus(data[i, c(rating_columns)])
    }) %>% unlist
    barplot(table(dec), xlab="Decision", ylab="Frequency",
            main=paste("Amplification curves rated by humans\n htPCR data set",
                       name), col=c(2,3,1), ylim=c(0,6000))
}

decision_plot(decision_res_htPCR, name="", rating_columns=2:4)
decision_plot(decision_res_htPCR_reevaluation, name="", rating_columns=2:4)
```

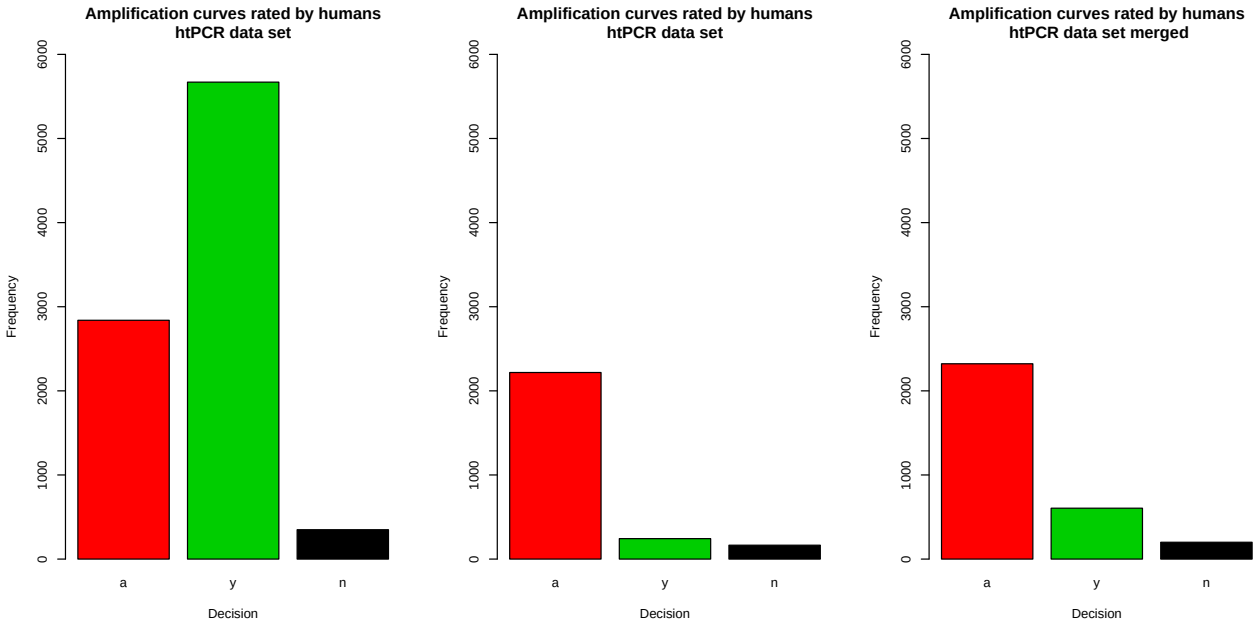

Figure 5: decision plot.

```
decision_plot(decision_res_htPCR_merged, name="merged", rating_columns=2:7)
```

3.3 `earlyreg()` - A Function to Calculate the Slope and Intercept of Amplification Curve Data from a qPCR Experiment

There are numerous segments of an amplification curve, that are potentially useful to extract a feature for an amplification curve classification. Some amplification curves deviate from the curvature as expected from a sigmoid model. In particular, some amplification curve have just a positive (linear) trend without typical infliction points. Others show a step signal increase even in the background phase (cycle 1 to 10) (compare Rödiger, Burdukiewicz, and Schierack (2015), Figure S12). The slope and the intercept in background region is an interesting target for a classification.

Gunay, Goceri, and Balasubramaniyan (2016) used a "modified sigmoid function for CT [Cq] prediction". The fundamental assumption of this approach is, that such a model can be applied to any data set. There are several reasons to why such assumption is flawed. For example is shown in section 3.5 about the `hookreg()` and `hookregNL()` functions. There it becomes evident that a three parameter model fit will give wrong predictions. Moreover, non-linear function tend to fit even noise (Figure 6).

```
# Load the drc package for the five-parameter log-logistic fit.
library(drc)
```

```
##
## 'drc' has been loaded.

## Please cite R and 'drc' if used for a publication,

## for references type 'citation()' and 'citation('drc')'.

##
## Attaching package: 'drc'

## The following object is masked from 'package:qpcR':
##
```

```
##     mselect

## The following objects are masked from 'package:stats':
##
##     gaussian, getInitial
```

```
library(chipPCR)

# Load the testdat data set from the qpcR package without
# loading the entire package.
data(testdat, package="qpcR")

# Arrange graphs in an orthogonal matrix and set the plot parmeters.
par(mfrow = c(2,2))

# Apply the the amptester function to all amplification curve data
# and write the results to the main of the plots.

curve_data_columns <- c(2,4,9,20)

for(i in 1L:length(curve_data_columns)) {
    res.ampt <- suppressMessages(amptester(testdat[, curve_data_columns[i]]))

    # Make a logical connection by two tests (shap.noisy, lrt.test and
    # tht.dec) of amptester to decide if an amplification reaction is
    # positive or negative.
    decision <- ifelse(!res.ampt@decisions[1] &&
                       res.ampt@decisions[2] &&
                       res.ampt@decisions[4],
                       "positive", "negative")

    plot(testdat[, 1], testdat[, curve_data_columns[i]],
         xlab = "Cycle", ylab = "RFU", pch = 19, main = "")
    mtext(LETTERS[i], cex = 2, side = 3, adj = 0, font = 2)
    legend("bottomright", paste0(colnames(testdat)[curve_data_columns[i]],
                                 "\nDecision: ", decision),
           bty="n", cex=1.25, col="red")
    # Use the drm function with a five-parameter log-logistic fit model.

        try(lines(predict(drm(testdat[, curve_data_columns[i]] ~ testdat[, 1],
                              fct = LL.3())), col = 2), silent=TRUE)

        try(lines(predict(drm(testdat[, curve_data_columns[i]] ~ testdat[, 1],
                              fct = LL.4())), col = 3), silent=TRUE)
 }
```

```
## Warning in sqrt(diag(varMat)): NaNs wurden erzeugt

## Warning in summary.lm(lm.dat.model): essentially perfect fit: summary may
## be unreliable

## Warning in summary.lm(object): essentially perfect fit: summary may be
## unreliable
```

We suggest using the slope and the intercept for the differentiation of the different amplification curves. Therefore, we developed the function `earlyreg()`. This wrapper function uses a MM-type estimator for a robust linear regression (Todorov and Filzmoser 2009).

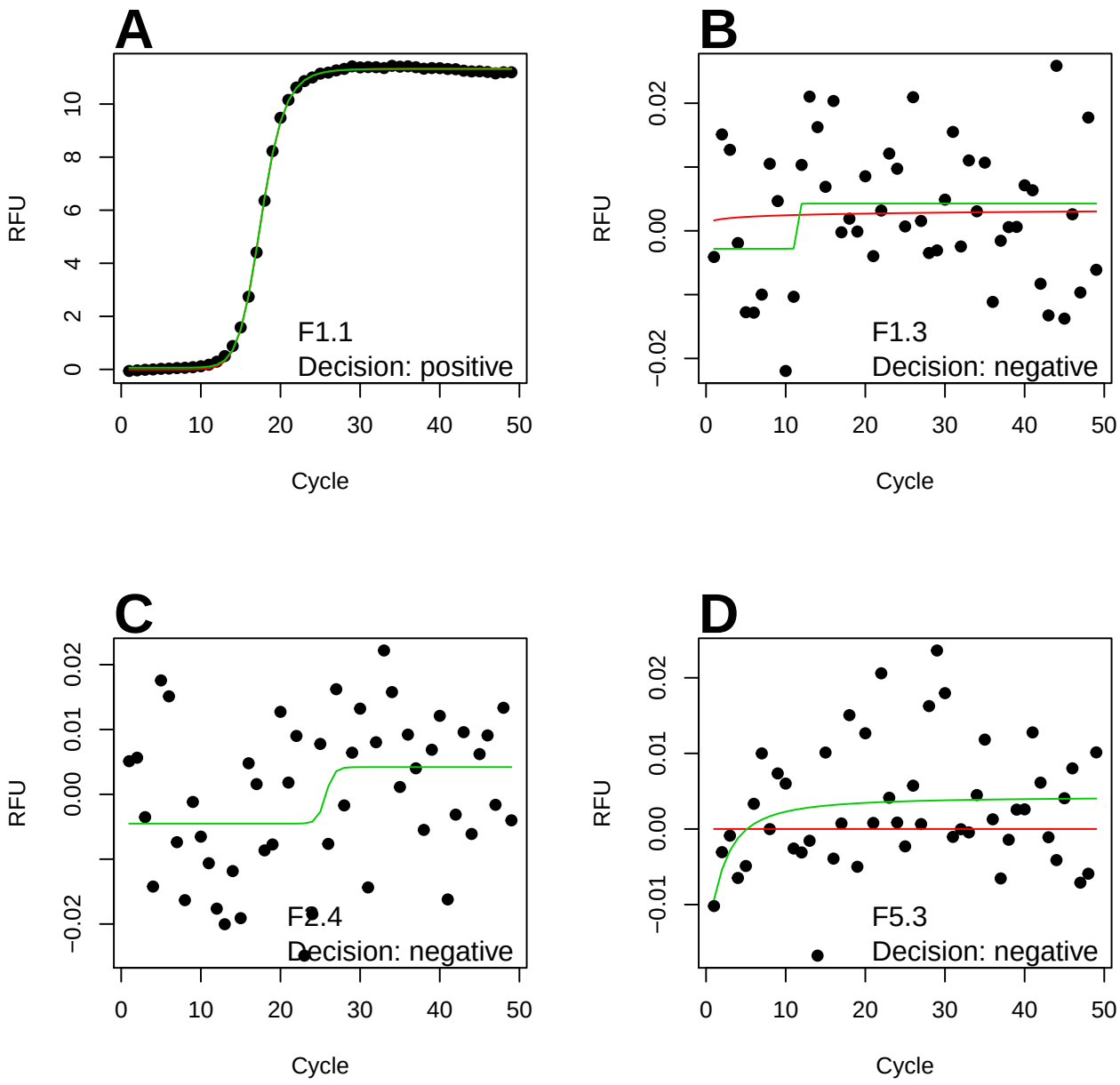

Figure 6: Sample data from the **testdat** data set (*qpcR* package, (Ritz and Spiess 2008)) with negative and positive amplification curves.

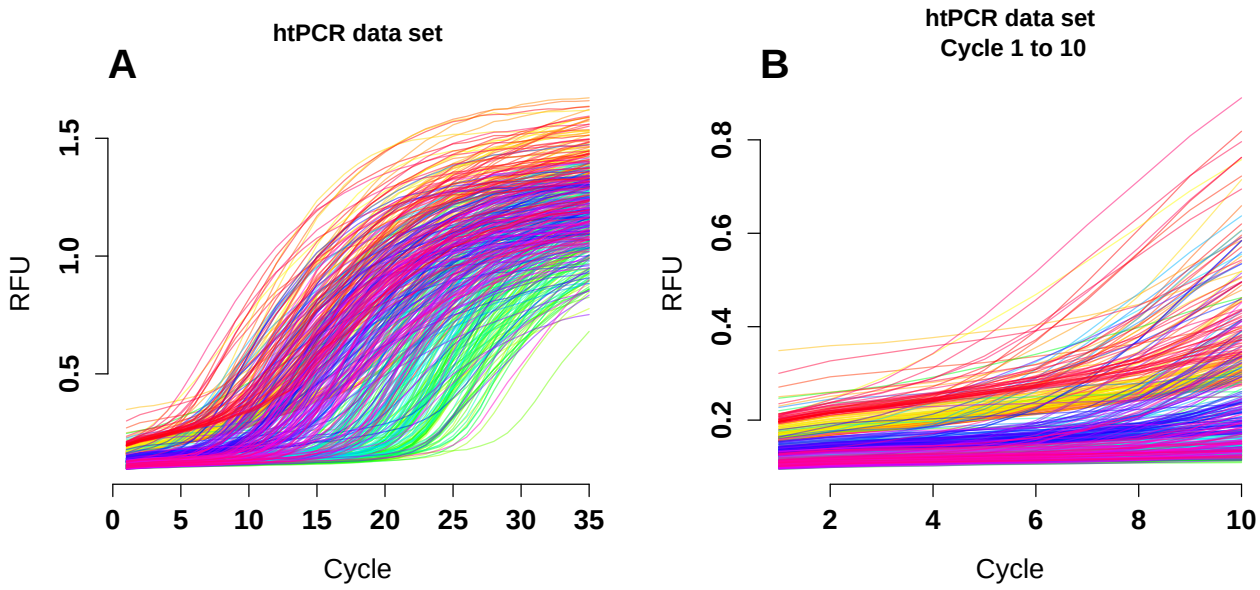

Figure 7: Amplification curves from the **htPCR** data set (*qpcR* package). The amplification curves show different slopes in the early phase (cycle 1 to 10) of the qPCR.

`earlyreg()` ignores the first cycle value and includes by default the next six cycle-dependent amplitude values to perform the regression on them. The exclusion of the first values is based on empirical expert knowledge. Many qPCR system report initial values in cycle one, which largely deviate from subsequent values in the background phase. Therefore, extreme values are removed. For the analysis we used arbitrarily the first 500 amplification curves of the `htPCR` data set.

The following example illustrates a potential use of the `earlyreg()` function on the `htPCR` data set. Figure 7 A shows all amplification curves. Figure 7 B shows the first ten cycles only.

```
par(bty="n", font=2, font.axis=2, las=1, cex.axis=1.3, cex.lab=1.3, lwd=1.3,
    bg="white", oma=c(.5,.5,.5,.5))
data <- htPCR[, 1:501]
curve_colors <- c(rainbow(ncol(data)-1, alpha=.5))
range <- 1:10

par(mfrow=c(1,2), las=0, bty="n", oma=c(1,1,1,1))

matplot(data[, 1], data[, -1], col= curve_colors, pch=19, lty=1, type="l",
        xlab="Cycle", ylab="RFU", main="htPCR data set")
mtext("A", cex = 2, side = 3, adj = 0, font = 2)

matplot(data[range, 1], data[range, -1], col= curve_colors, pch=19, lty=1,
        type="l", xlab="Cycle", ylab="RFU",
        main="htPCR data set\n Cycle 1 to 10")
mtext("B", cex = 2, side = 3, adj = 0, font = 2)
```

Next we calculated of the slope and intercept with the `earlyreg()` function for the first six cycles of the qPCR data. The results were subjected to a cluster analysis by k-means clustering. For the analysis we focused on the slope, because the intercept is representing the background fluorescence in the early phase of the qPCR. As shown in Figure 8B exhibited the density function of all intercept values a mono-modal shaped bell curve. In contrast, the density function of all slope values had a bi-modal shaped bell curve (figure 8C). Therefore, the slope appears to be an indicator for differences between the qPCR amplification curves.

The cluster analysis by k-means (k=3, assumed weak slope, moderate slope, strong slope) on the slope data was plotted. The clusters grouped the amplification curve data in curves with no-background region and high slope, amplification curves with no-background region and a weaker slope and amplification curves with background region and a weak slope (figure 8D-F).

To conclude, this approach might be of use for the classification of amplification curves.

```
# Normalize each amplification curve to their 0.99 percentile and use the
# earlyreg function to determine the slope and intercept of the first
# 6 cycles

res_slope_intercept <- do.call(rbind, lapply(2L:ncol(data), function(i){
                      earlyreg(x=data[, 1], y=data[, i], range=7, normalize=TRUE)
                     }))
# Label the sample with their original names
rownames(res_slope_intercept) <- colnames(htPCR)[2L:ncol(data)]
# Use k-means for cluster analysis
res_kmeans <- kmeans(res_slope_intercept[, "slope"], 3)$"cluster"

par(mfrow=c(2,3))
plot(res_slope_intercept, col=res_kmeans)
mtext("A", cex = 2, side = 3, adj = 0, font = 2)

plot(density(res_slope_intercept[, "intercept"]), main="Intercept")
mtext("B", cex = 2, side = 3, adj = 0, font = 2)

plot(density(res_slope_intercept[, "slope"]), main="Slope")

for(i in unique(res_kmeans)) {
    abline(v=min(res_slope_intercept[res_kmeans==i, "slope"]), col=i)
}
mtext("C", cex = 2, side = 3, adj = 0, font = 2)

matplot(data[, 1], data[, which(res_kmeans==1)+1], col= curve_colors, pch=19, lty=1,
 type="l", xlab="Cycle", ylab="RFU",
 main="htPCR data set (1-500)\n Cluster 1", ylim=c(min(data[, -1]), 0.4))
mtext("D", cex = 2, side = 3, adj = 0, font = 2)

matplot(data[, 1], data[, which(res_kmeans==2)+1], col= curve_colors, pch=19, lty=1,
 type="l", xlab="Cycle", ylab="RFU",
 main="htPCR data set (1-500)\n Cluster 2", ylim=c(min(data[, -1]), 0.4))
mtext("E", cex = 2, side = 3, adj = 0, font = 2)

matplot(data[, 1], data[, which(res_kmeans==3)+1], col= curve_colors, pch=19, lty=1,
 type="l", xlab="Cycle", ylab="RFU",
 main="htPCR data set (1-500)\n Cluster 3", ylim=c(min(data[, -1]), 0.4))
mtext("F", cex = 2, side = 3, adj = 0, font = 2)
```

3.4 `head2tailratio()` - A Function to Calculate the Ratio of the Head and the Tail of a Quantitative PCR Amplification Curve

The `head2tailratio()` function calculates the ratio of the head and the tail of a quantitative PCR amplification curve. Positive amplification curves have different slopes and intercepts at the begin of the amplification curve (head, background region) and the end of the amplification curve (tail, plateau region). Therefore,

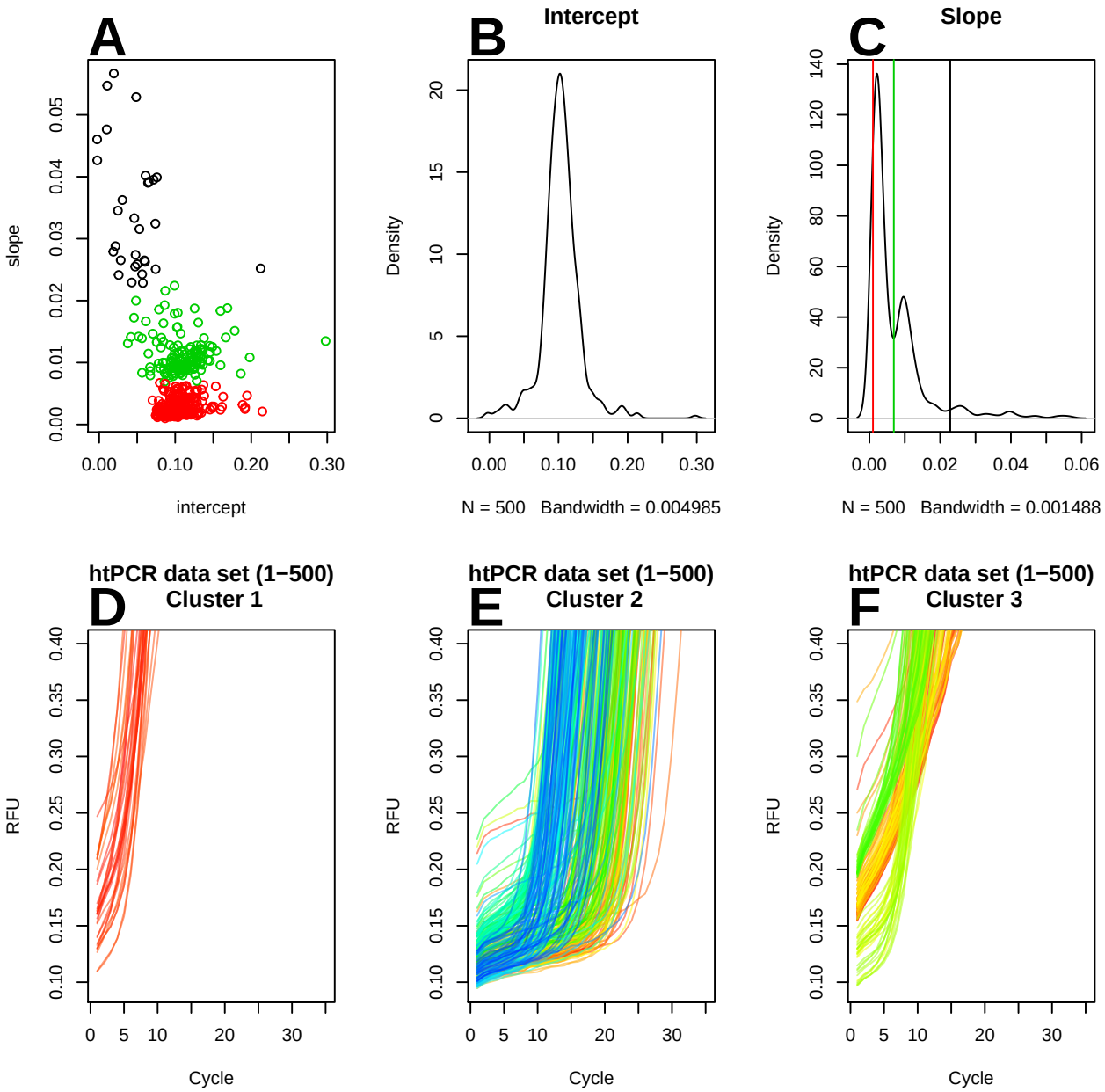

Figure 8: Clusters of samples according to their slope and intercept. The **htPCR** data set (*qpcR* package) was used. The amplification curves show different slopes in the early phase (cycle 1 to 10) of the qPCR. Both the slope and the intercept were used for a cluster analysis (k-means).

these segments are potentially useful to extract a feature for an amplification curve classification.

```
suppressMessages(library(qpcR))


colors <- c("#FF00004D", "#80FF004D", "#00FFFF4D", "#8000FF4D")

data <- dil4reps94[, c(1, 2, 98, 194,290)]

res_head2tailratio <- lapply(2L:ncol(data), function(i) {
    head2tailratio(y=data[, i], normalize=TRUE, slope_normalizer=TRUE,
                   verbose=TRUE)
})

data_normalized <- cbind(data[, 1],
              sapply(2L:ncol(data), function(i){
                  data[, i] / quantile(data[, i], 0.999)
             })
              )

par(mfrow=c(1,2), las=0, bty="n", oma=c(1,1,1,1))

matplot(data_normalized[, 1], data_normalized[, -1],
        xlab="Cycle", ylab="normalized RFU", main="dil4reps94 data set\nsubset",
        type="l", lty=1, lwd=2, col=colors
       )
for(i in 1L:(ncol(data_normalized)-1)) {
    points(res_head2tailratio[[i]]$x_roi, res_head2tailratio[[i]]$y_roi,
           col=colors[i], pch=19, cex=1.5)
    abline(res_head2tailratio[[i]]$fit, col=colors[i], lwd=2)
}

mtext("A", cex = 2, side = 3, adj = 0, font = 2)

colors <- c(rep("#FF00004D", 94),
            rep("#80FF004D", 94),
            rep("#00FFFF4D", 94),
            rep("#8000FF4D", 94)
            )

matplot(dil4reps94[, 1], dil4reps94[, -1], xlab="Cycle", ylab="RFU",
        main="dil4reps94 data set", type="l", lty=1, lwd=2, col=colors)
mtext("B", cex = 2, side = 3, adj = 0, font = 2)
```

3.5 `hookreg()` and `hookregNL()` - Functions to Detect Hook Effekt-like Curvatures

`hookreg()` and `hookregNL()` are functions to detect amplification curves bearing a hook effect or negative slope at the end of the amplification curve. Both functions calculate the slope and intercept of an amplification curve data. The idea is that a strong negative slope at the end of an amplification curve is indicative for a hook effect.

Amplification curves with a hook effect like curvature are characterized by a negative trend in the late phase of the amplification reaction (Figure 10 A, curve F1.1, F1.2, F2.1, F2.2, F3.1 and F3.2).

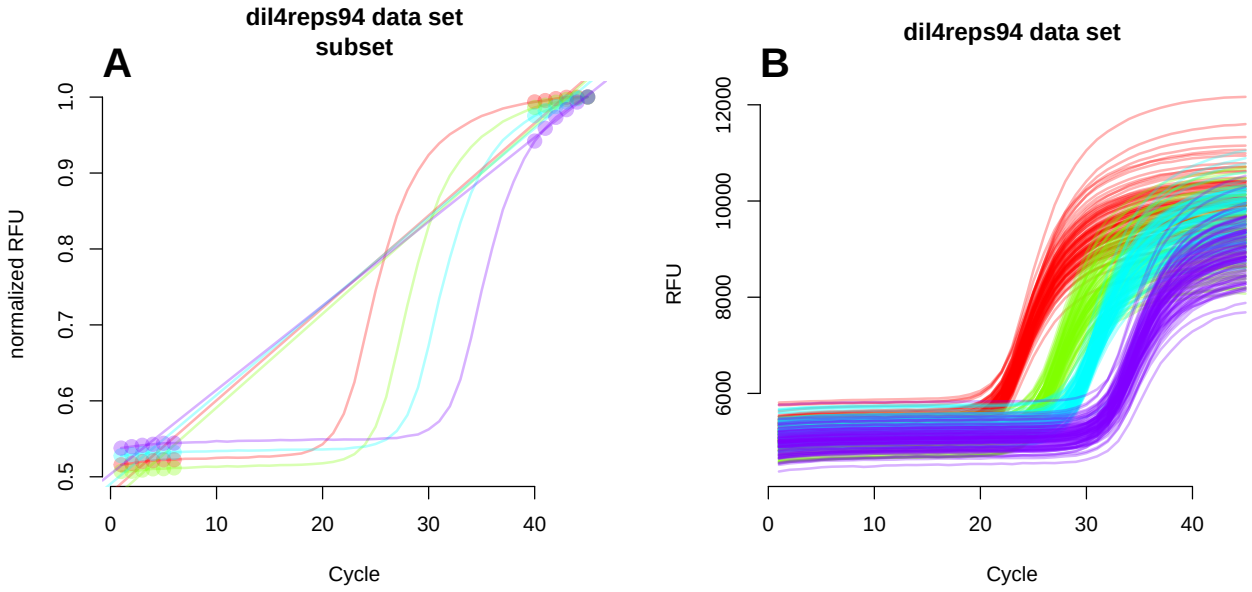

Figure 9: Calculation of the ratio between the head and the tail of a quantitative PCR amplification curve.

```
# Calculate slope and intercept on noise (negative) amplification curve data
# for the last eight cycles.
library(qpcR)
library(magrittr)

res_hook <- sapply(2L:ncol(boggy), function(i) {
    hookreg(x=boggy[, 1], y=boggy[, i])}) %>% t %>%
    data.frame(sample=colnames(boggy)[-1],.)

    data_colors <- rainbow(ncol(boggy[, -1]), alpha=0.5)
    cl <- kmeans(na.omit(res_hook[, 2:3]), 2)$cluster
```

The results of the `hookreg()` analysis were transferred to a tabular format.

```
library(xtable)
print(xtable(res_hook, caption = "Screening results for the analysis with the
             hookreg algorithm. Samples withe a value of 1 in the hook column
             had all a hook effect like curvature. The samples F4.1, F4.2, F5.1,
             F5.2, F6.1 and F6.2 miss entries because the hoogreg algorithm
             could not fit a linear model. This is an expected behavior, since these
             amplification curve did not have a hook effect like curvature.",
             label='tablehookreg'), include.rownames = FALSE)
```

% latex table generated in R 3.4.2 by xtable 1.8-2 package % Mon Nov 20 16:52:55 2017

In Table 1 is shown that the first amplification curves (F1.1, F1.2, F2.1, F2.2, F3.1 and F3.2) appear the have a hook effect-like curvature ("hook" column = 1.00). The function estimate reliably the start of the of the hook effect-like region.

The clusters for amplification curve were determined by k-means clustering in this example. Next we plot the results of the analysis (Figure 10). For the visualization the intercepts was plotted against the slope with the clusters as determined by k-means clustering.

sample	intercept	slope	hook.start	hook.delta	p.value	CI.low	CI.up	hook.fit	hook.CI	hook
F1.1	1.17	-0.01	26.00	15.00	0.00	1.21	-0.01	1.00	1.00	1.00
F1.2	1.20	-0.01	26.00	15.00	0.00	1.23	-0.01	1.00	1.00	1.00
F2.1	1.16	-0.01	32.00	9.00	0.00	1.22	-0.00	1.00	1.00	1.00
F2.2	1.17	-0.01	32.00	9.00	0.00	1.21	-0.00	1.00	1.00	1.00
F3.1	1.05	-0.00	35.00	6.00	0.05	1.17	0.00	0.00	0.00	0.00
F3.2	1.08	-0.00	35.00	6.00	0.02	1.21	0.00	0.00	0.00	0.00
F4.1								0.00	0.00	0.00
F4.2								0.00	0.00	0.00
F5.1								0.00	0.00	0.00
F5.2								0.00	0.00	0.00
F6.1								0.00	0.00	0.00
F6.2								0.00	0.00	0.00

Table 1: Screening results for the analysis with the hookreg algorithm. Samples withe a value of 1 in the hook column had all a hook effect like curvature. The samples F4.1, F4.2, F5.1, F5.2, F6.1 and F6.2 miss entries because the hoogreg algorithm could not fit a linear model. This is an expected behavior, since these amplification curve did not have a hook effect like curvature.

```
    par(mfrow=c(1,2))
    matplot(x=boggy[, 1], y=boggy[, -1], xlab="Cycle", ylab="RFU",
            main="boggy Data Set", type="l", lty=1, lwd=2, col=data_colors)
    legend("topleft", as.character(res_hook$sample), pch=19,
           col=data_colors, bty="n")
    mtext("A", cex = 2, side = 3, adj = 0, font = 2)

    plot(res_hook$intercept, res_hook$slope, pch=19, cex=2, col=data_colors,
         xlab="intercept", ylab="Slope",
         main="Hook Effect-like Curvature\nboggy Data Set")
    mtext("B", cex = 2, side = 3, adj = 0, font = 2)
    points(res_hook$intercept, res_hook$slope, col=cl, pch=cl, cex=cl)
    legend("topright", c("Strong Hook effect", " Weak Hook effect"),
           pch=c(1,2), col=c(1,2), bty="n")
    text(res_hook$intercept, res_hook$slope, res_hook$sample)
```

Further examples are shown in the manual for the `hookregNL()` function.

3.6 `mblrr()` - A Function Perform the Quantile-filter Based Local Robust Regression

`mblrr()` is a function to perform the Median based Local Robust Regression (m b l r r) from a quantitative PCR experiment. In detail, this function attempts to break the amplification curve in two parts (head (~background) and tail (~plateau)). Subsequent, a robust linear regression analysis (`lmrob()`) is preformed individually on both parts. The rational behind this analysis is that the slope and intercept of an amplification curve differ in the background and plateau region. In the example shown below, the samples "P01.W19", "P06.W35", "P33.W66", "P65.W90", "P71.W23" and "P87.W01" were arbitrarily selected for demonstration purposes.

```
suppressMessages(library(qpcR))
par(mfrow=c(3,2))
data <- htPCR[, c(1, 20, 500, 3000, 6000, 6500, 8000)]
for(i in 2L:ncol(data)){
        x <- data[, 1]
```

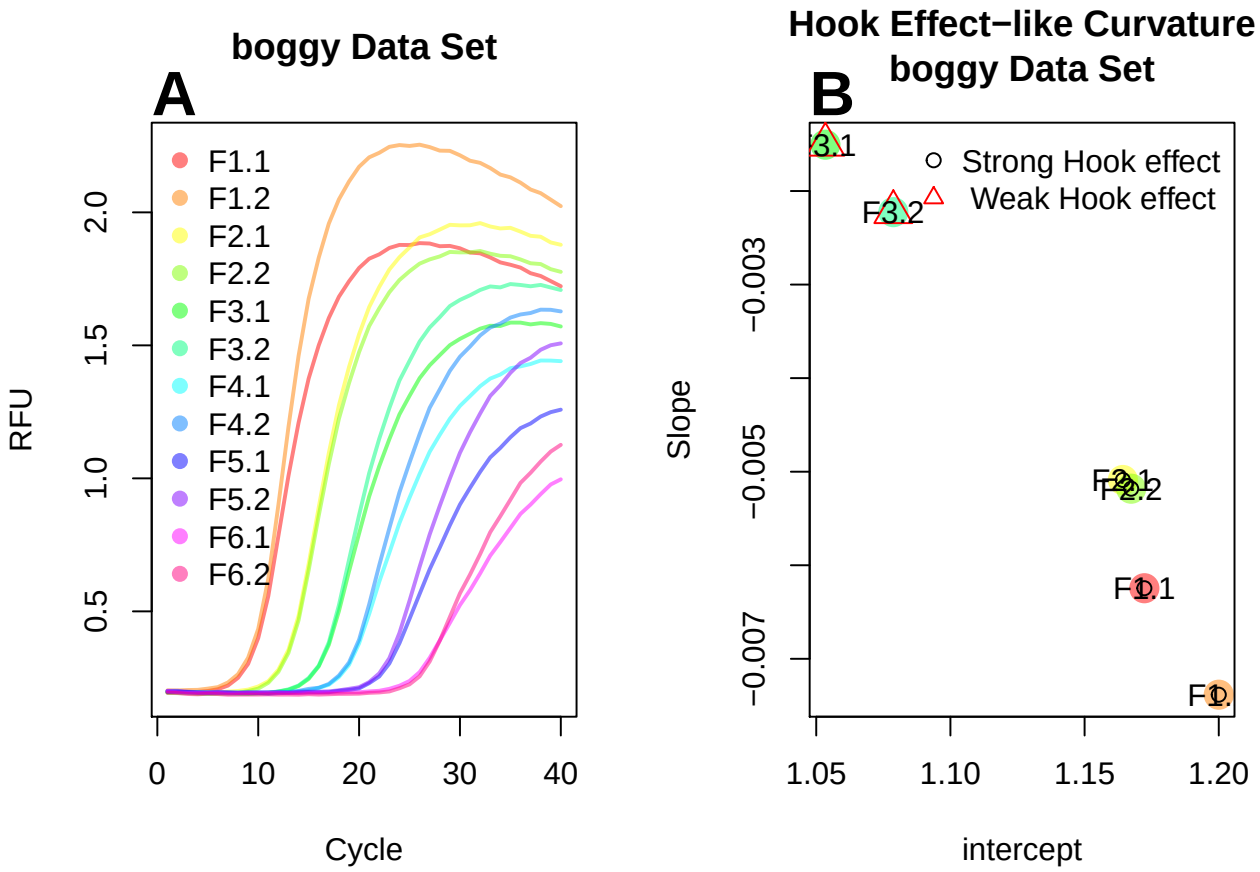

Figure 10: Analysis of amplification curves for hook effect-like structures.

```
        y_tmp <- data[, i]/quantile(data[, i], 0.999)
        res_q25 <- y_tmp < quantile(y_tmp, 0.25)
        res_q75 <- y_tmp > quantile(y_tmp, 0.75)
        res_q25_lm <- try(suppressWarnings(lmrob(y_tmp[res_q25] ~ x[res_q25])),
                          silent=TRUE)
        res_q75_lm <- try(suppressWarnings(lmrob(y_tmp[res_q75] ~ x[res_q75])),
                          silent=TRUE)

        plot(x, y_tmp, xlab="Cylce", ylab="RFU (normalized)",
             main=colnames(data)[i],
             type="b", pch=19)
        abline(res_q25_lm, col="red")
        points(x[res_q25], y_tmp[res_q25], cex=2.5, col="red")
        abline(res_q75_lm, col="green")
        points(x[res_q75], y_tmp[res_q75], cex=2.5, col="green")
    }
```

Finally we print the results of the analysis in tabular format.

```
# Load the library xtable for an appealing table output
library(xtable)

# Analyze the data via the mblrr() function

res_mblrr <- do.call(cbind, lapply(2L:ncol(data), function(i) {
                suppressMessages(mblrr(x=data[, 1], y=data[, i],
                      normalize=TRUE)) %>% data.frame
}))
colnames(res_mblrr) <- colnames(data)[-1]

# Transform the data for a tabular output and assign the results to the object
# output_res_mblrr.

output_res_mblrr <- res_mblrr %>% t

# The output variable names of the mblrr() function are rather long. For better
# readability the variable names were changed to "nBG" (intercept of head region),
# "mBG" (slope of head region), "rBG" (Pearson correlation of head region),
# "nTP" (intercept of tail region), "mTP" (slope of tail region), "rBG" (Pearson
# correlation of tail region)

colnames(output_res_mblrr) <- c("nBG", "mBG", "rBG",
                                "nTP", "mTP", "rTP")

print(xtable(output_res_mblrr, caption = "mblrr text intro. nBG, intercept of
             head region; mBG, slope of head region; rBG, Pearson
             correlation of head region; nTP, intercept of tail region; mTP,
             slope of tail region; rBG, Pearson correlation of tail region",
             label='tablemblrrintroduction'))
```

% latex table generated in R 3.4.2 by xtable 1.8-2 package % Mon Nov 20 16:52:56 2017

In another example, we combined the results from the `mblrr()` function with the classification (positive, negative) by a human rater to apply them in an analysis with Fast and Frugal Trees (FFTrees). A general introduction to decision trees is given in (Quinlan 1986, Luan, Schooler, and Gigerenzer (2011)). FFTrees

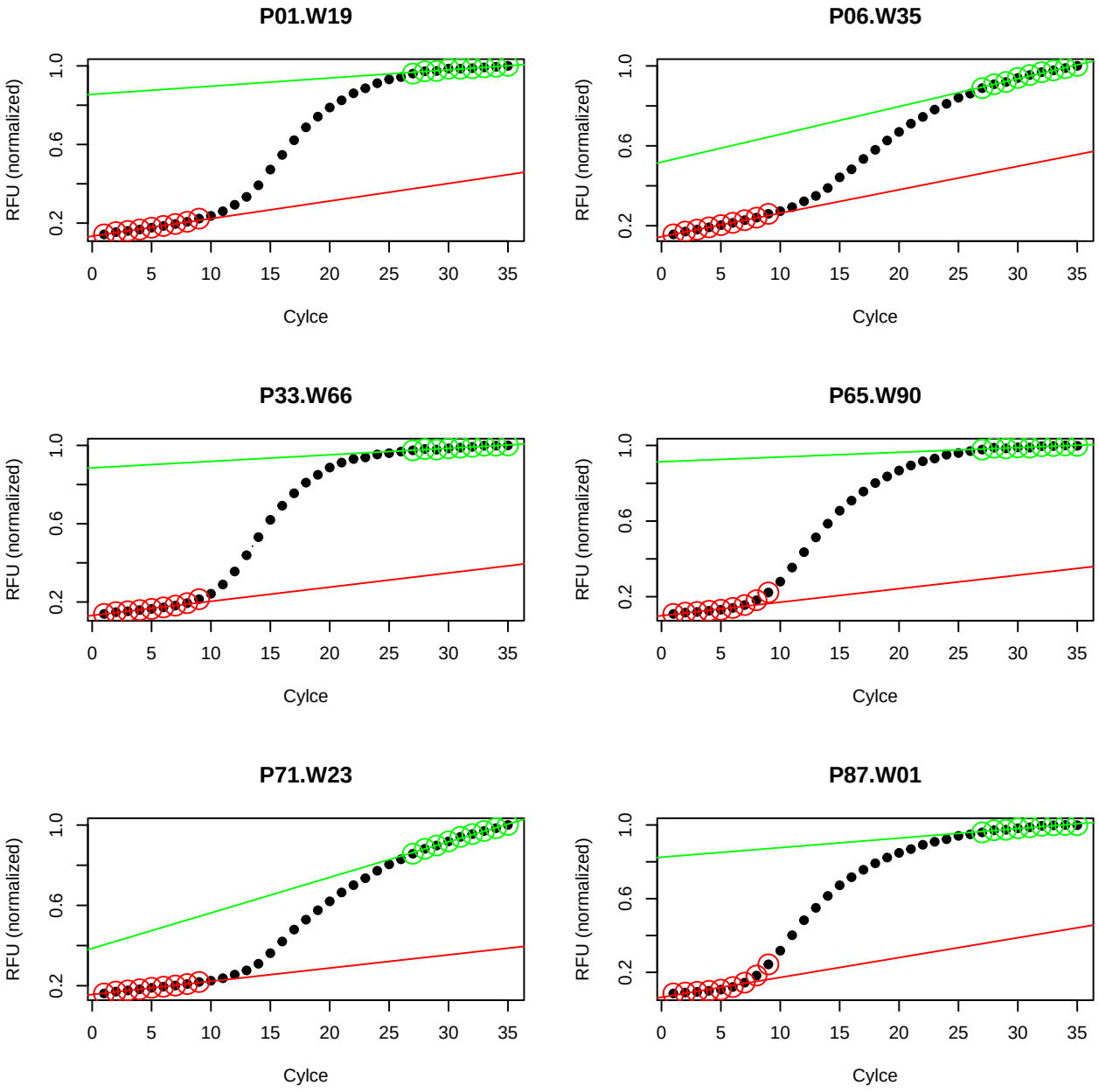

Figure 11: Robust local regression to analyze amplification curves. The amplification curves were arbitrarily selected from the htPCR data set. Not the differences in slopes and intercepts (red and green lines). The mblrr() function is presumably useful for data sets which are accompanied by noise and artifacts.

	nBG	mBG	rBG	nTP	mTP	rTP
P01.W19	0.13	0.01	0.99	0.85	0.00	0.96
P06.W35	0.15	0.01	1.00	0.52	0.01	0.99
P33.W66	0.13	0.01	0.97	0.89	0.00	0.97
P65.W90	0.10	0.01	0.92	0.91	0.00	0.93
P71.W23	0.16	0.01	1.00	0.39	0.02	1.00
P87.W01	0.06	0.01	0.89	0.83	0.01	0.96

Table 2: mblrr text intro. nBG, intercept of head region; mBG, slope of head region; rBG, Pearson correlation of head region; nTP, intercept of tail region; mTP, slope of tail region; rBG, Pearson correlation of tail region

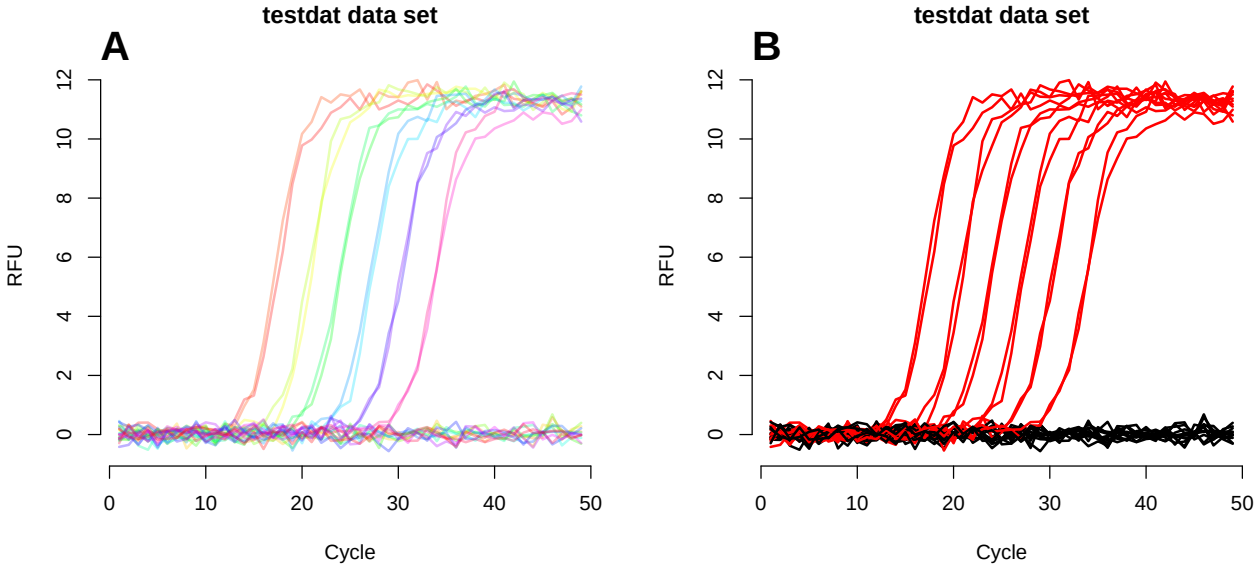

Figure 12: Fast and Frugal Trees.

belong to class of simple decision rules. In many situation, FFTrees make fast decisions based on a few features (1 - 5). In our example we have six feature which are considered for the analysis.

The `FFTrees` package (Phillips et al. 2017) provides an implementation for the **R** statistical computing language. All that is needed for the present example are:

- the data assed by the `mblrr()` function,
- the classification of the amplification curve data by a human rater,
- and a standard formula, which looks like $outcome \leftarrow var1 + var2 + \ldots$ along with the data arguments.

The function `FFTrees()` returns a fast and frugal tree object. This rich object contains the underlying trees and many classification statistics (similar to section 3.8).

In the following example we use the `testdat` data set from the `qpcR` package. The show the usefulness and robustness of the `mblrr()` function we replicated the `testdat` with an additional noise source (normal distribution, mean=0, standard deviation=1).

```
# Load the library xtable for an appealing table output
suppressMessages(library(FFTrees))

# Analyze the testdat data via the mblrr() function

res_mblrr <- do.call(cbind, lapply(2L:ncol(data), function(i) {
    suppressMessages(mblrr(x=data[, 1], y=data[, i],
```

```
                          normalize=TRUE)) %>% data.frame
}))
colnames(res_mblrr) <- colnames(data)[-1]

# Transform the data for a tabular output and assign the results to the object
# output_res_mblrr.

output_res_mblrr <- res_mblrr %>% t

# The output variable names of the mblrr() function are rather long. For better
# readability the variable names were changed to "nBG" (intercept of head region),
# "mBG" (slope of head region), "rBG" (Pearson correlation of head region),
# "nTP" (intercept of tail region), "mTP" (slope of tail region), "rBG" (Pearson
# correlation of tail region)

colnames(output_res_mblrr) <- c("nBG", "mBG", "rBG",
                                "nTP", "mTP", "rTP")

output_res_mblrr <- data.frame(class=human_rater_classification,
                           output_res_mblrr
                          )

output_res_mblrr.fft <- suppressMessages(FFTrees(formula = class ~.,
                                    data = output_res_mblrr[, c(1,2,3,5,6)]))
```

The table 3 output.

```
library(xtable)
print(xtable(output_res_mblrr, caption = "mblrr text. nBG, intercept of
             head region; mBG, slope of head region; rBG, Pearson
             correlation of head region; nTP, intercept of tail region; mTP,
             slope of tail region; rBG, Pearson correlation of tail region",
             label='tablemblrr'), include.rownames = FALSE)
```

% latex table generated in R 3.4.2 by xtable 1.8-2 package % Mon Nov 20 16:52:59 2017

Figure 13 shows the Fast and Frugal Trees by using the features nBG (intercept of head region), mBG (slope of head region), rBG (Pearson correlation of head region), nTP (intercept of tail region), mTP (slope of tail region), and rBG (Pearson correlation of tail region).

3.7 `pcrfit_single()` - A Function to Extract Features from an Amplification Curve

The functionality is provided by packages developed by others and us. The following table gives an overview and some comments on the internal algorithms used by the `pcrfit_single()` function.

Function	Package	Source	Details
`bcp()`	`bcp`	Erdman, Emerson, and others (2007)	Performs a change point analysis based on a Bayesian approach.
`amptester()`	`chipPCR`	Rödiger, Burdukiewicz, and Schierack (2015)	Test with different statistical methods, if the amplification is positive of negative.

Function	Package	Source	Details
bg.max()	chipPCR	Rödiger, Burdukiewicz, and Schierack (2015)	Calculates estimates for the start and end of the background region and the end of the exponential amplification reaction. The estimates of the bg.max function are normalized to the total cycle number.
e.agglo()	ecp	James and Matteson (2013)	Performs a multiple change point analysis based on agglomerative hierarchical estimation.
diffQ()	MBmca	Rödiger, Böhm, and Schimke (2013)	
diffQ2()	MBmca	Rödiger, Böhm, and Schimke (2013)	
mcaPeaks()	MBmca	Rödiger, Böhm, and Schimke (2013)	
head2tailratio()	PCRedux		Calculates an estimate for the head to tail fluorescence, normalized to the slope. See Section 3.4.
earlyreg()	PCRedux		Calculates the slope and intercept of an amplification curve data from a quantitative PCR experiment. See Section 3.3.
hookreg() & hookregNL()	PCRedux		
autocorrelation_test()	PCRedux		Test for an autocorrelation of amplification curve data from a quantitative PCR experiment. See Section 3.1.
mblrr()	PCRedux		See Section 3.6.
polyarea()	pracma	Borchers (2017)	Calculates the area of a polygon defined by the vertices with coordinates of the cycles and fluorescence, based on the Gauss polygon area formula.
efficiency()	qpcR	Ritz and Spiess (2008)	Determine the Cq (first derivative maximum, second derivative maximum) and other parameters.
LRE()	qpcR	Ritz and Spiess (2008)	Calculates the qPCR efficiency by the 'linear regression of efficiency' method.
sliwin()	qpcR	Ritz and Spiess (2008)	Calculates the qPCR efficiency by the 'window-of-linearity' method.
takeoff()	qpcR	Ritz and Spiess (2008)	Calculates the qPCR takeoff point.

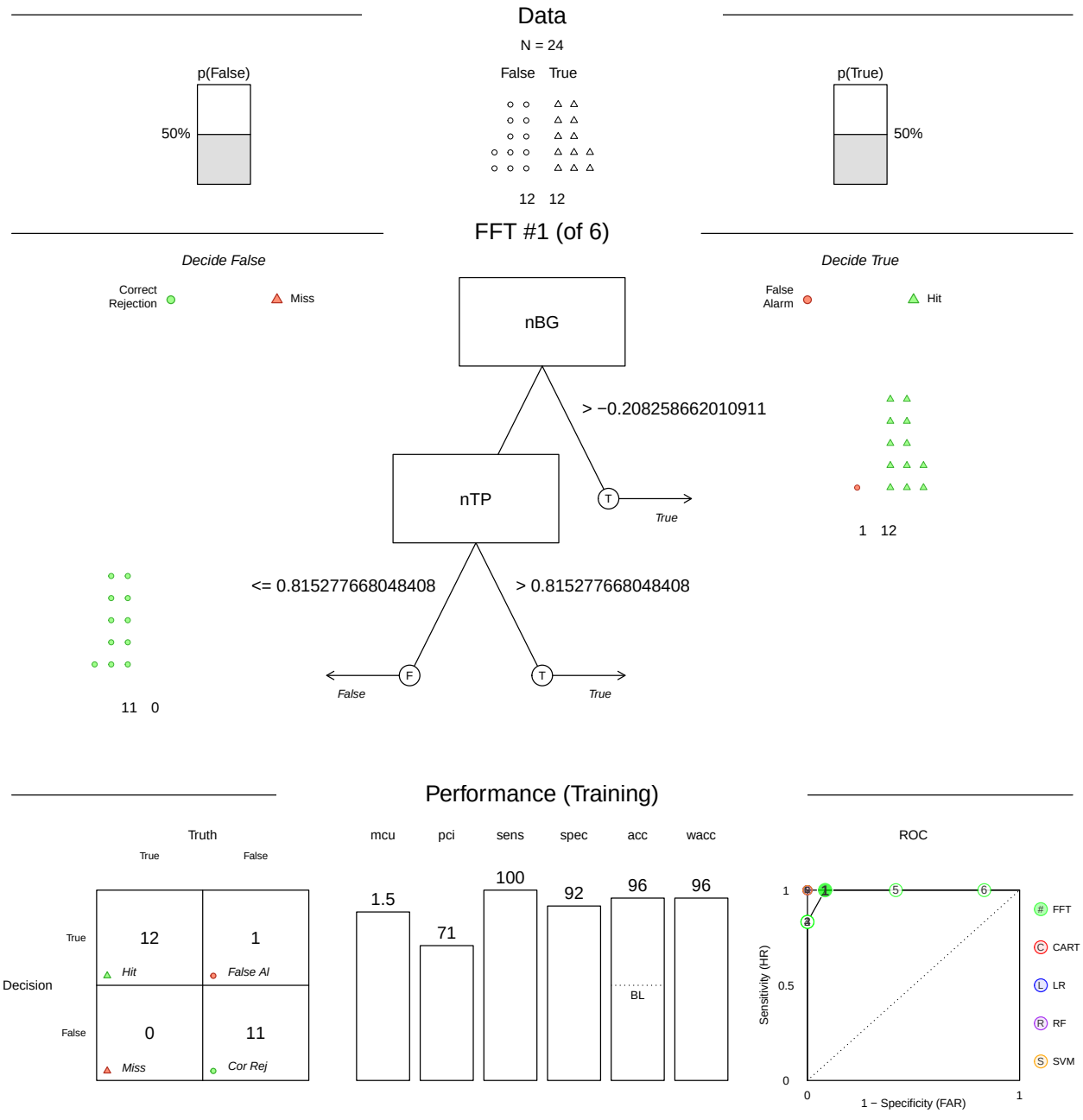

Figure 13: Fast and Frugal Trees. nBG, intercept of head region; mBG, slope of head region; rBG, Pearson correlation of head region; nTP, intercept of tail region; mTP, slope of tail region; rBG, Pearson correlation of tail region.

class	nBG	mBG	rBG	nTP	mTP	rTP
TRUE	0.02	-0.00		0.97	0.00	
TRUE	-0.01	0.00		0.97	-0.00	
FALSE	-0.38	0.00		0.27	0.00	
FALSE	-0.35	0.00		0.70	-0.00	
TRUE	0.00	0.00		0.99	-0.00	
TRUE	-0.00	0.00		1.00	-0.00	
FALSE	-0.37	0.00		0.37	0.00	
FALSE	-0.75	-0.00		0.82	-0.01	
TRUE	-0.01	-0.00		0.95	0.00	
TRUE	-0.01	0.00		0.96	0.00	
FALSE	-0.88	0.01		0.77	-0.00	
FALSE	-0.21	-0.01		0.54	0.00	
TRUE	-0.00	-0.00		0.99	0.00	
TRUE	-0.01	-0.00		0.96	0.00	
FALSE	-0.65	0.00		0.71	0.00	
FALSE	-0.72	-0.00		0.62	-0.00	
TRUE	-0.02	0.00		0.95	0.00	
TRUE	-0.01	0.00		0.90	0.00	
FALSE	-0.44	-0.00		0.74	-0.00	
FALSE	-0.39	-0.00		0.56	0.01	
TRUE	-0.00	-0.00		0.70	0.01	0.76
TRUE	-0.00	-0.00		0.68	0.01	
FALSE	-0.63	0.00		0.72	-0.00	
FALSE	-0.46	-0.02		0.63	-0.00	

Table 3: mblrr text. nBG, intercept of head region; mBG, slope of head region; rBG, Pearson correlation of head region; nTP, intercept of tail region; mTP, slope of tail region; rBG, Pearson correlation of tail region

An example is given for the internal parameter `diffQ2_slope` which is calculated from the slope determined by a linear model of the data points from the minimum and maximum of the second derivative. The coordinates of the minimum and the maximum were determined as described in Rödiger, Böhm, and Schimke (2013). In the following example, the data

```
# Load example data (sample F6.1) from the testdat data set
library(qpcR)
library(magrittr)
# Load MBmca package to calculate the minimum and the maximum of the second
# derivative

library(MBmca)
data <- testdat[, c(1,22)]

# Calculate the minimum and the maximum of the second
# derivative and assign it to the object res_diffQ2

res_diffQ2 <- suppressMessages(diffQ2(data, plot=FALSE, fct=min))

# Build a linear model from der second derivative of res_diffQ2
res_diffQ2_lm <- lm(res_diffQ2[["yTm1.2.D2"]] ~ res_diffQ2[["xTm1.2.D2"]])

par(mfrow=c(1,2))

data %>% plot(., xlab="Cycle", ylab="RFU", main="F6.1 (testdat)", type="l",
```

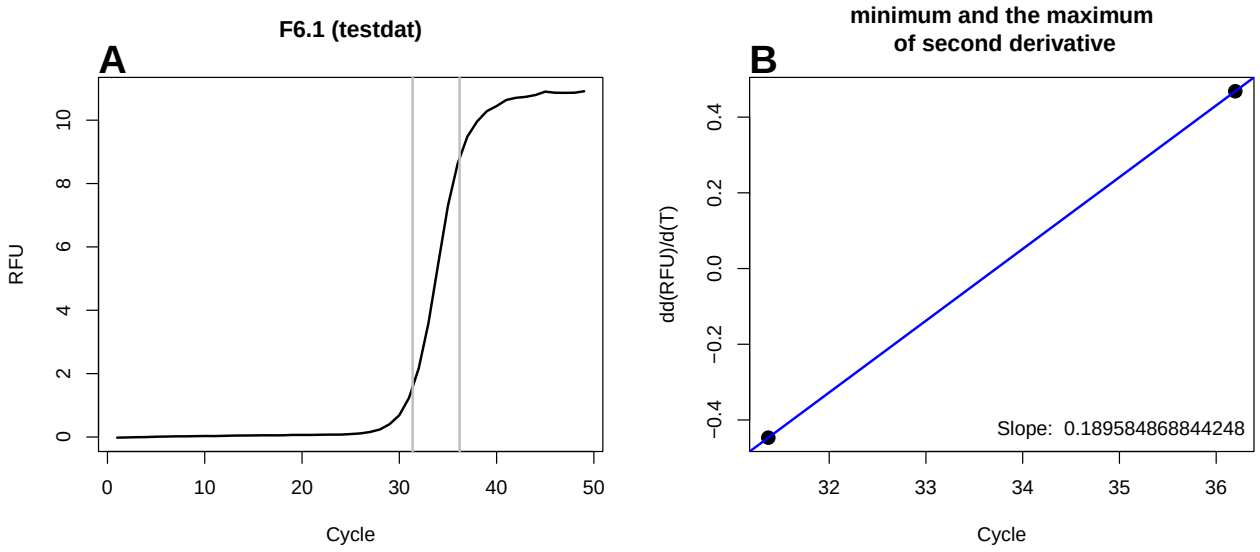

Figure 14: Analysis of the amplification curve via diffQ2.

```
                  lty=1, lwd=2, col="black")
abline(v=res_diffQ2[["xTm1.2.D2"]], col="grey", lwd=2)
mtext("A", cex = 2, side = 3, adj = 0, font = 2)


plot(res_diffQ2[["xTm1.2.D2"]], res_diffQ2[["yTm1.2.D2"]], pch=19, cex=1.5,
     xlab="Cycle", ylab="dd(RFU)/d(T)",
     main="minimum and the maximum\n of second derivative")
abline(res_diffQ2_lm, col="blue", lwd=2)
legend("bottomright", paste("Slope: ", coefficients(res_diffQ2_lm)[2]), bty="n")
mtext("B", cex = 2, side = 3, adj = 0, font = 2)
```

`encu()` (ENcode CUrves) is a relative of the `pcrfit_single()` function. Similarly, this function calculates numerous but with an emphasis on features extraction of large amplification curve data sets. The `pcrfit_single()` function is performing the analysis for a single process and the `pblapply()` function from the `pbapply` package is used internally to is delivers a progress bar and leverages parallel processing. Examples are given in the documentation of the `encu()` function.

`pcrfit_single()` is a powerful function, which calculates numerous features from amplification curve data. This comes at a cost, since several internal functions are computational very intensive (Porzelius, Knaus, and Schwarzer 2009, Schmidberger et al. (2009)). Recent information about parallelization in **R** is available from Eddelbuettel (2017). An example is the `pcrfit()` function from the `qpcR` package. This function fits and optimizes eight non-linear (sigmoid) models to the amplification curve data.

The code block below shows an example for a parallelized version of `pcrfit_single()`. We call this function `pcrfit_parallel()` is presumably beneficial for users who wish to calculate the features of a large amplification curve data set. We found that this function works on Linux systems. On Windows systems we received selectively error messages. `pcrfit_parallel()` makes use of parallelized code to make use of multi-core architectures. In this function we import from the `parallel` package the `detectCores()` function. This function determines the number of available cores. Function from the `foreach` package (e.g., `%dopar%`, `foreach()`) are used for the further organization of the CPU usage. The `pcrfit_single()` performs the analysis for a single process.

- The parameter `data` is the data set containing the cycles and fluorescence amplitudes.

- The parameter `n_cores` defines the numbers of cores that should be left unused by this function.

By default `pcrfit_parallel()` is using only one core (`n_cores = 1`). `n_cores = "all"` uses all available cores. The output of the `pcrfit_parallel()` function is similar to the `pcrfit_single()` function.

```
# Copy and paste the code to an R console to evaluate it

library(parallel)
library(doParallel)

pcrfit_parallel <- function(data, n_cores = 1) {
    # Determine the number of available cores and register them
    if(n_cores == "all")
        n_cores <- detectCores()

        registerDoParallel(n_cores)

        # Prepare the data for further processing
        # Normalize RFU values to the alpha percentile (0.999)
        cycles <- data.frame(cycles=data[, 1])
        data_RFU <- data.frame(data[, -1])
        data_RFU_colnames <- colnames(data_RFU)
        data_RFU <- sapply(1L:ncol(data_RFU), function(i) {
            data_RFU[, i] / quantile(data_RFU[, i], 0.999, na.rm=TRUE)
        })
        colnames(data_RFU) <- data_RFU_colnames

        # just to shut RCHeck for NSE we define ith_cycle
        ith_cycle <- 1

        run_res <- foreach::foreach(ith_cycle = 1L:ncol(data_RFU),
                            .packages=c("bcp", "changepoint", "chipPCR", "ecp", "MBmca",
                                        "PCRedux", "pracma", "qpcR", "robustbase",
                                        "zoo"),
                            .combine = rbind) %dopar% {
                                suppressMessages(pcrfit_single(data_RFU[, ith_cycle]))
                            }


                            res <- cbind(runs = colnames(data_RFU), run_res)

                            rownames(res) <- NULL

                            res

}

# Calculate curve features of an amplification curve data. Note that not all
# available CPU cores are used. If need set "all" to use all available cores.
# In this example the testdat data set from the qpcR package is used.
# The samples F1.1 and F1.2 are positive amplification curves. The samples
# F1.3 and F1.4 are negative.

library(qpcR)
res_pcrfit_parallel <- pcrfit_parallel(testdat[, 1:5])
```

```
res_pcrfit_parallel
```

3.8 `performeR()` - Performance Analysis for Binary Classification

Statistical modeling and machine learning can be powerful but expose the user to the risk of introducing unexpected biases. This may lead to an overestimation of the performance. The assessment of the performance by the sensitivity (percentage of true decisions that are identified) and specificity (percentage of negative decision that are correctly identified) is important to characterize the performance of a classifier or screening test (James et al. 2013).

Abbreviations: TP, true positive; FP, false positive; TN, true negative; FN, false negative

Measure	Formula
Sensitivity - TPR, true positive rate	$TPR = \frac{TP}{TP+FN}$
Specificity - SPC, true negative rate	$SPC = \frac{TN}{TN+FP}$
Precision - PPV, positive predictive value	$PPV = \frac{TP}{TP+FP}$
Negative predictive value - NPV	$NPV = \frac{TN}{TN+FN}$
Fall-out, FPR, false positive rate	$FPR = \frac{FP}{FP+TN} = 1 - SPC$
False negative rate - FNR	$FNR = \frac{FN}{TN+FN} = 1 - TPR$
False discovery rate - FDR	$FDR = \frac{FP}{TP+FP} = 1 - PPV$
Accuracy - ACC	$ACC = \frac{(TP+TN)}{(TP+FP+FN+TN)}$
F1 score - F1	$F1 = \frac{2TP}{(2TP+FP+FN)}$
Matthews correlation coefficient - MCC	$MCC = \frac{(TP*TN-FP*FN)}{\sqrt{(TP+FP)*(TP+FN)*(TN+FP)*(TN+FN)}}$
Likelihood ratio positive - LRp	$LRp = \frac{TPR}{1-SPC}$
Cohen"s kappa (binary classification)	$\kappa = \frac{p_0-p_c}{1-p_0}$

3.9 `qPCR2fdata()` - A Helper Function to Convert Amplification Curve Data to the `fdata` Format

`qPCR2fdata()` is a helper function to convert qPCR data to the functional `fdata` class as introduced by Febrero-Bande and Oviedo de la Fuente (2012). This function prepares the data for further analysis, which includes utilities for functional data analysis. For example, it this can be used to determine the similarity measures between amplification curves shapes by the Hausdorff distance (Charpiat, Faugeras, and Keriven 2003).

The `qPCR2fdata()` function takes a data set containing the amplification cycles (first column) and the fluorescence amplitudes (subsequent columns) as input. Since noise and missing values may affect the analysis adversely we integrated an instance of `CPP()` function from the `chipPCR` package (Rödiger, Burdukiewicz, and Schierack 2015) in `qPCR2fdata()`.

The following example illustrates the usage for the `testdat` data set.

```
# Calculate slope and intercept on noise (negative) amplification curve data.
# Load additional packages for data and pipes.
library(qpcR)
library(chipPCR)
library(fda.usc)
library(magrittr)

# Convert the qPCR data set to the fdata format
```

```
# Use unprocessed data from the testdat data set
res_fdata <- qPCR2fdata(testdat)

# Use preprocessed data (preprocess=TRUE) from the testdat data set
res_fdata_preprocessed <- qPCR2fdata(testdat, preprocess=TRUE)

# Extract column names and create rainbow color to label the data
res_fdata_colnames <- testdat[-1] %>% colnames()
data_colors <- rainbow(length(res_fdata_colnames), alpha=0.5)
```

```
# Plot the converted qPCR data
par(mfrow=c(2,2))
res_fdata %>% plot(., xlab="Cycle", ylab="RFU", main="testdat", type="l",
                   lty=1, lwd=2, col=data_colors)
legend("topleft", as.character(res_fdata_colnames), pch=19,
       col=data_colors, bty="n", ncol=2)
mtext("A", cex = 2, side = 3, adj = 0, font = 2)
# Calculate the Hausdorff distance (fda.usc) package and plot the distances
# as clustered data.

res_fdata_hclust <- metric.hausdorff(res_fdata)
plot(hclust(as.dist(res_fdata_hclust)), main="Clusters of the amplification\n
curves as calculated by the Hausdorff distance")
mtext("B", cex = 2, side = 3, adj = 0, font = 2)
rect(0.5,-3,12.25,0.5, border = "red")
text(7, 1, "negative", col="red")
rect(12.5,-3,24.5,0.5, border = "green")
text(14, 1, "positive", col="green")

# Repeat the plot and the cluster analysis for the preprocessed data
res_fdata_preprocessed %>% plot(., xlab="Cycle", ylab="RFU", main="testdat", type="l",
                   lty=1, lwd=2, col=data_colors)
legend("topleft", as.character(res_fdata_colnames), pch=19,
       col=data_colors, bty="n", ncol=2)
mtext("C", cex = 2, side = 3, adj = 0, font = 2)
# Calculate the Hausdorff distance (fda.usc) package and plot the distances
# as clustered data from preprocessed amplification curves.

res_fdata_hclust_preprocessed <- metric.hausdorff(res_fdata_preprocessed)
plot(hclust(as.dist(res_fdata_hclust_preprocessed)),
     main="Clusters of the preprocessed amplification\n
curves as calculated by the Hausdorff distance")
rect(0.5,-3,12.25,0.5, border = "red")
text(7, 1, "negative", col="red")
rect(12.5,-3,24.5,0.5, border = "green")
text(14, 1, "positive", col="green")
mtext("D", cex = 2, side = 3, adj = 0, font = 2)
```

The following example illustrates the usage for the `HCU32_aggR.csv` data set from the 32 channel VideoScan heating and cooling unit. In this experiment the bacterial gene *aggR* from *E. coli* was amplified in 32 identical reactions. The ambition was to test if the 32 amplification curves of the qPCR reaction are identical. As before, the data were processed with the `qPCR2fdata()` function and compared by the the Hausdorff distance. Ideally, the amplification curves form only few clusters.

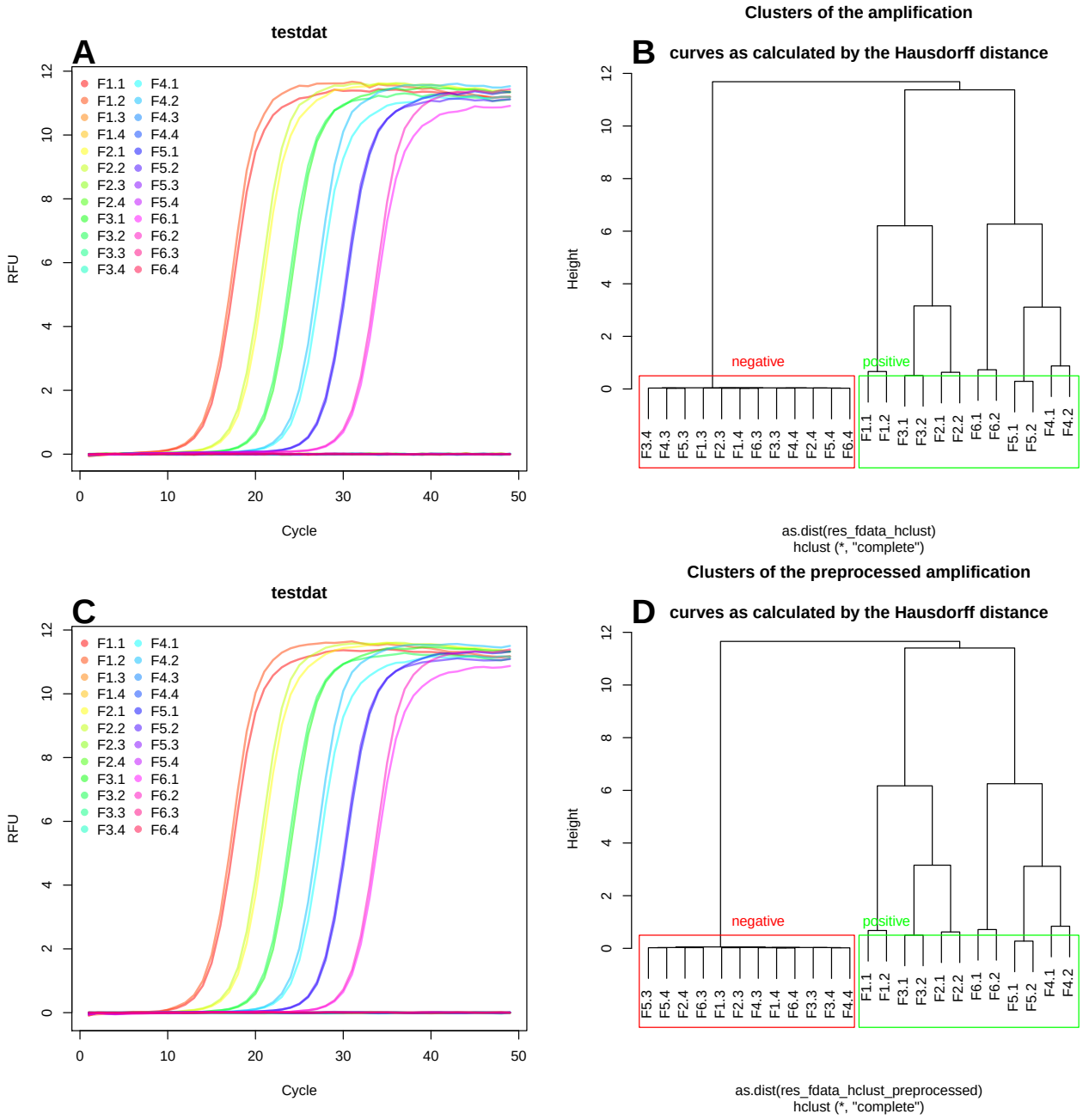

Figure 15: Clustering of amplification curves via Hausdorff distance. The amplification curves were preprocessed with the qPCR2fdata() function and subsequent processed by a cluster analysis and Hausdorff distance analysis.

```
# Calculate slope and intercept on positive amplification curve data from the
# VideoScan 32 cavity real-time PCR device.
# Load additional packages for data and pipes.
suppressMessages(library(data.table))
library(fda.usc)
library(magrittr)

# Load the qPCR data from the HCU32_aggR.csv data set

filename <- system.file("HCU32_aggR.csv", package="PCRedux")
data_32HCU <- fread(filename) %>% as.data.frame()

# Convert the qPCR data set to the fdata format
res_fdata <- qPCR2fdata(data_32HCU)

# Extract column names and create rainbow color to label the data
res_fdata_colnames <- data_32HCU[-1] %>% colnames()
data_colors <- rainbow(length(res_fdata_colnames), alpha=0.55)
```

In advance the the Cq values were calculated by the following code:

```
# Load the qpcR package to calculate the Cq values by the second derivative
# maximum method.

suppressMessages(library(qpcR))

res_Cq <- sapply(2L:ncol(data_32HCU), function(i){
    efficiency(pcrfit(data_32HCU, cyc=1, fluo=i, model=l6))
})

data.frame(sample=colnames(data_32HCU)[-1], Cq=unlist(res_Cq["cpD2", ]), eff=unlist(res_Cq["eff", ]))

#        Result
#
# sample    Cq      eff
# 1      A1 14.89 1.092963
# 2      B1 15.68 1.110480
# 3      C1 15.63 1.111474
# 4      D1 15.50 1.103089
# 5      E1 15.54 1.100122
# 6      F1 15.37 1.091367
# 7      G1 15.78 1.118713
# 8      H1 15.24 1.080062
# 9      A2 15.94 1.095004
# 10     B2 15.88 1.107878
# 11     C2 15.91 1.112694
# 12     D2 15.77 1.106286
# 13     E2 15.78 1.108201
# 14     F2 15.74 1.110697
# 15     G2 15.84 1.110749
# 16     H2 15.78 1.107229
# 17     A3 15.64 1.107543
# 18     B3 15.61 1.100984
# 19     C3 15.66 1.110703
```

```
# 20     D3 15.63 1.115996
# 21     E3 15.77 1.113885
# 22     F3 15.71 1.113985
# 23     G3 15.70 1.094108
# 24     H3 15.79 1.124223
# 25     A4 15.80 1.119774
# 26     B4 15.72 1.112124
# 27     C4 15.70 1.121453
# 28     D4 15.82 1.121809
# 29     E4 15.62 1.108028
# 30     F4 15.71 1.109634
# 31     G4 15.70 1.110373
# 32     H4 15.73 1.117827
```

Next follows the plotting of all data and the cluster analysis based on the Hausdorff distance (Figure 16). Note that the raw data were not altered to preserve the curvature. As shown in Figure 16A there are some amplification curves, which have jumps in the amplification curve signal. A comparison of the signal (plateau phase) to noise (background phase) shows that the samples `E1`, `F1` and `H1` have the lowest signals and lowest signal differences (Figure 16B). The comparison of the Cq and amplification efficiencies showed that most curves cluster in a similar range. Some samples had a larger deviation from median Cq (Figure 16C). The analysis by clustering the the Hausdorff distance showed no specific pattern for the amplification curves (Figure 16D).

```
library(fda.usc)
library(magrittr)

calculated_Cqs <- c(14.89, 15.68, 15.63, 15.5, 15.54, 15.37, 15.78, 15.24, 15.94,
                    15.88, 15.91, 15.77, 15.78, 15.74, 15.84, 15.78, 15.64, 15.61,
                    15.66, 15.63, 15.77, 15.71, 15.7, 15.79, 15.8, 15.72, 15.7, 15.82,
                    15.62, 15.71, 15.7, 15.73)

calculated_effs <- c(1.09296326515231, 1.11047987547324, 1.11147389307153, 1.10308929700635,
                     1.10012176315852, 1.09136717687619, 1.11871308210321, 1.08006168654712,
                     1.09500422011318, 1.1078777171126, 1.11269436700649, 1.10628580163733,
                     1.1082009954558, 1.11069683827291, 1.11074914659374, 1.10722949813473,
                     1.10754282514113, 1.10098387264025, 1.1107026749644, 1.11599641663658,
                     1.11388510347017, 1.11398547396991, 1.09410798249025, 1.12422338092929,
                     1.11977386646464, 1.11212436173214, 1.12145338871426, 1.12180879952503,
                     1.1080276005651, 1.10963449004393, 1.11037302758388, 1.11782689816295)

# Plot the converted qPCR data
layout(matrix(c(1,2,3,4,4,4), 2, 3, byrow = TRUE))
res_fdata %>% plot(., xlab="Cycle", ylab="RFU", main="HCU32_aggR", type="l",
                   lty=1, lwd=2, col=data_colors)
legend("topleft", as.character(res_fdata_colnames), pch=19,
       col=data_colors, bty="n", ncol=4)
mtext("A", cex = 2, side = 3, adj = 0, font = 2)

# Plot the background and plateau phase.

boxplot(data_32HCU[,-1] - apply(data_32HCU[, -1], 2, min),
        col=data_colors, las=2, main="Signal to noise ratio",
        xlab="Sample", ylab="RFU")
mtext("B", cex = 2, side = 3, adj = 0, font = 2)
```

```
# Plot the Cqs and the amplification efficiencies.
# Determine the median of the Cq values and label all Cqs, which a less 0.1 Cqs
# of the median or more then 0.1 Cqs of the median Cq.

plot(calculated_Cqs, calculated_effs, xlab="Cq (SDM)",
     ylab="eff", main="Cq vs. Amplification Efficiency",
     type="p", pch=19, lty=1, lwd=2, col=data_colors)

median_Cq <- median(calculated_Cqs)
abline(v=median_Cq)

text(median_Cq + 0.01, 1.085, expression(paste(tilde(x))))
labeled <- c(which(calculated_Cqs < median_Cq - 0.1),
             which(calculated_Cqs > median_Cq + 0.1))

text(calculated_Cqs[labeled], calculated_effs[labeled], as.character(res_fdata_colnames)[labeled])
mtext("C", cex = 2, side = 3, adj = 0, font = 2)

# Calculate the Hausdorff distance using the fda.usc package and cluster the
# the distances.

res_fdata_hclust <- metric.hausdorff(res_fdata)
cluster <- hclust(as.dist(res_fdata_hclust))

# plot the distances as clustered data and label the leafs with the Cq values and
# colored dots.

plot(cluster, main="Clusters of the amplification\n
curves as calculated by the Hausdorff distance")
mtext("D", cex = 2, side = 3, adj = 0, font = 2)
```

The analysis gives an overview of the variation of the amplification curve data.

3.10 `visdat_pcrfit()` - A Function to Visualize the Content of Data From an Analysis with the `pcrfit_single()` Function

The `visdat_pcrfit()` function makes use of the `vis_dat` function from the `visdat` package by Tierney (2017). The samples "A01", "A06" and "A10" from the `C126EG685` of the `chipPCR` package were analyzed.

```
# Calculate curve features of an amplification curve data set.
# Use the C126EG685 data set from the chipPCR package and analyze the samples
# A01, A02, A04 and B05.

library(chipPCR)

res_1 <- cbind(runs="A01", pcrfit_single(C126EG685[, 2]))
```

```
## Loading required package: bcp
```

```
## Loading required package: grid
```

```
res_2 <- cbind(runs="A02", pcrfit_single(C126EG685[, 3]))
res_3 <- cbind(runs="A04", pcrfit_single(C126EG685[, 5]))
res_4 <- cbind(runs="B04", pcrfit_single(C126EG685[, 17]))
```

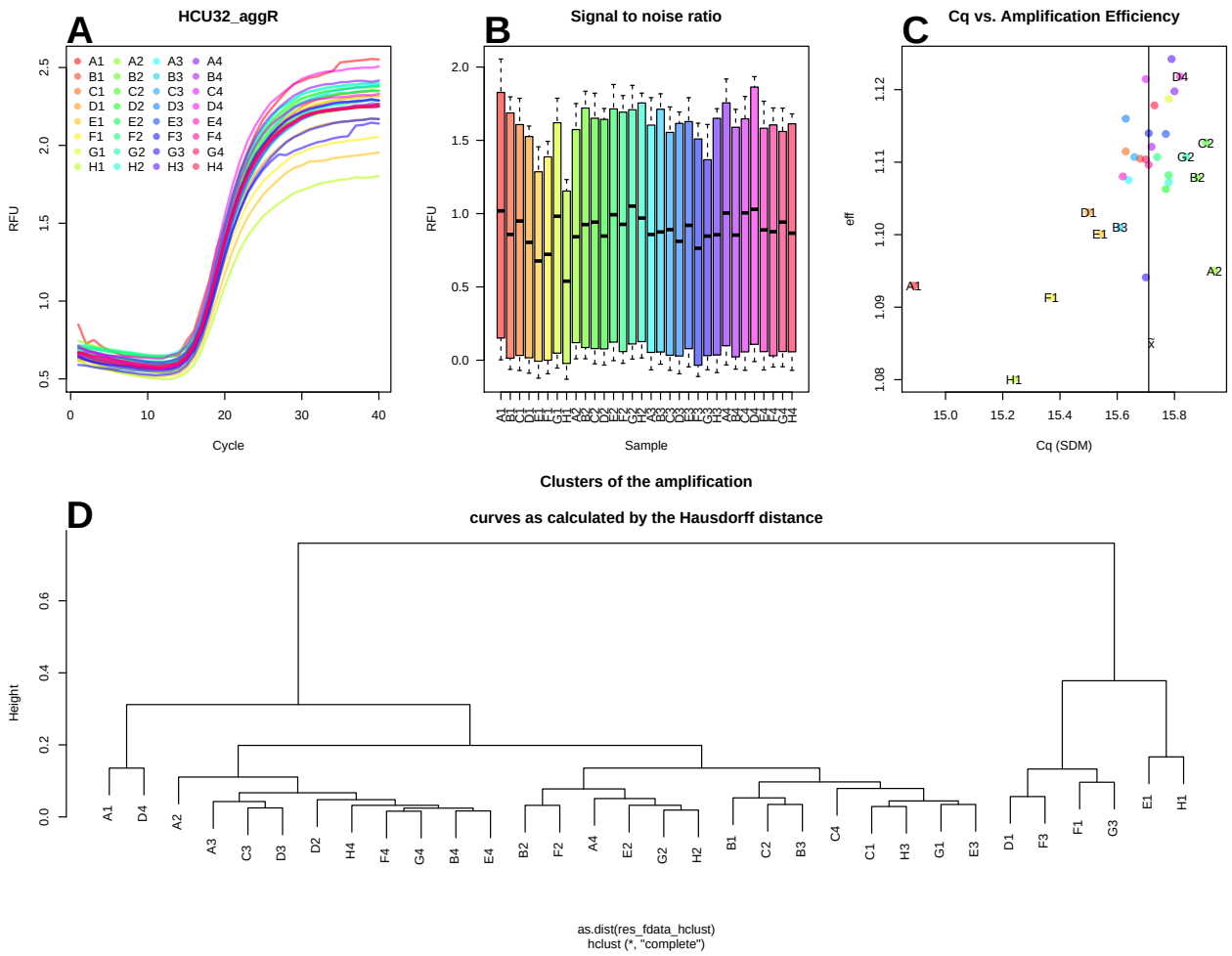

Figure 16: Clustering of amplification curves. The amplification curves from the 32HCU were processed with the `qPCR2fdata()` function and subsequent processed by a cluster analysis and Hausdorff distance analysis. A) Amplification curves were plotted from the raw data. B) Overall, the signal to noise ratios of the amplification curves were comparable between all cavities. C) The Cqs (Second Derivative Maximum) and the amplification efficiency (eff) were calculated with the `efficiency(pcrfit())` functions from the `qpcR` package. The median Cq is indicated as vertical line. Cqs larger or less than 0.1 of the Cq $\tilde{x}$ are indicated with the labels of the corresponding sample. D) The clusters according to the Hausdorff distance show no specific pattern regarding the amplification curve signals. It appears that the samples D1, E1, F1, F3, G3 and H1 deviate most from the other amplification curves.

```
res <- rbind(A01=res_1, A02=res_2, A04=res_3, B04=res_4)
```

Finally, the data were visualized with the `visdat_pcrfit()` function. In this example the static plot is shown (Figure 17). It is also possible to run the function interactively by setting the parameter "interactive = TRUE". In this case starts a `ggplotly` instance.

```
# Show all results in a plot. Not that the interactive parameter is set to
# FALSE.

visdat_pcrfit(res, type="all", interactive=FALSE)
```

4 Data Sets

4.1 Classified Data Sets

For the supervised learning and methods validation it was important to asses the amplification curves individually. In Rödiger, Burdukiewicz, and Schierack (2015) the function *humanrater* was introduced. *humanrater* is a graphical interface to rate curves (e.g., amplification curves, melting curves). This tool was be used to prepare data sets for data mining and machine learning purposes. The user has to define characteristics (e.g., negative ("n"), ambiguous ("a"), positive ("p")) which are used to rate in an interactive, semi-blinded session to the presented data. The `PCRedux` package contains data set of amplification curves, which were classified for data sets from the `chipPCR`, `qpcR` and `PCRedux` packages. The following table gives an overview about the available data sets.

Decision Data Sets in PCRedux	qPCR Data Set	Package
decision_res_2017_08_07___RAS002.csv	2017_08_07___RAS002.rdml	[PCRedux]
decision_res_2017_08_11___RAS003.csv	2017_08_11___RAS003.rdml	[PCRedux]
decision_res_batsch1.csv	batsch1	[qpcR]
decision_res_batsch2.csv	batsch2	[qpcR]
decision_res_batsch3.csv	batsch3	[qpcR]
decision_res_batsch4.csv	batsch4	[qpcR]
decision_res_batsch5.csv	batsch5	[qpcR]
decision_res_boggy.csv	boggy	[qpcR]
decision_res_C126EG595.csv	C126EG595	[chipPCR]
decision_res_C127EGHP.csv	C127EGHP	[chipPCR]
decision_res_C316.amp.csv	C316.amp	[chipPCR]
decision_res_C317.amp.csv	C317.amp	[chipPCR]
decision_res_C60.amp.csv	C60.amp	[chipPCR]
decision_res_CD74.csv	CD74	[chipPCR]
decision_res_competimer.csv	competimer	[qpcR]
decision_res_dil4reps94.csv	dil4reps94	[qpcR]
decision_res_guescini1.csv	guescini1	[qpcR]
decision_res_guescini2.csv	guescini2	[qpcR]
decision_res_htPCR.csv	htPCR	[qpcR]
decision_res_karlen1.csv	karlen1	[qpcR]
decision_res_karlen2.csv	karlen2	[qpcR]
decision_res_karlen3.csv	karlen3	[qpcR]
decision_res_lievens1.csv	lievens1	[qpcR]
decision_res_lievens2.csv	lievens2	[qpcR]
decision_res_lievens3.csv	lievens3	[qpcR]

Decision Data Sets in PCRedux	qPCR Data Set	Package
decision_res_reps.csv	reps	[qpcR]
decision_res_reps2.csv	reps2	[qpcR]
decision_res_reps3.csv	reps3	[qpcR]
decision_res_reps384.csv	reps384	[qpcR]
decision_res_rutledge.csv	rutledge	[qpcR]
decision_res_stepone_std.csv	stepone_std	[qpcR]
decision_res_testdat.csv	testdat	[qpcR]
decision_res_vermeulen1.csv	vermeulen1	[qpcR]
decision_res_vermeulen2.csv	vermeulen2	[qpcR]
decision_res_VIMCFX96_60.csv	VIMCFX96_60	[chipPCR]
decision_res_HCU32_aggR.csv	HCU32_aggR.csv	[PCRedux]

4.2 Amplification Curve Data in the `RDML` Format

Selected amplification cure data sets were stored in the RDML format as described in (Rödiger et al. 2015, Rödiger et al. (2017)).

RDML data file	Device	Target gene	Detection chemistry
2017_08_07___RAS002.rdml	CFX96	HPV16, HPV18, HPRT1	Taqman
2017_08_11___RAS003.rdml	CFX96	HPV16, HPV18, HPRT1	Taqman
hookreg.rdml	Bio-Rad	various	Taqman, intercalating dyes
HCU32_aggR.rdml	32HCU	various	intercalating dye (EvaGreen)

32HCU: VideoScan (Attomol GmbH), CFX96: Bio-Rad.

Table_human_rated.xlsx

5 Summary and Conclusions

The `PCRedux` enables the user to extract features from amplification curve data. We implemented numerous features that can be extracted from the amplification curve data. Some of them have not been described in the literature. We consider `PCRedux` as enabling technology for further research. For example, the proposed features can be used for machine learning applications or quality assessment of data.

The `pcrfit_single()` function is a wrapper that can be extended if new features emerge.

Of note, we would like to emphasis that the functionality of this package is not limited to amplification curve data from qPCR experiments. As stated before, amplification curves have a sigmoid curve. So do many other processes.

References

Borchers, Hans Werner. 2017. *pracma: Practical Numerical Math Functions*. https://CRAN.R-project.org/package=pracma.

Bustin, Stephen. 2017. "The Continuing Problem of Poor Transparency of Reporting and Use of Inappropriate Methods for RT-qPCR." *Biomolecular Detection and Quantification* 12 (June): 7–9.

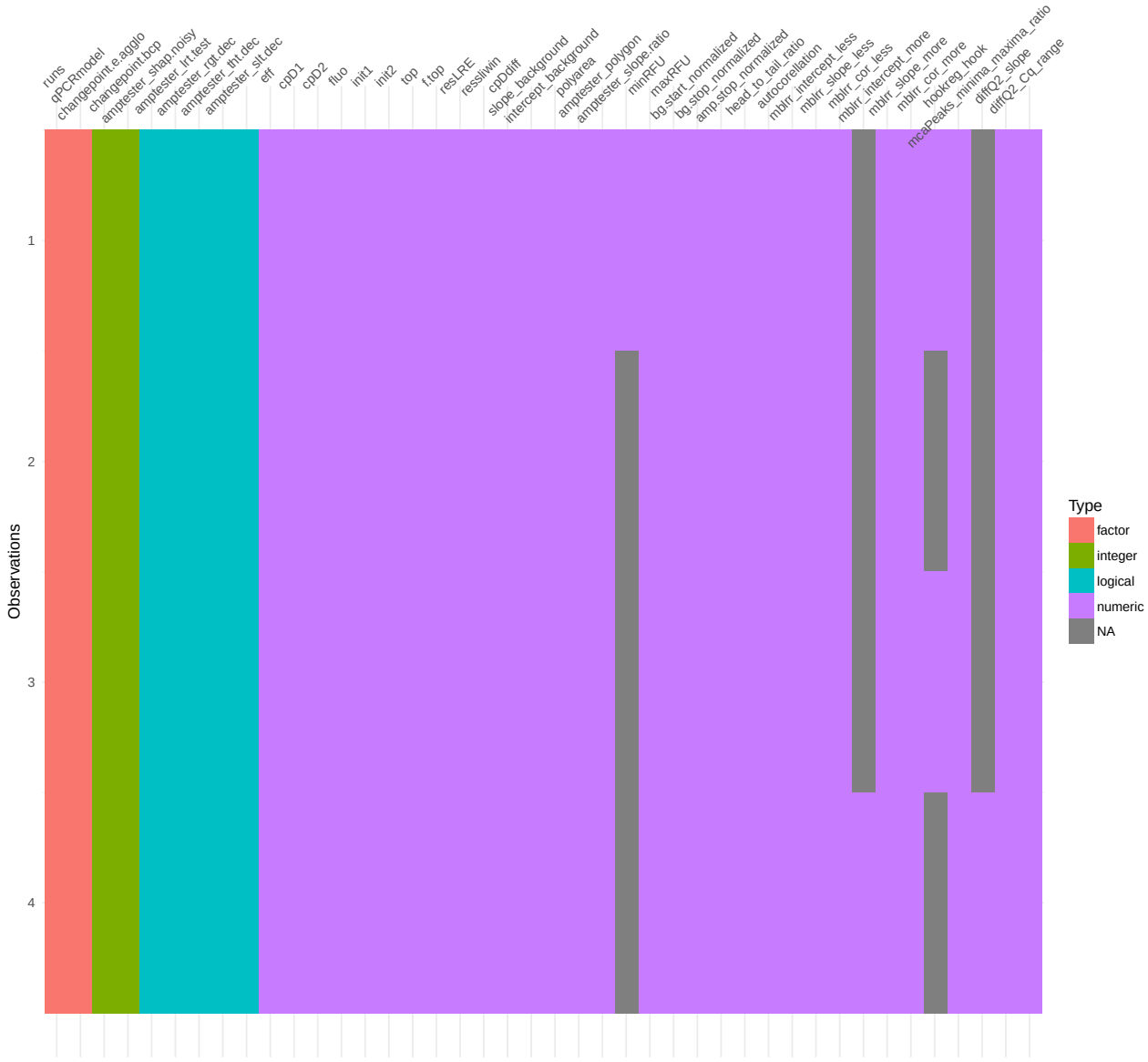

Figure 17: Application of visdat_pcrfit() for the visualization of the data structure after an analysis by pcrfit_single().

doi:10.1016/j.bdq.2017.05.001.

Bustin, Stephen A. 2014. "The Reproducibility of Biomedical Research: Sleepers Awake!" *Biomolecular Detection and Quantification* 2 (December): 35–42. doi:10.1016/j.bdq.2015.01.002.

Charpiat, Guillaume, Olivier Faugeras, and Renaud Keriven. 2003. "Shape Metrics, Warping and Statistics." In *Image Processing, 2003. ICIP 2003. Proceedings. 2003 International Conference on*, 2:II–627. IEEE. http://ieeexplore.ieee.org/abstract/document/1246758/.

Dvinge, Heidi, and Paul Bertone. 2009. "HTqPCR: High-Throughput Analysis and Visualization of Quantitative Real-Time PCR Data in R." *Bioinformatics* 25 (24): 3325–6. doi:10.1093/bioinformatics/btp578.

Eddelbuettel, Dirk. 2017. "CRAN Task View: High-Performance and Parallel Computing with R," September. https://CRAN.R-project.org/view=HighPerformanceComputing.

Erdman, Chandra, John W. Emerson, and others. 2007. "bcp: An R Package for Performing a Bayesian Analysis of Change Point Problems." *Journal of Statistical Software* 23 (3): 1–13. https://www.jstatsoft.org/article/view/v023i03/v23i03.pdf.

Febrero-Bande, Manuel, and Manuel Oviedo de la Fuente. 2012. "Statistical Computing in Functional Data Analysis: The R Package fda.usc." *Journal of Statistical Software* 51 (4): 1–28. http://www.jstatsoft.org/v51/i04/.

Feuer, Ronny, Sebastian Vlaic, Janine Arlt, Oliver Sawodny, Uta Dahmen, Ulrich M. Zanger, and Maria Thomas. 2015. "LEMming: A Linear Error Model to Normalize Parallel Quantitative Real-Time PCR (qPCR) Data as an Alternative to Reference Gene Based Methods." *PLOS ONE* 10 (9): e0135852. doi:10.1371/journal.pone.0135852.

Gunay, Melih, Evgin Goceri, and Rajarajeswari Balasubramaniyan. 2016. "Machine Learning for Optimum CT-Prediction for qPCR." In *Machine Learning and Applications (ICMLA), 2016 15th IEEE International Conference on Machine Learning and Applications (Icmla)*, 588–92. IEEE. doi:10.1109/ICMLA.2016.0103.

Huggett, Jim, Justin O'Grady, and Stephen Bustin. 2014. "How to Make Mathematics Biology's Next and Better Microscope." *Biomolecular Detection and Quantification* 1 (1): A1–A3. doi:10.1016/j.bdq.2014.09.001.

James, Gareth, Daniela Witten, Trevor Hastie, and Robert Tibshirani. 2013. *An Introduction to Statistical Learning*. Vol. 103. Springer Texts in Statistics. New York, NY: Springer New York. http://link.springer.com/10.1007/978-1-4614-7138-7.

James, Nicholas A., and David S. Matteson. 2013. "ecp: An R package for nonparametric multiple change point analysis of multivariate data." *arXiv Preprint arXiv:1309.3295*. https://arxiv.org/abs/1309.3295.

Lefever, Steve, Jan Hellemans, Filip Pattyn, Daniel R. Przybylski, Chris Taylor, René Geurts, Andreas Untergasser, Jo Vandesompele, and on behalf of the RDML Consortium. 2009. "RDML: Structured Language and Reporting Guidelines for Real-Time Quantitative PCR Data." *Nucleic Acids Research* 37 (7): 2065–9. doi:10.1093/nar/gkp056.

Luan, Shenghua, Lael J. Schooler, and Gerd Gigerenzer. 2011. "A signal-detection analysis of fast-and-frugal trees." *Psychological Review* 118 (2): 316–38. doi:10.1037/a0022684.

Mallona, Izaskun, Anna Díez-Villanueva, Berta Martín, and Miguel A. Peinado. 2017. "Chainy: An Universal Tool for Standardized Relative Quantification in Real-Time PCR." *Bioinformatics*. doi:10.1093/bioinformatics/btw839.

Mallona, Izaskun, Julia Weiss, and Marcos Egea-Cortines. 2011. "pcrEfficiency: A Web Tool for PCR Amplification Efficiency Prediction." *BMC Bioinformatics* 12: 404. doi:10.1186/1471-2105-12-404.

Martins, C., G. Lima, Mr. Carvalho, L. Cainé, and Mj. Porto. 2015. "DNA quantification by real-time PCR in different forensic samples." *Forensic Science International: Genetics Supplement Series* 5 (December): e545–e546. doi:10.1016/j.fsigss.2015.09.215.

McCall, Matthew N., Helene R. McMurray, Hartmut Land, and Anthony Almudevar. 2014. "On Non-Detects

in qPCR Data." *Bioinformatics* 30 (16): 2310–6. doi:10.1093/bioinformatics/btu239.

Neve, Jan De, Joris Meys, Jean-Pierre Ottoy, Lieven Clement, and Olivier Thas. 2014. "unifiedWMWqPCR: The Unified Wilcoxon–Mann–Whitney Test for Analyzing RT-qPCR Data in R." *Bioinformatics* 30 (17): 2494–5. doi:10.1093/bioinformatics/btu313.

Pabinger, Stephan, Stefan Rödiger, Albert Kriegner, Klemens Vierlinger, and Andreas Weinhäusel. 2014. "A Survey of Tools for the Analysis of Quantitative PCR (qPCR) Data." *Biomolecular Detection and Quantification* 1 (1): 23–33. doi:10.1016/j.bdq.2014.08.002.

Pabinger, Stephan, Gerhard G. Thallinger, René Snajder, Heiko Eichhorn, Robert Rader, and Zlatko Trajanoski. 2009. "QPCR: Application for Real-Time PCR Data Management and Analysis." *BMC Bioinformatics* 10 (1): 268. doi:10.1186/1471-2105-10-268.

Perkins, James R., John M. Dawes, Steve B. McMahon, David LH Bennett, Christine Orengo, and Matthias Kohl. 2012. "ReadqPCR and NormqPCR: R Packages for the Reading, Quality Checking and Normalisation of RT-qPCR Quantification Cycle (Cq) Data." *BMC Genomics* 13 (1): 296. doi:10.1186/1471-2164-13-296.

Phillips, Nathaniel, Hansjoerg Neth, Jan Woike, and Wolfgang Gaissmaer. 2017. *FFTrees: Generate, Visualise, and Evaluate Fast-and-Frugal Decision Trees*. https://CRAN.R-project.org/package=FFTrees.

Porzelius, Christine, Harald Binder Jochen Knaus, and Guido Schwarzer. 2009. "Easier Parallel Computing in R with Snowfall and sfCluster." *The R Journal* 1 (1): 54–59. http://journal.r-project.org/archive/2009-1/RJournal_2009-1_Knaus+et+al.pdf.

Quinlan, J. Ross. 1986. "Induction of Decision Trees." *Machine Learning* 1 (1): 81–106. http://link.springer.com/article/10.1007/BF00116251.

Ritz, Christian, and Andrej-Nikolai Spiess. 2008. "qpcR: An R Package for Sigmoidal Model Selection in Quantitative Real-Time Polymerase Chain Reaction Analysis." *Bioinformatics* 24 (13): 1549–51. doi:10.1093/bioinformatics/btn227.

Rödiger, Stefan, Alexander Böhm, and Ingolf Schimke. 2013. "Surface Melting Curve Analysis with R." *The R Journal* 5 (2): 37–53. http://journal.r-project.org/archive/2013-2/roediger-bohm-schimke.pdf.

Rödiger, Stefan, Michał Burdukiewicz, and Peter Schierack. 2015. "chipPCR: An R Package to Pre-Process Raw Data of Amplification Curves." *Bioinformatics* 31 (17): 2900–2902. doi:10.1093/bioinformatics/btv205.

Rödiger, Stefan, Michał Burdukiewicz, Konstantin A. Blagodatskikh, and Peter Schierack. 2015. "R as an Environment for the Reproducible Analysis of DNA Amplification Experiments." *The R Journal* 7 (2): 127–50. http://journal.r-project.org/archive/2015-1/RJ-2015-1.pdf.

Rödiger, Stefan, Michał Burdukiewicz, Andrej-Nikolai Spiess, and Konstantin Blagodatskikh. 2017. "Enabling Reproducible Real-Time Quantitative PCR Research: The RDML Package." *Bioinformatics*, August. doi:10.1093/bioinformatics/btx528.

Rödiger, Stefan, Peter Schierack, Alexander Böhm, Jörg Nitschke, Ingo Berger, Ulrike Frömmel, Carsten Schmidt, et al. 2013. "A Highly Versatile Microscope Imaging Technology Platform for the Multiplex Real-Time Detection of Biomolecules and Autoimmune Antibodies." *Advances in Biochemical Engineering/Biotechnology* 133: 35–74. doi:10.1007/10_2011_132.

Ronde, Maurice W. J. de, Jan M. Ruijter, David Lanfear, Antoni Bayes-Genis, Maayke G. M. Kok, Esther E. Creemers, Yigal M. Pinto, and Sara-Joan Pinto-Sietsma. 2017. "Practical Data Handling Pipeline Improves Performance of qPCR-Based Circulating miRNA Measurements." *RNA* 23 (5): 811–21. doi:10.1261/rna.059063.116.

Ruijter, J M, C Ramakers, W M H Hoogaars, Y Karlen, O Bakker, M J B van den Hoff, and A F M Moorman. 2009. "Amplification Efficiency: Linking Baseline and Bias in the Analysis of Quantitative PCR Data." *Nucleic Acids Research* 37 (6): e45. doi:10.1093/nar/gkp045.

Ruijter, Jan M., Steve Lefever, Jasper Anckaert, Jan Hellemans, Michael W. Pfaffl, Vladimir Benes, Stephen A. Bustin, Jo Vandesompele, Andreas Untergasser, and on behalf of the RDML Consortium.

2015. “RDML-Ninja and RDMLdb for Standardized Exchange of qPCR Data.” *BMC Bioinformatics* 16 (1): 197. doi:10.1186/s12859-015-0637-6.

Ruijter, Jan M., Michael W. Pfaffl, Sheng Zhao, Andrej N. Spiess, Gregory Boggy, Jochen Blom, Robert G. Rutledge, et al. 2013. “Evaluation of qPCR Curve Analysis Methods for Reliable Biomarker Discovery: Bias, Resolution, Precision, and Implications.” *Methods* 59 (1): 32–46. doi:10.1016/j.ymeth.2012.08.011.

Ruijter, Jan M., Adrián Ruiz Villalba, Jan Hellemans, Andreas Untergasser, and Maurice J. B. van den Hoff. 2015. “Removal of Between-Run Variation in a Multi-Plate qPCR Experiment.” *Biomolecular Detection and Quantification*, Special Issue: Advanced Molecular Diagnostics for Biomarker Discovery – Part I, 5 (September): 10–14. doi:10.1016/j.bdq.2015.07.001.

Sauer, Eva, Ann-Kathrin Reinke, and Cornelius Courts. 2016. “Differentiation of five body fluids from forensic samples by expression analysis of four microRNAs using quantitative PCR.” *Forensic Science International: Genetics* 22 (May): 89–99. doi:10.1016/j.fsigen.2016.01.018.

Schmidberger, Markus, Martin Morgan, Dirk Eddelbuettel, Hao Yu, Luke Tierney, and Ulrich Mansmann. 2009. “State-of-the-Art in Parallel Computing with R.” *Journal of Statistical Software* 47 (1).

Spiess, Andrej-Nikolai, Claudia Deutschmann, Michał Burdukiewicz, Ralf Himmelreich, Katharina Klat, Peter Schierack, and Stefan Rödiger. 2015. “Impact of Smoothing on Parameter Estimation in Quantitative DNA Amplification Experiments.” *Clinical Chemistry* 61 (2): 379–88. doi:10.1373/clinchem.2014.230656.

Spiess, Andrej-Nikolai, Stefan Rödiger, Michał Burdukiewicz, Thomas Volksdorf, and Joel Tellinghuisen. 2016. “System-Specific Periodicity in Quantitative Real-Time Polymerase Chain Reaction Data Questions Threshold-Based Quantitation.” *Scientific Reports* 6 (December): 38951. doi:10.1038/srep38951.

Tierney, Nicholas. 2017. “visdat: Visualising Whole Data Frames.” *The Journal of Open Source Software* 2 (16). The Open Journal. doi:10.21105/joss.00355.

Todorov, Valentin, and Peter Filzmoser. 2009. “An Object-Oriented Framework for Robust Multivariate Analysis.” *Journal of Statistical Software* 32 (3). doi:10.18637/jss.v032.i03.

Wilson, Greg, Jennifer Bryan, Karen Cranston, Justin Kitzes, Lex Nederbragt, and Tracy K. Teal. 2016. “Good Enough Practices in Scientific Computing,” August. https://arxiv.org/abs/1609.00037.