Stefan Rödiger* and Michał Burdukiewicz and Peter Schierack

Supplement to: “chipPCR: an R Package to Pre-Process Amplification Curve Data”

Contents

Abstract

Background: The quantitative real-time polymerase chain reaction (qPCR) and isothermal amplification are standard methods for quantification of nucleic acids. Numerous real-time read-out technologies with different technical foundation have been developed. However, the amplification curve analysis consists of cascaded steps, which are carried out similarly in all technologies. Despite the continuous interest in amplification based techniques, there are only few transparent tools for amplification data pre-processing. It is a major setback especially during development of new instruments, when the precise and reproducible processing of raw data is indispensable.

Results and Conclusion: *chipPCR* is an **R** package for pre-processing and quality analysis of amplification curve data from conventional quantitative polymerase chain reactions (qPCR) and quantitative isothermal amplification (qIA). Isothermal amplifications are monocyclic reactions at a constant temperature. Conversely, PCR is a polycyclic reaction with thermal cycling condition steps (denaturation, annealing, elongation) and measurements at discrete cycle steps. The curve shape of a qIA and ccPCR do not necessarily follow a S-shaped structure and the measurement is time-based (continuous, not mandatory equidistant) in contrast to a cycle-based (discontinuous) measurement of qPCRs. The number of measure points in most cases higher than qPCR.

This supplement provides further details and examples for the *chipPCR* package. The package contains several data sets, which were generated by helicase dependent amplification (HDA) or polymerase chain reaction (PCR) under various temperature conditions and detection systems, such as hydrolysis probes and intercalating dyes. Examples for their usage are presented herein. We have developed *chipPCR*, which is a versatile software tailored for the pre-processing of amplification curve data. Its utility is elaborated on both real and simulated data sets. The structure of the packages is open for integration to Web based and standalone *shiny* applications. The **R** package along codes used for creation of figures used in publication is freely available.

1 Availability, requirements and setting up a working environment

Availability and requirements are an **R** installation and optionally a working Internet connection and web browser:

- Project name: chipPCR,
- Project homepage (development): https://github.com/michbur/chipPCR,
- Project homepage at CRAN: http://cran.r-project.org/web/packages/chipPCR/index.html,
- Operating System: Platform independent,
- Other requirement: R 3.1.0 or higher,
- License: GPL-3

We use **R**'s object model *S4* class system (see *methods* package) to separate between interface and implementation. Other than in **R**'s *S3* class system, needs *S4* to declare classes, slots, methods relationships explicitly and to establish formal declarations of methods. Therefore, the number and types of slots in an instance of a class have to be established at the time the class definition and the objects from the class are validated against this definition and have to comply to it at any time. *S4* methods are declared by calls to *setMethod* together with the names and signatures of the arguments. Signatures are used for identification of classes of one or more arguments of the methods. *setGeneric* can be used to declare generic functions. *S4* classes require a higher effort than *S3* classes, but assure strictly objects in a class have the required slots, that data in the slots have consistent names and classes, and enable to include additional information (e.g., results, parameters). Moreover, it assures better control on the object structure and the method dispatch [16]. As perquisite for high-throughput technologies we avoided loops in the core structures and use partially parallel computing (*smoother* function) to keep the code fast.

The vignette can be viewed from **R** using command: vignette("chipPCR"). Further details of the experimental set-up for the data sets are described in the manual of the *chipPCR* package. Before the start of any analysis, a user must choose a data set, as shown in example below.

```r
# Load chipPCR
require(chipPCR)
# Load package for table formatting
require(xtable)
# Print table
print(xtable(head(C60.amp[, 1L:5]), caption = "First five cycles of imported data."))
```

	Index	Vim.0.1	Vim.0.2	Vim.1.1	Vim.1.2
1	0	0.00	0.00	-0.03	-0.03
2	1	0.00	0.00	-0.03	-0.03
3	2	0.00	-0.00	-0.02	-0.02
4	3	-0.00	-0.00	-0.01	-0.01
5	4	-0.00	-0.00	0.01	0.01
6	5	-0.00	-0.00	0.05	0.05

Table S1: First five cycles of imported data.

All datasets used in following examples can be loaded in the same manner. They are also automatically available after loading whole package. *chipPCR* relies on the **R** workspaces, and dedicated **R** packages as default data format and standard import and export as described elsewhere [3, 28, 30]. The *chipPCR* can be used to read the Real-time PCR Data Markup Language (RDML) from various qPCR cycler systems. This means that basically any RDML compliant data set can be directly used with the *chipPCR* package.

Graphical user interface (GUI) are important to spread software and to make it available also for researchers not fluent in **R**. We implemented core functionality of our package in various **R** offers several GUI projects to chose from [35]. Recently, the *shiny* [37] framework to build and deploy GUIs for the desktop (web browser) or services for interactive web applications emerged. *shiny* enables to build plugin-like applications with highly customizable widgets (e.g., sliders, plots, reports) for a efficient extension. *shiny* applications update live and interactively. The user interfaces can be built entirely using **R** and operates in any **R** environment (cross-platform). Currently, the functions *AmpSim*, *th.cyc*, *bg.max* and *amptester* are part of *shiny* GUIs.

2 Pre-processing of amplification curve data

Quantitative polymerase chain reaction (qPCR) and quantitative isothermal amplification (qIA) are standard methods to amplify nucleic acids (e.g., genomic DNA, copy DNA). Recently amplification methods with a continuous temperature gradient (e.g., microfluidics, capillary convective PCR (ccPCR)) emerged [6, 41]. All these amplification methods are used in different real-time monitoring technologies, such as our previously reported VideoScan technology, microfluidic systems, point-of-care devices, microbead-chip technologies and commercial real-time thermo cyclers [5, 36, 41, 42].

Real-time technologies enable the quantification of nucleic acids by calculation of specific curve parameters like the quantification point (Cq) and the amplification efficiency (AE) [40, 49]. Novel qPCR and qIA technologies initially depend on tools to pre-process the raw data. Pre-processing specifically addresses raw data inspection, steps to transform raw data in a compatible format for successive analysis steps (e.g., smoothing, imputation of missing values), data reduction (e.g., removal of invalid sets) and data quality management. The data quality of experimental instruments is often not ready for end-user analysis and presentation. It is important to use as many raw data as possible. Pre-processing algorithms remove stochastic errors and artefacts (e.g., noise, photo-bleaching effects, degassing effects, different signal levels) illustrated in Figure S1. Misinterpretations are more likely if non or manual ("arbitrary") corrections are performed. A manual alteration is in contradiction to reproducible research. The presence of noise may lead to false conclusions and performance estimation. Noise is challenging because derivative processes as used for "cycle of quantification" methods (e.g., Second Derivative Maximum method) lead to an amplification of noise [20, 34, 40, 49, 52].

R belongs to the most used bioinformatics tools and is known to be an early adopter of emerging technologies such as digital PCR and NanoString nCounter Platform [26, 54, 26]. At same time it is widely used for established methods, such as qPCR. Most **R** packages focus on the read-in and (post)-processing of data sets, which originate from commercial qPCR systems. Fundamental steps of amplification curve analysis are: (1) raw data read-in, (2) amplification curve pre-processing (e.g., noise reduction, outlier removal), (3) amplification curve processing (e.g., Cq and AE calculation), (4) post-processing and quantification of secondary parameters (e.g., Delta-Delta-Ct for gene expression analysis), (5) data export, (6) visualization and (7) report generation. Sophisticated **R** packages for the steps 1. and 3.–7. are available from Bioconductor and CRAN [8, 11, 14, 15, 23, 26, 28, 56, 57]. However, there is no **R** package for pre-processing and quality analysis of amplification curve data. Similar applies to other software solutions (compare [26]). Pre-processing is in most commercial cyclers a "black box". This has limitations, such as hard to reproduce analysis on other platforms, difficulties during the adoption to changing experimental setups and limited sophisticated statistical tools. Moreover, it is desirable set up work-flows in an open environment, which enables downstream analyses and which offers powerful tools for data visualizations and automatic report generation.

The *chipPCR* package (*chipPCR*: "Lab-on-a-**Chip**" & **PCR**) was developed to automatize pre-processing, to ease data analysis/visualization and to offer a quality control for the statistical data analysis of qPCR and qIA experiments by a comprehensive set of functions (Section 3). *chipPCR* is primarily targeted at developers of experimental nucleic acid amplification systems but non the less at users who have access to the raw data of commercial systems. This supplement highlights pinnacles of the *chipPCR* package. Due to the vast amount of information we refer the interested reader to the *chipPCR* manual for further information and use-case examples on the parameters of the different functions.

Pre-processing is in most commercial cyclers an black box. This has limitations, such as hard to reproduce analysis on other platforms, difficulties during the adoption to changing experimental setups and limited sophisticated statistical tools. Moreover, it is desirable set up work-flows in an open environment, which enables downstream analyses and which offers powerful tools for data visualizations and automatic report generation.

```r
# Use AmpSim to generate an amplification curve with 40
# cycles and a different Cq.
res.pos <- AmpSim(cyc = 1:40, noise = TRUE, b.eff = -12, nnl = 0.02)
res.pos[5, 2] <- res.pos[5, 2] * 6

res.low <- AmpSim(cyc = 1:40, noise = TRUE, b.eff = -20, bl = 0.5,
    ampl = 0.58, Cq = 33)
# Add missing value to res.low at cycle 31
res.low[31, 2] <- NA

res.neg <- AmpSim(cyc = 1:40, b.eff = -0.1, bl = 0.05, ampl = 0.4,
    Cq = 1, noise = FALSE, nnl = 0.5)

res.pos.CPP <- cbind(1:40, CPP(res.pos[, 1], res.pos[, 2], bg.outliers = TRUE,
    smoother = TRUE, method = "smooth", method.norm = "minm",
    method.reg = "lmrob")$y)

res.low.NA <- cbind(1:40, CPP(res.low[, 1], res.low[, 2], smoother = TRUE,
    method = "smooth", bg.outliers = TRUE, method.norm = "minm",
    method.reg = "lmrob")$y)

res.neg.exc <- cbind(1:40, amptester(res.neg[, 2]))

par(mfrow = c(1, 2), las = 0, bty = "n", cex.axis = 1.5, cex.lab = 1.5,
    font = 2, cex.main = 1.8, oma = c(1, 1, 1, 1))
plot(NA, NA, xlim = c(1, 40), ylim = c(0, max(res.pos[, 2])),
    xlab = "Cycle", ylab = "Raw fluorescence")
mtext("A", cex = 2, side = 3, adj = 0, font = 2)

lines(res.pos, lwd = 2)
lines(res.low, col = 2, lwd = 2)
arrows(38, min(res.low[, 2], na.rm = TRUE), 38, max(res.low[,
    2], na.rm = TRUE), code = 3, lwd = 3, angle = 90, col = "grey")
text(38, max(res.low[, 2], na.rm = TRUE) * 0.7, "SNR", cex = 1.2)

arrows(29, 0.42, 31, 0.51, lwd = 2)
text(29, 0.38, "NA", cex = 1.2)

points(res.pos[5, 1], res.pos[5, 2], pch = 21, cex = 4, lwd = 5,
    col = "orange")
text(res.pos[5, 1], res.pos[5, 2] * 1.2, "Outlier", cex = 1.2)

lines(res.neg, col = 4, lwd = 2)
text(20, mean(res.neg[, 2]) * 0.9, "No amplification", cex = 1.2,
    col = "blue")


plot(NA, NA, xlim = c(1, 40), ylim = c(0, max(res.pos[, 2])),
    xlab = "Cycle", ylab = "Pre-processed fluorescence")
abline(h = 0.03, lty = 2, lwd = 2)
mtext("B", cex = 2, side = 3, adj = 0, font = 2)

lines(res.pos.CPP, lwd = 2)
lines(res.low.NA, col = 2, lwd = 2)
lines(res.neg.exc, col = 4, lwd = 2)

legend(1, 1, c("Positive (outlier removed)", "Positive (scaled)",
    "Negative", "Threshold line nof Cq"), col = c("black", "red",
    "blue", "black"), lty = c(1, 1, 1, 2), lwd = 2, bty = "n")
lines(c(15.1, 15.1), c(-1, 0.03), lwd = 2, col = "black")
text(14, 0.06, "Cq")
lines(c(28.5, 28.5), c(-1, 0.03), lwd = 2, col = "red")
text(27, 0.06, "Cq", col = "red")
```

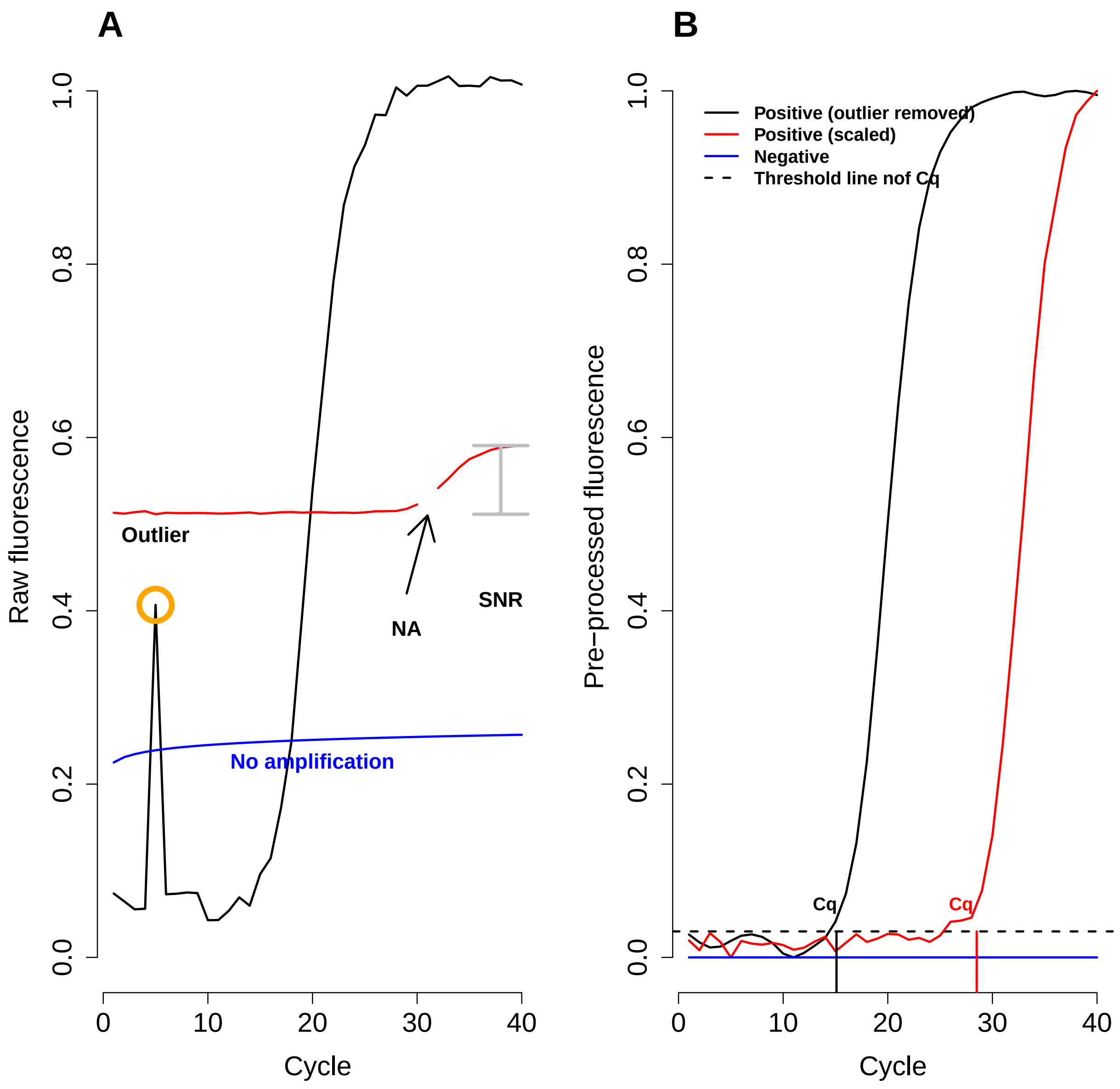


Figure S1: Analysis and interpretation of real-time amplification curves. *(A)* The fluorescence values are plotted against the cycle. The amplification curve has a sigmoidal shape (–, –). Amplification curve raw data are affected by many influences. This includes noise introduced by the detection system and sensor errors. Measurements can occasionally contain missing values (“NA”, –) and outliers (orange circle, –). Outliers are often present in the first cycle due to sensor adjustments. The signal difference between the background phase (first cycles) and the plateau phase (last cycles) can be expressed as signal-to-noise ratio (SNR). The SNR between different between samples (e.g., – and –) can vary. For interpretation it is better to compensate the differences. Negative samples (–) need to be (automatically) identified. *(B)* pre-processed raw data. NAs were imputed and the noise slightly removed. The curves were adjusted to have the same baseline and plateau level. The quantification point (Cq) of the positive reactions are determined in the exponential phase (“Threshold method” is used in this example). Negative sample are automatically set to zero.

3 Functions of the chipPCR package

Main functions of the *chipPCR* package are:

- *AmpSim*: a 5-parameter model for a S-shaped amplification curves accompanied by *AmpSim.gui* a *shiny* GUI, for *AmpSim*,
- *bg.max*: a function to detect the start and end of an amplification reaction,
- *CPP*: wrapper for several pre-processing functions,
- *fixNA*: to impute missing values in a data column,
- *inder*: for interpolating first and second derivatives interpolation using the five-point stencil (accompanied by *rounder* function),
- *MFIaggr*: to analyze a bulk of replicates of an amplification reaction,
- *smoother*: to smooth the curve data by different methods (e.g., moving average, Savitzky-Golay smoothing).

In addition, further the (auxiliary) functions:

- *amptester*: to detect the start and end of an amplification curve,
- *effcalc*: to calculate the amplification efficiency,
- *humanrater*: a graphical interface to rate curves,
- *lm.coefs*: to compute linear model coefficients,
- *normalizer*: to normalize data between a user defined range,
- *plotCurves*: to plots many curves on one plot in separate cells allowing quick assessment,
- *th.cyc*): to calculate the number of cycles at which the fluorescence exceeds a defined threshold, called the threshold cycle (Ct),

are available. These functions are typically used for post-processing (e.g., Cq calculation, amplification efficiency calculation) and quality analysis are available from the *chipPCR* package. For more information please refer to the following sections. Selected functionality is used in the *RDML* [3] package. Most of the functions are element of other functions in this package. For example, *fixNA* is embedded in most functions to prevent potential problem during processing due to missing values.

Graphical user interface (GUI) are important to spread software. **R** offers several GUI projects to chose from [35]. Some functionality of *chipPCR* originates from experimental plugins for the **RKWard** GUI [26]. Recently, the *shiny* [37] framework to build and deploy GUIs for the desktop (web browser) or services for interactive web applications emerged. *shiny* enables to build plugin-like applications with highly customizable widgets (e.g., sliders, plots, reports) for a efficient extension. *shiny* applications update live and interactively. The user interfaces can be built entirely using **R** and operates in any **R** environment (cross-platform). Currently, the functions *AmpSim*, *th.cyc*, *bg.max* and *amptester* are part of *shiny* GUIs. It is possible to run the applications as service on a server. Examples for these use case scenarios are given in the following sections. The following section give details on the application of the different functions.

4 *AmpSim* - a function to simulate amplification curves

The function *AmpSim* is a simulator for amplification reactions. Use cases include teaching, algorithm testing or the comparison of an experimental system to the predicted ("optimal") model. *AmpSim* uses a 5-parameter model (Equation S1). The function is commonly used for the simulation of amplification curves [12, 33, 47].

$$fluo = bl + \frac{ampl - bl}{1 + \exp\left(b.eff * (\log cyc - \log Cq)\right)} \tag{S1}$$

It is an intrinsic property of *AmpSim* to generate unique results if the *noise* parameter is *bg.max* set $TRUE$. This is due the use of the *rnorm* (*stats*) function to simulate noise. If data need to be replicated identically use *set.seed(123)* to alter the random number generator (RNG) state. For example, the amplification curves of Figure S25 *A* are generated with the same starting parameter of *AmpSim* but noise was added. *AmpSim.gui* is a *shiny* GUI (graphical user interface [37]) implementation for *AmpSim*. The code example below is an example how-to invoke the *AmpSim.gui*. Further details on *shiny* are described in Section 1. *AmpSim* was also used to illustrate the *inder* function (Figure S19), the *fixNA* function (Figure S12) and the use of the *smoother* (Figure S14) function.

There are several ways to implement *chipPCR* GUIs. An example to launch a graphical user interface that allows simulating and analyzing amplification reactions is shown below. This approach is usable to run a GUI on a local machine without the requirement to connect to the Internet. Prerequisites are a local installation of **R**, with installed *shiny* and *chipPCR* packages and a modern web browser. It is important to note that any ad-blocking software may be cause of malfunctions. The GUI is invoked by pasting the following code snippet into a **R** console.

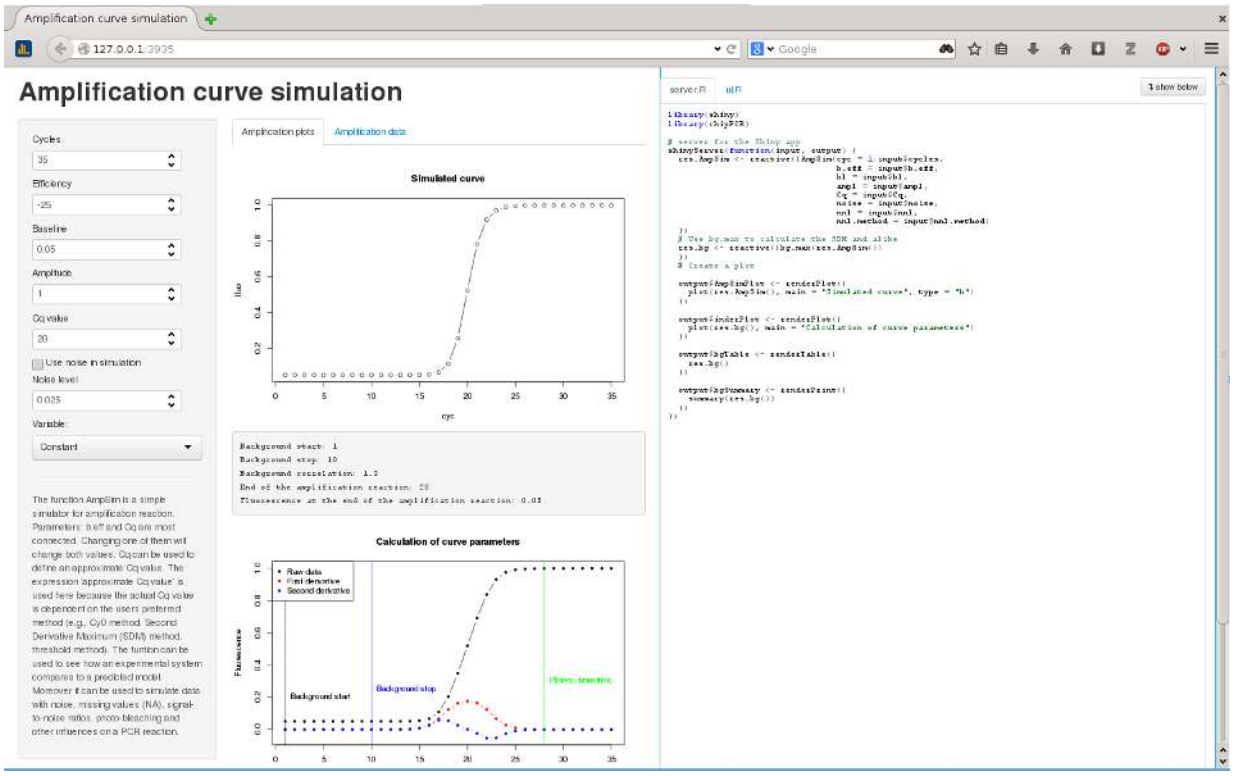
```r
# Load the shiny package (chipPCR should already be loaded).
# Run from a R console following commands.
require(shiny)

# Invoke the shiny AmpSim app in the default browser.
runApp(paste(find.package("chipPCR")[1], "/AmpSim.gui", sep = ""))

# Call shiny app AmpSim directly from gist
runGist("https://gist.github.com/michbur/e1def41598f1d0c1e2e6")
```

The function will open the GUI in a *chipPCR* webpage of the default browser (Figure S2). All parameters of the *AmpSim* function can be used. In addition, shows the GUI some information calculated by the *bg.max* in a summary field and a plot below the simulated amplification curve.

AmpSim has several parameters, which can be used to simulate an amplification curve. $b.eff$ and Cq are most connected with another. Thus changing one of them will change both values. Cq can be used to define an approximate Cq value. The expression "approximate Cq" value is used because the calculated Cq value will vary depending on the preferred Cq quantification method (e.g., Second Derivative Maximum (SDM) method, threshold method). *AmpSim* can be used to simulate data with noise (based on *rnorm*, *stats*), signal-to-noise ratios, photo-bleaching and other influences on a qPCR reaction. The following example illustrates the use of *AmpSim* (Figure S3).

Warnings in following code chunks were suppressed.

```r
# Draw an empty plot for 40 cycles with user defined
# parameters.

par(las = 0, bty = "n", oma = c(0.5, 0.5, 0.5, 0.5))
plot(NA, NA, xlim = c(1, 40), ylim = c(0, 1.1), xlab = "Cycle",
    ylab = "RFU")
colors <- rainbow(8)

# Create eight amplification curves. The approximate Cqs are
# synthesized as temporary Cqs by adding a random value to a
# starting Cq of 25. Note: ``noise'' is set TRUE with a level
# of nnl = 0.03. This adds some scatter to the amplification
# curves.

sim <- sapply(1L:8, function(i) {
    Cq.tmp <- 25 + rnorm(1) * 5
```



Cycles: 35
Efficiency: -25
Baseline: 0.05
Amplitude: 1
Cq value: 20
☐ Use noise in simulation
Noise level: 0.025
Variable: Constant

The function AmpSim is a simple simulator for amplification reaction. Parameters: b.eff and Cq are most connected. Changing one of them will change both values. Cq can be used to define an approximate Cq value. The expression 'approximate Cq value' is used here because the actual Cq value is dependent on the users preferred method (e.g., Cy0 method, Second Derivative Maximum (SDM) method, threshold method). The funtion can be used to see how an experimental system compares to a predicted model. Moreover it can be used to simulate data with noise, missing values (NA), signal-to-noise ratios, photo-bleaching and other influences on a PCR reaction.

```
Background start: 1
Background stop: 10
Background correlation: 1.3
End of the amplification reaction: 28
Fluorescence at the end of the amplification reaction: 0.05
```

```
library(shiny)
library(chipPCR)

# server for the Shiny app
shinyServer(function(input, output) {
  res.AmpSim <- reactive({AmpSim(cyc = 1:input$cycles,
                                 b.eff = input$b.eff,
                                 bl = input$bl,
                                 ampl = input$ampl,
                                 Cq = input$Cq,
                                 noise = input$noise,
                                 nnl = input$nnl,
                                 nnl.method = input$nnl.method)
  })
  # Use bg.max to calculate the SDM and alike
  res.bg <- reactive({bg.max(res.AmpSim())
  })
  # Create a plot

  output$AmpSimPlot <- renderPlot({
    plot(res.AmpSim(), main = "Simulated curve", type = "b")
  })

  output$inderPlot <- renderPlot({
    plot(res.bg(), main = "Calculation of curve parameters")
  })

  output$bgTable <- renderTable({
    res.bg()
  })

  output$bgSummary <- renderPrint({
    summary(res.bg())
  })
})
```

Figure S2: Locally running *shiny AmpSim.gui* app. *(Top)* The plot of the *AmpSim.gui* is shown in a standard browser (**Iceweasel**, v. 29.0.1) along with the parameters (left panel) and the estimation by the *th.cyc* function. The code ("server.R", "ui.R") of the *shiny* app is also shown. All parameters (e.g., Cq value, baseline) of the *AmpSim* function are accessible. *(Bottom)* In addition, shows *AmpSim.gui* the plot output and the textual results of *bg.max*.

```
    tmp <- AmpSim(1:40, Cq = Cq.tmp, noise = TRUE, nnl = 0.03)
    lines(tmp, col = colors[i], lwd = 2)

    # Add the approximate Cq values to the plot
    text(3, 1 - i/10, paste("Cq ", round(Cq.tmp, 2)), col = colors[i])
})
```

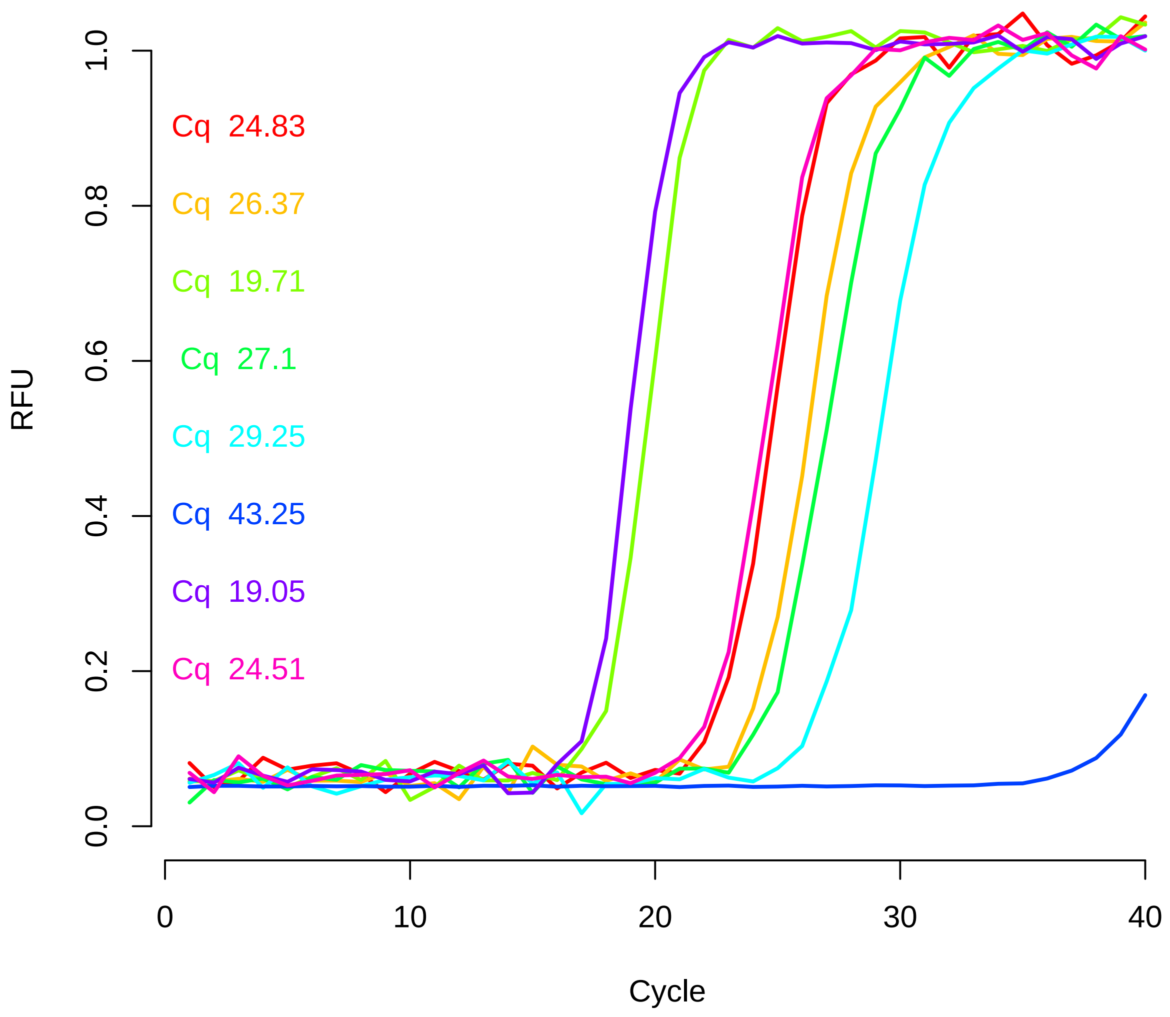
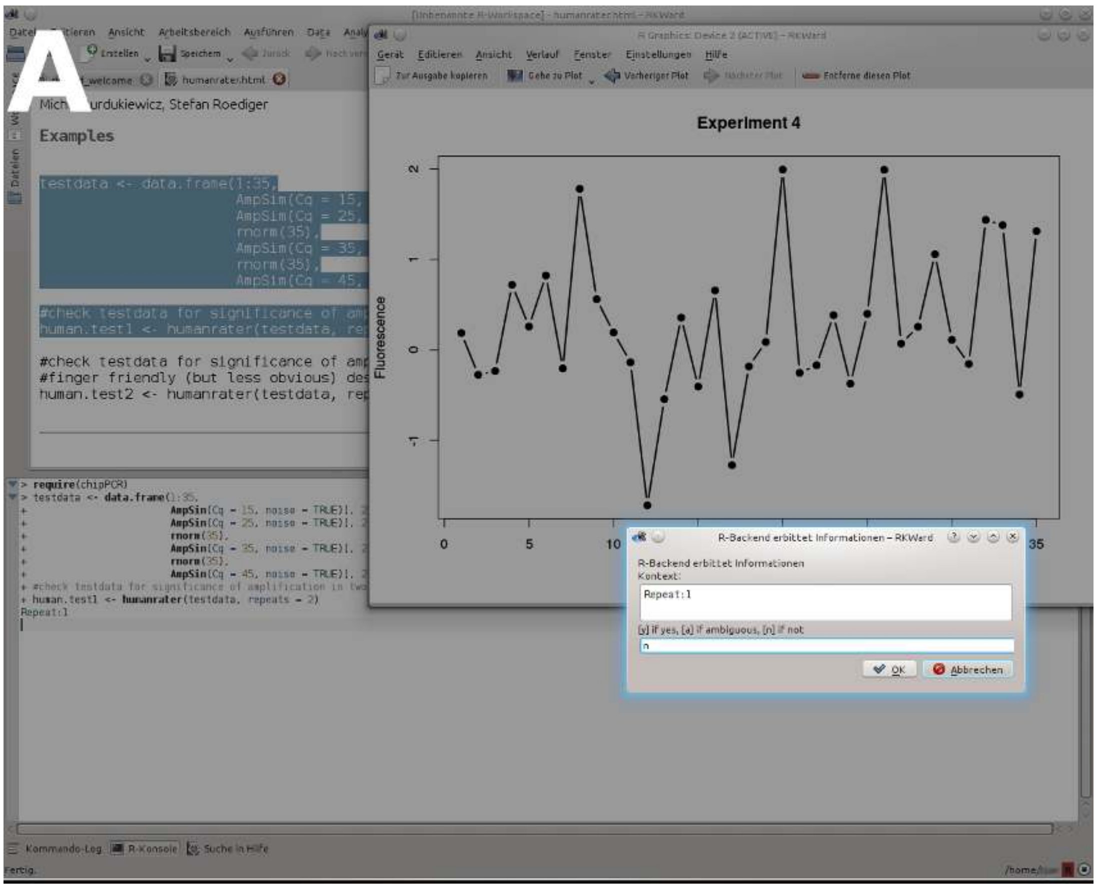


Figure S3: The amplification curves were generated with the *AmpSim* function. All Cqs are unique due to the use of random value, which were added to the starting Cq of 25. The parameter *noise* = 0.03 adds some scatter to the amplification curve data.

5 Single-blinded, randomized judging of amplification curves

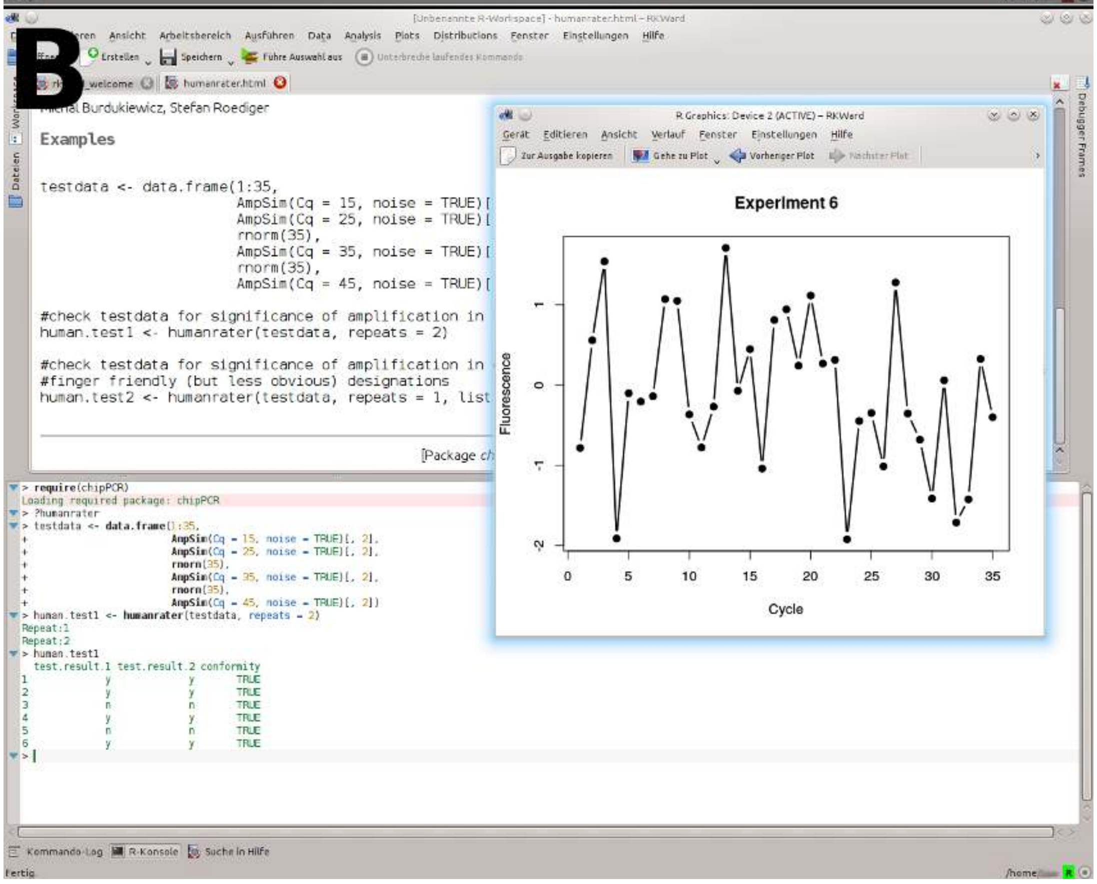
Humans tend to bias the interpretation towards a desirable outcome. A single-blinded and randomized experiment aims to reduce bias in the results, which concealed from the human tester. The bias towards a certain result may be intentional or unconscious. We developed the *humanrater* function to evaluate the quality of an amplification curve in a randomized, half-blind manner. *humanrater* is an interactive function, which can be used to rate a curve for a certain characteristic. *humanrater* draws individual graphs of a curve and prompts an input field for the user. Application of this function are numerous. For example, this function can be used to compare the human rating and the rating of a machine or the rating of several individual experts. It is possible to specify a list of designations, which seem appropriate to characterize the amplification curve, where the names of elements specify short designations used during rating. Defaults are *y* for "yes", *a* for "ambiguous" and *n* for "no". It is possible to supply longer or shorter designations lists. In our example we used *humanrater* in the **RKWard** GUI (Figure S4). We aimed to characterize amplification curves which were randomly drawn from "testdata" to avoid a bias by the user. However, it should be noted that *humanrater* can be used to rate other input data such as melting curves.

```r
# Create a set of data to be analyzed by humanrater.  The
# function AmpSim will create amplification curves which
# follow a nearly optimal sigmoidal curve shape or just
# noise.

testdata <- data.frame(1:35, AmpSim(Cq = 15, noise = TRUE)[,
    2], AmpSim(Cq = 25, noise = TRUE)[, 2], rnorm(35), AmpSim(Cq = 35,
    noise = TRUE)[, 2], rnorm(35), AmpSim(Cq = 45, noise = TRUE)[,
    2])

# Use testdata as input for humanrater and assign the results
# to the object human.test.  check testdata for significance
# of amplification in two repeats.

human.test1 <- humanrater(testdata, repeats = 2)
```


A
Experiment 4
Fluorescence
R-Backend erbittet Informationen – RKWard
R-Backend erbittet Informationen
Kontext:
Repeat:1
[y] if yes, [a] if ambiguous, [n] if not
n
OK
Abbrechen

```
Examples

testdata <- data.frame(1:35,
                       AmpSim(Cq = 15,
                       AmpSim(Cq = 25,
                       rnorm(35),
                       AmpSim(Cq = 35,
                       rnorm(35),
                       AmpSim(Cq = 45,

#check testdata for significance of am
human.test1 <- humanrater(testdata, re

#check testdata for significance of am
#finger friendly (but less obvious) de
human.test2 <- humanrater(testdata, re
```

```
> require(chipPCR)
> testdata <- data.frame(1:35,
+                        AmpSin(Cq = 15, noise = TRUE)[, 2
+                        AmpSin(Cq = 25, noise = TRUE)[, 2
+                        rnorm(35),
+                        AmpSin(Cq = 35, noise = TRUE)[, 2
+                        rnorm(35),
+                        AmpSin(Cq = 45, noise = TRUE)[, 2
+ #check testdata for significance of amplification in two
+ human.test1 <- humanrater(testdata, repeats = 2)
Repeat:1
```



```
Examples

testdata <- data.frame(1:35,
                       AmpSim(Cq = 15, noise = TRUE)[
                       AmpSim(Cq = 25, noise = TRUE)[
                       rnorm(35),
                       AmpSim(Cq = 35, noise = TRUE)[
                       rnorm(35),
                       AmpSim(Cq = 45, noise = TRUE)[

#check testdata for significance of amplification in
human.test1 <- humanrater(testdata, repeats = 2)

#check testdata for significance of amplification in
#finger friendly (but less obvious) designations
human.test2 <- humanrater(testdata, repeats = 1, list
```

```
> require(chipPCR)
Loading required package: chipPCR
> ?humanrater
> testdata <- data.frame(1:35,
+                        AmpSim(Cq = 15, noise = TRUE)[, 2],
+                        AmpSim(Cq = 25, noise = TRUE)[, 2],
+                        rnorm(35),
+                        AmpSim(Cq = 35, noise = TRUE)[, 2],
+                        rnorm(35),
+                        AmpSim(Cq = 45, noise = TRUE)[, 2])
> human.test1 <- humanrater(testdata, repeats = 2)
Repeat:1
Repeat:2
> human.test1
  test.result.1 test.result.2 conformity
1             y             y       TRUE
2             y             y       TRUE
3             n             n       TRUE
4             y             y       TRUE
5             n             n       TRUE
6             y             y       TRUE
> 
```

Figure S4: Application of *humanrater* in a instance of **RKWard**. *humanrater* was used to analyze a row of amplification curves. *(A)* All data are anonymous and can be randomized during the rating. The number of repeats for the rating and the categories (e.g., *y* for "yes", *a* for "ambiguous" and *n* for "no") can be defined by the user. The function has an option to present the curves at random (default). *(B)* The user gets as result a tabular output, including the result of each run and the conformity of the runs (see table in console).

6 Inspection and analysis of amplification curve data

The following section briefly describes function from the *chipPCR* to visualize and analyze amplification curve data. In particular, the functions *MFIaggr* and *plotCurves* (Section 6.2) were developed for a rapid and convenient inspection of raw data.

MFIaggr is a powerful analytical and graphical tool for fast multiple comparison of the cycle dependent signal dispersion and distribution. The continuous response variable y is used to describe the relationships to n continuous predictor variables x_i, where $i \in \{1, ..., n\}$. Use cases include the comparison of independent reaction vessels or the analysis of replicate experiments. The basic idea is to analyze only a region of interest (ROI) from a data set. *MFIaggr* is a relative of the *MFIerror* function from the *MBmca* ([34]) package. However, this functions enables a fine grained analysis of specific parts of the curve data. The function returns an object of the class list with the columns "Cycle", "Location" (Mean, Median), "Deviation" (Standard Deviation, Median Absolute Deviation) and "Coefficient of Variation". Using the option $rob = TRUE$ the median and the median absolute deviation (MAD) are calculated instead of the mean and standard deviation. *MFIaggr* has parameter *llul* to define the lower and upper data limit (cycle) for a ROI. The results for the ROI can be invoke by *@stats*. The output includes the mean, median, standard deviation (sd), median absolute deviation (mad), inter quartile range (IQR), medcouple (robust measure of skewness), skewness (Pearson's second skewness coefficient; $skewness = 3\ (mean(x) - median(x))\ /\ sd(x)$), signal-to-noise ratio (SNR), variance-to-mean ratio (VRM), number of missing values (NAs) and results from a linear fit of the ROI (intercept, slope, r.squared). Moreover, we included the Breusch-Pagan test to test for heteroskedasticity in a linear regression model (see Section 6). In our example we analyzed the raw fluorescence from 96 replicates of a qPCR experiment for the human gene *Vimentin*. The *MFIaggr* plot shows that the first ten cycles (noise) follow a normal distribution (Figure S5). In contrast, the analysis of all cycles shows expectedly a distribution, which significantly differs from a normal distribution (Figure S7). Setting the option $CV = FLASE$ shows the relative standard deviation (RSD, %). Ideally the variance between the amplification curves is low. Other results of *MFIaggr* include the density analysis (*@density*), the quantile (*@qqnorm.data*), and the results of the linear regression (*@lm.roi*) from the ROI. In particular, this function might be useful for quality management during the development of high-throughput technologies. An analysis via a the *shiny* *MFIaggr.gui* app is shown in Figure S6.

```r
par(las = 0, bty = "n", cex.axis = 1.2, cex.lab = 1.2, font = 2,
    cex.main = 1.2, oma = c(1, 1, 1, 1))

plot(MFIaggr(VIMCFX96_60[, 1], VIMCFX96_60[, 2:ncol(VIMCFX96_60)],
    llul = c(1, 10)), CV = FALSE)

# plot(MFIaggr(VIMCFX96_60[, 1], VIMCFX96_60[,
# 2:ncol(VIMCFX96_60)], llul = c(1,40)), CV = FALSE)
```

6.1 Parameters of *MFIaggr*

MFIaggr is a powerful analytical and graphical tool for fast multiple comparison of the cycle dependent signal dispersion and distribution. This function enables the analysis of specific parts of the curve data as defined by the *llul* parameter. *llul* defines the lower and upper data limit (cycles) for a region of interest (ROI). The function returns an object with the columns "Cycle", "Location" (Mean, Median), "Deviation" (Standard Deviation, Median Absolute Deviation) and "Coefficient of Variation". Using the option $rob = TRUE$ the median and the median absolute deviation (MAD) are calculated instead of the mean and standard deviation. *MFIaggr* has the parameter *llul* to define the lower and upper data limit (cycle) for a ROI. Invoked by *@stats* reports *MFIaggr* further information such as inter quartile range (IQR), medcouple (robust measure of skewness), skewness (Pearson's second skewness coefficient), signal-to-noise ratio (SNR), variance-to-mean ratio (VRM), number of missing values (NAs), results from a linear fit of the ROI (intercept, slope, R squared) and the Breusch–Pagan test to test for heteroscedasticity in a linear regression model.

In our example we analyzed the raw fluorescence from 96 replicates (”VIMCFX96_60” data set) of a qPCR experiment for the human gene *Vimentin*. The *MFIaggr* plot shows that the analysis of all cycles is non-normal distributed (Figure S7).

```r
plot(MFIaggr(VIMCFX96_60[, 1], VIMCFX96_60[, 2:ncol(VIMCFX96_60)],
    llul = c(1, 40)), CV = FALSE)
```

MFIaggr can be used to analyze the heteroskedasticity in a given data set. Heteroskedasticity ("hetero" = different, "skedasis" = dispersion) is present if the variance (error term) is not constant. In case the variance

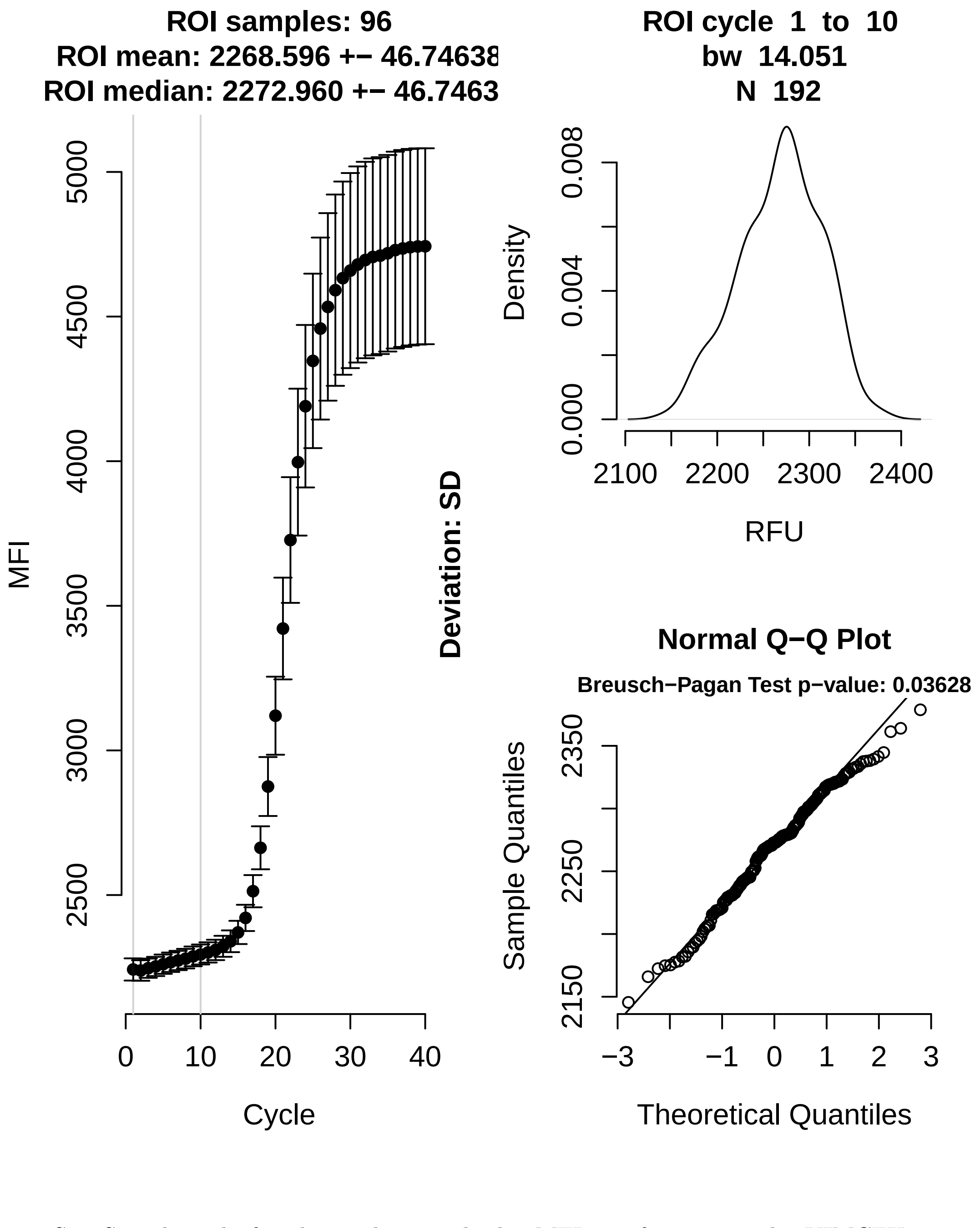
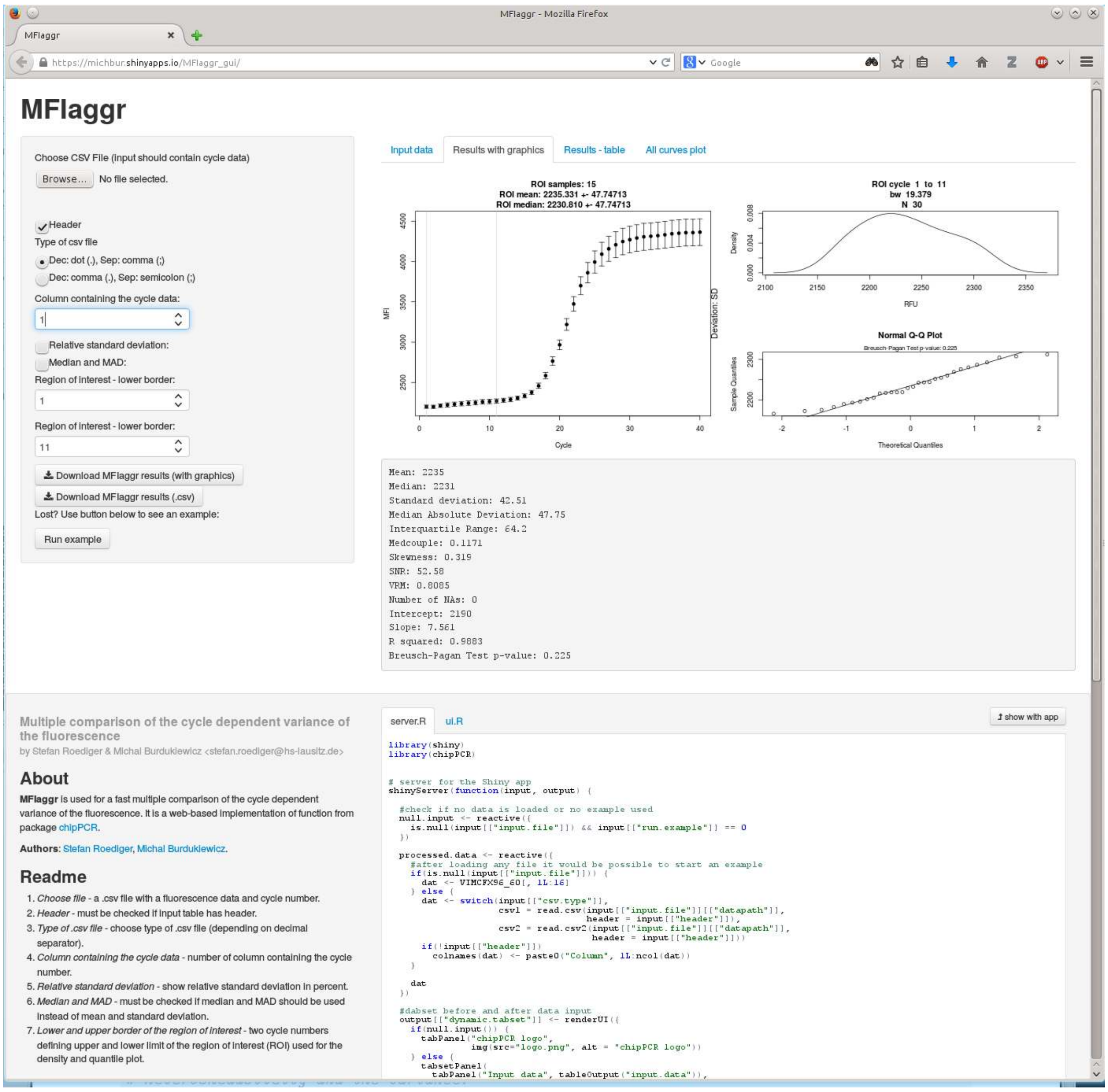


Figure S5: Sample code for the analysis with the *MFIaggr* function. The VIMCFX96_60 data set (96-well plate cycler (Bio–Rad CFX96)) was used. Either all a subset of cycles (ROI: 1 – 10) or all cycles (ROI: 1 – 40) (Figure S7) were analyzed. The density plot (right upper panel) and quantile-quantile analysis (right lower panel) show no normal distribution. Due to the sigmoidal curve structure is the density function bimodal.



MFlaggr

Choose CSV File (input should contain cycle data)

Browse... No file selected.

[✓] Header

Type of csv file

[● Dec: dot (.), Sep: comma (;)]

Dec: comma (.), Sep: semicolon (;)

Column containing the cycle data: 1

Relative standard deviation:

Median and MAD:

Region of interest - lower border: 1

Region of interest - lower border: 11

Download MFlaggr results (with graphics)

Download MFlaggr results (.csv)

Lost? Use button below to see an example:

Run example

```
Mean: 2235
Median: 2231
Standard deviation: 42.51
Median Absolute Deviation: 47.75
Interquartile Range: 64.2
Medcouple: 0.1171
Skewness: 0.319
SNR: 52.58
VRM: 0.8085
Number of NAs: 0
Intercept: 2190
Slope: 7.561
R squared: 0.9883
Breusch-Pagan Test p-value: 0.225
```

Multiple comparison of the cycle dependent variance of the fluorescence

by Stefan Roediger & Michal Burdukiewicz <stefan.roediger@hs-lausitz.de>

About

MFlaggr is used for a fast multiple comparison of the cycle dependent variance of the fluorescence. It is a web-based implementation of function from package chipPCR.

Authors: Stefan Roediger, Michal Burdukiewicz.

Readme

1. *Choose file* - a .csv file with a fluorescence data and cycle number.
2. *Header* - must be checked if input table has header.
3. *Type of .csv file* - choose type of .csv file (depending on decimal separator).
4. *Column containing the cycle data* - number of column containing the cycle number.
5. *Relative standard deviation* - show relative standard deviation in percent.
6. *Median and MAD* - must be checked if median and MAD should be used instead of mean and standard deviation.
7. *Lower and upper border of the region of interest* - two cycle numbers defining upper and lower limit of the region of interest (ROI) used for the density and quantile plot.

```
library(shiny)
library(chipPCR)


# server for the Shiny app
shinyServer(function(input, output) {

  #check if no data is loaded or no example used
  null.input <- reactive({
    is.null(input[["input.file"]]) && input[["run.example"]] == 0
  })

  processed.data <- reactive({
    #after loading any file it would be possible to start an example
    if(is.null(input[["input.file"]])) {
      dat <- VIMCFX96_60[, 1L:16]
    } else {
      dat <- switch(input[["csv.type"]],
                    csv1 = read.csv(input[["input.file"]][["datapath"]],
                                    header = input[["header"]]),
                    csv2 = read.csv2(input[["input.file"]][["datapath"]],
                                     header = input[["header"]]))
      if(!input[["header"]])
        colnames(dat) <- paste0("Column", 1L:ncol(dat))
    }

    dat
  })

  #dabset before and after data input
  output[["dynamic.tabset"]] <- renderUI({
    if(null.input()) {
      tabPanel("chipPCR logo",
               img(src="logo.png", alt = "chipPCR logo"))
    } else {
      tabsetPanel(
        tabPanel("Input data", tableOutput("input.data")),
```

Figure S6: Example of *shiny MFIaggr.gui* app. *(Top)* The plot of the *AmpSim.gui* is shown in a standard browser (**Iceweasel**, v. 32.0) along with the parameters (left panel). The code ("server.R", "ui.R") of the *shiny* app is also shown. Most parameters of the *MFIaggr.gui* function are accessible.

is constant data are considered to be homoskedastic. If the error terms do not have constant variance, they are said to be homoskedastic. Analysis of the heteroskedasticity can give some insight into the characteristics of a system. In the following example we compared the VIMCFX96_60 and VIMCFX96_69 data sets. Both data set were obtained from the same qPCR run in a Bio-Rad CFX96. However, data from the VIMCFX96_60 data set were obtained during the annealing phase at 60 degree Celsius and during the elongation phase at 69 °C. The heteroskedasticity increased expectedly during the amplification reaction. The variance in the elongation phase (Figure S7C and D) was lower than in the annealing phase (Figure S7A and B). The heteroskedasticity was significant in during the (Figure S7A) first 15 cycles at 60 °C.

```
par(mfrow = c(2, 2), bty = "n")
# Create a helper function 'hsk.test' to analyze the
# heteroskedasticity and the variance.
hsk.test <- function(x, y, llul = c(1, 15), main = "") {
    res <- MFIaggr(x, y, llul = llul)
    head(res)
    plot(res[, 1], res[, 3]^2, xlab = "Cycle", ylab = "Variance of refMFI",
        xlim = llul, ylim = c(min(res[llul[1]:llul[2], 3]^2),
            max(res[llul[1]:llul[2], 3]^2)), main = main, pch = 19,
```

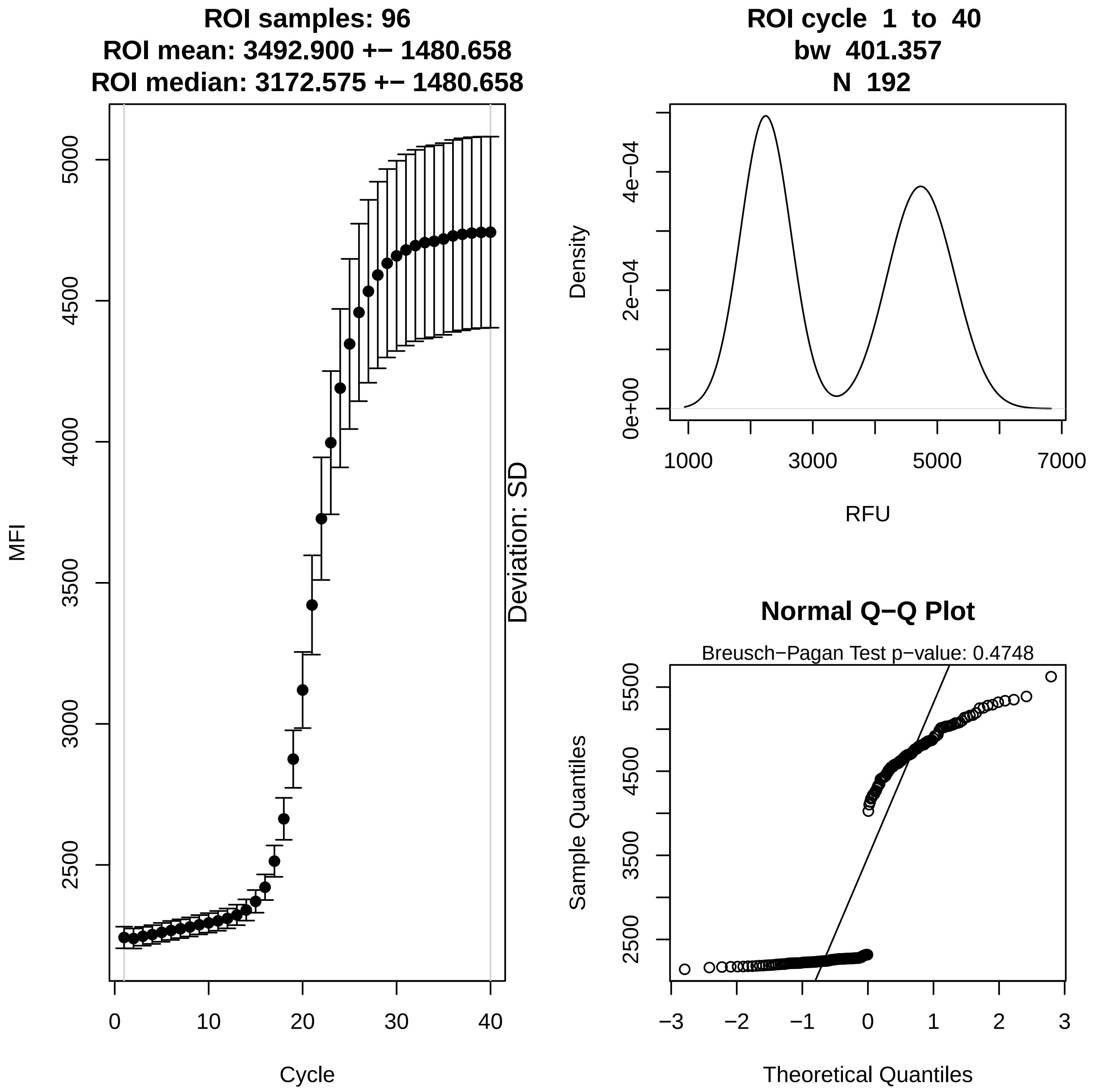


Figure S7: Signal analysis using the VIMCFX96_60 data set (96-well plate cycler (Bio-Rad CFX96)). All cycles (ROI: 1 − 40) were analyzed by the *MFIaggr* function. The density plot (right upper panel) and quantile-quantile analysis (right lower panel) show no normal distribution. Due to the sigmoidal curve structure is the density function bimodal.

```r
        type = "b")
    abline(v = llul, col = "grey", lty = 2, lwd = 2)
    legend("top", paste0("Breusch-Pagan test p-value: \n", format(summary(res,
        print = FALSE)[14], digits = 2)), bty = "n")
}

hsk.test(VIMCFX96_60[, 1], VIMCFX96_60[, 2:ncol(VIMCFX96_60)],
    llul = c(1, 15), main = "ROI Cycle 1 to 15\nAnnealing phase")
mtext("A", cex = 2, side = 3, adj = 0)

hsk.test(VIMCFX96_60[, 1], VIMCFX96_60[, 2:ncol(VIMCFX96_60)],
    llul = c(1, 40), main = "ROI Cycle 1 to 40\nAnnealing phase")
mtext("B", cex = 2, side = 3, adj = 0)

hsk.test(VIMCFX96_69[, 1], VIMCFX96_69[, 2:ncol(VIMCFX96_69)],
    llul = c(1, 15), main = "ROI Cycle 1 to 15\nElongation phase")
mtext("C", cex = 2, side = 3, adj = 0)

hsk.test(VIMCFX96_69[, 1], VIMCFX96_69[, 2:ncol(VIMCFX96_69)],
    llul = c(1, 40), main = "ROI Cycle 1 to 40\nElongation phase")
mtext("D", cex = 2, side = 3, adj = 0)
```

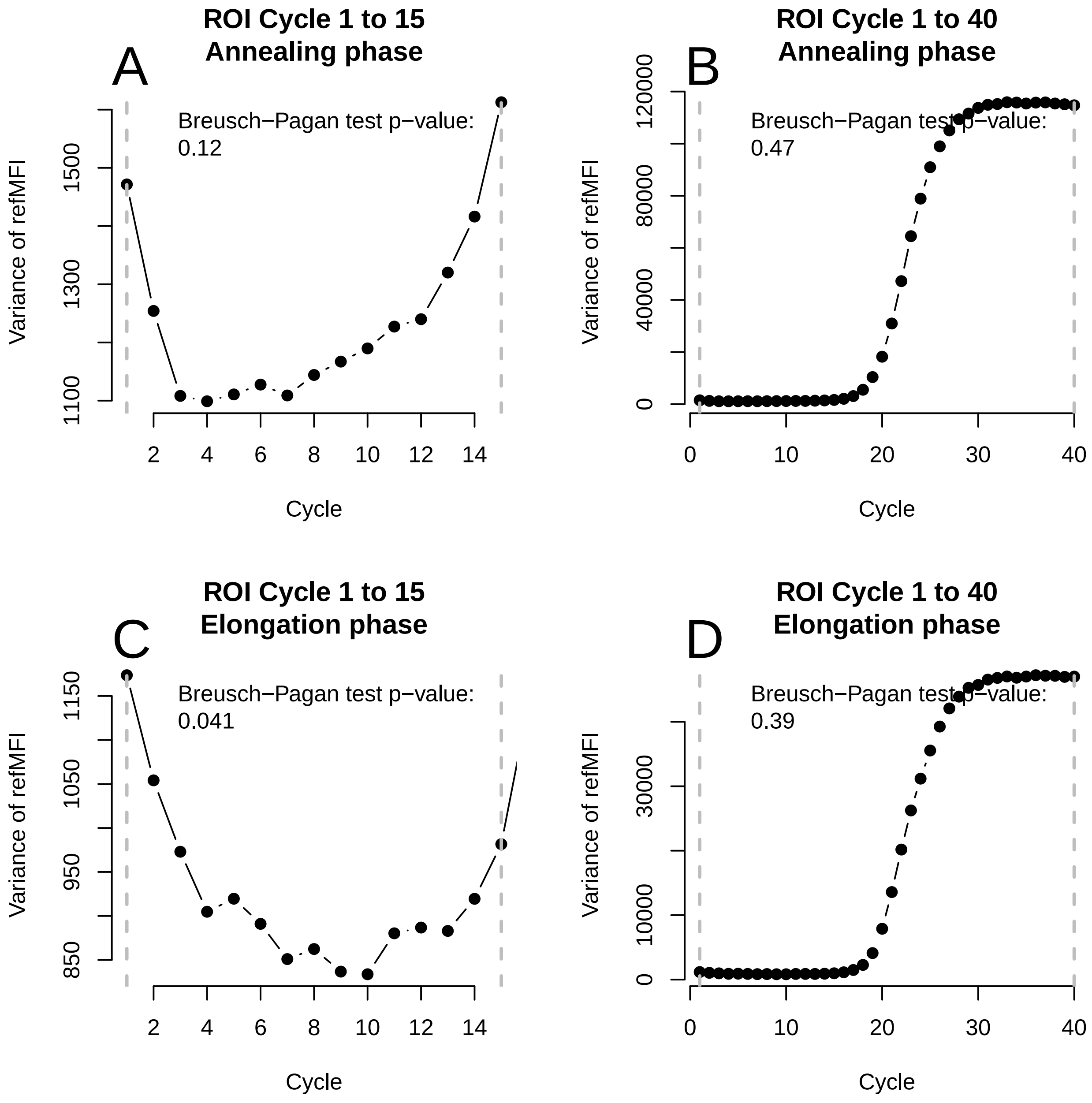


Figure S8: Use of *MFIaggr* to test for heteroskedasticity using the Breusch-Pagan test. The data were aggregated with the *MFIaggr* function and assigned to the object res. The standard deviation was transformed to the variance. The plot shows the cycle dependent variance measured at 60 degree Celsius (annealing phase; A, B) and 69 degree Celsius (elongation phase, C, D). First cycles 1 to 10 of 96 qPCR replicate amplification curves were analyzed. Next the cycles 1 to 40 of the same amplification curve data were analyzed. The Breusch-Pagan confirmed the heteroskedasticity in the amplification curve data. The VIMCFX96_60 and VIMCFX96_69 data sets were used.

6.2 Data overview - *plotCurves*

plotCurves visualizes many curves on one plot in separate cells allowing quick experiment assessment (Figure S9). In addition, *plotCurves* has an option to run an unsupervised *CPP* pre-processing step on the raw data. This will smooth the data (Savitzky-Golay Smoothing), remove missing values (spline interpolation by default) and perform a background subtraction (base-lining to zero). *plotCurves* has a colored indicator for rapid visualization of dataset with potentially problematic amplification curves. The output of the plot is arranged in an orthogonal matrix.

Warnings in following code chunks were suppressed.

```
y <- VIMCFX96_60[, 2L:9]
# Introduce some missing values.
y[c(10, 22, 3, 25, 26, 15, 27, 23, 4), c(5, 7, 4, 2, 1)] <- NA

# Show plot with raw data and missing values (black line) and
# show plots with pre-processed data and imputed missing
# values (red line).
plotCurves(VIMCFX96_60[, 1], y, nrow = 2, type = "l", CPP = TRUE)
```

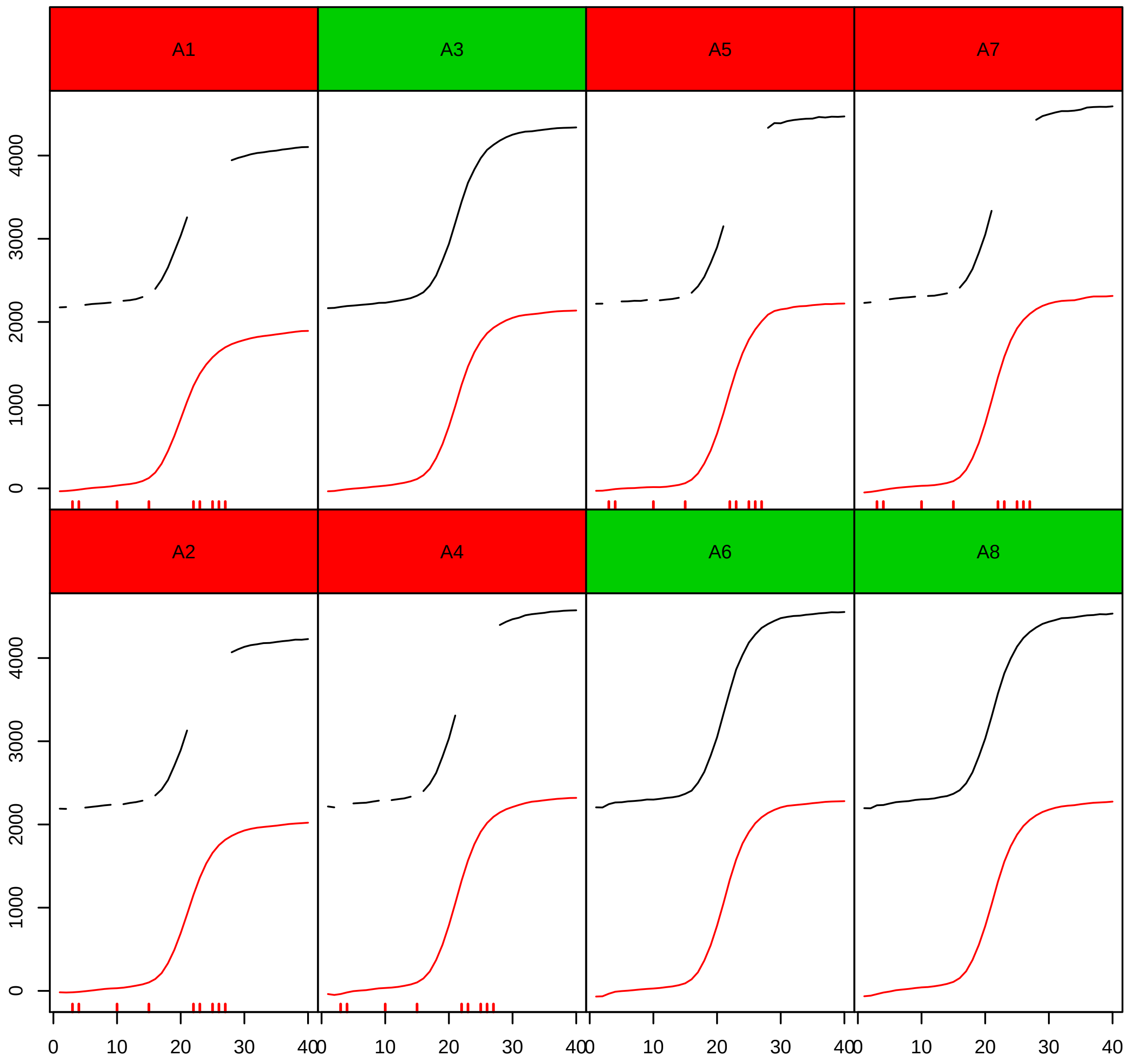


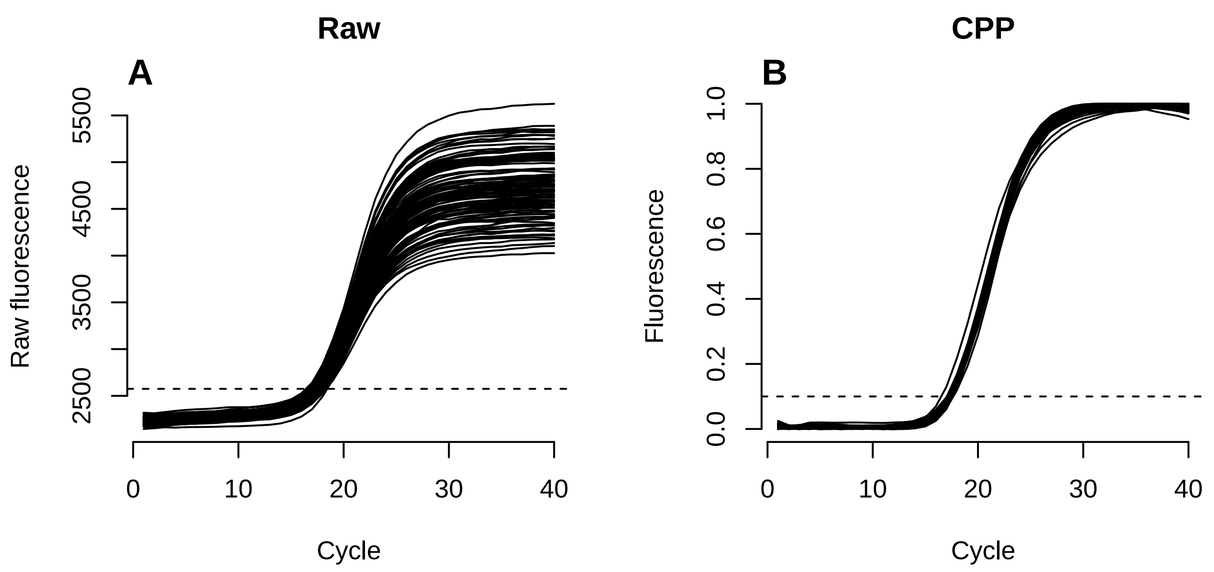
Figure S9: The plotCurves function. Plots many curves on one plot in separate cells allowing quick assessment. Missing values were artificially introduced at random position to selected curves of the VIMCFX96_60 data set (solid black line). A colored box (topleft of each plot) indicates the sample name and if the data contain missing values. The red rug indicates the position of the missing values. The red lined shows the amplification curve after unsupervised pre-processing (using an instance of *CPP*).

7 Proposed workflow

In the previous section we showed different methods to investigate specific properties of the measured data. Next, we focus on methods of the *chipPCR* package to pre-process the data. We have chosen to state function names and give some information on the working principle. However, details will be explained in the subsequent sections to avoid confusion of the reader. In particular, we show the application of the *CPP* function, which can be seen as a proposed workflow for customized pre-processor functions. Data were taken from the VIMCFX96_60 data set. This data set was measured with a Bio-Rad CFX96 thermo-cycler with 96 replicates (see *chipPCR* manual for experiential details).

The function *CPP* encompasses a set of functions to pre-process an amplification curve. The pre-processing includes options to normalize curve data, to remove background, to remove outliers (*fixNA* function) in the background range and to test if an amplification is significant. The function uses the *bg.max* function to estimate automatically the start of the amplification process. In the background range there is often noise which makes it harder to determine a meaningful background value. Therefore, *CPP* can optionally remove outliers by finding the value with the smallest and largest difference from the mean as provided by the *rm.outlier* function from the *outlier* package [19]. *rm.outlier* detects these outliers by a simple rule without statistical testing and replaces it by the sample mean. Outliers herein referrers to the smallest and largest value, which has maximum difference from the sample mean. The slope of the background range is often unequal to zero. By setting the parameter *trans* it is possible to apply a simple correction of the slope. Thereby either a robust linear regression by computing MM-type regression estimators, a nonparametric rank-based estimator or a standard linear regression model. *CPP* uses by default a robust linear regression (MM-type estimator) as integrated in the *lm.coefs* function. A defined range of the amplification curve (typically the background range) is used to extrapolate the linear trend over the entire data set. However, this step his to be performed with caution since this operation effects the amplification efficiency. The background is assumed to be constant for the entire measurement. Care is also needed when using *trans* with time series (see *lm* from the *stats* package for details). In addition, all data are normalized between there minimum and maximum. This is taken care of by the *normalizer* function. Smoothing of the data is finally done based on an instance of the *smoother* function. By default, a Savitsky-Golay filter was used to smooth the data. The following code is a representative example for the use of *CPP* (Figure S10). Warnings in following code chunks were suppressed.

```r
layout(matrix(c(1, 2, 3, 3), 2, 2, byrow = TRUE), respect = TRUE)

par(las = 0, bty = "n", oma = c(0.5, 0.5, 0.5, 0.5))

th.cyc.raw <- apply(VIMCFX96_60[, -1], 2, function(i) {
    th.cyc(VIMCFX96_60[, 1], i, r = 2575)[1, 1]
})

res.CPP <- apply(VIMCFX96_60[, -1], 2, function(i) {
    CPP(VIMCFX96_60[, 1], i, trans = TRUE, method.norm = "minm")[["y.norm"]]
})

th.cyc.CPP <- apply(res.CPP, 2, function(i) {
    th.cyc(VIMCFX96_60[, 1], i, r = 0.1)[1, 1]
})

matplot(VIMCFX96_60[, -1], type = "l", pch = 19, col = 1, lty = 1,
    xlab = "Cycle", ylab = "Raw fluorescence", main = "Raw")
abline(h = 2575, lty = 2)
mtext("A", cex = 1.2, side = 3, adj = 0, font = 2)

matplot(res.CPP, type = "l", pch = 19, col = 1, lty = 1, xlab = "Cycle",
    ylab = "Fluorescence", main = "CPP")
abline(h = 0.1, lty = 2)
mtext("B", cex = 1.2, side = 3, adj = 0, font = 2)

boxplot(data.frame(Raw = th.cyc.raw, CPP = th.cyc.CPP), ylab = "Cq (Ct)",
    notch = TRUE)
mtext("C", cex = 1.2, side = 3, adj = 0, font = 2)
```



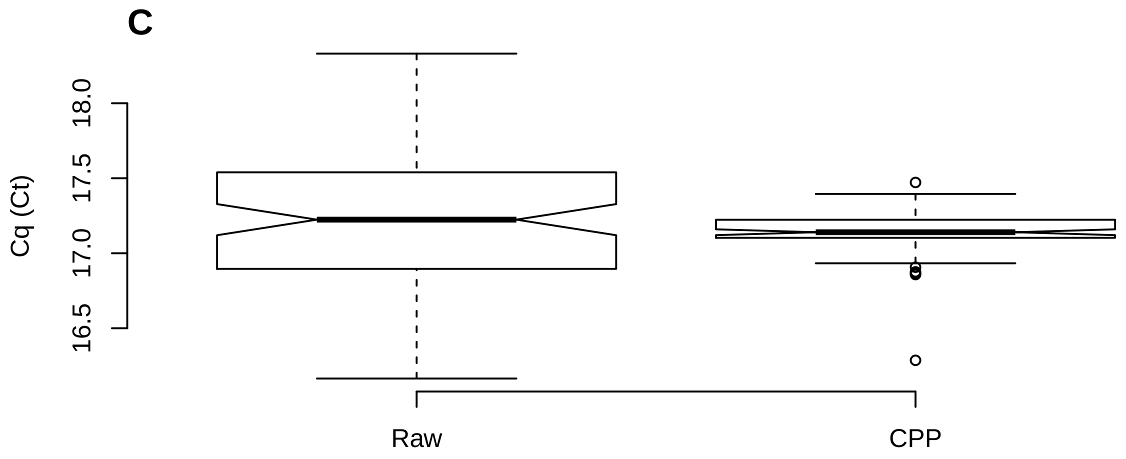


Figure S10: Application of the *CPP* and *th.cyc* functions. **A)** The raw data of the VIMCFX96_60 data set were plotted without pre-processing. **B)** All amplification curve data were pre-processed with the CPP function. The parameter *trans* was set to $TRUE$, which lead to a linear trend correction and base-lining. By default a Savitzky-Golay filter was used to smooth the data. The data were normalized between 0 and 1 ($method.norm =' minm'$). **C)** All Cqs were calculated with *th.cyc* function. The Cq for the raw data was 17.25 ± 0.5 (at $r = 2575$) and 17.1 ± 0.1 (at $r = 0.1$) for the pre-processed data. Our results indicate that the dispersion of the Cq values was slightly lower.

8 Imputation of missing values in amplification curve data - *fixNA*

Experimental technologies may produce missing values (NA) at random due to sensor drop-outs or other technical difficulties. Many analytical functions stop to progress or discard entire data sets. This behavior is rational for unknown data structures. However, in case of amplification curve data it is justified to impute NAs because the structure generally resembles an S-shaped curve. Standard approaches include substitution with most frequent values, mean value imputation, last value carried forward, bootstrapping, or substitution by correlation with replicate measurements [13]. In case of amplification curves other approaches are favorable. Particularly, the transitions phases (e.g., background phase to exponential phase) is potentially prone to bias.

The NAs may be caused by detector problems, acquisition error or other assorted problems. There are different ways to handle missing values. One approach is to ignore NAs, which is generally acceptable. However, in case of further calculation it is often necessary to handle cases of missing values in a way that the next calculation steps can be performed. Missing values can be eliminated by a imputation, which encompasses various approaches. This includes to calculate a location parameter (e.g., mean, median) or other significant values (e.g., minimum, maximum, modus) of a data column. However, in non-linear processes such as amplification processes its is reasonable to estimate the missing values from a trend.

The function *fixNA* imputes missing values in a single column of data (response). The imputation is based either on a linear approximation or an approximation by cubic splines (default) (Figure S12). Other smoothing functions such as the Savitzky-Golay smoothing filter have the intrinsic capability to remove missing values [9, 43]. However, such functionality was not yet implemented. This linear approach is useful but may be problematic on the phases other then background or plateau phases of an amplification reaction. The parameter spline on *fixNA* enables a trend estimation on cubic splines and may be more appropriate in most scenarios. The function *fixNA* imputes missing values in a single column of data. The imputation is based on a linear approximation by default. However, the data can also be estimated from an approximation by splines.

We used the "reps384" data set from the *qpcR* package [46] to compare the influence of imputation on real-world data. The "reps384" data set consists of 379 replicate amplification curves (see documentation of the *qpcR* package for details). Our in-silico experiment was designed as followed: Either one or three missing values were artificially to each amplification curve at random positions. We separated the amplification curve in three fixed regions ("Linear phase" (cycle 1 − 10), "Exponential phase" (cycle 11 − 34), "Plateau phase" (cycle 34 − 40), Figure S11) and investigated the impact on the pPCR parameters "Cq (SDM, Cy0)" and curve background parameter "bg". The performance of the imputed models was analyzed the goodness-of-fit in this region by the commonly normalized used root-mean-squared-error (NRMSE). The smaller the NRMSE the better the imputation is. Using this approach we compared the imputation by "linear approximation" ($fixNA(x, y, spline = FALSE)$) and "spline approximation" ($fixNA(x, y, spline = TRUE)$). Our results show that the imputation with the spline method worked reliably and introduced no significant bias to all investigated parameters. To compare the linear and spline-based imputation we performed a Kruskal-Wallis rank sum test (*kruskal.test*, *stats* package [29]).

Our in-silico experiments showed that cubic spline interpolation yielded the most probable values and therefore led to the least effect on tested statistical parameters (Cq, background signal, Pearson correlation coefficient) the exponential phase and is therefore the recommended approach to remove missing values (Figure S11). We observed no significant bias by cubic spline interpolation (Table S2). The performance of *fixNA* using cubic splines was better than a linear interpolation (Figure S12). However, the linear approximation might be applicable in measurements with high sample rates (e.g., isothermal amplification) (not shown). Any method requires a minimum number of data points as foundation for a meaningful imputation. *fixNA* attempts to take care of such pitfalls. By rule of thumbs we determined that the number of missing elements in relation to the total number of elements. In case more than 30 % of all values are NAs gives *fixNA* a warning.

Our results for the given experiential setting support following statements. (I) The imputation of missing values by spline interpolation and liner methods introduce no significant bias on the tested parameters "Cq (SDM, Cy0)", "bg" and the accompanied quality measure NRMSE (Table S2). (II) We found that the difference between the linear and spline imputation is negligible (p 1).

```r
library(qpcR)
library(chipPCR)
cols <- adjustcolor(2:4, 0.6)
plot(NA, NA, xlim = c(1, 45), ylim = c(min(reps384[, -1]), max(reps384[,
    -1])), col = 1, pch = 19, type = "b", xlab = "Cycle", ylab = "Fluorescence")
rect(0.8, min(reps384[, -1]), 10.2, max(reps384[, -1]), border = NA,
    col = cols[1])
rect(10.8, min(reps384[, -1]), 33.2, max(reps384[, -1]), border = NA,
    col = cols[2])
rect(33.8, min(reps384[, -1]), 45, max(reps384[, -1]), border = NA,
    col = cols[3])

apply(reps384[, -1], 2, function(i) lines(reps384[, 1], i))

## NULL
```

	Background - raw data	Background - fixed NA	cpD2 - raw data	cpD2 - fixed NA
Linear phase - 1 NA	4606.43 ± 186	4606.53 ± 186	18.20 ± 0.145	18.20 ± 0.145
Exponential phase - 1 NA	7555.31 ± 468	7555.28 ± 468	18.20 ± 0.145	18.20 ± 0.145
Plateau phase - 1 NA	11736.60 ± 1032	11736.82 ± 1032	18.20 ± 0.145	18.20 ± 0.145
Linear phase - 3 NA	4606.43 ± 186	4607.05 ± 186	18.20 ± 0.145	18.20 ± 0.145
Exponential phase - 3 NA	7555.31 ± 468	7555.15 ± 468	18.20 ± 0.145	18.19 ± 0.148
Plateau phase - 3 NA	11736.60 ± 1032	11736.86 ± 1033	18.20 ± 0.145	18.20 ± 0.145

	Cy0 - raw data	Cy0 - fixed NA	NRMSE
Linear phase - 1 NA	11.73 ± 1.06	11.73 ± 1.06	0.00012 ± 0.000133
Exponential phase - 1 NA	11.73 ± 1.06	11.73 ± 1.06	0.00018 ± 0.000176
Plateau phase - 1 NA	11.73 ± 1.06	11.73 ± 1.06	0.00026 ± 0.000237
Linear phase - 3 NA	11.73 ± 1.06	11.73 ± 1.06	0.00033 ± 0.00031
Exponential phase - 3 NA	11.73 ± 1.06	11.71 ± 1.06	0.00078 ± 0.000601
Plateau phase - 3 NA	11.73 ± 1.06	11.73 ± 1.06	0.00072 ± 0.000475

Table S2: Results of fixNA data imputation. $1NA$ (= one) or $3NA$ (= three) missing value per amplification curve, respectively.

```
# Simulation of an ideal amplification curve with 40 cycles
# The other parameter of the AmpSim function are identical to
# the default.

res <- AmpSim(cyc = 1:40)

# Introduce a missing value (cycle 18) in the transition
# between the background and the exponential phase.

res.NA <- res
res.NA[18, 2] <- NA

# Helper function to highlight the position of the missing
# value.
abliner <- function(x1 = 17.5, x2 = 18.5, y1 = 0.09, y2 = 0.14) {
    abline(v = c(x1, x2), col = "red")
    abline(h = c(y1, y2), col = "red")
}

par(las = 0, mfrow = c(2, 2), bty = "n")
plot(res, xlab = "Cycles", ylab = "refMFI", type = "b", pch = 20,
    main = "Without NA")
abliner()
mtext("A", cex = 1.2, side = 3, adj = 0, font = 2)
res.NA.linear <- fixNA(res.NA[, 1], res.NA[, 2], spline = FALSE,
    verbose = FALSE)

plot(res.NA, xlab = "Cycles", ylab = "refMFI", type = "b", pch = 20,
    main = "With NA during transition")
abliner()
mtext("B", cex = 1.2, side = 3, adj = 0, font = 2)

res.NA.spline <- fixNA(res.NA[, 1], res.NA[, 2], spline = TRUE,
    verbose = FALSE)
```

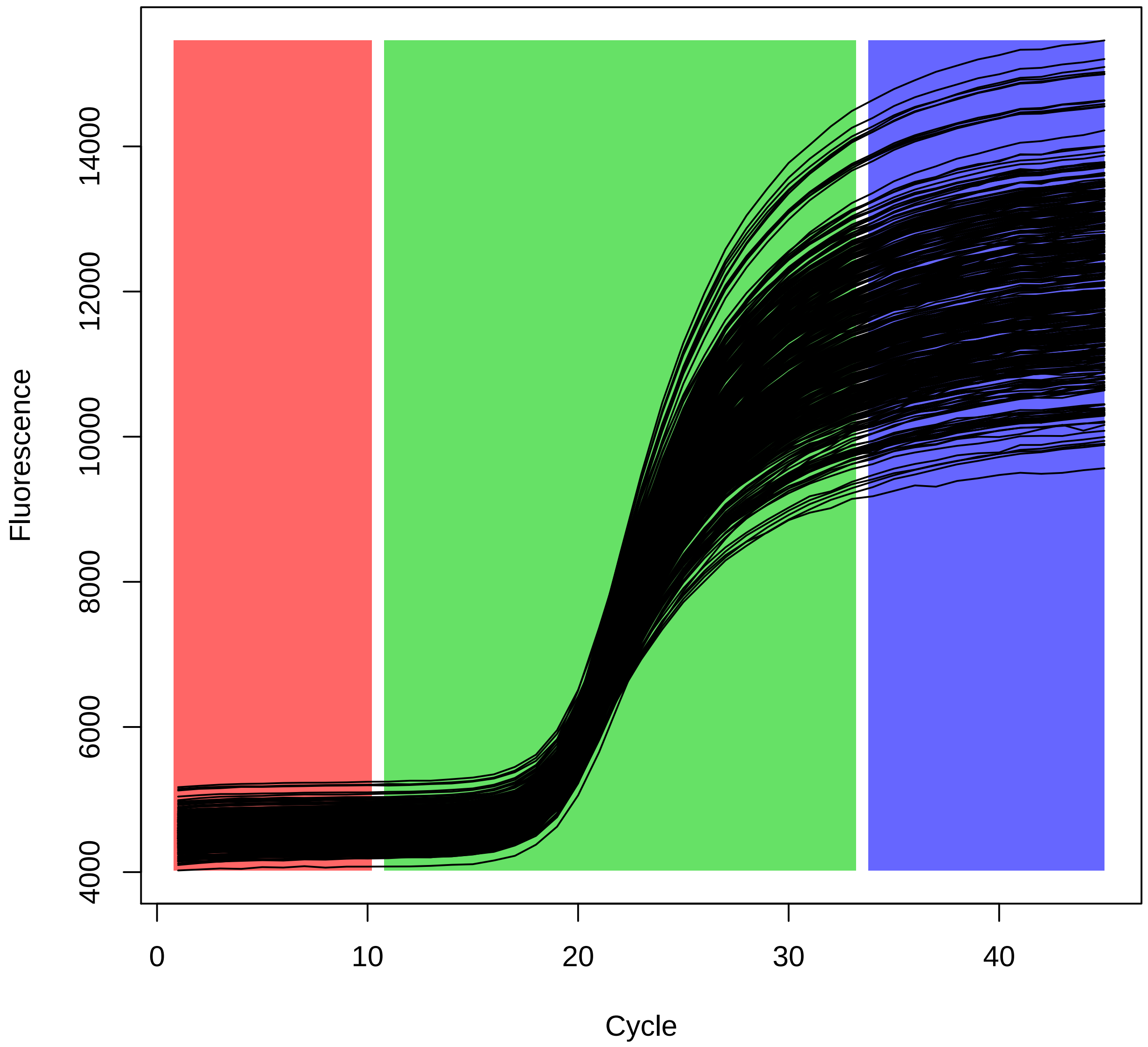


Figure S11: Inspection of the reps384 data set. The reps384 data set was used for the analysis of the impact of imputed missing values. Three areas of the curve data were defined as “Linear phase” (red, cycle 1 – 10), “Exponential phase” (blue, cycle 11 – 33), “Plateau phase” (green, cycle 34 – 40).

```r
plot(res.NA.linear, xlab = "Cycles", ylab = "refMFI", type = "b",
    pch = 20, main = "Linear imputed\n NA")
abliner()
mtext("C", cex = 1.2, side = 3, adj = 0, font = 2)

plot(res.NA.spline, xlab = "Cycles", ylab = "refMFI", type = "b",
    pch = 20, main = "Spline imputed\n NA")
abliner()
mtext("D", cex = 1.2, side = 3, adj = 0, font = 2)
par(mfrow = c(1, 1))
```

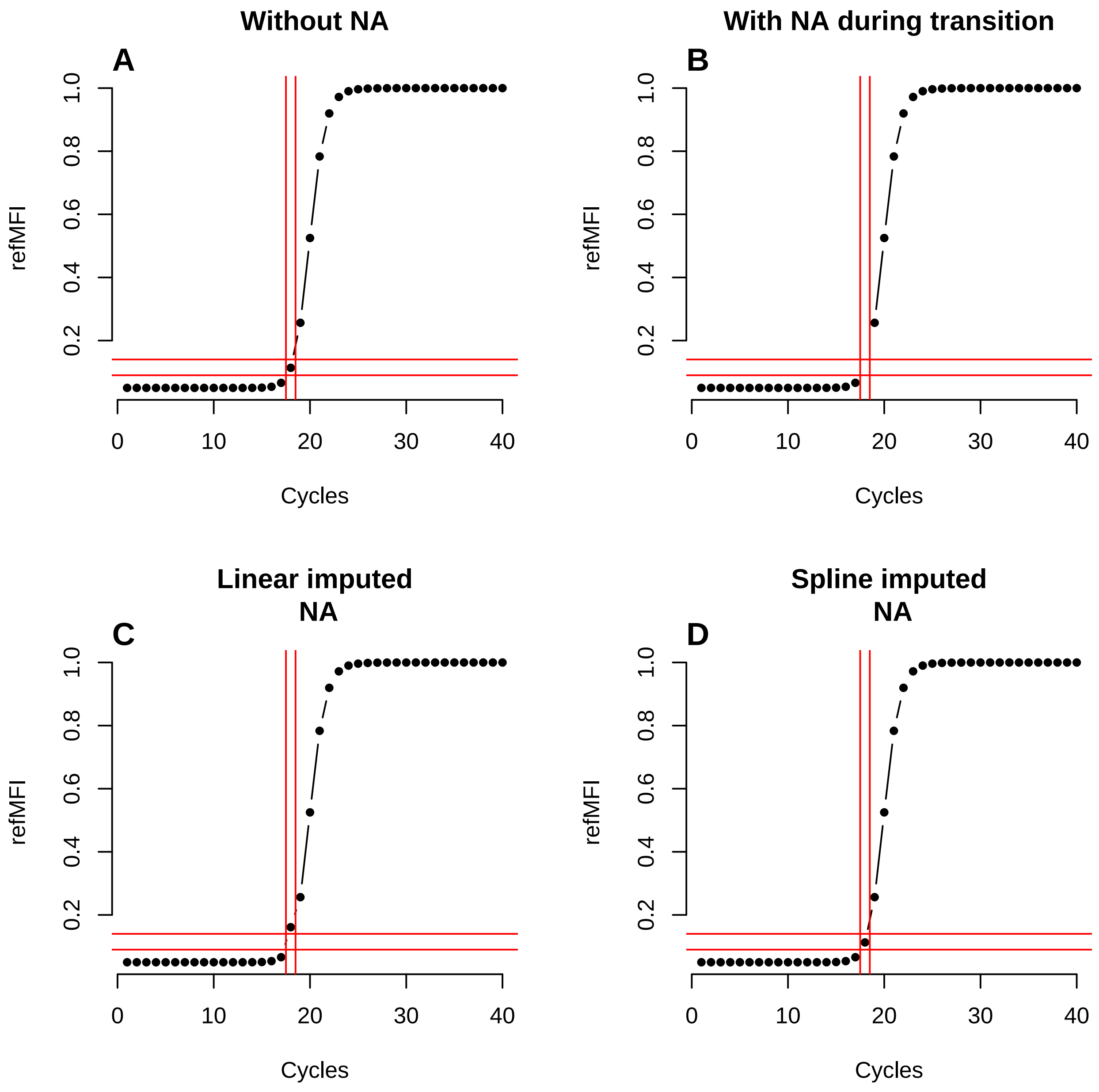


Figure S12: Imputation of missing values in amplification curve data. *(A)* Raw data were generated using the *AmpSim* simulation function. *(B)* A missing value was introduced in the transition phase. The missing value was imputed either by *(C)* linear approximation or *(D)* a cubic spline approximation. The spline approximation nearly reconstituted the original curve.

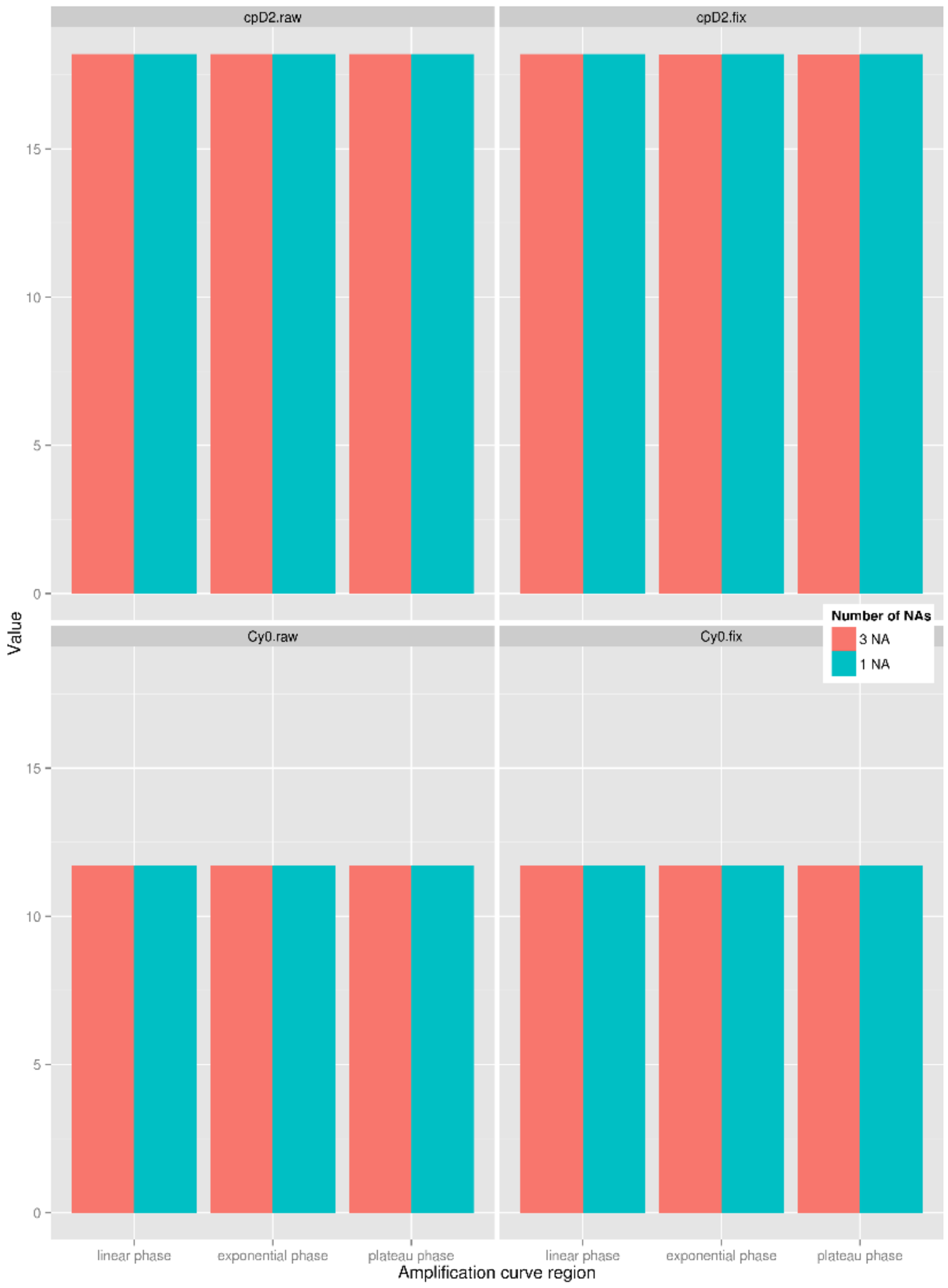


Figure S13: Test of the *fixNA* function using the spline and linear imputation for qPCR data with missing values. Missing values were artificially introduced into the "reps384" data set from the *qpcR* package. We found no significant difference between raw data and data with imputed missing values. R^2 and correlation coefficients of curves were close to 1 with p-value $< 10^{-6}$

9 Smoothing and filtering

Amplification curve data of experimental thermo-cyclers may deliver results, which are hard to interpret due to noise. For data presentation it is often useful to smooth or filter the data. Smoothing and filtering are different approaches with a similar outcome. Both pre-process an input signal as output for subsequent analysis steps. Filtering uses methods of signal processing and takes a data input and apply a function to form an output. Smoothing in contrast uses statistical approaches, like local regression models (e.g., least squares estimate) or cubic splines. Therefore, we developed the *smoother* function, which is a wrapper for smoother functions and filters commonly used to process amplification curve data. *smoother* inherited traits (Table 9) of the parent functions. However, the functionality of *smoother* greatly outgrowths applications only in amplification curve analysis. Incorporating most of the best proven algorithms, we offer the user a powerful tool to access the methods while minimizing the drawback of learning syntax of specific functions. *smoother* was enhanced by functionality of *fixNA* and *CPP*. Figure S14 shows results of the *smoother* function an amplification curve data.

```r
# Simulate and amplification curve with the AmpSim function
tmp <- AmpSim(cyc = 1:35, bl = 0)

par(las = 0, bty = "n", cex.axis = 1.5, cex.lab = 1.5, font = 2,
    cex.main = 1.8, oma = c(1, 1, 1, 1), fig = c(0, 1, 0.55,
        1))
plot(tmp, type = "b", col = 1, pch = 20, xlab = "", ylab = "RFU",
    main = "Raw data")
mtext("A", cex = 2, side = 3, adj = 0, font = 2)

# Apply all (parameter method = 'all') smoothers/filter with
# the default setting to the amplification curve of the
# object tmp. Smoothers / Filters: Savitzky-Golay smoothing
# filter locally-weighted polynomial regression moving
# average, windowsize 3 cubic spline smooth standard cubic
# spline smooth Friedman's SuperSmoother weighted Whittaker
# smoothing with first order finite difference penalty
# weighted Whittaker smoothing with a second order finite
# difference penalty
res <- smoother(tmp[, 1], tmp[, 2], method = "all", CPP = FALSE)

# Calculate the difference between the ideal curve (tmp) and
# the smoothed curves (res) and assign the results to the
# object res.out
res.out <- cbind(cycle = tmp[, 1], tmp[, 2] - res)

# Plot the smoothed curves
par(fig = c(0, 1, 0, 0.65), new = TRUE)
plot(NA, NA, type = "b", col = 2, pch = 15, xlim = c(1, 35),
    ylim = c(-0.1, 0.1), xlab = "Cycle", ylab = "delta refMFI (raw - smoothed)",
    main = "Smoothed / Filtered data")

mtext("B", cex = 2, side = 3, adj = 0, font = 2)
legend(1.5, 0.1, ncol = 2, colnames(res.out[, 2:9]), pch = 15:22,
    lwd = 2, col = c(2:9))

# Plot the results.
tmp <- sapply(2:9, function(i) {
    lines(res.out[, 1], res.out[, i], type = "b", col = i, pch = i +
        13)
})
```

Many functions (e.g., Savitsky-Golay filter) of *chipPCR* assume uniform (equally spaced) sampling. Therefore, it is recommended to pre-process the data to have equally spaced values. The function *smoother* and *CPP* (inherited from *smoother*) give a warning in such cases. The *smoother* function enables users to tune behavior of the chosen smoothing algorithm by using nearly all parameters available in called subroutines and at the same time uniforms input and output. It should be noted that smoothing may alter the curve shape and thus lead to

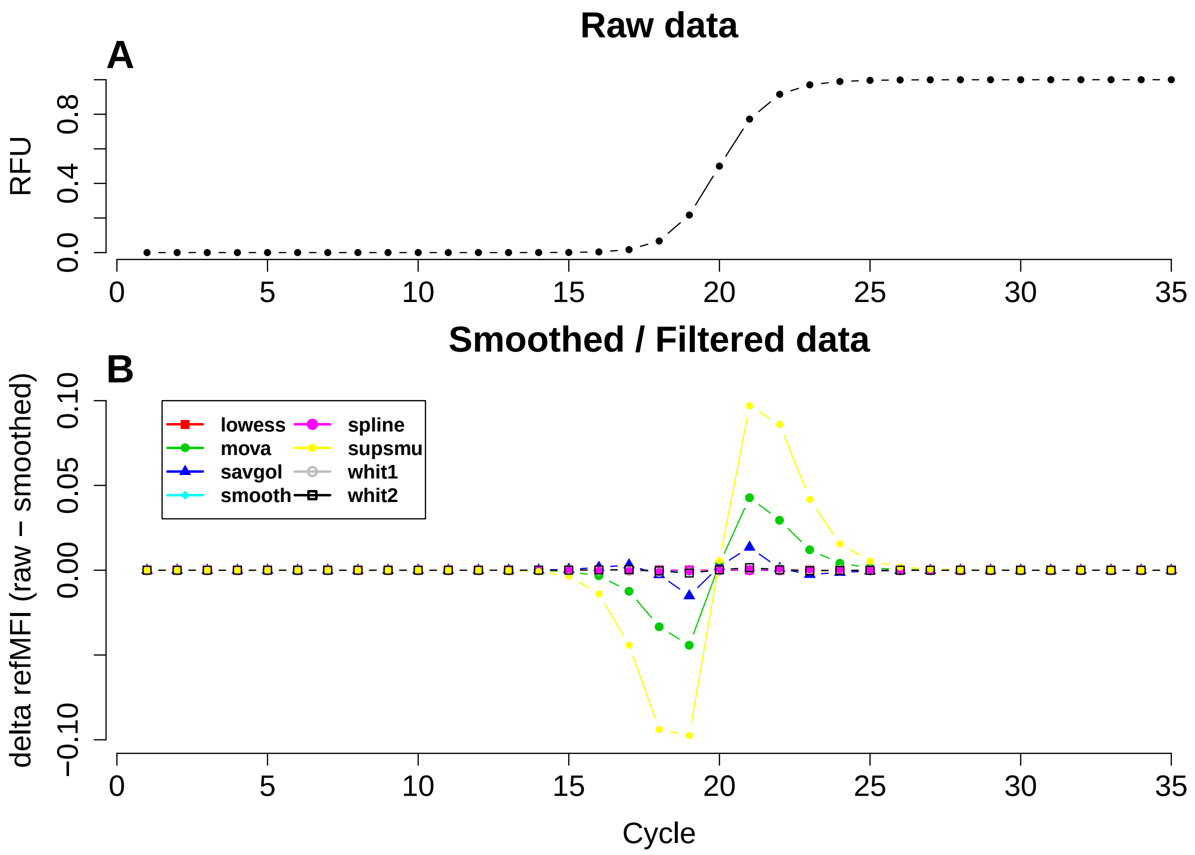


Figure S14: Smoother and filter methods of the *chipPCR* package. *(A)* Raw data were generated using the *AmpSim* simulation function. *(B)* The difference of the raw data to the smoothed data was plotted. “savgol” (Savitzky-Golay Smoothing), “lowess” (locally-weighted polynomial regression), “mova3” (moving average with window size of 3), “smooth” (cubic smoothing spline), “spline” (Interpolating cubic spline), “supsmu” (Friedman’s SuperSmoother), “whit1” (weighted Whittaker smoothing with a finite difference penalty of order 1), “whit2” (weighted Whittaker smoothing with a finite difference penalty of order 2). The “savgol”, “smooth”, “spline” “whit1” , and “whit2” nearly preserved the original curve. The other functions resulted in alteration in the transition phases of the amplification curve. Optimized time series smoother, like the Kalman filter [53], are not yet integrated.

artificial results. Smoothed data are easier to read but introduce a bias to the pre-processed data. Therefore, the prime use of smoothers is processing data for visualization purposes. However, it is not recommended to smooth signals unsupervised prior to statistical procedures (e.g., least-squares curve fitting). All smoothing algorithms are “lossy” to a certain extent and may change the curve shape significantly. In particular, the residual evaluation of a fit may lead to false prediction, because noise after smoothing may be mistaken for signal. Signals after curve smoothing can be used to locate peaks but it should cautiously be used to measure peaks.

Tables

10 *bg.max* - a function to estimate the start and end of an amplification reaction

The following paragraphs describe methods from the literature to detect the background range of amplification curves. Background range herein refers to a level of fluorescence measured before any specific amplification is detectable. The raw data (e.g., fluorescence intensity) measured after each step (cycle or time point) follow a non-linear progress. Currently none of them is implemented as **R** function. The easiest way to classify them is the extend of assumptions made before applying of a method.

The simplest approach is to treat the background fluorescence as a value constant during whole amplification reaction. In this case the noise could be approximated as the mean or median of fluorescence values in lag phase [10] or their standard deviations [27]. The more sophisticated way of approximating constant background fluorescence requires optimizing its value to achieve linearity of the model fit on the semi logarithmic plot in log-linear phase [10]. The later procedure is greatly enhanced by performing further computations only on a subset of consecutive measurements for which calculated efficiencies have the lowest variance. Other methods loosen the assumption that background fluorescence is a constant value and instead describe it as a function of the cycle number. For example the algorithm used in SoFar [55] fits a nonlinear saturation function to measurement points before the start of the exponential growth phase. Parameters of the saturation function are chosen to minimize the sum of squared residuals of the fitted function. Then the value of saturation function is calculated for all data points and subtracted from measured values giving corrected values of fluorescence, which are used in next calculations.

Some approaches make even less assumptions regarding the form of the background noise. The taking-difference linear regression method has a premise that changes of fluorescence between subsequent cycles are exclusively caused by the amplification of the product [31]. The corrected values are calculated by simply subtracting the fluorescence value in the former cycle from fluorescence in the latter. Of course in this case the real fluorescence value in first cycle is unknown, so the number of cycles that can be used in following computations is reduced by one.

The Real-Time PCR Miner algorithm is also nearly assumption-free ([58]). The principle is that background fluorescence is similar in the small groups of subsequent measurements. So the first step of the algorithm is division of subsequent measurement points belonging to the exponential phase of amplification in at least four-element groups. For each set of points is calculated a pair of the estimate of the efficiency and the significance of model representing relation between the fluorescence value and the cycle number. The estimates paired with the highest significance are the most influential in the computation of the final efficiency.

To find the beginning of the lag phase and end of plateau phase is important for the goodness-of-fit for both exponential-phase-only and S-shaped models. There are two strategies. The first narrows the area of the search to the neighborhood of their theoretical values determined by a fitted model of the amplification reaction. To this group belongs SoFar (Wilhelm et al. (2003) [55]). The algorithm looks for the start and the end of the exponential phase near the second derivatives of the function representing the relation between logarithm of the fluorescence and the cycle number. The available correction guarantees that the start of amplification has higher value than background noise. The very similar procedure is implemented in Real-Time PCR Miner [58], where background noise is also used as parameter in implemented models to calculate theoretical the start of the amplification process. The end of amplification process is detected by calculating the third derivative of implemented S-shaped model. The second approach does not require theoretical values. A very intuitive solution, designated take-off point, by Tichopad et al. (2003) [50] describes the lag phase using a linear function. Random deviations are taken into account as standardized residuals. The method starts with a fitting of a linear function to first three measurement points. If none of residuals is considered an outlier with a statistical test, the algorithm fits a new linear model to the first four measurement points and so on. The procedure stops when two last points are designated as outliers. The first of aforementioned outliers is considered the end of lag phase. It is worth noting that this algorithm is versatile enough to also detect the beginning of the plateau phase.

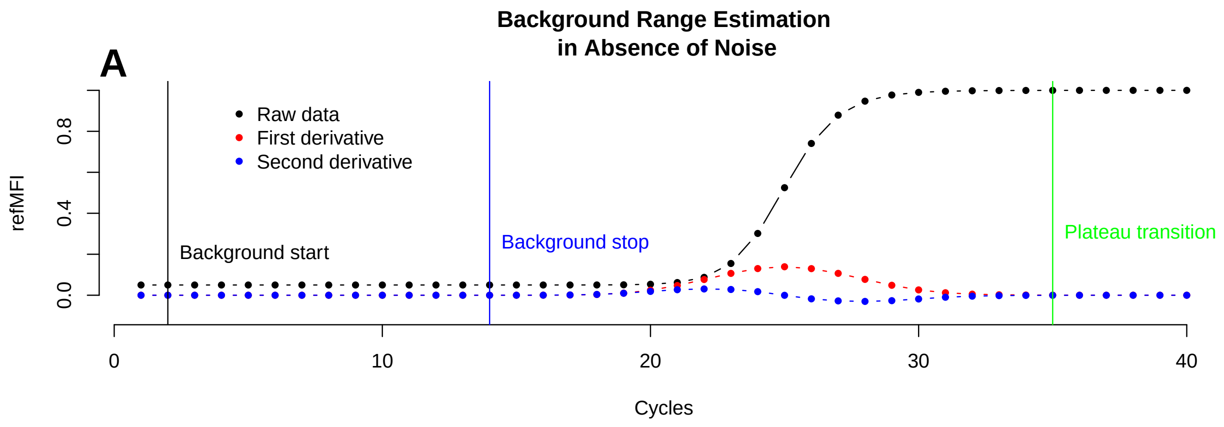
The algorithm of *bg.max* is based on the assumption that the signal difference of successive cycles in the linear ground phase is approximately constant. After transition in the early exponential phase the signal changes drastically. First data are smoothed by Friedman's 'super smoother' (as found in "supsmu". Thereof the approximate first and second derivative are calculated by a five-point stencil *inder*. The difference of cycles at the maxima of the first and second approximate derivative and a correction factor are used to estimate the range before the exponential phase. This simple function finds the background range without modeling the function. The start of the background range is defined be a "fixed" value. Since many signals tend to overshot in the first cycles a default value of 2 (for qPCR) is chosen. *bg.max* tries also to estimate the end of an amplification reaction (Figure S15). See section *bg.max* "Details" of the *chipPCR* manual for further details. Application of this function is for example a rational basis for trimming of unneeded data.

```
par(las = 0, mfrow = c(2, 1), bty = "n", oma = c(0.5, 0.5, 0.5,
    0.5))

res <- AmpSim(cyc = 1:40, Cq = 25)
plot(res, xlim = c(1, 40), ylim = c(-0.1, 1), xlab = "Cycles",
    ylab = "refMFI", main = "Background Range Estimation\n in Absence of Noise",
    type = "b", pch = 20)
background <- bg.max(res[, 1], res[, 2])
mtext("A", cex = 2, side = 3, adj = 0, font = 2)

points(background[, 3], col = "red", type = "b", pch = 20)
points(background[, 4], col = "blue", type = "b", pch = 20)
abline(v = background@bg.start)
text(background@bg.start, 0.2, "Background start", pos = 4)
abline(v = background@bg.stop, col = "blue")
text(background@bg.stop, 0.25, "Background stop", pos = 4, col = "blue")
abline(v = background@amp.stop, col = "green")
text(background@amp.stop, 0.3, "Plateau transition", pos = 4,
    col = "green")
legend(4, 1, c("Raw data", "First derivative", "Second derivative"),
    pch = rep(20, 3), col = c(1, 2, 4), bty = "n")

res <- AmpSim(cyc = 1:40, Cq = 25, noise = TRUE)
plot(res, xlim = c(1, 40), ylim = c(-0.1, 1), xlab = "Cycles",
    ylab = "refMFI", main = "Background Range Estimation\n in Presence of Noise",
    type = "b", pch = 20)
mtext("B", cex = 2, side = 3, adj = 0, font = 2)
background <- bg.max(res[, 1], res[, 2])

points(background[, 3], col = "red", type = "b", pch = 20)
points(background[, 4], col = "blue", type = "b", pch = 20)
abline(v = background@bg.start)
text(background@bg.start, 0.2, "Background start", pos = 4)
abline(v = background@bg.stop, col = "blue")
text(background@bg.stop, 0.25, "Background stop", pos = 4, col = "blue")
abline(v = background@amp.stop, col = "green")
text(background@amp.stop, 0.3, "Plateau transition", pos = 4,
    col = "green")
legend(4, 1, c("Raw data", "First derivative", "Second derivative"),
    pch = rep(20, 3), col = c(1, 2, 4), bty = "n")
par(mfrow = c(1, 1))
```

We used to the *bg.max* algorithm to analyze amplification curve data from an capillary convective PCR (capillaryPCR *chipPCR* data set). The data were used as raw data (Figure S16 A) and pre-processed data (Figure S16 B) using the *CPP* function. For both cases it was possible to receive results, which can be used for further processing. We observed no significant difference between the raw and pre-processed data.

Warnings in code chunk below were suppressed.

```
# Set parameter for the plot.
par(mfrow = c(2, 1), las = 0, bty = "n")

# Use of bg.max for time-dependent measurements.
# Amplification curves from the capillaryPCR data set were
# processed in a loop. The results of bg.max are added to the
# plot.

colors <- rainbow(8)

plot(NA, NA, xlim = c(0, 75), ylim = c(-200, 1300), xlab = "Time (min)",
    ylab = "Voltage (micro V)", main = "ccPCR - Raw data")
mtext("A", cex = 1.5, side = 3, adj = 0)
```



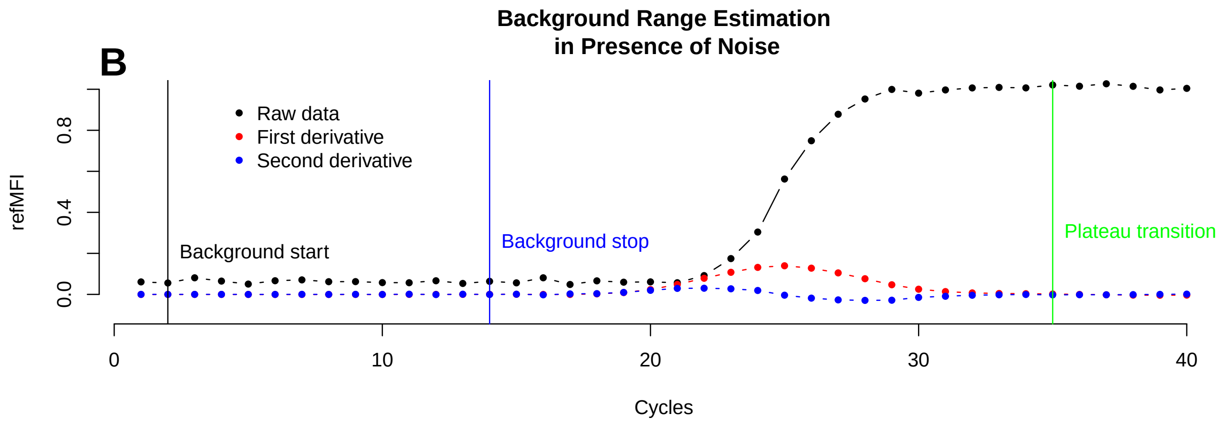
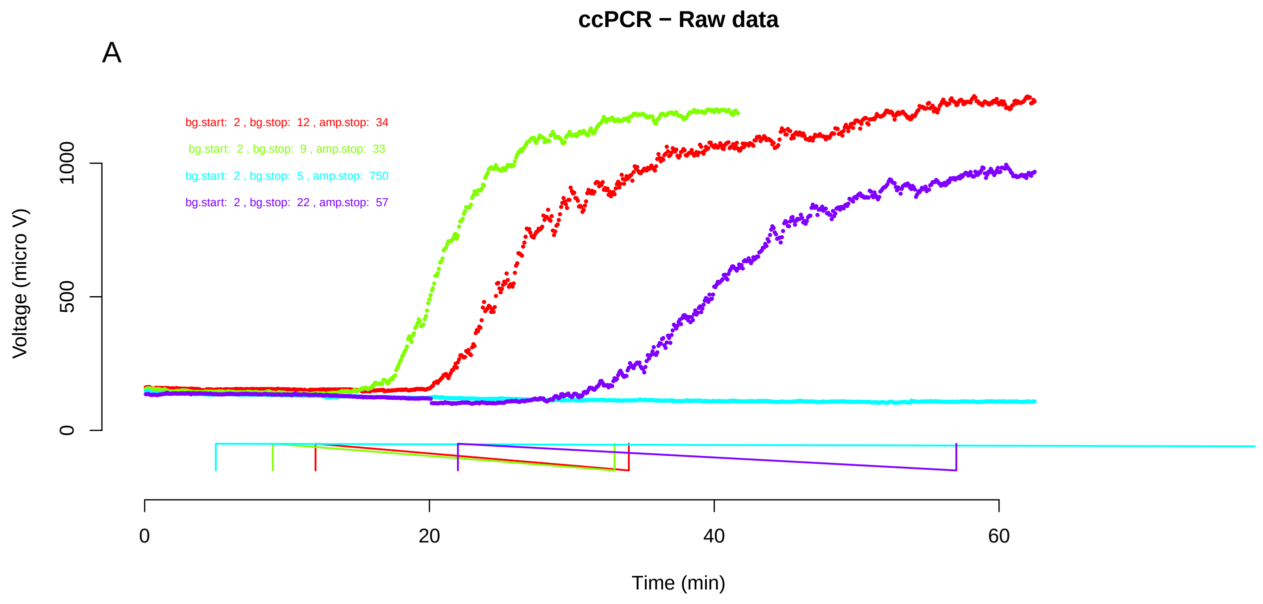


Figure S15: *bg.max* tries to estimate the range between the background and the plateau phase of an amplification reaction. *(A)* in absence and *(B)* presence of noise. The data were simulated with the *AmpSim* function.

```r
for (i in c(1, 3, 5, 7)) {
    x <- capillaryPCR[1L:750, i]
    y <- capillaryPCR[1:750, i + 1]
    res.bg <- summary(bg.max(x, y))
    lines(x, y, type = "b", pch = 20, col = colors[i], cex = 0.5)
    lines(c(res.bg[2], res.bg[2], res.bg[4], res.bg[4]), c(-150,
        -50, -150, -50), col = colors[i], lwd = 1.5)
    text(10, 1200 - i * 50, paste("bg.start: ", res.bg[1], ", bg.stop: ",
        res.bg[2], ", amp.stop: ", res.bg[4]), col = colors[i],
        cex = 0.6)
}

plot(NA, NA, xlim = c(0, 75), ylim = c(-200, 1300), xlab = "Time (min)",
    ylab = "Voltage (micro V)", main = "ccPCR - Pre-processed")
mtext("B", cex = 1.5, side = 3, adj = 0)
for (i in c(1, 3, 5, 7)) {
    x <- capillaryPCR[1L:750, i]
    y <- CPP(capillaryPCR[1L:750, i], capillaryPCR[1:750, i +
        1], method = "mova", trans = TRUE, bg.range = c(1, 105),
        bg.outliers = TRUE)[["y.norm"]]
    res.bg <- summary(bg.max(x, y))
    lines(x, y, type = "b", pch = 20, col = colors[i], cex = 0.5)
    lines(c(res.bg[2], res.bg[2], res.bg[4], res.bg[4]), c(-150,
        -50, -150, -50), col = colors[i], lwd = 1.5)
    text(10, 1200 - i * 50, paste("bg.start: ", res.bg[1], ", bg.stop: ",
        res.bg[2], ", amp.stop: ", res.bg[4]), col = colors[i],
        cex = 0.6)
}
```



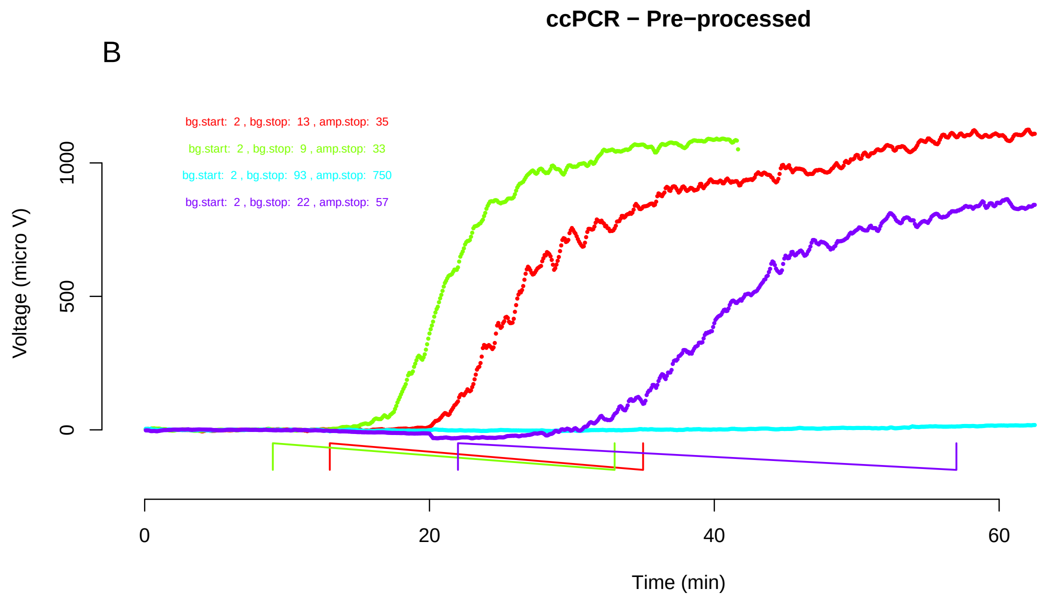


Figure S16: Application of the *bg.max* function. Amplification curve data from a capillary convective PCR were used *(A)* as raw data and *(B)* pre-processed (smoothed (moving average, window size 3), base-lined and trend corrected (robust MM-estimator)) with the CPP function. The output of the was used by *bg.max* to detected the start and the end of the amplification reaction. The start and end were reliably estimated (range between “bg.stop” and “amp.stop”). There was no significant difference between raw data and pre-processed data.

Table S3: Smoothing and filter methods of the *chipPCR* package. The parameter *lowess* for LOWESS smoother (locally-weighted polynomial regression) can be tuned by the parameters f and $iter$. The parameter *mova* for moving average can be tuned by the parameter *movaww*. *movaww* is the window size used for the moving average. The parameter *savgol* for Savitzky-Golay smoothing filter can be tuned by the parameter p (see *sgolayfilt* (*signal*) for details). The parameter *smooth* for cubic spline smooth can be tuned by the parameter $df.fact$. A $df.fact$ value of 1 will leave the raw data almost unaffected while a value 0.5 will smooth the curve considerably. The parameter *spline* for standard cubic spline smooth has currently no additional parameter. The parameter *supsmu* for Friedman's SuperSmoother can be tuned by the parameter *span*. The parameter $whit1$ (first order finite difference penalty) and $whit2$ (second order finite difference penalty) for Weighted Whittaker smoother smoothing filter, derived from the *ptw* package, can be tuned by the parameter *lambda*. For further details on the smoothers refer to the documentation of the parent functions.

Method	Parameter	value	Parent
LOWESS	*lowess*	*f*	*lowess*, *stats*
Cubic spline	*smooth*	*df.fact*	*smooth.spline*, *stats*
Interpolating Splines	*spline*	-	*spline*, *stats*
Friedman's "super smoother"	*supsmu*	*span*	*supsmu*, *stats*
Savitsky-Golay	*savgol*	-	*sgolayfilt*, *signal*
Moving Average	*mova*	*movaww* (3, 5, ...)	*filter*, *stats*
Whittaker	$whit1$, $whit2$	*lambda*	*whit1*, *whit2*, *ptw*
All smoother	*all*	defaults	

11 Normalization of amplification curve data

It is a common characteristic of amplification curve data that the fluorescence values between samples vary due to high background, sample inhomogeneities and variances in dye quantities (Figure S17 *A*). Data within an experiment have in most cases different minimum and maximum values. For the visualization of the data it is often better to scale the data within a defined range. Eventually, this helps to grasp the data faster. In particular, a comparison of data from different measurements and/or scaling is easier if data are normalized. *normalizer* is a function to normalize any data set. It is possible to chose from different methods (see Details). This is recommended if the data from an experiment have considerable variation regarding the background and plateau signal. Therefore, normalization of amplification curve data is a common task during the data analysis. To scale the fluorescence between 0 and 1 a *Min-Max normalization* (Equation S2) can be used [34]. We propose an alternative normalization based on quantiles (Equation S4). Quantiles are less affected by outliers. The method can be invoked by the parameter $norm$ = ”*luqn*”. Although this does not scale all values between zero and one we found it to be useful for noisy data. The parameter qnL is symmetrically used to set the level for the quantiles. By default the 3 % and 97 % quantiles are used for the normalization. In addition, a normalization to maximum (Equation S3, Figure S17 *D*) and by standard score (Equation S5, Figure S17 *F*).

$$RFU_{minmax} = \frac{RFU - \min(RFU)}{\max(RFU) - \min(RFU)} \tag{S2}$$

$$RFU_{max} = \frac{RFU}{\max(RFU)} \tag{S3}$$

$$RFU_{luqn} = \frac{RFU - Q_p(RFU)}{Q_{1-p}(RFU) - Q_p(RFU)} \tag{S4}$$

$$RFU_{zscore} = \frac{RFU - \bar{x}_{RFU}}{s_{RFU}} \tag{S5}$$

The parameter qnL is a user defined quantile, which is used for the quantile normalization.

- A quantile normalization herein refers to an approach, which is less prone to outliers than a normalization based on the minimum and the maximum of an amplification curve.
- minm does a min-max normalization between 0 and 1 (see [34] for explanation).
- max does a normalization to the maximum value (MFI/max(MFI)).
- lugn does a quantile normalization based on a symmetric proportion as defined by the qnL parameter (e.g., qnL = 0.03 equals 3 and 97 percent quantiles).
- zscore performs a z-score normalization with a mean of 0 and a standard deviation of 1.

```r
par(mfrow = c(2, 3), las = 0, bty = "n", oma = c(0.5, 0.5, 0.5,
    0.5))
tmp <- VIMCFX96_60

plot(NA, NA, xlim = c(1, 40), ylim = c(0, 6000), xlab = "Cycle",
    ylab = "RFU", main = "Raw data")
mtext("A", cex = 1.2, side = 3, adj = 0, font = 2)
lin <- apply(tmp[, -1], 2, function(x) lines(tmp[, 1], x))
abline(lm(rowMeans(tmp[2:10, 2L:ncol(tmp)]) ~ tmp[2:10, 1]),
    col = 2)

plot(NA, NA, xlim = c(1, 40), ylim = c(0, 3300), xlab = "Cycle",
    ylab = "RFU", main = "Baselined data")
mtext("B", cex = 1.2, side = 3, adj = 0, font = 2)
lin <- apply(tmp[, -1], 2, function(x) lines(tmp[, 1], CPP(tmp[,
    1], x, method.norm = "none")$y))


plot(NA, NA, xlim = c(1, 40), ylim = c(0, 1.15), xlab = "Cycle",
    ylab = "RFU", main = "MinMax-Normalization")
mtext("C", cex = 1.2, side = 3, adj = 0, font = 2)
```

```r
lin <- apply(tmp[, -1], 2, function(x) lines(tmp[, 1], CPP(tmp[,
    1], x, method.norm = "minm")$y))

plot(NA, NA, xlim = c(1, 40), ylim = c(0, 1.15), xlab = "Cycle",
    ylab = "RFU", main = "Max-Normalization")
mtext("D", cex = 1.2, side = 3, adj = 0, font = 2)
lin <- apply(tmp[, -1], 2, function(x) lines(tmp[, 1], CPP(tmp[,
    1], x, , method.norm = "max")$y))

plot(NA, NA, xlim = c(1, 40), ylim = c(0, 1.15), xlab = "Cycle",
    ylab = "RFU", main = "luqn-Normalization")
mtext("E", cex = 1.2, side = 3, adj = 0, font = 2)
lin <- apply(tmp[, -1], 2, function(x) lines(tmp[, 1], CPP(tmp[,
    1], x, method.norm = "luqn", qnL = 0.03)$y))

plot(NA, NA, xlim = c(1, 40), ylim = c(-1.5, 1.5), xlab = "Cycle",
    ylab = "RFU", main = "zscore-Normalization")
mtext("F", cex = 1.2, side = 3, adj = 0, font = 2)
lin <- apply(tmp[, -1], 2, function(x) lines(tmp[, 1], CPP(tmp[,
    1], x, method.norm = "zscore")$y))
```

The slope in a curve can be corrected by a linear regression. *CPP* and *lm.coefs* offers four linear regression models to calculate the slope based on the background range. This includes a ordinary least squares method (*lm*, *stats*) but also three robust methods. The robust regression methods are considered to be less vulnerable to outliers. This feature is especially useful, when the background range contains considerable noise. The methods are (I) a nonparametric rank-based estimator [17], (II) quantile regression [18] and (III) a MM-type estimators for linear regression [51]. By default the MM-type estimator is used. In all cases takes *CPP* a defined range of the amplification curve to extrapolate the linear trend over the entire data set. However, this step his to be performed with caution since this operation effects the amplification efficiency. The background is assumed to be constant for the entire measurement.

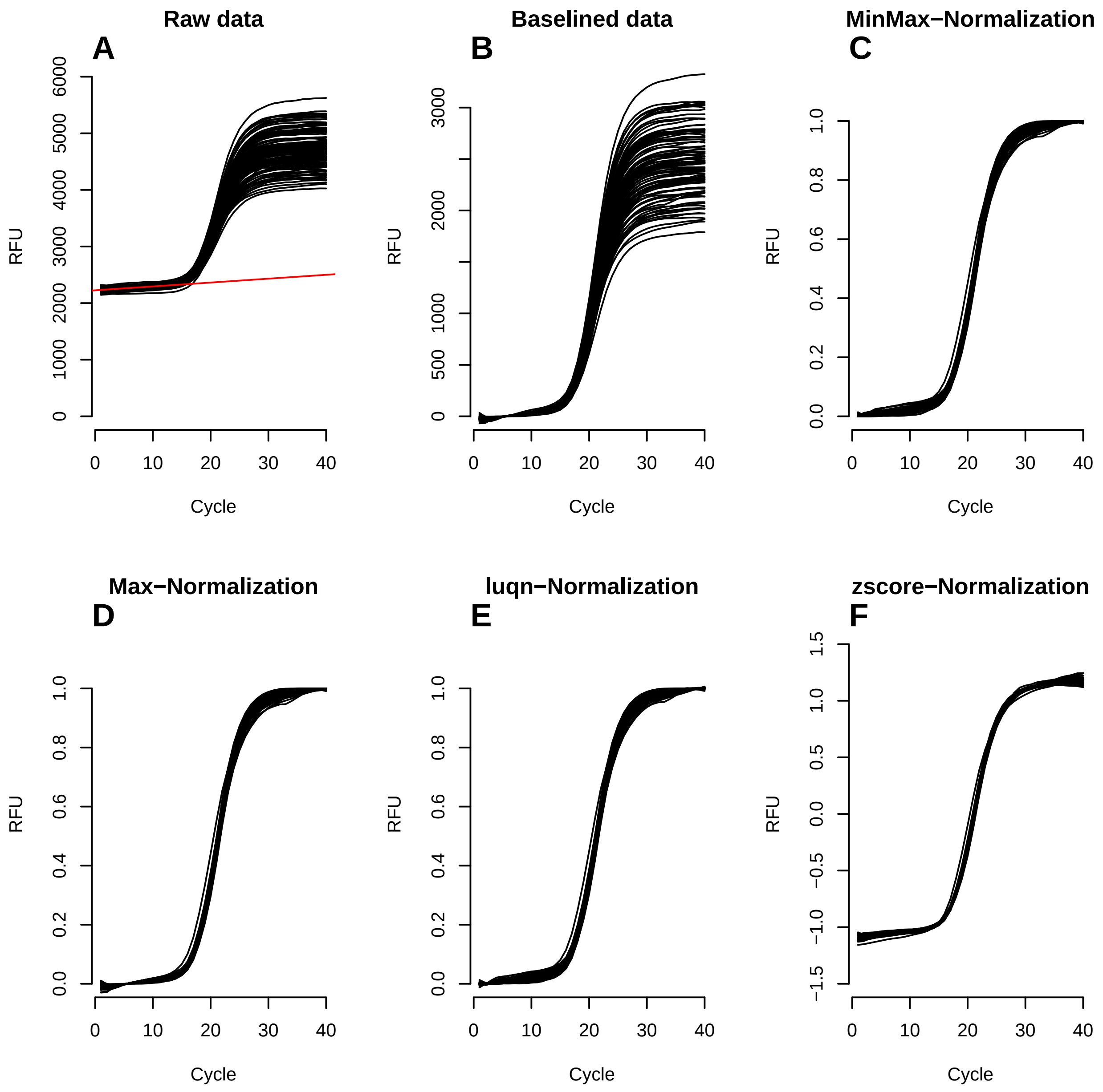


Figure S17: Comparison of the normalization functions from *CPP*. The VIMCFX96_60 data set (96-well plate cycler, Bio-Rad CFX96, EvaGreen detection) was used. *(A)* Raw data of all amplification curves. The signals are superimposed to circa 2200 RFU and the inter-sample baseline and plateau shift is high. Note the positive trend (–, fitted with an ordinary least squares method) in the background range of cycles 1 to 15. All subsequent plots were processed with the CPP function. By default, the curves are base-lined, smoothed (Savitzky-Golay smoother) and the slop corrected by a linear regression ($trans = TRUE$). *(B)* base-lined raw data, *(C) Min-Max normalization*, *(D) Max normalization*, *(E) lugn-normalization* with a cut off 3% and *(F) zscore-normalization*.

12 Compute linear model coefficients - Background subtraction based on linear models

The slope of the background range is often unequal to zero and in most cased accompanied by a positive or negative trend. By using a linear trend extrapolation it is possible to apply a correction of the slope.

lm.coefs is a wrapper around functions performing normal (linear least squares) and robust linear regression. If the robust linear regression is impossible, *lm.coefs* will perform linear regression using the least squares method. This function can be used to calculate the background of an amplification curve. The coefficients of the analysis can be used for a trend based correction of the entire data set. Thereby either a robust linear regression by computing MM-type regression estimators, a nonparametric rank-based estimator or a standard linear regression model. Care is needed when using trans with time series (see *lm* from the *stats* package for details).

```r
par(bty = "n")
plot(VIMCFX96_69[, 1], VIMCFX96_69[, 2], type = "l", xlab = "Cycle",
    ylab = "Fluorescence")
rect(1, 0, 10, 5000)
method <- c("lmrob", "rq", "least", "rfit")
for (i in 1:4) {
    tmp <- lm.coefs(VIMCFX96_69[1:10, 1], VIMCFX96_69[1:10, 2],
        method.reg = method[i])
    text(9, 3200 - i * 100, paste(method[i], ":", "m: ", round(tmp[1,
        1], 4), "n: ", round(tmp[2, 1], 3)))
    abline(a = tmp[1, 1], b = tmp[2, 1], col = i + 1, lwd = 1.5)
}

## Warning:  Chosen method rfit failed to converge.
##                    Performed linear regression.

legend("right", c("Data", "lmrob", "rq", "least", "rfit"), lty = 1,
    col = 1:5, cex = 0.95)
```

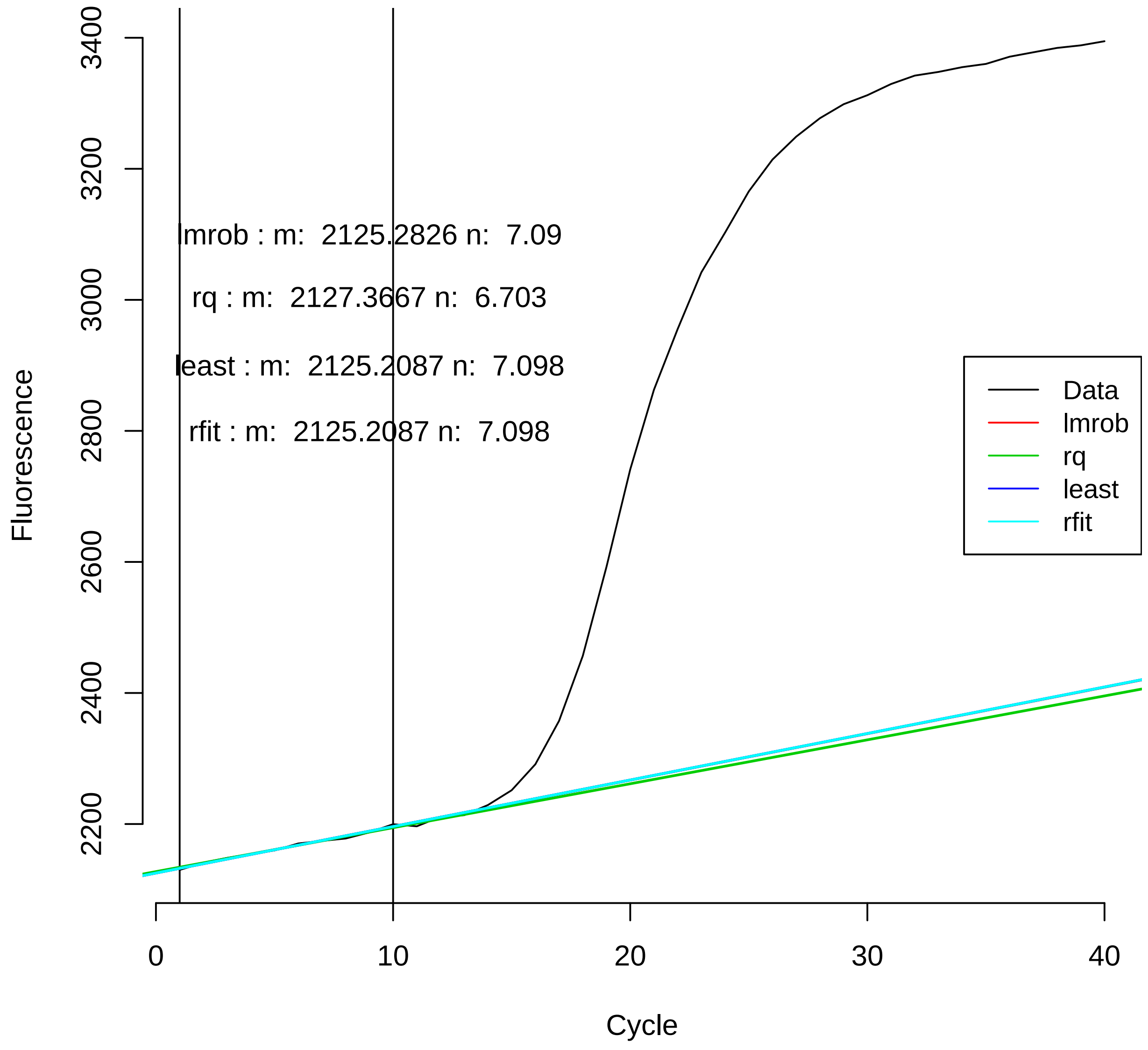


Figure S18: *lm.coefs* a function to compute linear model coefficients. The function is a convenient wrapper around few functions performing normal (least squares) and robust linear regression. If the robust linear regression is impossible, *lm.coefs* will perform linear regression using the least squares method. This function can be used to calculate the background of an amplification curve. The coefficients of the analysis can be used for a trend based correction of the entire data set.

13 The inder function - an interpolating five-point stencil

Many methods for curves analysis require the calculation of derivatives. It is possible to solve this by fitting a curve to a function and performing symbolic derivation. Unfortunately, this approach causes information loss through the fit and unnecessary adds additional assumptions regarding the relation between cycle number and fluorescence level. Hence, we integrated the *inder* function. *inder* ("in" and "der" = interpolate derivatives) finds numeric derivatives by a five-point stencil, a commonly used finite difference method. These methods approximate derivative in a given point by adding up products of nearby values of function and their weights [7]. This function can be used to estimate the approximate cycle of quantification (Cq). Differentiation is a method for background suppression and reduction of the inter sample background amplitude variations (Figure S21 A and B). Smoothing may enhance the calculation of derivatives calculation and optimize the signal-to-noise ratio. Therefore, we implemented spline interpolation. Friedman's SuperSmoother is also implemented. However, the use of this smoother is limited for the use in other functions such as *bg.max*. The parameter Nip (default $Nip = 4$) is used to define how often an interpolation takes place at n equidistant points within the first and the last cycle. A high Nip may improve the precision. However, Nip less than 2 and higher than 20 are not meaningful for conventional qPCR with 30 to 50 cycles. In context of qIA, a higher Nip might be appropriate.

13.1 Quantitative description of amplification reactions

According to the MIQE guidelines ("Minimum Information for publication of Quantitative real-time PCR Experiments", [4]) is the Ct referred to as quantification cycle (Cq). The calculated Cq is a relative value, which depends on the template copy number, instrument, reagents, amplification efficiency and probe technology. Low Cqs correlate with high quantities template copy numbers. Real-time technologies enable the quantification of nucleic acids by calculation of specific curve parameters like the quantification point (Cq) and the amplification efficiency (AE) based on the kinetics of the amplification curve. The Cq represents the number of cycles (time for qIA) needed to reach a defined fluorescence signal level in the exponential phase of the amplification curve. The Cq can be determined from a fixed threshold value or by various analytical algorithm as described elsewhere [38, 40, 49]. The output of *inder* includes the first derivative maximum (FDM) and second derivative maximum (SDM), which are commonly used in qPCR experiments. Figure S19 shows a typical result of the *inder* function. Following we show three examples explain properties of *inder* and to illustrate applications of the function in combination with other functions.

Function *inder* calculates numeric derivatives on smoothed data, which results in data points not observable in reality. The *rounder* function averages such result to the real values of cycle number.

Warnings in following code chunks were suppressed.

```
# Simulate an amplification curve with 40 cycles using the
# AmpSim function.
isPCR <- AmpSim(cyc = 1:40)

# Use inder to calculate the derivatives and assign the
# results to the object res
res <- inder(isPCR)

# Process res by rounder and assign the results to the object
# rd
rd <- rounder(res)

# Print details of res and rd. Due to the internal use of
# interpolating splines in inder are the number of elements
# in the object res the n-th time of the raw data. In this
# case 200 virtual instead of 40 real cycles.
head(res)

##             x    y           d1y           d2y
## [1,] 1.000000 0.05 -4.350617e-13  8.697680e-13
## [2,] 1.245283 0.05 -2.217106e-13  8.704119e-13
## [3,] 1.490566 0.05 -8.206265e-15  8.702197e-13
## [4,] 1.735849 0.05  2.086810e-13  8.842231e-13
## [5,] 1.981132 0.05  4.222938e-13  7.730801e-13
## [6,] 2.226415 0.05  4.929158e-13 -2.900925e-13

summary(res)
```

```
## Smoothing method: spline
## First derivative maximum: 19.89
## Second derivative maximum: 18.91
## Second derivative minimum: 21.11
## Second derivative center: 19.98

head(rd)

##      cyc    y           d1y           d2y
## [1,]   1 0.05 -2.216595e-13  8.701332e-13
## [2,]   2 0.05  3.471567e-13 -5.632357e-14
## [3,]   3 0.05 -1.083221e-12 -4.552932e-13
## [4,]   4 0.05  3.944521e-12  2.505165e-12
## [5,]   5 0.05 -1.449063e-11 -1.173032e-11
## [6,]   6 0.05  5.390452e-11  5.562310e-11

# summary(rd)
```

Figure S19 illustrates the most important parameters of the *inder* function. We used the *AmpSim* function to simulate an ideal "noise-free" amplification curve with the default setting to calculated the second derivative maximum (SDM) with *inder*. If *logy* is $TRUE$ than a semi-decadic log scale graph (corresponds to the linear phase) to illustrate the exponential dynamic of the qPCR amplification is used. The parameter *logy* is $FALSE$ by default. To the best of our knowledge, is *inder* the first tool in **R**, which allows user to numerically derive his data without fitting them to any function or combination of functions. The universality of stencil approach can find an application even in problems not related to the analysis of amplification curve.

```
# Use AmpSim to generate an amplification curve with 40
# cycles and an approximate Cq of 20 and assign it to the
# object isPCR.  isPCR is an object of the class
# 'data.frame'.
isPCR <- AmpSim(cyc = 1:40, Cq = 20)

# Invoke the inder function for the object isPCR to
# interpolate the derivatives of the simulated data as object
# res. The Nip parameter was set to 5. This leads to smoother
# curves. res is an object of the class 'der'.
res <- inder(isPCR, Nip = 5)

# Plot the object res and add descriptions to the elements.

par(las = 0, bty = "n", oma = c(0.5, 0.5, 0.5, 0.5))

plot(isPCR, xlab = "Cycle", ylab = "RFU", ylim = c(-0.15, 1),
    main = "", type = "b", pch = 20, lwd = 2)
colors <- rainbow(4)
# Add graphical elements for the derivatives and the
# calculated Cq values FDM, SDM, SDm and SDC.

lines(res[, "x"], res[, "d1y"], col = "blue", lwd = 2)
lines(res[, "x"], res[, "d2y"], col = "red", lwd = 2)

# Fetch the Cq values from res with the summary function
summ <- summary(res, print = FALSE)

abline(v = summ, col = colors, lwd = 2)
text(15, 0.3, paste("FDM ~ ", round(summ["FDM"], 2)), cex = 1.1,
    col = colors[1])
text(15, 0.2, paste("SDM ~ ", round(summ["SDM"], 2)), cex = 1.1,
    col = colors[2])
text(15, -0.1, paste("SDm ~ ", round(summ["SDm"], 2)), cex = 1.1,
    col = colors[3])
text(15, 0.7, paste("SDC ~ ", round(summ["SDC"], 2)), cex = 1.1,
    col = colors[4])

legend(1.1, 0.9, c("raw", "first derivative", "second derivative"),
    col = c(1, 4, 2), lty = c(2, 1, 1), bty = "n")

# Summary of the object res.
summ

##      FDM      SDM      SDm      SDC
## 19.81407 19.03015 20.98995 19.98604
```

inder is a helper function, which can be part of other routines. Recently, we added this approach to the *diffQ* function of the *MBmca* for improved predictions. *diffQ* function is part of a routine to calculate the melting points of nucleic acids [34]. The FDM and SDM are peak values to determine the Cq. We used the *inder* function in *diffQ* to compare the Cq values between a quantification experiment where the samples were either detected with a gene specific hydrolysis probe or the intercalating dye EvaGreen. For the analysis we focused on the SDM. We found that the samples detected with EvaGreen had a slightly lower Cq (Figure S20 A) than samples detected with the hydrolysis probe (Figure S20 B). The spread of the Cq was also less in samples where EvaGreen was used for the monitoring.

```
# Plot all data from C127EGHP and calculate the SDM (Second
# Derivative Maximum) values with the diffQ2() function
# (Note: the inder parameter is set as TRUE) first plot the
# samples detected with EvaGreen and next the samples
# detected with the Hydrolysis probe
require(MBmca)

pointer <- function(x, pos = 1, w = 5, stat = TRUE) {
    xx <- pos + rep(seq(-0.1, 0.1, length.out = w), ceiling(length(x)/w))
    yy <- sort(x)
    points(xx[1:length(yy)], yy, pch = 19)

    if (stat == TRUE)
        x.median <- median(x, na.rm = T)
    x.mad <- mad(x, na.rm = T) * 2
    param <- c(length = 0, code = 3, pch = 15, cex = 1.2)
    arrows(xx[1] * 0.98, x.median, tail(xx, 1) * 1.02, x.median,
        param, lwd = 3, col = 2)
    arrows(xx[1] * 1.01, x.median + x.mad, tail(xx, 1) * 0.99,
        x.median + x.mad, param, lwd = 2, lty = 2, col = 4)
    arrows(xx[1] * 1.01, x.median - x.mad, tail(xx, 1) * 0.99,
        x.median - x.mad, param, lwd = 2, lty = 2, col = 4)
}

amp.liner <- function(range, input, colors = "black") {
    sapply(range, function(i) {
        lines(input[, 2], input[, i], col = colors, pch = 19)
        tmpP <- mcaSmoother(input[, 2], input[, i])
        SDM <- diffQ2(tmpP, inder = TRUE)[["xTm1.2.D2"]][1]
        abline(v = SDM)
        SDM
    })
}

layout(matrix(c(1, 3, 2, 3), 2, 2, byrow = TRUE), respect = TRUE)
par(las = 0, bty = "n")
plot(NA, NA, xlim = c(1, 40), ylim = c(0, 10), xlab = "Cycle",
    ylab = "Fluorescence", main = "EvaGreen")
mtext("A", cex = 1.1, side = 3, adj = 0, font = 2)

EG <- amp.liner(range = 3L:34, input = C127EGHP)
```

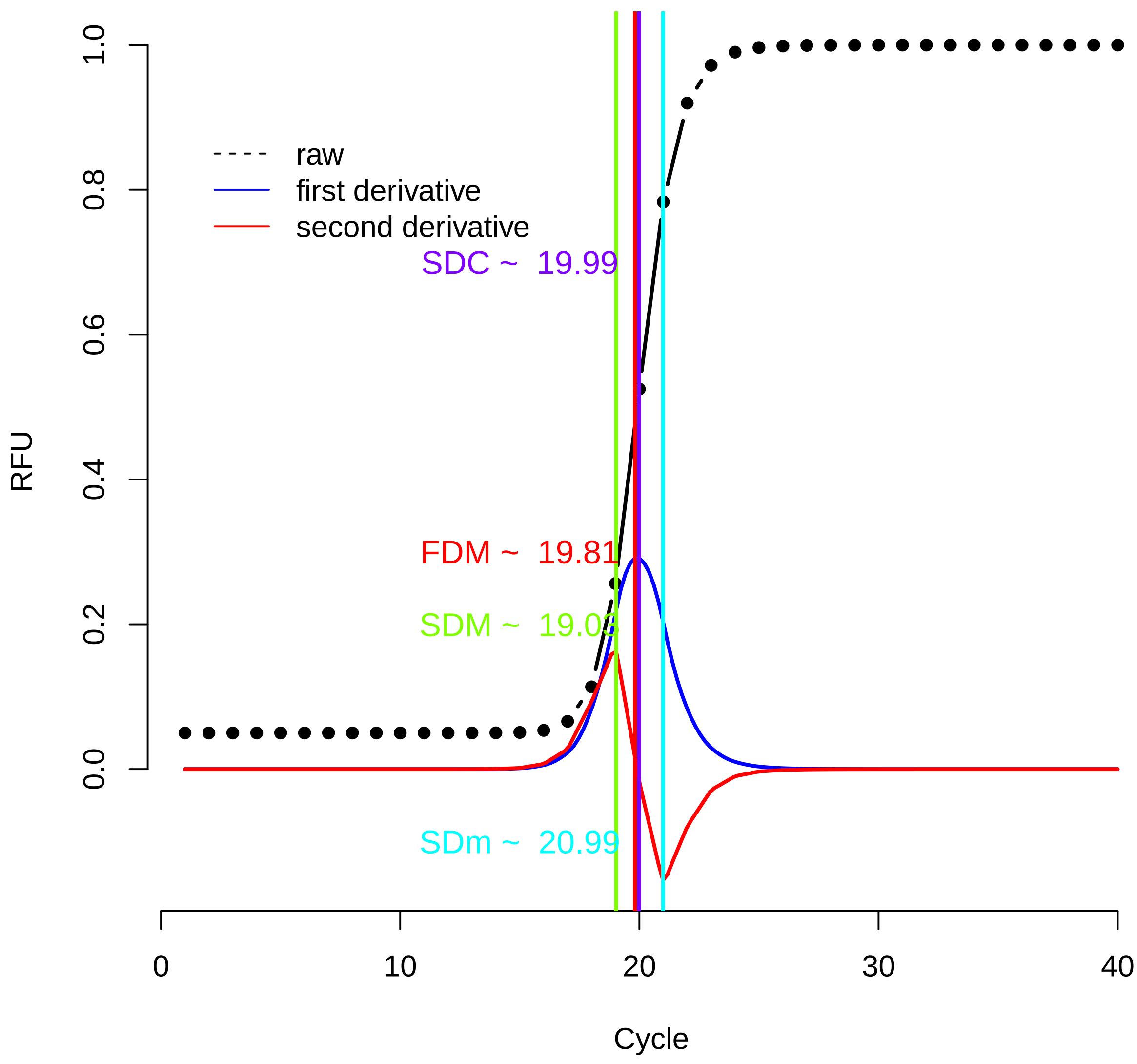


Figure S19: Cycle of quantification by the second derivative maximum method. Raw data (●) were generated using the *AmpSim* simulation function (see example main text). The inflection point is the point where the slope is maximum and the curvature is zero. The first derivative of the amplification curve has a first derivative maximum (FDM) at the inflection point. The second derivative maximum method (SDM) needs to differentiate a curve to the second order prior to quantification. The second derivative exhibits a zero-crossing at the FDM. The function $y = f(x)$ is numerically derived by five-point stencil. This method do not require any assumptions regarding the function f. *inder* calculates the approximate SDM. The SDM might in addition be useful for isothermal amplification processes. The SDM is calculated from a derived cubic spline. Similarly the first approximate derivative maximum (FDM), second derivative minimum (SDm), and approximate second derivative center (SDC, geometric mean of SDM and SDm) are available. FDM, SDm and SDC values can be used to further characterize the amplification process.

```r
plot(NA, NA, xlim = c(1, 40), ylim = c(0, 10), xlab = "Cycle",
    ylab = "Fluorescence", main = "Hydrolysis probe")
mtext("B", cex = 1.1, side = 3, adj = 0, font = 2)

HP <- amp.liner(range = 35L:66, input = C127EGHP)

plot(NA, NA, xlim = c(0.8, 2.2), ylim = c(13, 14), xaxt = "n",
    xlab = "", ylab = "Cq (SDM, diffQ2)")
text(c(1.05, 2), c(13.05, 13.05), c("EG", "HP"), cex = 1.2)
mtext("C", cex = 1.1, side = 3, adj = 0, font = 2)
pointer(EG, pos = 1, w = 8)
pointer(HP, pos = 2, w = 8)
```

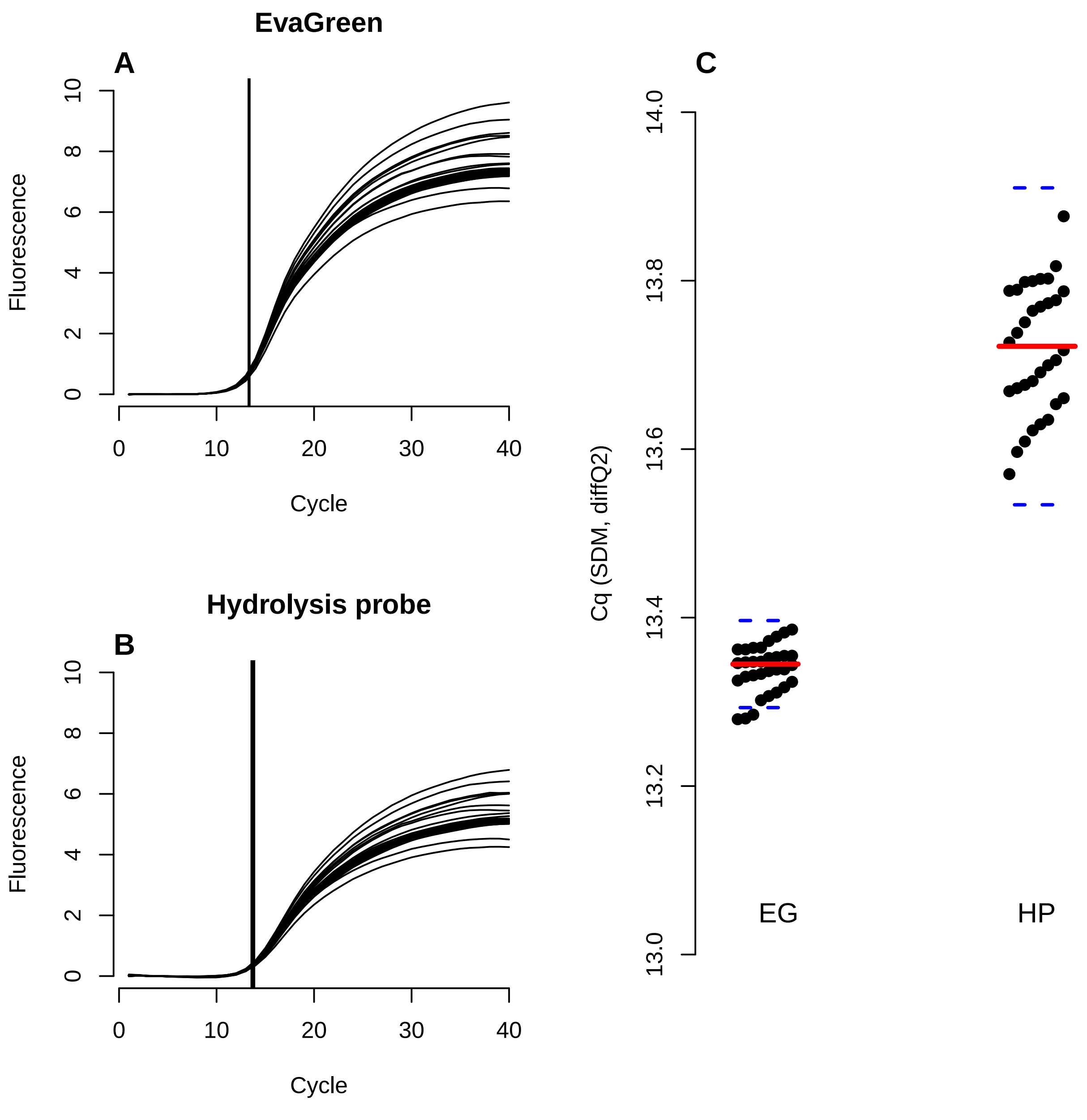


Figure S20: Plot all data from C127EGHP and calculate the *SDM* (Second Derivative Maximum) values with the *diffQ2* function. *(A)* Plot the samples detected with EvaGreen and *(B)* shows the same samples detected with the Hydrolysis probe for MLC-2v. *(C)* Stripchart of the Cq values (●) with the median (–) and the median absolute deviation (– –). This result indicates, that the variance of the derived from the detection with hydrolysis probes is higher than the samples detected with EvaGreen. Note: the *inder* parameter is set as TRUE.

13.2 Quantification cycle calculation by the inder function

The presence of noise may cause many false estimates for the FDM and SDM. To minimize this problem, it is possible to smooth the first derivative of the amplification curve. Many methods integrated the moving average as first pre-processing step (e.g., [44]). The moving average filter is linear filter, which replaces sequentially data points with the average of the neighbor data points. The average is calculated from a defined span ("window") of odd count (e.g., 3, 5). The "average" herein may refer to the arithmetic mean, the median, the geometric or the exponential mean. The *smoother* function uses exclusively the arithmetic mean. Moving average is intuitive and easy to implement but it lags behind a trend and ignores rapid changes. For example, the 3- and 5-window moving average (running mean) filters are useful to pre-process data, but always leads to a forerun of few cycles. This is in particular problematic in the exponential phase. Splines apply non-parametric regression by local cubic polynomials between knot points [25]. Other examples for smoothers include Savitzky-Golay smoothing filter, Friedman's SuperSmoother, and the Weighted Whittaker smoother (see the *smoother* function for details).

Provided that the smoother is properly adjusted, it is possible to detect only the significant peaks while small or to narrow peaks are ignored. *smoother* is used by other functions of *chipPCR* like *CPP*. The example for Figure S20 illustrates the use of the *diffQ* and *diffQ2* function from the *MBmca* and the integration of the *inder* function. In contrast to the original publication [34] is the *inder* function in *diffQ* and *diffQ2* used for a precise peak location while the approximate SDM is calculated from the derivative of a quadratic function at the approximate SDM.

13.3 The *inder* function in combination with a 5-parameter curve fit function

In the previous example we used smoothing and the *inder* method to calculate the SDM. But, smoothing may alter peak signal considerably. For example, peak height reduction and peak width increase are a common problem. An alternative technique to determine the FDM or SDM is by fitting the raw data. In the next example we used the *drm* function from the *drc* package [32] to fit a five-parameter log-logistic function (S-shaped). The *inder* function was used to calculate the SDM of the predicted models (Figure S21).

```
fit.amp <- function(cyc, fluo, plot = FALSE) {

    ampl <- quantile(fluo, 0.999)
    bl <- quantile(fluo, 0.001)
    Cq <- round(mean(cyc))
    b.eff <- 1

    fit <- nls(fluo ~ bl + ampl/(1 + exp(-(cyc - Cq)/b.eff)),
        start = list(Cq = Cq, b.eff = b.eff, ampl = ampl, bl = bl))

    res.pred <- data.frame(cyc, predict(fit))
    res <- inder(res.pred[, 1], res.pred[, 2])
    if (plot) {
        lines(res[, 1], res[, 4])
    }
    # SDM
    summary(res)[2]
}

tmp <- C126EG595

out <- apply(tmp[, -1], 2, function(x) fit.amp(tmp[, 1], x))

layout(matrix(c(1, 2, 1, 3), 2, 2, byrow = TRUE))

plot(NA, NA, xlim = c(1, 40), ylim = c(min(tmp[, 2L:97]), max(tmp[,
    2L:97])), xlab = "Cycle", ylab = "Raw fluorescence")
mtext("A", cex = 1.2, side = 3, adj = 0, font = 2)
for (i in 2L:97) {
    lines(tmp[, 1], tmp[, i], col = ifelse(out[i - 1] < 15.5,
        "red", "black"), lwd = 2)
}
abline(v = out)

plot(NA, NA, xlab = "Cycle", ylab = "RFU''(Cycle)", main = "",
    xlim = c(0, 40), ylim = c(-850, 850))
abline(v = 15.5, lty = 2)
invisible(apply(tmp[, -1], 2, function(x) {
    fit.amp(tmp[, 1], x, plot = TRUE)
}))
mtext("B", cex = 1.2, side = 3, adj = 0, font = 2)

hist(out, xlab = "Cq (SDM)", main = "", breaks = seq(14.8, 15.8,
    0.05), col = rainbow(96))
abline(v = 15.5, lty = 2)
mtext("C", cex = 1.2, side = 3, adj = 0, font = 2)
```

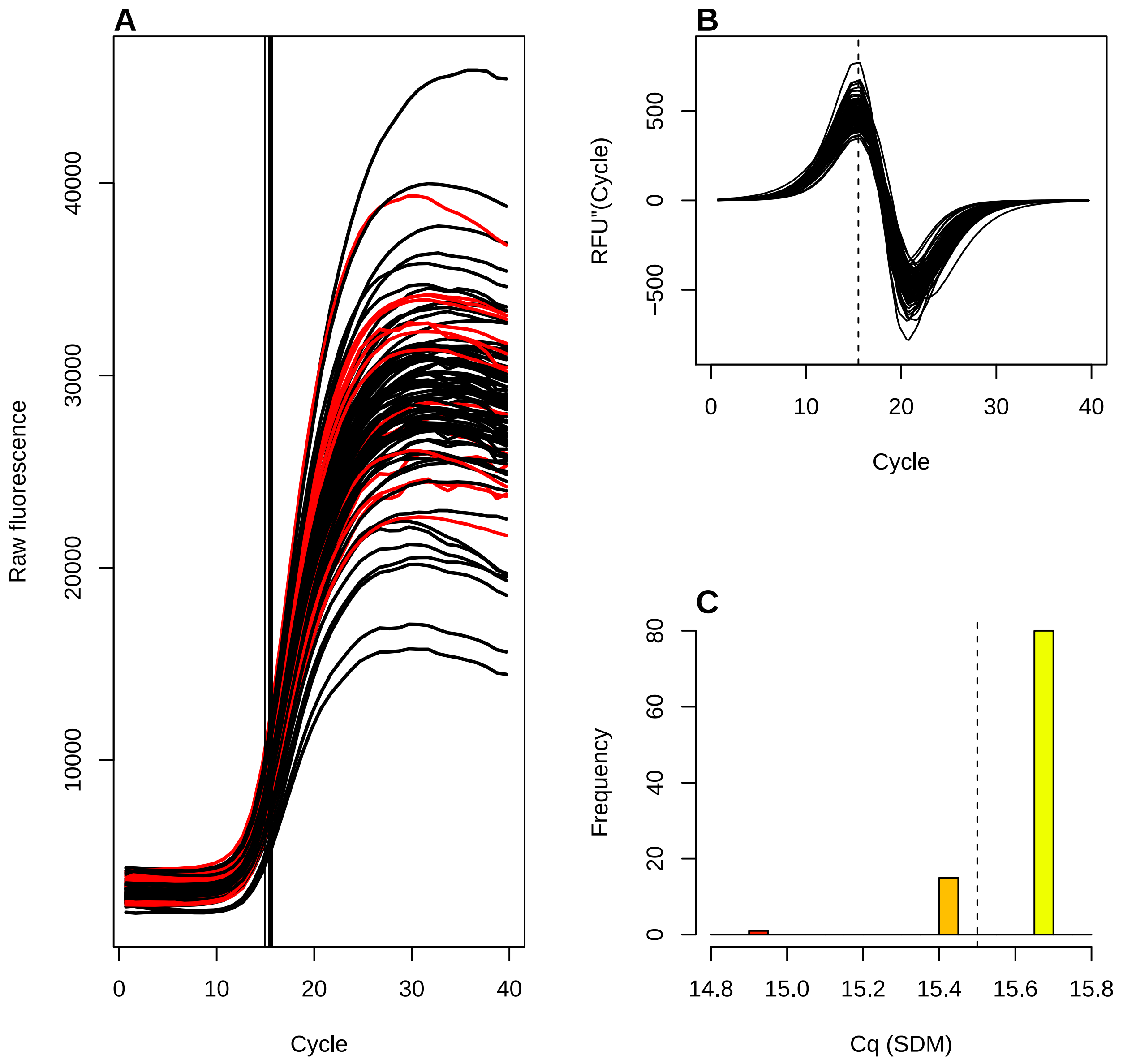


Figure S21: Amplification curve profiles from the Bio-Rad iQ5 thermo cycler for the human gene *HPRT1*. *(A)* The *C126EG595* data set was used with 96 replicates of equal starting numbers of template molecules. Vertical lines represent the Cq (SDM method) determined with inder method on amplification curves fitted with a 5-parameter curve function. Curves with Cqs less than 14.5 are indicated in red (–). *(B)* Second derivatives of the amplification curves. Note that after differentiation all inter sample baseline and plateau shifts are similar. *(C)* Histogram (class width = 0.05 Cq) of the Cq values (SDM). Cqs were mainly at circa 15.7 (N = 80) while some amplification curves had a Cq less than 15.5 (N = 16).

14 Threshold cycle method

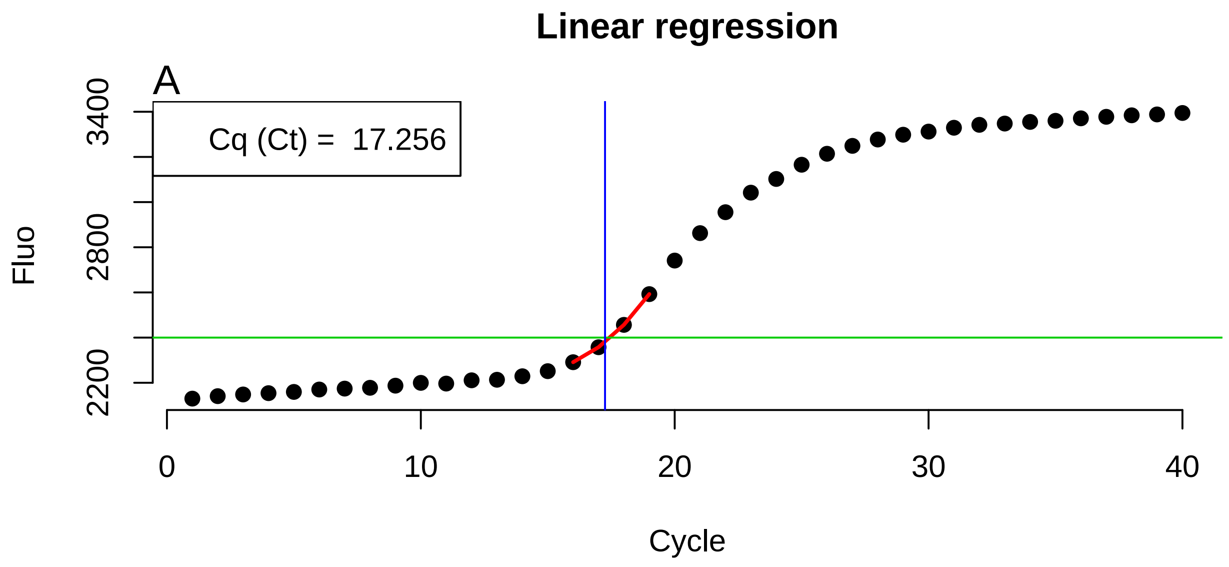
The *th.cyc* function can also be used to calculate the quantification cycle. However, in contrast to the *FDM* and *SDM* requires this method a proper baselining [38, 40, 49]. In qPCR analysis are other approaches the method of choice. This function was implemented primarily for the analysis of amplification from qIA but also usable for qPCR if used cautiously. We implement a symmetrically approximation algorithm based on linear and quadratic least squares regression. The Threshold Cycle (Ct) (Cq according to MIQE, see) is the cycle number at which the fluorescence exceeds significantly a point above the baseline and defined threshold in a particular samples. Thus the Ct is the cycle when sufficient numbers of amplicons have accumulated. The *th.cyc* calculates the intersection of the user defined Ct value (r) and a linear regression or quadratic polynomial in the range of the user defined Ct value. In contrast to other methods is does *th.cyc* have no requirement to fit a "complex" non linear model to the entire data set but rather focuses on the specific area. The polynomial is calculated from four neighbor values at the fluorescence threshold.

Warnings in following code chunks were suppressed.

```r
# Raw data from the VIMCFX96_69 data set.  Cycles x and
# Fluoresce values y
x <- VIMCFX96_69[, 1]
y <- VIMCFX96_69[, 2]

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

# Plot the raw data
plot(x, y, xlab = "Cycle", ylab = "Fluo", main = "Linear regression",
    pch = 19)
mtext("A", cex = 1.3, side = 3, adj = 0)
# Calculate the Cq (Ct) value
res <- th.cyc(x, y, r = 2400, linear = TRUE)
lines(res@input, col = 2, lwd = 2)

# Threshold fluorescence value
abline(h = res[2], col = 3)

# Calculated Ct value
abline(v = res[1], col = 4)
legend("topleft", paste("Cq (Ct) = ", round(res[1], 3)))

plot(x, y, xlab = "Cycle", ylab = "Fluo", main = "Quadratic regression",
    pch = 19)
mtext("B", cex = 1.3, side = 3, adj = 0)

# Calculate the Ct value
res <- th.cyc(x, y, r = 2400, linear = FALSE)
lines(res@input, col = 2, lwd = 2)

# Threshold fluorescence value
abline(h = res[2], col = 3)

# Calculated Ct value
abline(v = res[1], col = 4)
legend("topleft", paste("Cq (Ct) = ", round(res[1], 3)))
```

14.1 Application of the *th.cyc* on ccPCR data

Similarly to the previous experiments we used the *CPP* and *th.cyc* functions to analyze a continuous amplification reaction. The data used herein were taken from the *capillaryPCR* data set, which were measured with capillary convective PCR (ccPCR) technology. A modified device of the ccPCR system as described by Chou et al. 2011 [6] was used. As heating system a conventional block heat was used. On the top of the heating block, we placed for the uptake of the capillaries an aluminum block (8 mm height) in which four holes (3.2 mm diameter and 3.0 mm depth with round shaped bottom) were drilled. The capillaries are regular 100 μL Roche LightCycler(R).



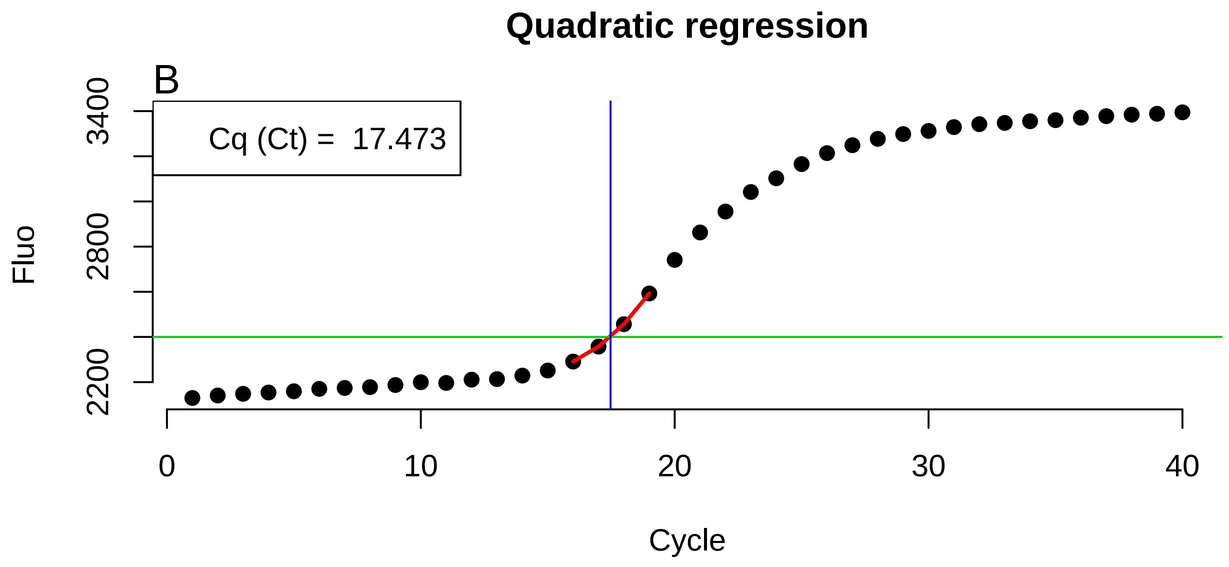


Figure S22: Working principle of *th.cyc*. The function provides two modes (**A**) is the linear regression. **B**) Quadratic regression) for the calculation of the Cq. In both cases is the highest R squared value determining how many left and right neighbors above and the below the used defined threshold level are use.

These glass capillaries have a round shaped closed bottom (2.3 mm inner diameter and 3.2 mm outer diameter). An “ESE-Log” detector (QIAGEN Lake Constance) was used for the real time fluorescent measurements, which was mounted in a distance of 5-10 mm next to the capillary. The PCR was performed with SYBR(R) Green fluorescent intercalating dye. Thereof the ESE-Log has in one channel the excitation at 470 nm and the detection at 520 nm. The data was recorded by the FL Digital Software (QIAGEN Lake Constance) and the exported text based raw data. The raw data were accompanied by noise, showed an off-set of circa 150 micro Volt and slight negative trend (Figure S16 A). Therefore, the data were first pre-processed with *CPP*. Note that the amplification curve were baselined and slightly smoother after pre-processing. Finally, we used the *th.cyc* function to determine the time required to reach a certain threshold level above the defined value (Threshold level “r” (80 μVolts)) (Figure S23 B).

```r
# Application of the th.cyc method to determine the Cq from a
# continuous amplification reaction.
par(las = 0, bty = "n", oma = c(0.5, 0.5, 0.5, 0.5))

plot(NA, NA, xlim = c(0, 80), ylim = c(0, 1200), xlab = "Time (min)",
    ylab = "Voltage [micro V]", main = "ccPCR - Pre-processed Data")
mtext("B", cex = 2, side = 3, adj = 0)
# Threshold level 'r' (50 micro Volts)
for (i in c(1, 3, 5, 7)) {
    y.tmp <- CPP(capillaryPCR[, i], capillaryPCR[, i + 1], trans = TRUE,
        bg.range = c(1, 150))$y.norm
    Ct.tmp <- th.cyc(capillaryPCR[, i], y.tmp, r = 80, linear = FALSE)
    abline(v = Ct.tmp[1])
    text(Ct.tmp[1] * 1.125, 1200, paste(round(Ct.tmp[1], 1),
        "\nmin"), cex = 0.8)
    lines(capillaryPCR[, i], y.tmp, type = "b", pch = 20 - i)
    points(Ct.tmp@input, col = "red", pch = 19)
}
abline(h = 80)
legend("topleft", c("Run 1", "Run 2", "Run 3", "Control"), pch = c(19,
    17, 15, 13), lwd = 1.3, bty = "n")
```

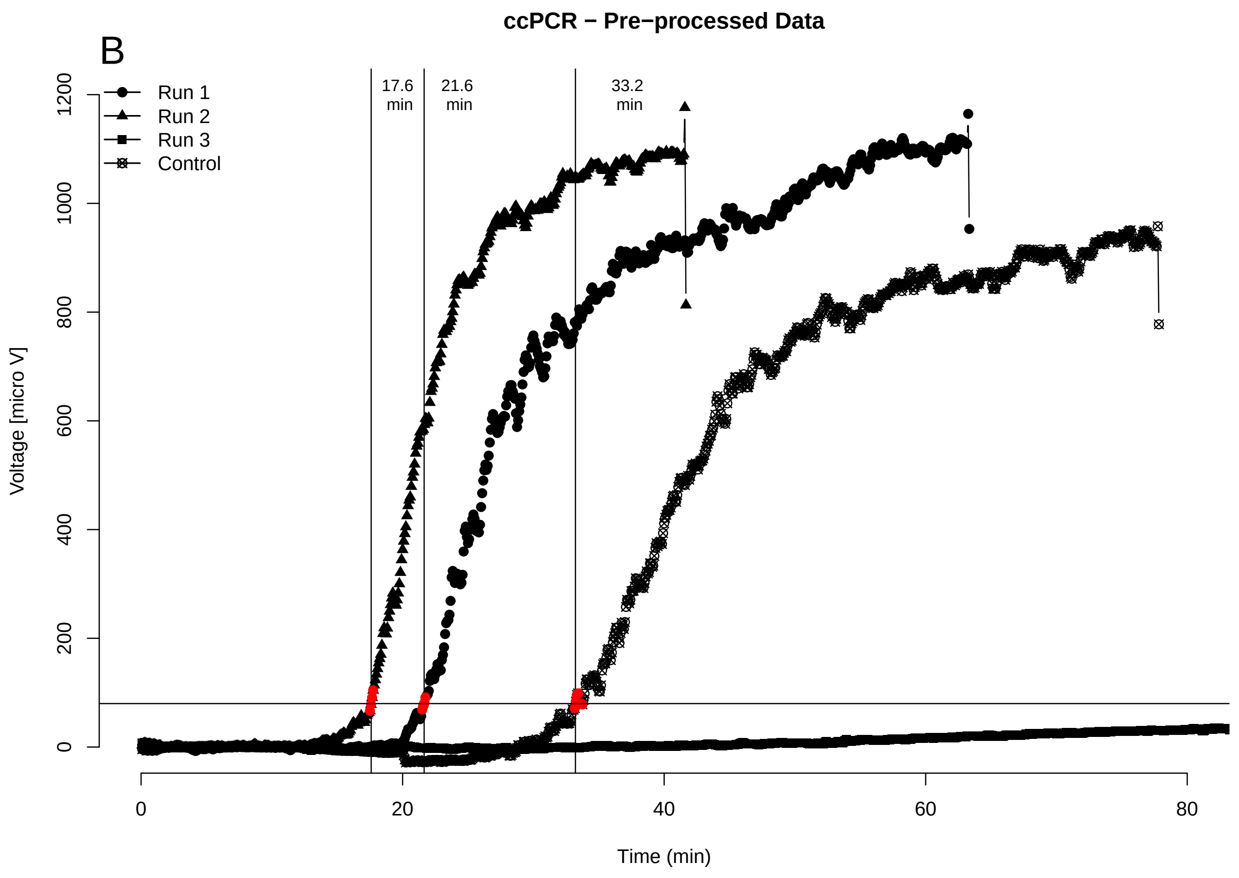


Figure S23: Application of *th.cyc* for the analysis of ccPCR data. The *CPP* function was used to pre-process the data. Susequently, the data were analyzed using the *th.cyc* function using the linear regression mode. The threshold level ($r = 50$) was identical for all data. The Cq (Ct) are given in minutes. The range used for the calculation of the Cq is indicated in red. Negative curves are automatically excluded from the analysis if the 90% percentile is lower or equal to the threshold level (r).

14.2 Application of the *th.cyc* and *CPP* function for Helicase Dependent Amplification

The *chipPCR* package was used to analyze a helicase dependent amplification (HDA) of Vimentin (Vim). The VideoScan Platform [42] was used to monitor the amplification. The HDA was performed at 65 °C. Three concentrations of input DNA (D1, D2, D3) were used (Figure S24).

To perform an isothermal amplification in VideoScan, standard conditions for the IsoAmp(R) III Universal tHDA Kit (Biohelix) were used. Primers and templates are described in [42]. The reaction was composed of reaction mix A) 10 μL A. bidest, 1.25 μL 10xbuffer, 0.75 μL primer(150 nM final), 0.5 μL template plasmid. Preincubation: This mixture was incubated for 2 min at 95 °C and immediately placed on ice. Reaction mix B) 5 μL A. bidest., 1.25 μL 10x buffer, 2 μL NaCl, 1.25 μL $MgSO_4$, 1.75 μL dNTPs, 0.25 μL EvaGreen, 1 μL enzyme mix. The mix was covered with 50 μL mineral oil. The fluorescence measurement in VideoScan HCU started directly after adding buffer B at 65 degree Celsius. A 1x (D1), a 1:10 dilution (D2) and a 1:100 (D3) dilution were tested. Temperature profile (after Preincubation):

- - 60 seconds at 65 °C,
- - 11 seconds at 55 °C & Measurement

Warnings in following code chunks were suppressed.

```r
par(mfrow = c(2, 1), bty = "n")
plot(NA, NA, xlim = c(0, 5000), ylim = c(0.45, 0.8), xlab = "Time (sec)",
    ylab = "Fluorescence", main = "HDA - Raw data")
mtext("A", cex = 2, side = 3, adj = 0)
lines(C85[, 2], C85[, 3], type = "b", col = 2, pch = 20)
lines(C85[, 4], C85[, 5], type = "b", col = 4, pch = 20)
lines(C85[, 6], C85[, 7], type = "b", col = 6, pch = 20)
legend("bottomright", c("D1, 1x", "D2, 1:10", "D3, 1:100"), col = c(2,
    4, 6), pch = rep(20, 3), bty = "n")

plot(NA, NA, xlim = c(0, 2000), ylim = c(0, 0.4), xlab = "Time (sec)",
    ylab = "Fluorescence", main = "HDA - Pre-processed data")
mtext("B", cex = 2, side = 3, adj = 0)
legend("topleft", c("D1, 1x", "D2, 1:10", "D3, 1:100"), col = c(2,
    4, 6), pch = rep(20, 3), bty = "n")

# Define the parameters for the pre-processing by CPP and the
# th.cyc function.  smoothing method
sm <- "mova"

# manual range for background
br <- c(2, 10)

# time range for analysis
xr <- 3L:200

# method for baseline normalization
lrg <- "least"

# threshold level for the th.cyc function
r <- 0.05
# Calculate in a loop the Cq values (Cycle threshold method)
# and add the calculated time (in minutes) to the plot.
for (i in c(2, 4, 6)) {
    y.tmp <- CPP(C85[xr, i], C85[xr, i + 1], method = sm, bg.range = br,
        trans = TRUE)$y.norm
    Ct.tmp <- th.cyc(C85[xr, i], y.tmp, r = r, linear = FALSE)
    abline(v = Ct.tmp[1], col = "grey")
    lines(C85[xr, i], y.tmp, col = i, lwd = 2)
    points(Ct.tmp@input, col = "red", pch = 19)
    text(Ct.tmp[1] * 1.1, 0.36, paste(round(Ct.tmp[1]/60, 1),
         "\nmin"))
}

# Show the fluorescence value, which defines the threshold.
abline(h = r, lty = 2)
```

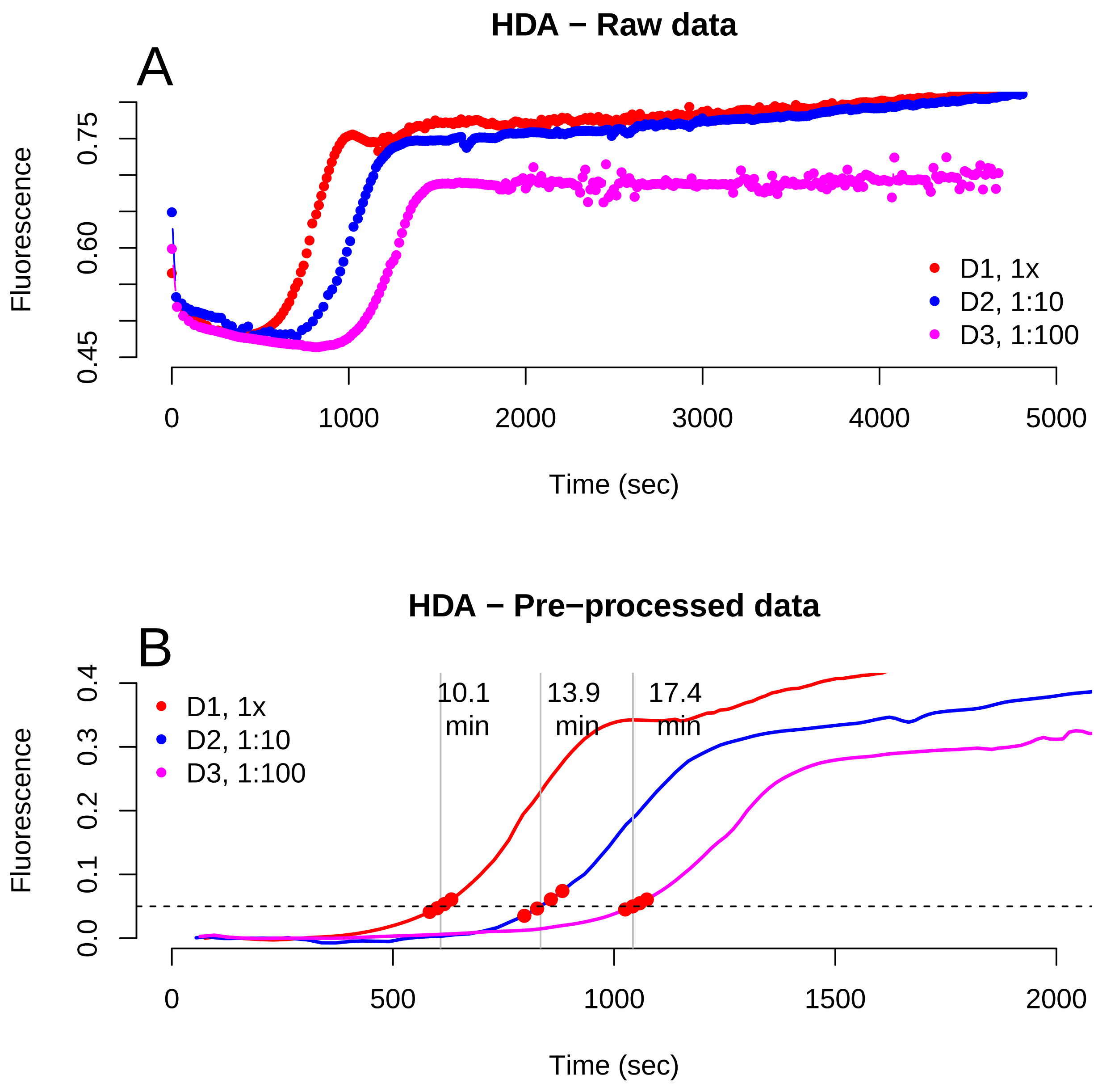


Figure S24: Helicase dependent amplification (HDA) of Vimentin (Vim). The VideoScan Platform was used to monitor the amplification. The HDA was performed at 65 degree Celsius. Three concentrations of input DNA (D1, D2, D3) were used. The amplification curves were smoothed by a moving average (windowsize 3) and base-lined by a a robust linear regression by computing MM-type regression estimator. The *th.cyc* function was used to determine the time required to reach the threshold level of 0.05 (–).

15 Amplification efficiency

Various influences alter amplification reactions. The amplification efficiency is controlled by a complex interaction of the intrinsic and extrinsic factors like reaction conditions, substrate consumption, primer dimmer formation and molecule specific reaction rates [24]. Some probe systems are perceived as bias-introducing. Therefore, qPCR reaction should be corrected based on the amplification efficiency [39, 52]. The amplification efficiency (AE) can be estimated from individual samples or a set of samples to compensate the presence of inhibitors and noise. Indirect methods use fitted mathematical models or estimate the AE from absolute fluorescence values [1, 2, 21, 22, 50, 45]. The *qpcR* has many functions included, which can be used for the indirect estimation of the AE. However, most commonly used is the "direct method" [21, 48]. Herein, the AE is estimated from dilution series of a template. The AE of a qPCR reaction is calculated from the slope of the standard curve (Equation S6).

$$AE = \frac{10^{(-1/m)}}{2} * 100 \tag{S6}$$

effcalc is used for the automatic calculation of the AE of a dilution series (Figure S25). An object of the class list contains the "Concentration", Cqs, deviation of the Cqs, "Coefficient of Variance" sequentially in the columns, the amplification efficiency (%) according to Equation S6, the results of the linear regression and the correlation test (Pearson) (Table S4). The *effcalc* has several options to enhance the plot. For example, it is possible to indicate the confidence interval (default CI = 95 %). Further options are described in the manual.

```r
# Load MBmca package (v. 0.0.3-3 or later)
require(MBmca)

# Create an graphic device for two empty plots.
par(mfrow = c(1, 2))
plot(NA, NA, xlim = c(1, 45), ylim = c(0.01, 1.1), xlab = "Cycles",
    ylab = "Fluorescence", main = "")
mtext("A", cex = 1.1, side = 3, adj = 0, font = 2)

# Create a sequence of 'targeted' Cq values (Cq.t) between 15
# and 34 cycles.

Cq.t <- rep(seq(15, 34, 3.5), 3)

# In-silico experiment set up: Define the levels for the
# decadic dilutions with concentrations from 100 to 0.001
# (six steps) as three replicates.

dilution <- rep(10^(2:-4), 3)

# Create an empty matrix for the results of the concentration
# dependent Cq values.

ma.out <- matrix(data = NA, nrow = 45, ncol = length(Cq.t))

# Use AmpSim to simulate amplification curves at different
# concentrations.  The simulation is performed with the
# addition of some noise. This generates unique
# (non-reproducible) amplification curves, even under
# identical parameter settings.

Cq.out <- vector()

# Simulate a qPCR reaction with AmpSim for 45 cycles and some
# noise.

for (i in 1L:18) {
    ma.out[1:45, i] <- AmpSim(cyc = c(1:45), b.eff = -50, bl = 0.001,
        ampl = 1, Cq = Cq.t[i], noise = TRUE, nnl = 0.02)[, 2]
    lines(1:45, ma.out[, i])
    tmpP <- mcaSmoother(1:45, ma.out[, i])
    # Calculate the pseudo Second Derivative Maximum (SDM) (Cq)
    # using the diffQ2 function from the MBmca package.
    Cq.tmp <- diffQ2(tmpP, inder = TRUE)$xTm1.2.D2[1]
    abline(v = Cq.tmp)
    Cq.out <- c(Cq.out, Cq.tmp)
}

# Assign the calculated Cqs to the corresponding
# concentrations.
tmp <- data.frame(dilution[1:6], Cq.out[1:6], Cq.out[7:12], Cq.out[13:18])

# Determine the amplification efficiency by using the effcalc
# function.
plot(effcalc(tmp[, 1], tmp[, 2:4]), CI = TRUE)
mtext("B", cex = 1.1, side = 3, adj = 0, font = 2)
```

Next we used *effcalc* to analyze the C54 data set from the *chipPCR* package. Herein, a qPCR Experiment for the amplification of MLC-2v using the VideoScan heating/cooling-unit was performed. To calculate the Cq it was necessary to pre-process the amplification curve data. One amplification curve contained a missing value (Figure S26 A) which was remove by the spline method in *CPP*. In addition, the data were baselined (linear model, robust MM-estimator) and smoothed by Savitzky-Golay Smoothing (Figure S26 B). The final analysis with the *effcalc* function showed that the amplification efficiency is circa 87.3 % for the gene MLC-2v in the VideoScan HCU (Figure S26 C). However, since only few measure points were tested in this experiment it just save to say that the hardware of the HCU works reliably.

```
require(MBmca)
par(las = 0, bty = "n", oma = c(0.5, 0.5, 0.5, 0.5))
par(fig = c(0, 0.5, 0, 1), new = TRUE)
plot(NA, NA, xlim = c(1, 55), ylim = c(0, 0.7), xlab = "Cycle",
    ylab = "refMFI", main = "Raw data")
just_line <- apply(C54[, c(2:4)], 2, function(y) lines(C54[,
    1], y))
mtext("A", cex = 1.2, side = 3, adj = 0, font = 2)

par(fig = c(0.5, 1, 0.5, 1), new = TRUE)
plot(NA, NA, xlim = c(1, 55), ylim = c(0, 0.55), xlab = "Cycle",
    ylab = "refMFI", main = "pre-processed data")
mtext("B", cex = 1.2, side = 3, adj = 0, font = 2)

D1 <- cbind(C54[1:35, 1], CPP(C54[1:35, 1], C54[1:35, 2], trans = TRUE,
    bg.range = c(1, 8))[["y.norm"]])
D2 <- cbind(C54[1:45, 1], CPP(C54[1:45, 1], C54[1:45, 3], trans = TRUE)[["y.norm"]])
D3 <- cbind(C54[1:55, 1], CPP(C54[1:55, 1], C54[1:55, 4], trans = TRUE)[["y.norm"]])

lines(D1, col = 1)
lines(D2, col = 2)
lines(D3, col = 3)

dilution <- c(1, 0.001, 1e-06)
Cq.D1 <- diffQ2(D1, inder = TRUE)[["xTm1.2.D2"]][1]

## Approximate and calculated Tm varri.
##           This is an expected behaviour
##
##           but the calculation should be confirmed with a plot
##           (see examples of diffQ).
## 15.638859833092433.74599536247882
## The distribution of the curve data indicates noise.
##
##           The data should be visually inspected with a plot
##           (see examples of diffQ).
```

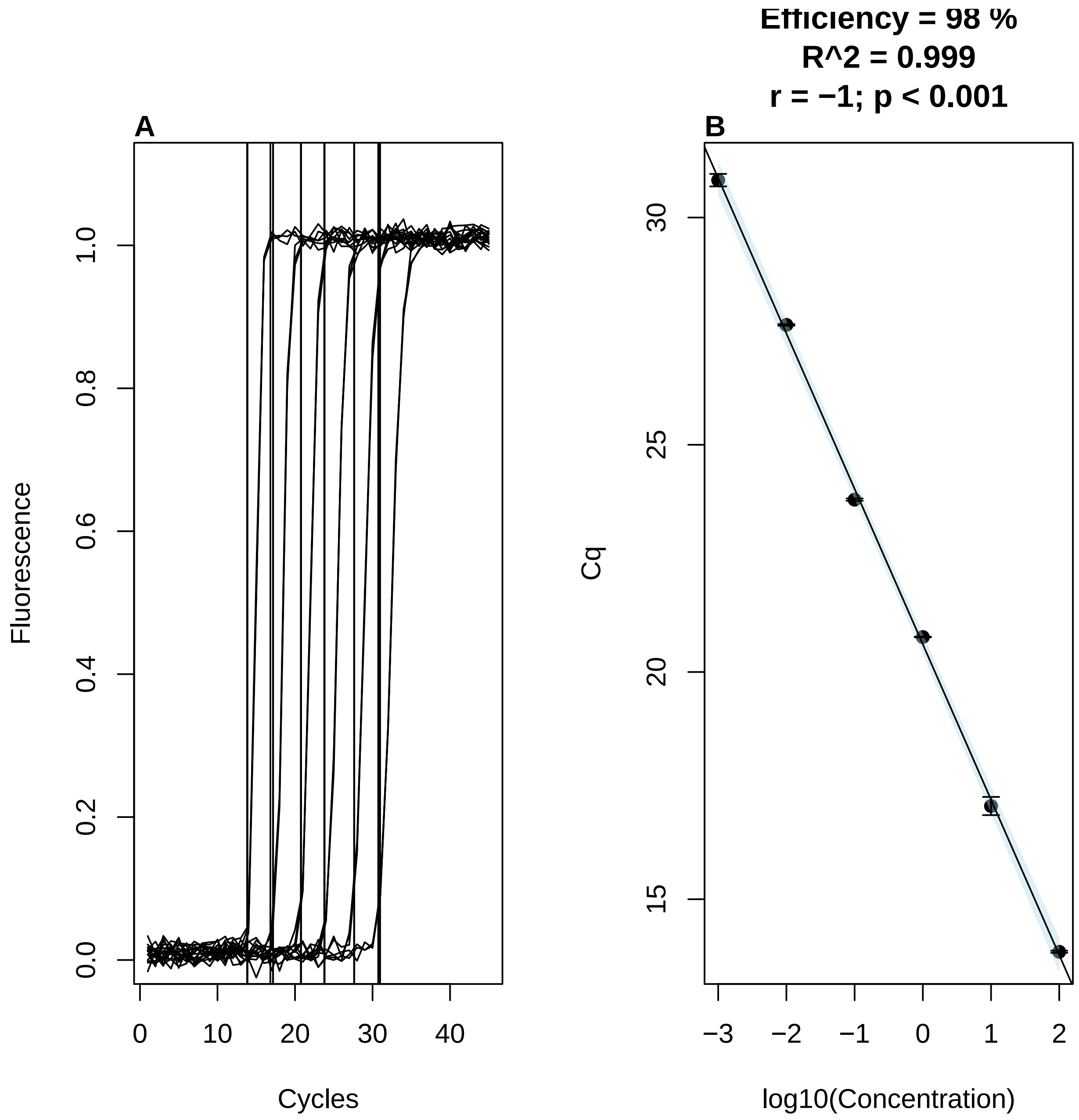


Figure S25: Amplification standard curve simulation and regression analysis. *(A) AmpSim* was used to synthesize a qPCR experiment of six dilutions (three replicates per dilution) standard samples. The Cqs were determined by the *SDM* method (solid black vertical lines). *(B)* effcalc was used to automatically perform a linear regression. The regression curve (–) was plotted as the decadic logarithm of input concentration versus the Cq. The 95% confidence interval is shown be the light-blue solid lines.

```
## Approximate and calculated Tm varri.
##             This is an expected behaviour
##
##             but the calculation should be confirmed with a plot
##             (see examples of diffQ).
## 12.139270996789612.307692307692310.3617975173511
## The distribution of the curve data indicates noise.
##
##             The data should be visually inspected with a plot
##             (see examples of diffQ).
## The distribution of the curve data indicates noise.
##
##             The data should be visually inspected with a plot
##             (see examples of diffQ).

Cq.D2 <- diffQ2(D2, inder = TRUE)[["xTm1.2.D2"]][1]

## Approximate and calculated Tm varri.
##             This is an expected behaviour
##
##             but the calculation should be confirmed with a plot
##             (see examples of diffQ).
## 16.4206537204211612.6709691596414
## The distribution of the curve data indicates noise.
##
##             The data should be visually inspected with a plot
##             (see examples of diffQ).
## Approximate and calculated Tm varri.
##             This is an expected behaviour
##
##             but the calculation should be confirmed with a plot
##             (see examples of diffQ).
## 6.5338668351225620 21.9375753419625

Cq.D3 <- diffQ2(D3, inder = TRUE)[["xTm1.2.D2"]][1]

## The Tm calculation (fit, adj.  R squared ~ 0.849, NRMSE ~ 0.088) is not optimal presumably due
to noisy data.
##
##                     Check raw melting curve (see examples of diffQ).

res.dil <- data.frame(dilution, rbind(Cq.D1, Cq.D2, Cq.D3))
par(fig = c(0.5, 1, 0, 0.5), new = TRUE)
plot(effcalc(res.dil[, 1], res.dil[, 2]))
```

	Concentration	Location (Mean)	Deviation (SD)	Coefficient of Variance (RSD [%])
1	0.00	10.36	0.00	0.00
2	-3.00	21.94	0.00	0.00
3	-6.00	35.15	0.00	0.00

Table S4: Output of the effcalc function.

In another example we used *effcalc* function to analyze the C60.amp data set from the *chipPCR* package. All data of the human genes Vimentin (Figure S27 A) and MLC-2v (Figure S27 B) were amplified in an Roche Light Cycler 1.5 and detected by the HRM dye EvaGreen in independent experiments. As shown in the code and Figure S27 it is possible to obtain a complete analysis with few commands. The amplification efficiencies for both qPCRs was higher than 94 % (Figure S27 C and D, Table S4).

```
colors <- rep(rainbow(7), each = 2)
par(mfrow = c(2, 2))

plot(NA, NA, xlim = c(0, 44), ylim = c(0, 6), xlab = "Cycles",
```

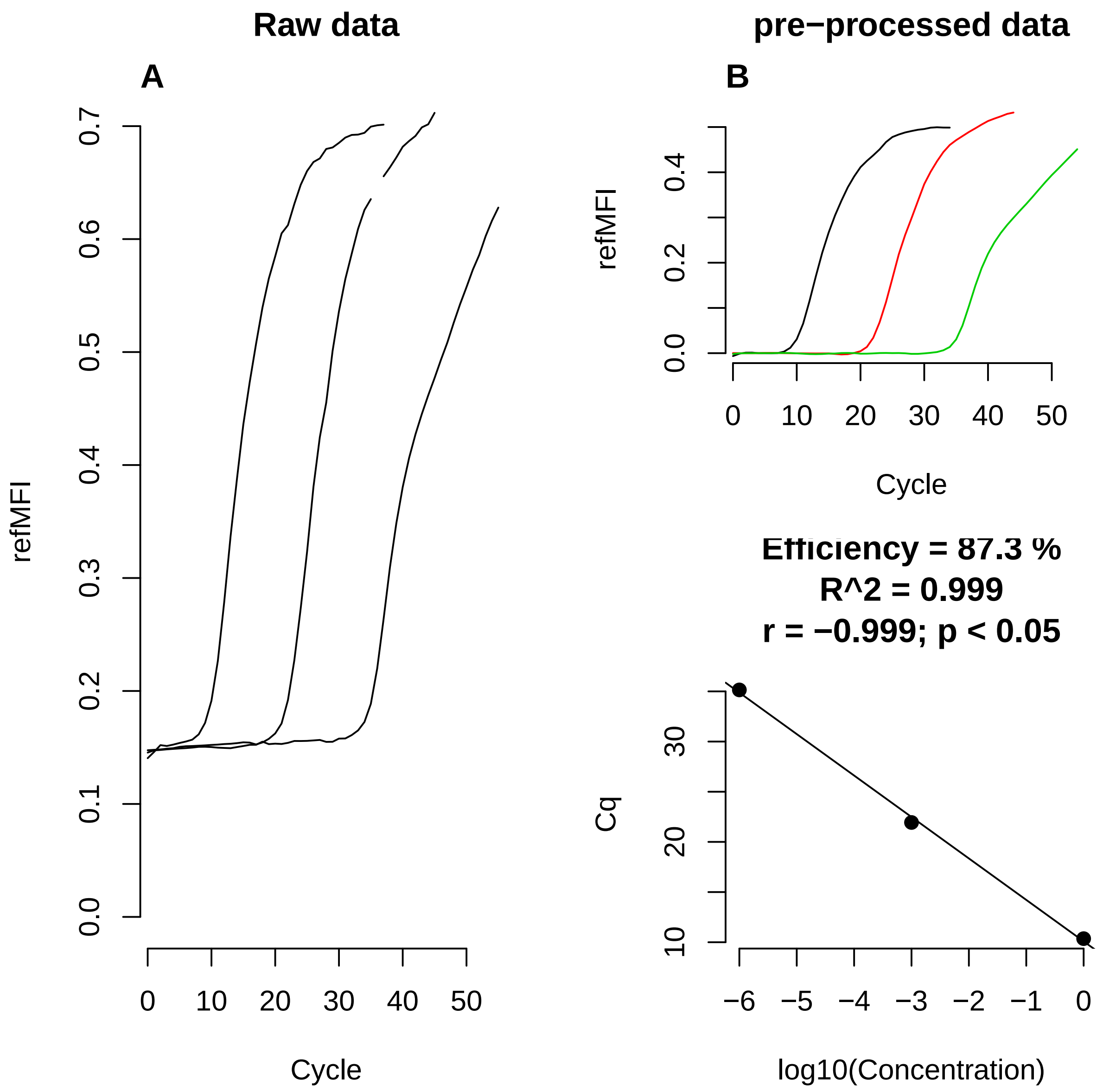


Figure S26: Calculation of the amplification efficiency. Data of a VideoScan HCU dilution experiment (C54 data set) were analyzed. *(A)* Visualization of the raw data. One of the three dilutions contains a missing value due to a sensor error. *(B, top panel)* The CPP function was used to baseline, to remove the missing value (–) and to smooth (–, –, –) the raw data. *(B, bottom panel)*. The Cqs (SDM) of the pre-processed data were calculated by diffQ2 (see main text) and analyzed with *effcalc*. The amplification efficiency approximately at 87.3 %.

```r
    ylab = "RFU")
legend(0, 6, colnames(C60.amp[, 4L:17]), ncol = 2, col = colors[1:14],
    pch = 19, bty = "n")
mtext("A", cex = 1.2, side = 3, adj = 0, font = 2)
SDM.vim <- sapply(4L:17, function(i) {
    lines(C60.amp[, 1], C60.amp[, i], col = colors[i - 3])
    SDM <- summary(inder(C60.amp[, 1], C60.amp[, i]), print = FALSE)[2]
})

plot(NA, NA, xlim = c(0, 44), ylim = c(0, 4), xlab = "Cycles",
    ylab = "RFU")
legend(0, 4, colnames(C60.amp[, 18L:31]), ncol = 2, col = colors[1:14],
    pch = 19, bty = "n")
mtext("B", cex = 1.2, side = 3, adj = 0, font = 2)
SDM.mlc2v <- sapply(18L:31, function(i) {
    lines(C60.amp[, 1], C60.amp[, i], col = colors[i - 17])
    SDM <- summary(inder(C60.amp[, 1], C60.amp[, i]), print = FALSE)[2]
})

# create vector of dilutions
dil <- sort(rep(10^(0L:-6), 2), TRUE)

res <- cbind(dil, SDM.vim, SDM.mlc2v)

plot(effcalc(res[, 1], res[, 2]))
mtext("C", cex = 1.2, side = 3, adj = 0, font = 2)

plot(effcalc(res[, 1], res[, 3]))
mtext("D", cex = 1.2, side = 3, adj = 0, font = 2)
```

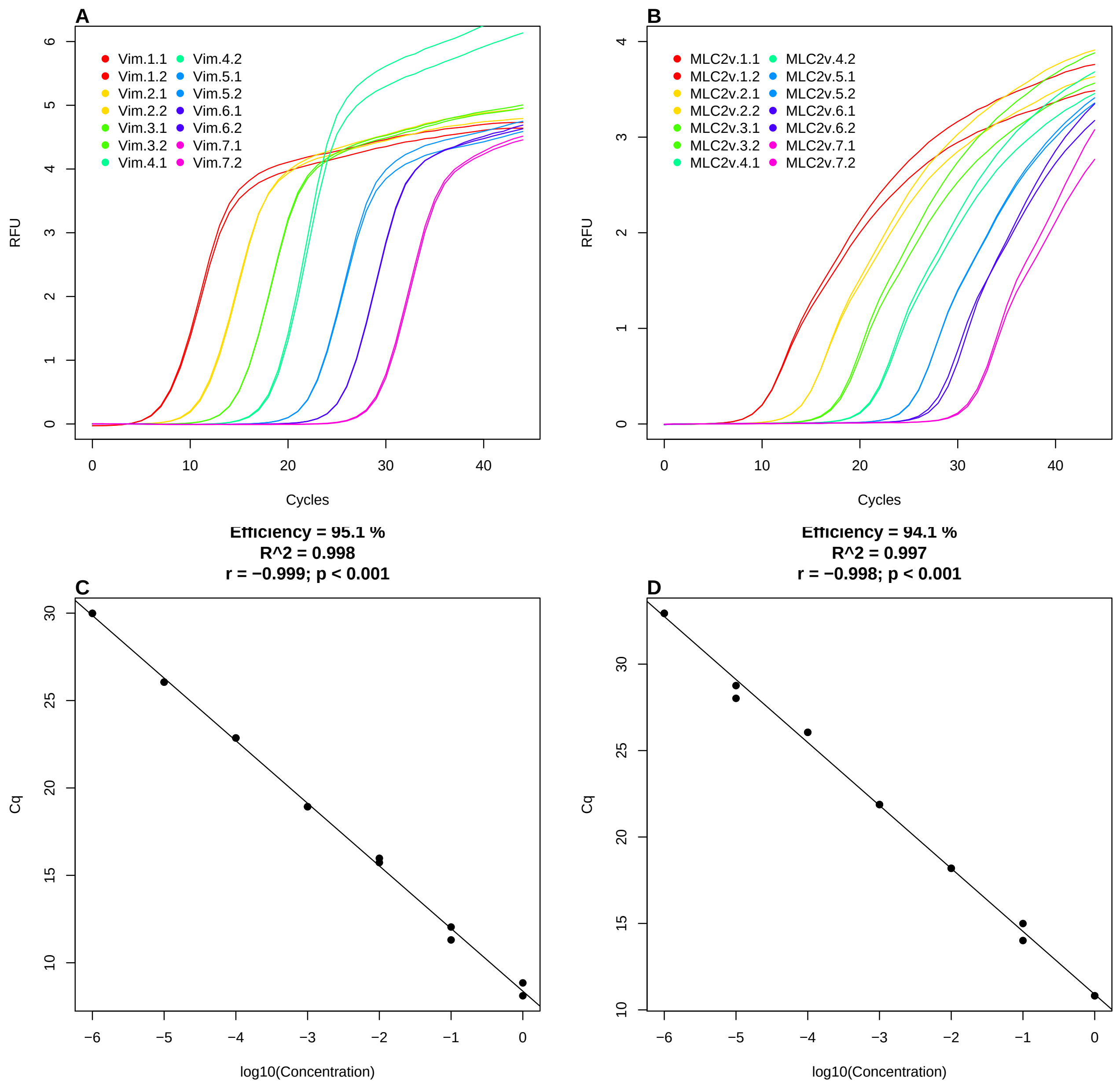


Figure S27: *lm.coefs* a function to compute linear model coefficients. The function is a convenient wrapper around few functions performing normal (least squares) and robust linear regression. If the robust linear regression is impossible, *lm.coefs* will perform linear regression using the least squares method. This function can be used to calculate the background of an amplification curve. The coefficients of the analysis can be used for a trend based correction of the entire data set.

16 Data sets

1. Data set: capillaryPCR:
   - Data set type: capillary convective PCR (ccPCR)
   - Description: The capillary convective PCR (ccPCR) is a modified device of the ccPCR system proposed by Chou et al. 2013.
   - Number of variables: 1844
   - Number of measurements: 10
2. Data set: C60.amp:
   - Data set type: standard qPCR - commercial thermo cyclers
   - Description: qPCR Experiment for the Amplification of MLC-2v and Vimentin (as decadic dilutions) Using the Roche Light Cycler 1.5.
   - Number of variables: 45
   - Number of measurements: 33
3. Data set: C60.melt:
   - Data set type: standard qPCR - commercial thermo cyclers
   - Description: Melt Curves MLC-2v and Vimentin for the qPCR experiment C60.amp using the Roche Light Cycler 1.5
   - Number of variables: 128
   - Number of measurements: 65
4. Data set: C126EG595:
   - Data set type: standard qPCR - commercial thermo cyclers
   - Description: A Quantitive PCR (qPCR) with the DNA binding dye (EvaGreen) (Mao et al. 2007) was performed in the Bio-Rad iQ5 thermo cycler. The cycle-dependent increase of the fluorescence was quantified at the elongation step (59.5 deg Celsius).
   - Number of variables: 40
   - Number of measurements: 97
5. Data set: C126EG685:
   - Data set type: standard qPCR - commercial thermo cyclers
   - Description: A Quantitive PCR (qPCR) with the DNA binding dye (EvaGreen) (Mao et al. 2007) was performed in the Bio-Rad iQ5 thermo cycler. The cycle-dependent increase of the fluorescence was quantified at the elongation step (68.5 deg Celsius).
   - Number of variables: 40
   - Number of measurements: 97
6. Data set: C127EGHP:
   - Data set type: standard qPCR - commercial thermo cyclers
   - Description: Quantitive PCR (qPCR) with a hydrolysis probe (Cy5/BHQ2) and DNA binding dye (EvaGreen) (Mao et al. 2007) performed in the Roche Light Cycler 1.5 thermo cycler.
   - Number of variables: 40
   - Number of measurements: 66
7. Data set: VIMCFX96.60: Human vimentin amplifcation curve data (measured during annealing phase at 60 deg Celsius) for 96 replicate samples in a Bio-Rad CFX96 thermo cycler. standard qPCR - commercial thermo cyclers 40 97
   - Data set type: standard qPCR - commercial thermo cyclers
   - Description: Human vimentin amplifcation curve data (measured during annealing phase at 60 deg Celsius) for 96 replicate samples in a Bio-Rad CFX96 thermo cycler.
   - Number of variables: 40

- Number of measurements: 97

8. Data set: VIMCFX96.69: Human vimentin amplifcation curve data (measured during elongatio phase at 69 deg Celsius) for 96 replicate samples in a Bio-Rad CFX96 thermo cycler. standard qPCR - commercial thermo cyclers 40 97
   - Data set type: standard qPCR - commercial thermo cyclers
   - Description: Human vimentin amplifcation curve data (measured during elongatio phase at 69 deg Celsius) for 96 replicate samples in a Bio-Rad CFX96 thermo cycler.
   - Number of variables: 40
   - Number of measurements: 97
9. Data set: VIMCFX96.meltcurve: Human vimentin melting curve data for 96 replicate samples in a Bio-Rad CFX96 thermo cycler. standard qPCR - commercial thermo cyclers 81 97
   - Data set type: standard qPCR - commercial thermo cyclers
   - Description: Human vimentin melting curve data for 96 replicate samples in a Bio-Rad CFX96 thermo cycler.
   - Number of variables: 81
   - Number of measurements: 97
10. Data set: VIMiQ5.595: Human vimentin amplifcation curve data (measured during annealing phase at 59.5 deg Celsius) for 96 replicate samples in a Bio-Rad iQ5 thermo cycler. standard qPCR - commercial thermo cyclers 40 97
    - Data set type: standard qPCR - commercial thermo cyclers
    - Description: Human vimentin amplifcation curve data (measured during annealing phase at 59.5 deg Celsius) for 96 replicate samples in a Bio-Rad iQ5 thermo cycler.
    - Number of variables: 40
    - Number of measurements: 97
11. Data set: VIMiQ5.685: Human vimentin amplifcation curve data (measured during elongatio phase at 68.5 deg Celsius) for 96 replicate samples in a Bio-Rad iQ5 thermo cycler. standard qPCR - commercial thermo cyclers 40 97
    - Data set type: standard qPCR - commercial thermo cyclers
    - Description: Human vimentin amplifcation curve data (measured during elongatio phase at 68.5 deg Celsius) for 96 replicate samples in a Bio-Rad iQ5 thermo cycler.
    - Number of variables: 40
    - Number of measurements: 97
12. Data set: VIMiQ5.melt: Human vimentin melting curve data for 96 replicate samples in a Bio-Rad iQ5 thermo cycler. standard qPCR - commercial thermo cyclers 81 97
    - Data set type: standard qPCR - commercial thermo cyclers
    - Description: Human vimentin melting curve data for 96 replicate samples in a Bio-Rad iQ5 thermo cycler.
    - Number of variables: 81
    - Number of measurements: 97
13. Data set: C54:
    - Data set type: standard qPCR - experimental thermo cyclers
    - Description: qPCR Experiment in the VideoScan heating/cooling-unit for the amplification using different concentrations of MLC-2v input cDNA quantities.
    - Number of variables: 56
    - Number of measurements: 4
14. Data set: CD74:
    - Data set type: standard qPCR - experimental thermo cyclers

- Description: Quantitive PCR with a hydrolysis probe and DNA binding dye (EvaGreen) for MLC-2v measured at 59.5 degree Celsius (annealing temperature), 68.5 degree Celsius (elongation temperature) and at 30 degree Celsius.
- Number of variables: 60
- Number of measurements: 19

15. Data set: Eff625:
    - Data set type: simulations
    - Description: Highly replicate number amplification curves with an approximate amplification efficiency of 62.5 percent at cycle number 18. The data were derived from a simulation such as the AmpSim function.
    - Number of variables: 40
    - Number of measurements: 1000
16. Data set: Eff750:
    - Data set type: simulations
    - Description: Highly replicate number amplification curves with an approximate amplification efficiency of 75 percent at cycle number 18. The data were derived from a simulation such as the AmpSim function.
    - Number of variables: 40
    - Number of measurements: 1000
17. Data set: Eff875:
    - Data set type: simulations
    - Description: Highly replicate number amplification curves with an approximate amplification efficiency of 87.5 percent at cycle number 18. The data were derived from a simulation such as the AmpSim function.
    - Number of variables: 40
    - Number of measurements: 1000
18. Data set: Eff1000:
    - Data set type: simulations
    - Description: Highly replicate number amplification curves with an approximate amplification efficiency of 100 percent at cycle number 18. The data were derived from a simulation such as the AmpSim function.
    - Number of variables: 40
    - Number of measurements: 1000
19. Data set: C67:
    - Data set type: Isothermal Amplifcation - Helicase Dependent Amplification
    - Description: A Helicase Dependent Amplification (HDA) of HRPT1 (Homo sapiens hypoxanthine phosphoribosyltransferase 1), performed at different input DNA quantities using the Bio-Rad iQ5 thermo cycler.
    - Number of variables: 43
    - Number of measurements: 6
20. Data set: CD75:
    - Data set type: Isothermal Amplifcation - Helicase Dependent Amplification
    - Description: Helicase Dependent Amplification in the VideoScan HCU of HRPT1 (Homo sapiens hypoxanthine phosphoribosyltransferase 1) measured at at 55, 60 or 65 degree Celsius.
    - Number of variables: 93
    - Number of measurements: 6
21. Data set: C81:

- Data set type: Isothermal Amplifcation - Helicase Dependent Amplification
- Description: Helicase Dependent Amplification (HDA) of pCNG1 using the VideoScan Platform (Roediger et al. (2013)). The HDA was performed at 65 degree Celsius. Two concentrations of input DNA were used.
- Number of variables: 351
- Number of measurements: 5

22. Data set: C85:

    - Data set type: Isothermal Amplifcation - Helicase Dependent Amplification
    - Description: Helicase Dependent Amplification (HDA) of Vimentin (Vim) in the VideoScan Platform (Roediger et al. (2013)). The HDA was performed at 65 degree Celsius with three dilutions of input DNA.
    - Number of variables: 301
    - Number of measurements: 7

17 Acknowledgment

Part of this work was funded by the BMBF InnoProfile-Projekt 03 IPT 611X. Grateful thanks belong to all authors of the cited **R** packages, the **R** community and **RKWard** developers. We would like to thank Claudia Deutschmann (Brandenburg University of Technology Cottbus - Senftenberg, Germany), Ralf Himmelreich (Fraunhofer ICT-IMM, Germany) and Katharina Klat (Darmstadt University of Applied Sciences, Germany) for the contribution of data.

Authors' contributions: SR conceived of the study, and participated in its design and coordination and wrote the manuscript. SR and MB jointly developed the software. All authors read and approved the final manuscript.

List of Figures

References

[1] Mariano J. Alvarez, Guillermo J. Vila-Ortiz, Mariano C. Salibe, Osvaldo L. Podhajcer, and Fernando J. Pitossi. Model based analysis of real-time PCR data from DNA binding dye protocols. *BMC Bioinformatics*, 8(1):85, March 2007.

[2] Anke Batsch, Andrea Noetel, Christian Fork, Anita Urban, Daliborka Lazic, Tina Lucas, Julia Pietsch, Andreas Lazar, Edgar Schömig, and Dirk Gründemann. Simultaneous fitting of real-time PCR data with efficiency of amplification modeled as gaussian function of target fluorescence. *BMC Bioinformatics*, 9(1):95, February 2008.

[3] Konstantin A. Blagodatskikh, Stefan Roediger, and Michal Burdukiewicz. *Importing real-time thermo cycler (qPCR) data from RDML format files*, September 2014.

[4] Stephen A. Bustin, Vladimir Benes, Jeremy A. Garson, Jan Hellemans, Jim Huggett, Mikael Kubista, Reinhold Mueller, Tania Nolan, Michael W. Pfaffl, Gregory L. Shipley, Jo Vandesompele, and Carl T. Wittwer. The MIQE guidelines: minimum information for publication of quantitative real-time PCR experiments. *Clinical Chemistry*, 55(4):611–622, April 2009.

[5] Chia-Chen Chang, Chien-Cheng Chen, Shih-Chung Wei, Hui-Hsin Lu, Yang-Hung Liang, and Chii-Wann Lin. Diagnostic devices for isothermal nucleic acid amplification. *Sensors (Basel, Switzerland)*, 12(6):8319–8337, 2012. PMID: 22969402 PMCID: PMC3436031.

[6] Wen Pin Chou, Ping Hei Chen, Ming Miao, Long Sheng Kuo, Shiou Hwei Yeh, and Pei Jer Chen. Rapid DNA amplification in a capillary tube by natural convection with a single isothermal heater. *BioTechniques*, 50(1):52–57, January 2011.

[7] Germund Dahlquist and Åke Björck. Numerical Methods in Scientific Computing. volume II. Society for Industrial and Applied Mathematics, 2008.

[8] Heidi Dvinge and Paul Bertone. HTqPCR: High-throughput analysis and visualization of quantitative real-time PCR data in R. *Bioinformatics*, 25(24):3325, 2009.

[9] Paul H. C. Eilers. A perfect smoother. *Analytical Chemistry*, 75(14):3631–3636, July 2003.

[10] Daniel N Frank. BARCRAWL and BARTAB: software tools for the design and implementation of barcoded primers for highly multiplexed DNA sequencing. *BMC Bioinformatics*, 10:362, 2009. PMID: 19874596 PMCID: PMC2777893.

[11] Nils Gehlenborg, Michael S. Noble, Gad Getz, Lynda Chin, and Peter J. Park. Nozzle: a report generation toolkit for data analysis pipelines. *Bioinformatics*, 29(8):1089–1091, April 2013.

[12] Daniel Gerhard, Melanie Bremer, and Christian Ritz. Estimating marginal properties of quantitative real-time PCR data using nonlinear mixed models. *Biometrics*, 70(1):247–254, March 2014.

[13] F E Harrell. Regression Modeling Strategies - with Applications to Linear Models, Logistic Regression, and Survival Analysis. Springer, New York, 2001.

[14] Lars-Henrik Heckmann, Peter B. Sørensen, Paul H. Krogh, and Jesper G. Sørensen. NORMA-gene: A simple and robust method for qPCR normalization based on target gene data. *BMC Bioinformatics*, 12(1):250, June 2011.

[15] Melanie A. Huntley, Jessica L. Larson, Christina Chaivorapol, Gabriel Becker, Michael Lawrence, Jason A. Hackney, and Joshua S. Kaminker. ReportingTools: an automated result processing and presentation toolkit for high-throughput genomic analyses. *Bioinformatics*, 29(24):3220–3221, December 2013.

[16] Alexandros Karatzoglou, Alexandros Smola, Kurt Hornik, and Achim Zeileis. kernlab - an S4 package for kernel methods in R. *Journal of Statistical Software*, 11(9):1–20, November 2004.

[17] John D. Kloke and Joseph W. McKean. Rfit: Rank-based Estimation for Linear Models. *The R Journal*, 4(2):57–64, December 2012.

[18] Roger Koenker. Censored quantile regression redux. *Journal of Statistical Software*, 27(6):1–25, July 2008.

[19] Lukasz Komsta. *outliers: Tests for outliers*, 2011. R package version 0.14.

[20] Alexey Larionov, Andreas Krause, and William Miller. A standard curve based method for relative real time PCR data processing. *BMC Bioinformatics*, 6(1):62, March 2005. PMID: 15780134.

[21] Weihong Liu and David A Saint. A new quantitative method of real time reverse transcription polymerase chain reaction assay based on simulation of polymerase chain reaction kinetics. *Analytical Biochemistry*, 302(1):52–59, March 2002. PMID: 11846375.

[22] Izaskun Mallona, Julia Weiss, and Marcos Egea-Cortines. pcrEfficiency: a web tool for PCR amplification efficiency prediction. *BMC Bioinformatics*, 12(1):404, October 2011.

[23] Matthew N. McCall, Helene R. McMurray, Hartmut Land, and Anthony Almudevar. On non-detects in qPCR data. *Bioinformatics (Oxford, England)*, April 2014.

[24] Sarika Mehra and Wei-Shou Hu. A kinetic model of quantitative real-time polymerase chain reaction. *Biotechnology and bioengineering*, 91(7):848–860, September 2005. PMID: 15986490.

[25] Zhenghua Nie and Jeffrey S Racine. The crs Package: Nonparametric Regression Splines for Continuous and Categorical Predictors. *The R Journal*, 4(2):48–56, December 2012.

[26] Stephan Pabinger, Stefan Rödiger, Albert Kriegner, Klemens Vierlinger, and Andreas Weinhäusel. A survey of tools for the analysis of quantitative PCR (qPCR) data. *Biomolecular Detection and Quantification*, accepted.

[27] Stuart N Peirson, Jason N Butler, and Russell G Foster. Experimental validation of novel and conventional approaches to quantitative real-time PCR data analysis. *Nucleic Acids Research*, 31(14):e73, July 2003. PMID: 12853650 PMCID: PMC167648.

[28] James R. Perkins, John M. Dawes, Steve B. McMahon, David LH Bennett, Christine Orengo, and Matthias Kohl. ReadqPCR and NormqPCR: R packages for the reading, quality checking and normalisation of RT-qPCR quantification cycle (cq) data. *BMC Genomics*, 13(1):296, July 2012. PMID: 22748112.

[29] R Core Team. R: A language and environment for statistical computing. Vienna, Austria, 2013.

[30] R Development Core Team. *R Data Import/Export*. R Foundation for Statistical Computing, Vienna, Austria, 2014. ISBN 3-900051-10-0.

[31] Xiayu Rao, Dejian Lai, and Xuelin Huang. A new method for quantitative real-time polymerase chain reaction data analysis. *Journal of computational biology: a journal of computational molecular cell biology*, 20(9):703–711, September 2013. PMID: 23841653 PMCID: PMC3762066.

[32] C. Ritz and J. C. Streibig. Bioassay analysis using r. *Journal of Statistical Software*, 12, 2005.

[33] Christian Ritz and Andrej-Nikolai Spiess. qpcR: an R package for sigmoidal model selection in quantitative real-time polymerase chain reaction analysis. *Bioinformatics*, 24(13):1549–1551, July 2008. PMID: 18482995.

[34] Stefan Rödiger, Alexander Böhm, and Ingolf Schimke. Surface melting curve analysis with R. *The R Journal*, 5(2):37–53, December 2013.

[35] Stefan Rödiger, Thomas Friedrichsmeier, Prasenjit Kapat, and Meik Michalke. RKWard: a comprehensive graphical user interface and integrated development environment for statistical analysis with R. *Journal of Statistical Software*, 49(9):1–34, 2012.

[36] Stefan Rödiger, Peter Schierack, Alexander Böhm, Jörg Nitschke, Werner Lehmann, and Christian Schröder. VideoScan - a microscope imaging technology platform for the multiplex real-time PCR. *qPCR & NGS 2013 Prodceedings; ISBN 9783000410246*, March 2013.

[37] RStudio and Inc. *shiny: Web Application Framework for R*, 2014. R package version 0.10.1.

[38] J M Ruijter, C Ramakers, W M H Hoogaars, Y Karlen, O Bakker, M J B van den Hoff, and A F M Moorman. Amplification efficiency: linking baseline and bias in the analysis of quantitative PCR data. *Nucleic Acids Research*, 37(6):e45, April 2009. PMID: 19237396 PMCID: PMC2665230.

[39] Jan M. Ruijter, Peter Lorenz, Jari M. Tuomi, Michael Hecker, and Maurice J. B. van den Hoff. Fluorescent-increase kinetics of different fluorescent reporters used for qPCR depend on monitoring chemistry, targeted sequence, type of DNA input and PCR efficiency. *Microchimica Acta*, pages 1–8, 2014.

[40] Jan M Ruijter, Michael W Pfaffl, Sheng Zhao, Andrej N Spiess, Gregory Boggy, Jochen Blom, Robert G Rutledge, Davide Sisti, Antoon Lievens, Katleen De Preter, Stefaan Derveaux, Jan Hellemans, and Jo Vandesompele. Evaluation of qPCR curve analysis methods for reliable biomarker discovery: bias, resolution, precision, and implications. *Methods (San Diego, Calif.)*, 59(1):32–46, January 2013. PMID: 22975077.

[41] Stefan Rödiger, Claudia Liebsch, Carsten Schmidt, Werner Lehmann, Ute Resch-Genger, Uwe Schedler, and Peter Schierack. Nucleic acid detection based on the use of microbeads: a review. *Microchimica Acta*, 181(11-12):1151–1168, August 2014.

[42] Stefan Rödiger, Peter Schierack, Alexander Böhm, Jörg Nitschke, Ingo Berger, Ulrike Frömmel, Carsten Schmidt, Mirko Ruhland, Ingolf Schimke, Dirk Roggenbuck, Werner Lehmann, and Christian Schröder. 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, 2013.

[43] Abraham. Savitzky and M. J. E. Golay. Smoothing and differentiation of data by simplified least squares procedures. *Analytical Chemistry*, 36(8):1627–1639, July 1964.

[44] Eric B Shain and John M Clemens. A new method for robust quantitative and qualitative analysis of real-time PCR. *Nucleic Acids Research*, 36(14):e91, August 2008. PMID: 18603594 PMCID: PMC2504305.

[45] Marjo V. Smith, Chris R. Miller, Michael Kohn, Nigel J. Walker, and Chris J. Portier. Absolute estimation of initial concentrations of amplicon in a real-time RT-PCR process. *BMC Bioinformatics*, 8(1):409, October 2007.

[46] Andrej-Nikolai Spiess. *qpcR: Modelling and analysis of real-time PCR data*, 2014. R package version 1.4-0.

[47] Andrej-Nikolai Spiess, Caroline Feig, and Christian Ritz. Highly accurate sigmoidal fitting of real-time PCR data by introducing a parameter for asymmetry. *BMC Bioinformatics*, 9(1):221, April 2008.

[48] Anders Ståålberg, Pierre Aman, Börje Ridell, Petter Mostad, and Mikael Kubista. Quantitative real-time PCR method for detection of b-lymphocyte monoclonality by comparison of kappa and lambda immunoglobulin light chain expression. *Clinical Chemistry*, 49(1):51–59, January 2003. PMID: 12507960.

[49] Joel Tellinghuisen and Andrej-Nikolai Spiess. Comparing real-time quantitative polymerase chain reaction analysis methods for precision, linearity, and accuracy of estimating amplification efficiency. *Analytical Biochemistry*, 449:76–82, March 2014. PMID: 24365068.

[50] Ales Tichopad, Michael Dilger, Gerhard Schwarz, and Michael W Pfaffl. Standardized determination of real-time PCR efficiency from a single reaction set-up. *Nucleic Acids Research*, 31(20):e122, October 2003. PMID: 14530455 PMCID: PMC219490.

[51] Valentin Todorov and Peter Filzmoser. An Object-Oriented Framework for Robust Multivariate Analysis. *Journal of Statistical Software*, 32(3):1–47, October 2009.

[52] Jari Michael Tuomi, Frans Voorbraak, Douglas L Jones, and Jan M Ruijter. Bias in the Cq value observed with hydrolysis probe based quantitative PCR can be corrected with the estimated PCR efficiency value. *Methods (San Diego, Calif.)*, 50(4):313–322, April 2010. PMID: 20138998.

[53] Fernando Tusell. Kalman Filtering in R. *Journal of Statistical Software*, 39(2):1–27, March 2011.

[54] Daryl Waggott, Kenneth Chu, Shaoming Yin, Bradly G. Wouters, Fei-Fei Liu, and Paul C. Boutros. NanoStringNorm: An extensible R package for the pre-processing of NanoString mRNA and miRNA data. *Bioinformatics*, page bts188, April 2012.

[55] Jochen Wilhelm, Alfred Pingoud, and Meinhard Hahn. Real-time PCR-based method for the estimation of genome sizes. *Nucleic Acids Research*, 31(10):e56, May 2003. PMID: 12736322 PMCID: PMC156059.

[56] Jitao David Zhang, Rudolf Biczok, and Markus Ruschhaupt. *ddCt: The ddCt Algorithm for the Analysis of Quantitative Real-Time PCR (qRT-PCR)*, 2010.

[57] Xiaohua Douglas Zhang and Zhaozhi Zhang. displayHTS: a r package for displaying data and results from high-throughput screening experiments. *Bioinformatics*, 29(6):794–796, March 2013.

[58] Sheng Zhao and Russell D Fernald. Comprehensive algorithm for quantitative real-time polymerase chain reaction. *Journal of computational biology: a journal of computational molecular cell biology*, 12(8):1047–1064, October 2005. PMID: 16241897 PMCID: PMC2716216.