Stefan Rödiger*, Michał Burdukiewicz and Peter Schierack

Supplement to: "chipPCR: an R Package to Pre-Process Raw Data of Amplification Curves"

Contents

Abstract

Background: Both the quantitative real-time polymerase chain reaction (qPCR) and isothermal amplification are standard methods used for the quantification of nucleic acids (DNA, RNA). Numerous real-time read-out technologies with different technical platforms have been developed so far. However, analysis of amplification curves consists of cascaded steps implemented in a similar manner across all existing technological platforms. Despite the growing interest in amplification-based techniques, there are only few open source tools for pre-processing real-time amplification data. The availability of a software for pre-processing raw amplification data is mandatory in different scenarios, for example during the development, optimization and improvement of the functionality of instruments.

Results and Conclusion: *chipPCR* is a versatile **R** package for pre-processing (e.g., imputation of missing values, smoothing) and quality analysis of raw data for amplification curves coming from conventional quantitative polymerase chain reactions (qPCR), and quantitative isothermal amplification (qIA) reactions. The package contains datasets, 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. The structure of the packages is amenable for integration to Web-based and standalone *shiny* applications.

1 Availability, requirements and setting up a working environment

Convention: According to the MIQE guidelines ("Minimum Information for publication of Quantitative real-time PCR Experiments" (Bustin et al.*, 2009)) is the threshold cycle (Ct) referred to as quantification cycle (Cq). We use the expression Cq exclusively regardless of the amplification method and mathematical principle. The abbreviations MFI, RFU and refMFI refer to arbitrary units of mean fluorescence intensities.*

The software is available in an **R** environment or through web browser applications:

- 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 *S4* class system (see *methods* package) to separate the interface and the implementation because, unlike the **R**'s *S3* class system, it requires the explicit declaration of classes and the inheritance and relationship for each class or method. Therefore, the number and types of objects in slots in an instance of a class have to be established at the time of the class definition. Objects belonging to the class are validated against this definition and have to fulfill it at any time. *setGeneric* declares generic functions. *S4* methods are declared by calls to *setMethod* together with the name of generics and signatures of the arguments. Signatures are used for identification of classes of one or more arguments of the methods.

S4 classes therefore require a higher development effort than *S3* classes, but offer more stringent control over the contents of created objects. Additional information (e.g., results, parameters) can also be included. *S4* assures better control on the object structure and chosen method dispatch (Karatzoglou *et al.*, 2004).

For high-throughput capability, we avoided loops in the core structures of the *chipPCR* package and partially used parallel computing (*smoother* function) to keep the code fast. This package supports the use of most popular R packages, thus providing a communication layer required for parallel computing: *parallel* and *snow*.

This vignette can be viewed from **R** using command: vignette("chipPCR"). All experimental details for the datasets are described in the *chipPCR* package manual and in the citations. To start an analysis it is required to choose a dataset (as shown 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."))
```

All datasets used in the following examples can be loaded similarly. *chipPCR* relies on the **R** environment, and dedicated **R** packages (e.g., *RDML*) as default data format and standard import and export formats (Perkins *et al.*, 2012; Blagodatskikh *et al.*, 2014; R Development Core Team, 2014).

	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.

Graphical user interfaces (GUIs) are important to make software usable for researchers not fluent in **R**. Several **R** GUI projects have been proposed (Rödiger *et al.*, 2012). Selected functionality of *chipPCR* originates from **RKWard** GUI plugins (Pabinger *et al.*, 2014). The *shiny* framework (RStudio and Inc., 2014) can be used to build and deploy GUIs for the desktop or as services for interactive web applications on servers. Prerequisites are an installation of **R**, with installed *shiny* and *chipPCR* packages and a modern web browser. It is possible to run a GUI in a web browser on a local machine without an Internet connection. Ad-blocking software may cause malfunctions and should be turned off. *shiny* enables the building of plugin-like applications with highly customizable widgets (e.g., sliders) for an efficient extension. *shiny* applications can be updated live and in an interactive manner. The user interfaces can be built entirely using **R** and operates in any **R** environment (cross-platform). The functions *AmpSim*, *th.cyc*, *bg.max* and *amptester* are a part of *shiny* GUIs. Examples and case studies for the mentioned functions are presented in the following sections.

2 Pre-processing raw data for DNA amplification plots

Quantitative polymerase chain reaction (qPCR) and quantitative isothermal amplification (qIA) are standard methods used for amplifying nucleic acids (e.g., genomic DNA, copy DNA) (Pabinger *et al.*, 2014). The Taguchi methods provide general optimization frameworks used in engineering optimization processes and other related disciplines. However, applications include PCR application too (Cobb and Clarkson, 1994; Thanakiatkrai and Welch, 2012). Basically, it is possible to determine the optimal conditions for PCR-based assays. A software implementation for **R** is available in the *qualityTools* package (Roth, 2012). Recently amplification methods with continuous temperature gradients (e.g., microfluidics, capillary convective PCR (ccPCR)) emerged (Chou *et al.*, 2011; Rödiger *et al.*, 2014; Spiess *et al.*, 2014). Isothermal amplification is a monocyclic reaction at a constant temperature. Conversely, PCR is a polycyclic reaction with repeated thermal cycling condition steps (denaturation, annealing, elongation) and measurements at discrete cycle steps. The curve shape of isothermal amplification reactions do not necessarily follow an 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 is usually higher than in qPCR experiments. All these amplification methods are used in different real-time monitoring technologies, such as our previously reported VideoScan technology (Rödiger *et al.*, 2013a), microfluidic systems, point-of-care devices, microbead-chip technologies and commercial real-time thermo cyclers (Chang *et al.*, 2012; Rödiger *et al.*, 2013c, 2014). Real-time technologies enable the quantification of nucleic acids by calculation of specific curve parameters like the quantification cycle (Cq) and the amplification efficiency (AE) (Ruijter *et al.*, 2013; Tellinghuisen and Spiess, 2014).

The data quality of experimental instruments is often not suitable for end-user analysis and presentation. Moreover, analysis of raw data may lead to misinterpretations and false performance estimates under certain conditions. Therefore, novel technologies usually depend on software for pre-procession of 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), noise reduction and data quality management. Noise is challenging because derivative processes as used for "cycle of quantification" methods (e.g., Second Derivative Maximum method) leads to an amplification of noise (Larionov *et al.*, 2005; Tuomi *et al.*, 2010; Rödiger *et al.*, 2013b; Ruijter *et al.*, 2013; Tellinghuisen and Spiess, 2014). Pre-processing algorithms remove stochastic errors and artefacts (e.g., noise, photo-bleaching effects, degassing effects, different signal levels) as illustrated in Figure S1. However, misinterpretations are more likely if arbitrary manual corrections are not performed. A manual alteration is in contradiction to reproducible research. In particular, open source scientific software and the associated input and output data are central structural elements to enable recomputability and reproducibility of results (Blanton and Lenhardt, 2014; Jacobs *et al.*, 2014; Stodden and Miguez, 2014).

R is widely used for the analysis of qPCR data. Most **R** packages focus on the read-in, processing and post-processing of datasets originating from commercial qPCR systems. The 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 (Dvinge and Bertone, 2009; Zhang *et al.*, 2010; Heckmann *et al.*, 2011; Perkins *et al.*, 2012; Gehlenborg *et al.*, 2013; Huntley *et al.*, 2013; Zhang and Zhang, 2013; McCall *et al.*, 2014; Pabinger *et al.*, 2014). However, there is no **R** package for pre-processing and quality analysis of raw data for amplification curves. This need also applies to other existing software solutions (compare (Pabinger *et al.*, 2014)). Pre-processing in most commercial cyclers is a "black box" model, where inner subroutines are not available for inspection. This approach limits reproduction of analysis on other platforms and introduces difficulties for transfer of experimental design and setup or for the adequate use of statistical tools. Moreover, it is desirable to set up work-flows in an open environment, which enables downstream analyses and offers powerful tools for data visualizations and automatic report generation.

The *chipPCR* package was developed to automatize pre-processing, ease data analysis/visualization and offer a quality control for the statistical data analysis of qPCR or qIA experiments (see Section 3). *chipPCR* is primarily targeted at developers of novel systems. Nonetheless, users who process raw data of commercial systems can also utilize its functionalities.

```r
# Use AmpSim to generate amplification curves with 40 cycles
# and different Cq's.
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)
```

```r
# 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")
```

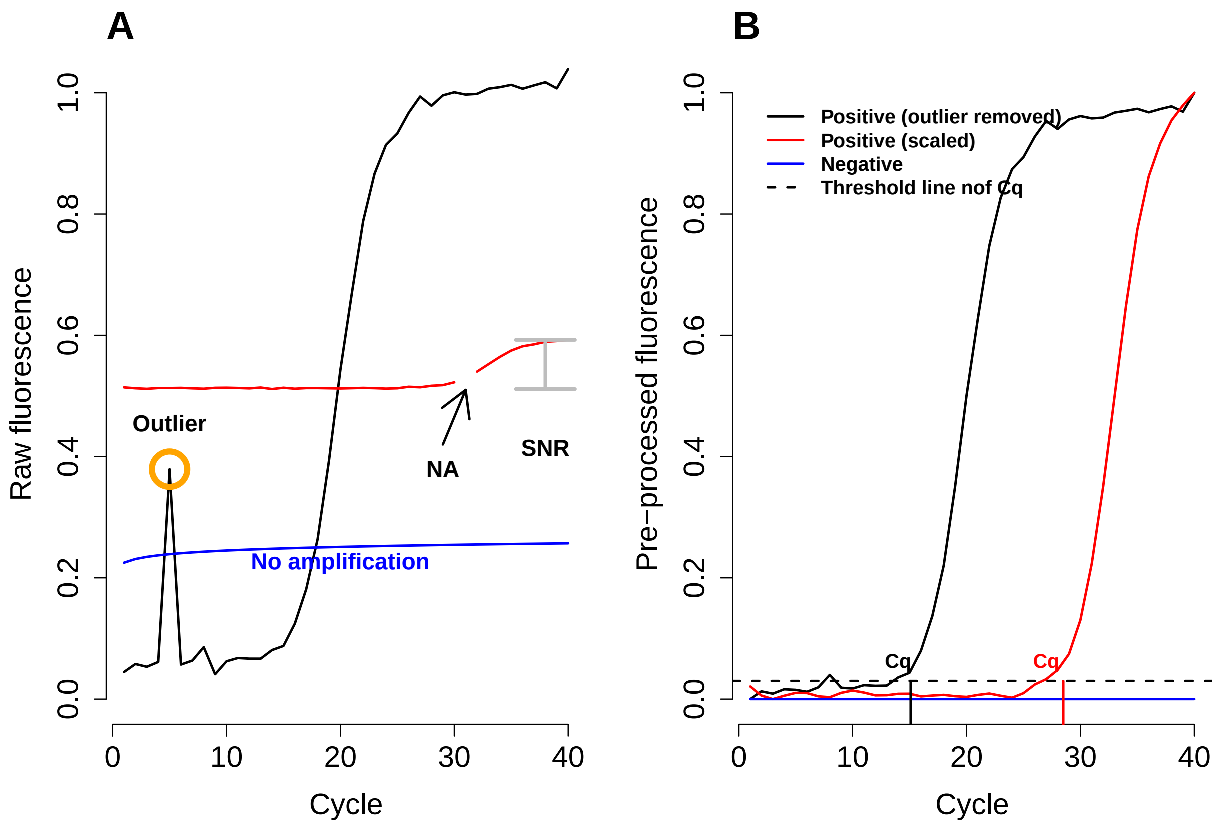


Figure S1: Analysis and interpretation of real-time amplification curves. *(A)* Before procession: The fluorescence values are plotted against the cycle, which results in sigmoidal shaped amplification curves (–, –). Measurements may occasionally contain missing values ("NA", –) and outliers (orange circle, –), due to noise introduced by the detection system or sensor errors. 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) is quantifiable as signal-to-noise ratio (SNR). The SNR between different samples (e.g., – and –) can vary. For interpretation it is recommended to normalize the data. Negative samples (–) need to be identified automatically. *(B)* Pre-processed raw data, where NAs were imputed and the noise slightly removed. The curves were adjusted to have the same baseline and plateau level. The quantification cycles (Cq) of the positive reactions are determined in the exponential phase ("threshold method" is used in this example). Negative samples are automatically set to zero.

3 Functions of the chipPCR package

The main functions of the *chipPCR* package are:

- *AmpSim*: simulatea S-shaped amplification plots based on a 5-parameter model accompanied by *AmpSim.gui* a *shiny* GUI, for *AmpSim*,
- *bg.max*: detects the start and end of an amplification reaction,
- *CPP*: easily accesses several pre-processing functions,
- *fixNA*: imputes missing values in a data column,
- *inder*: interpolates first and second derivatives interpolation using the five-point stencil (accompanied by *rounder* function),
- *MFIaggr*: analyzes a bulk of replicates of an amplification reaction,
- *smoother*: smoothens the data for curve plotting by different methods (e.g., moving average, Savitzky-Golay smoothing).

Additionally, further auxiliary functions are:

- *amptester*: detects the start and end of an amplification curve,
- *effcalc*: calculates the amplification efficiency,
- *humanrater*: rates curves using a graphical interface,
- *lm.coefs*: computes linear model coefficients,
- *normalizer*: normalizes data between a user defined range,
- *plotCurves*: plots many curves on one plot in separate cells allowing for quick assessment,
- *th.cyc*: calculates the number of cycles for which the fluorescence reporter signal exceeds a defined threshold, called the threshold cycle (Cq),

These auxillary functions are used for post-processing (e.g., Cq calculation, AE calculation) and quality analysis. Here we provide more information on the functionality of the chipPCR package. Selected functionality is used in the *RDML* (Blagodatskikh *et al.*, 2014) package. Most of the functions are central elements of other *chipPCR* functions. For example, *fixNA* is embedded in most functions to prevent errors due to missing values.

4 *AmpSim* - a function for simulating amplification curves

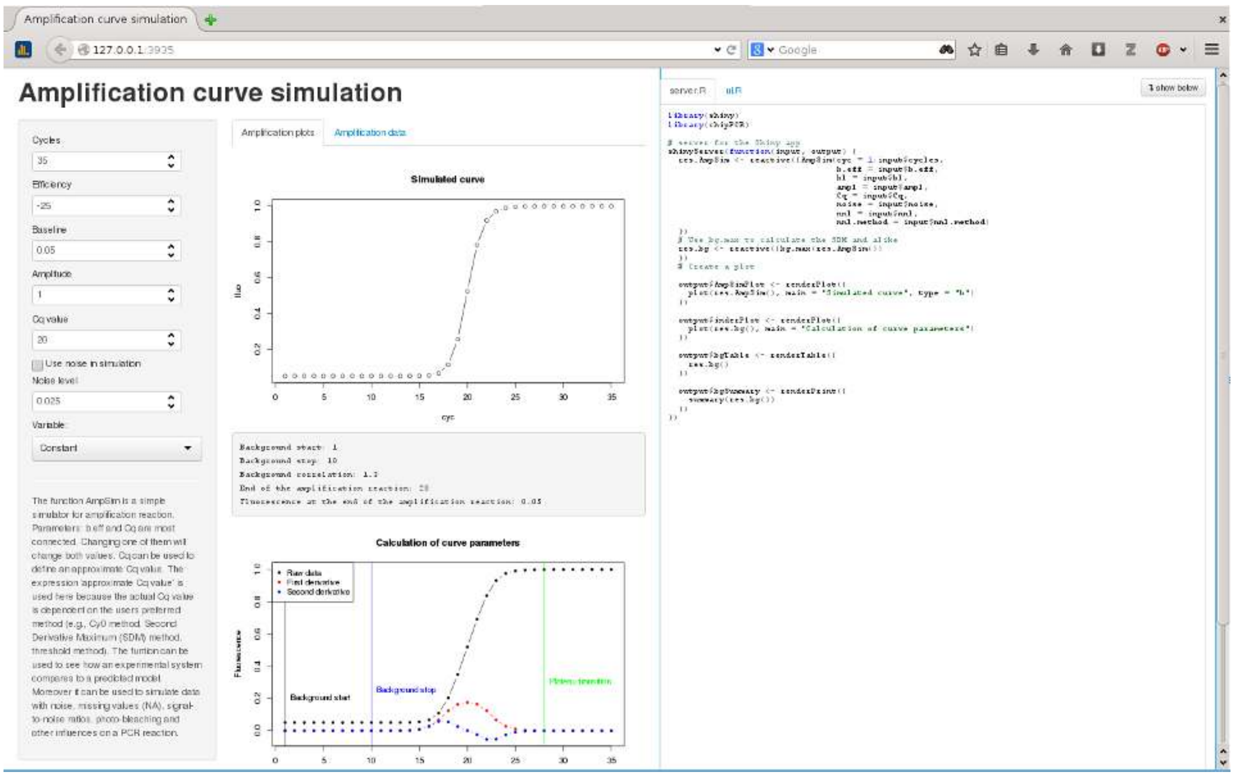
The *AmpSim* function simulates amplification reactions. Use cases include teaching, algorithm testing or the comparison of an experimental system to the predicted ("optimal") model. *AmpSim* uses a 5-parameter model (Equation S1), which is commonly used for the simulation of amplification curves (Ritz and Spiess, 2008; Spiess *et al.*, 2008; Gerhard *et al.*, 2014).

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

AmpSim has the intrinsic property to generate unique results if the *noise* parameter is set to $TRUE$. This is due to the addition of normally distributed noise (as per *rnorm* function from *stats* package), for identically replicated noisy dataset random seed (for example *set.seed(123)*). The amplification curves of Figure S26 *A* were generated with the same starting parameter of *AmpSim* with some noise added. *AmpSim.gui* is a *shiny* GUI (RStudio and Inc., 2014) implementation for *AmpSim*. A GUI for the simulation and analysis of amplification reactions can be invoked by pasting the following code snippet in an **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 web browser.
runApp(paste(find.package("chipPCR")[1], "/AmpSim.gui", sep = ""))

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

The function opens a *chipPCR* webpage in a default web browser (Figure S2). All parameters of the *AmpSim* function may be set in the left part of the interface. In addition, the GUI shows some information calculated by the *bg.max* function in a summary field and a plot below the simulated amplification curve.



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: 1.2
End of the amplification reaction: 28
Fluorescence at the end of the amplification reaction: 0.05
```

```r
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 web 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 shown in the right panel. All parameters (e.g., Cq value, baseline) of the *AmpSim* function are accessible. *(Bottom)* Additionally, *AmpSim.gui* shows the plot output and the textual results of the *bg.max* function.

AmpSim has several parameters controlling simulation of amplification curves. $b.eff$ and Cq are strongly connected. Thus changing one of them changes both values. Cq is used to define an approximate Cq value. The expression "approximate Cq" value is used because the calculated Cq value varies depending on the preferred Cq quantification method (e.g., second derivative maximum (SDM) method, threshold method). *AmpSim* is 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).

```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

    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])
})
```

AmpSim was used to illustrate the use of *inder* (Figure S20), the *fixNA* (Figure S13) and the *smoother* (Figure S15) functions.

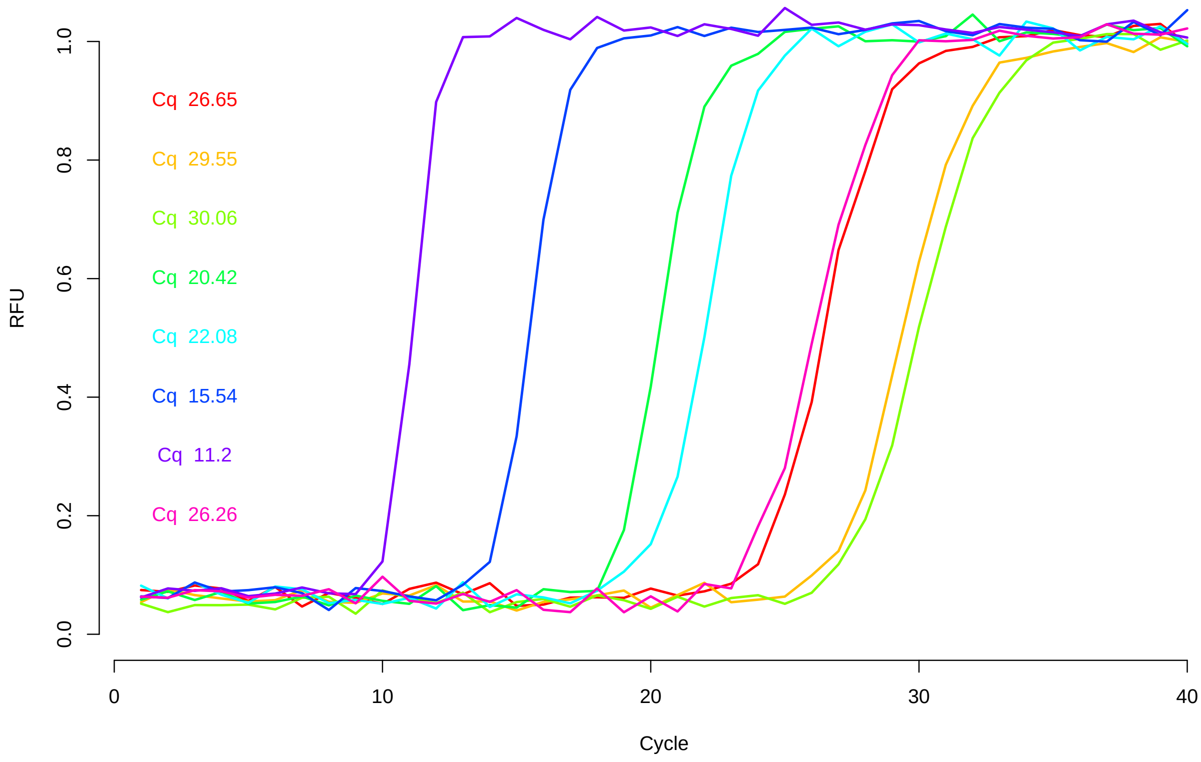
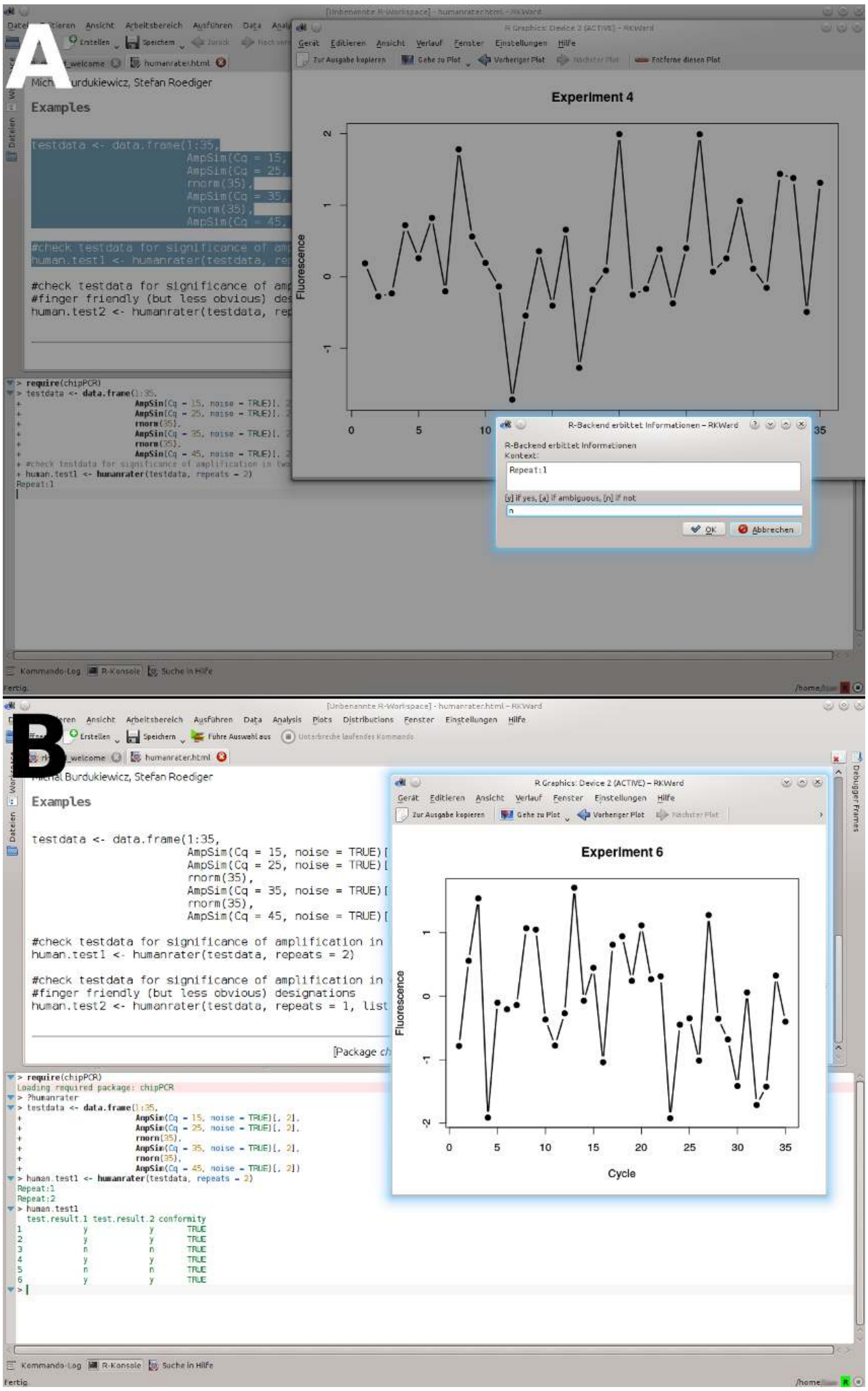


Figure S3: The amplification curves were generated with the *AmpSim* function. All Cqs are unique since random values were added to the starting Cq of 25. The parameter $noise = 0.03$ adds some scatter to the data used for plotting the amplification curves.

5 Single-blinded, randomized judging of amplification curves

Humans show bias towards interpreting data for a particular outcome. A single-blinded and randomized experiment aims to reduce bias in the results. We developed the *humanrater* function to evaluate the quality of curve data (e.g., amplification or melting curve data) in a randomized, half-blinded manner. The function allows interactive rating of a curve for a certain characteristic. *humanrater* draws individual graphs of a curve and prompts an input field for the user. The application of this function are numerous (e.g., comparing the human rating and the rating of a machine or the rating of several individual experts). A list of designations to characterize the amplification curve can be specified. The names of elements can be specified (e.g., by 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 for lists. In our example, we used *humanrater* in the **RKWard** GUI (Figure S4). We aimed to characterize amplification curves which were randomly drawn from our simulated "testdata" dataset.

```r
# Create a set of data to be analyzed by humanrater.  The
# function AmpSim creates 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
B
Experiment 4
Experiment 6
Fluorescence
Cycle
R-Backend erbittet Informationen – RKWard
Repeat:1
[y] if yes, [a] if ambiguous, [n] if not
OK
Abbrechen
Michał Burdukiewicz, Stefan Roediger
Examples
R-Konsole
Kommando-Log
Suche in Hilfe

```
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 working 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 data for amplification curves

The following section briefly describes *chipPCR* functions used for visualizing and analyzing data for amplification curves. The functions *MFIaggr* and *plotCurves* (Section 6.1) were developed for a rapid and convenient inspection of raw data. *MFIaggr* is a powerful analytical and graphical tool for fast multiple comparison of 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 idea is to analyze only a region of interest (ROI) from a dataset as defined by the parameter *llul* (**l**ower **l**imit and **u**pper **l**imit). A ROI can be cycles or a time frame. *MFIaggr* is a relative of the *MFIerror* function from the *MBmca* (Rödiger *et al.*, 2013b) package but allows a finer grained data analysis for specific parts of a plotted curve. *MFIaggr* returns an object of the class list with the columns "Cycle", "Location" (Mean, Median), "Deviation" (Standard Deviation, Median Absolute Deviation) and "Coefficient of Variation". If the option *rob* is $TRUE$ the function calculates out the median and the median absolute deviation (MAD) instead of the mean and standard deviation. *MFIaggr* has 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). 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 expectedly shows a distribution, which significantly differs from a normal distribution (Figure S8). Setting the option $CV = FALSE$ shows the relative standard deviation (RSD, %). The variance between the amplification curves of replicates should be 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.

```
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)
```

The function can be used to compare two conditions of a qPCR experiment. In our example we tried to spot differences between a measurement during the annealing phase and the elongation phase (Figure S6).

```
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(x = MFIaggr(VIMCFX96_60, fluo = c(2L:10)), y = MFIaggr(VIMCFX96_69,
    fluo = c(2L:10)))
```

An analysis via the *shiny MFIaggr.gui* app is shown in Figure S7.

In our example we analyzed the raw fluorescence from 96 replicates (*VIMCFX96_60* dataset) of a qPCR experiment for the human gene *vimentin*. The *MFIaggr* plot shows that the analysis of all cycles is non-normally distributed (Figure S8).

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

MFIaggr analyzes the heteroskedasticity. Heteroskedasticity ("hetero" = different, "skedasis" = dispersion) is present if the variance (error term) is not constant. If the error terms do not have a constant variance, they are homoskedastic. Analysis of the heteroskedasticity gives insight into the characteristics of a system. In the following example we compared the *VIMCFX96_60* and *VIMCFX96_69* datasets both obtained from the same qPCR run in a Bio-Rad CFX96 during an annealing phase of 60 °C and an elongation phase of 69 °C, respectively. The heteroskedasticity increased expectedly during the amplification reaction. The variance in the elongation

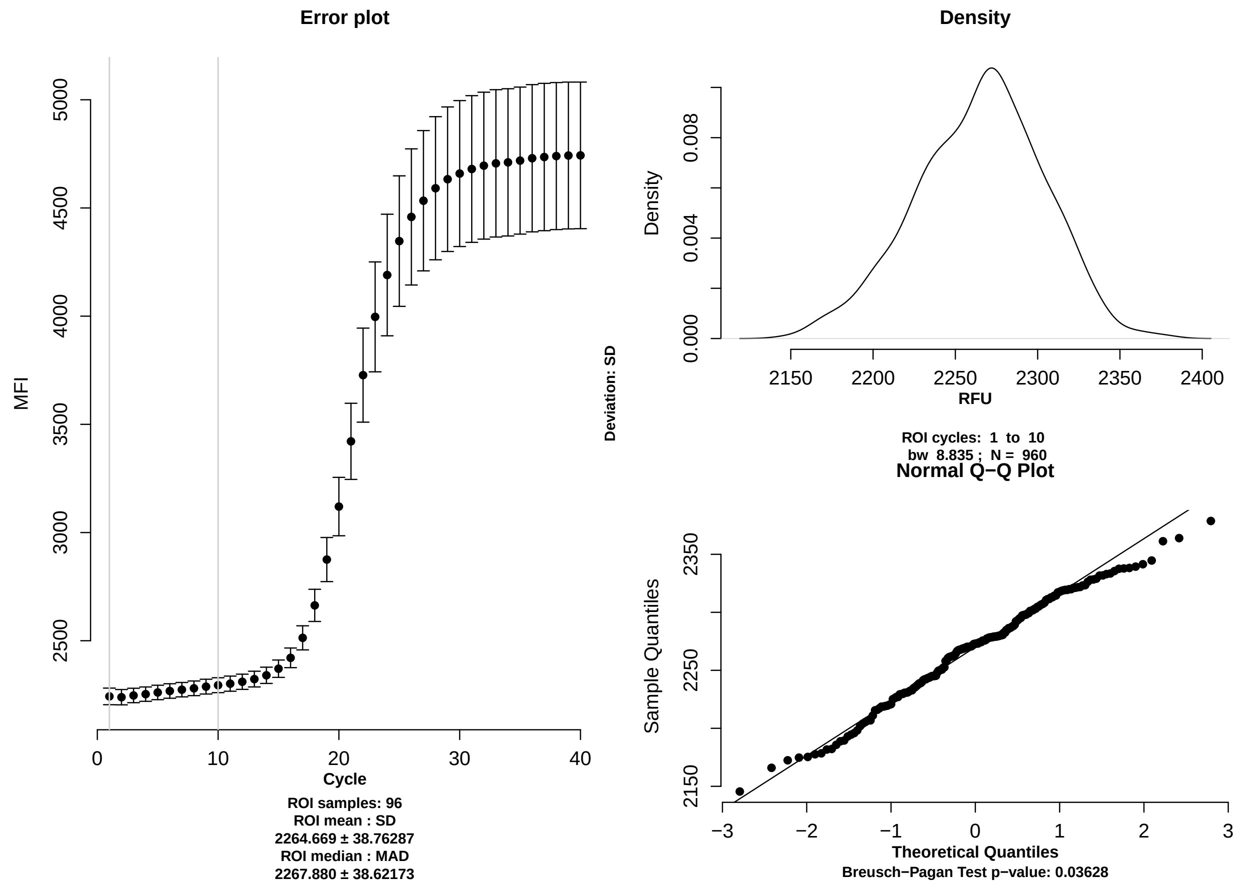


Figure S5: Sample for analysis using the *MFIaggr* function. The VIMCFX96_60 dataset (96-well plate cycler (Bio–Rad CFX96)) was used. Either all or a subset of the cycles (ROI: 1 − 10) or all cycles (ROI: 1 − 40) (Figure S8) were analyzed. The density plot (right upper panel) and quantile-quantile analysis (right lower panel) show no normal distribution. Due to the sigmoidal structure of the curve, the density function can be considered bimodal.

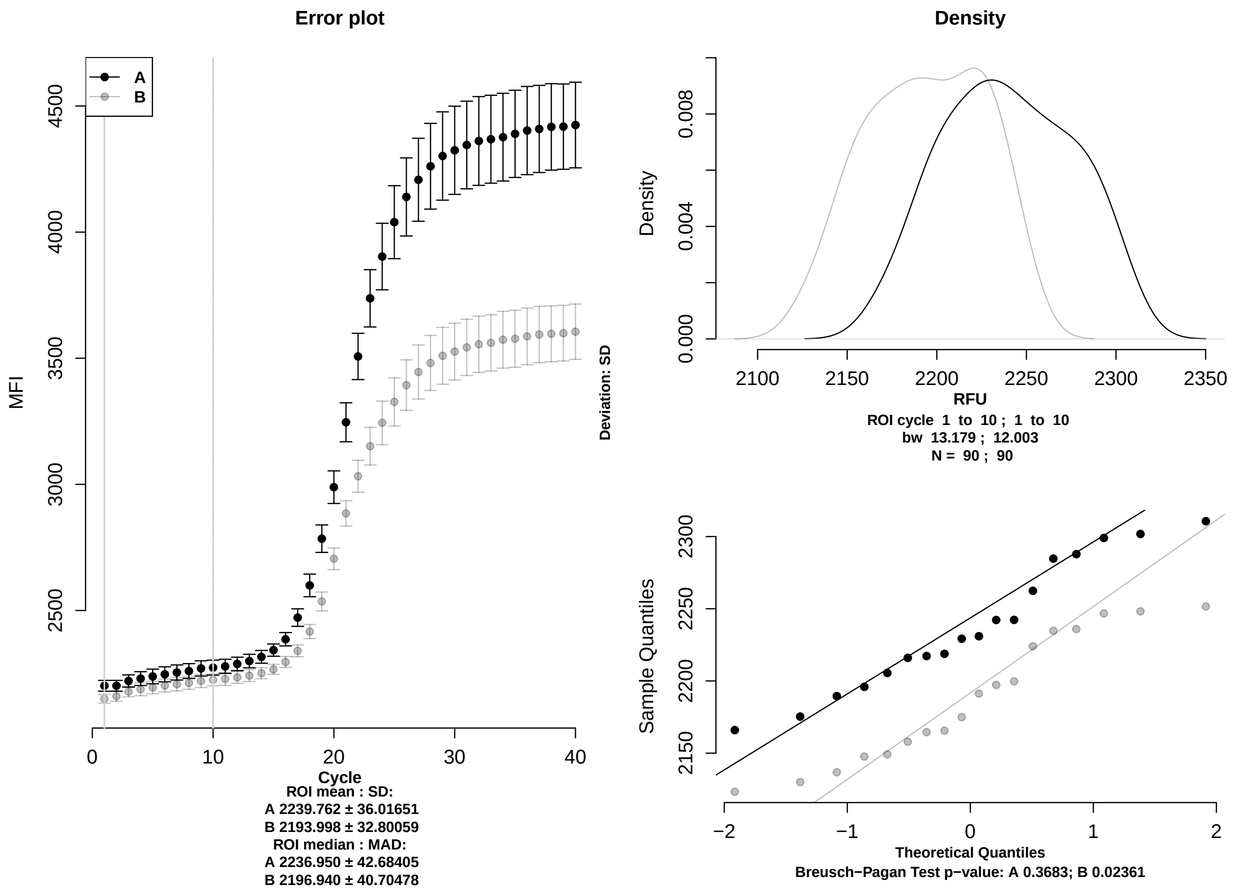
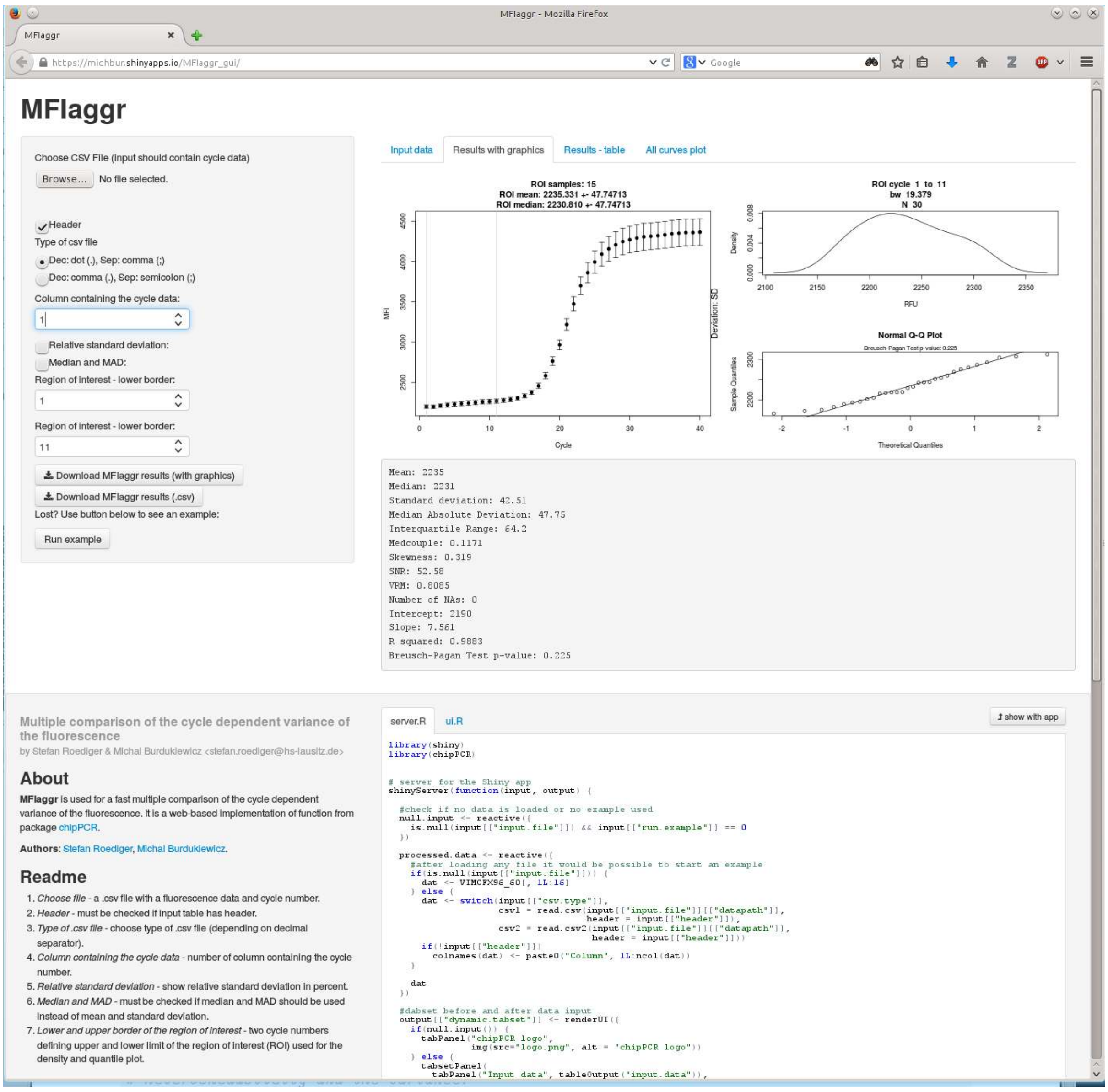


Figure S6: Analysis of two PCR conditions with the *MFIaggr* function. The VIMCFX96_60 and VIMCFX96_69 datasets (96-well plate cycler (Bio–Rad CFX96)) were used. A subset of cycles (ROI: 2 – 10) was analyzed for a qPCR measured during the annealing phase and elongation phase. The density plot (right upper panel) and quantile-quantile analysis (right lower panel) show no normal distribution. The measurements during the annealing phase and elongation phase show a similar distribution but differ in their mean fluorescence intensity (MFI) levels.



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.

server.R ui.R

```
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 S7: Example of *shiny MFIaggr.gui* app. *(Top)* The plot of the *AmpSim.gui* is shown in a web browser (**Iceweasel**, v. 32.0) along with the parameters (left panel) and the code ("server.R", "ui.R").

phase (Figure S8C and D) was lower than in the annealing phase (Figure S8A and B). The heteroskedasticity was significant during the first 15 cycles at 60 °C (Figure S8A).

```
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,
        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)],
```

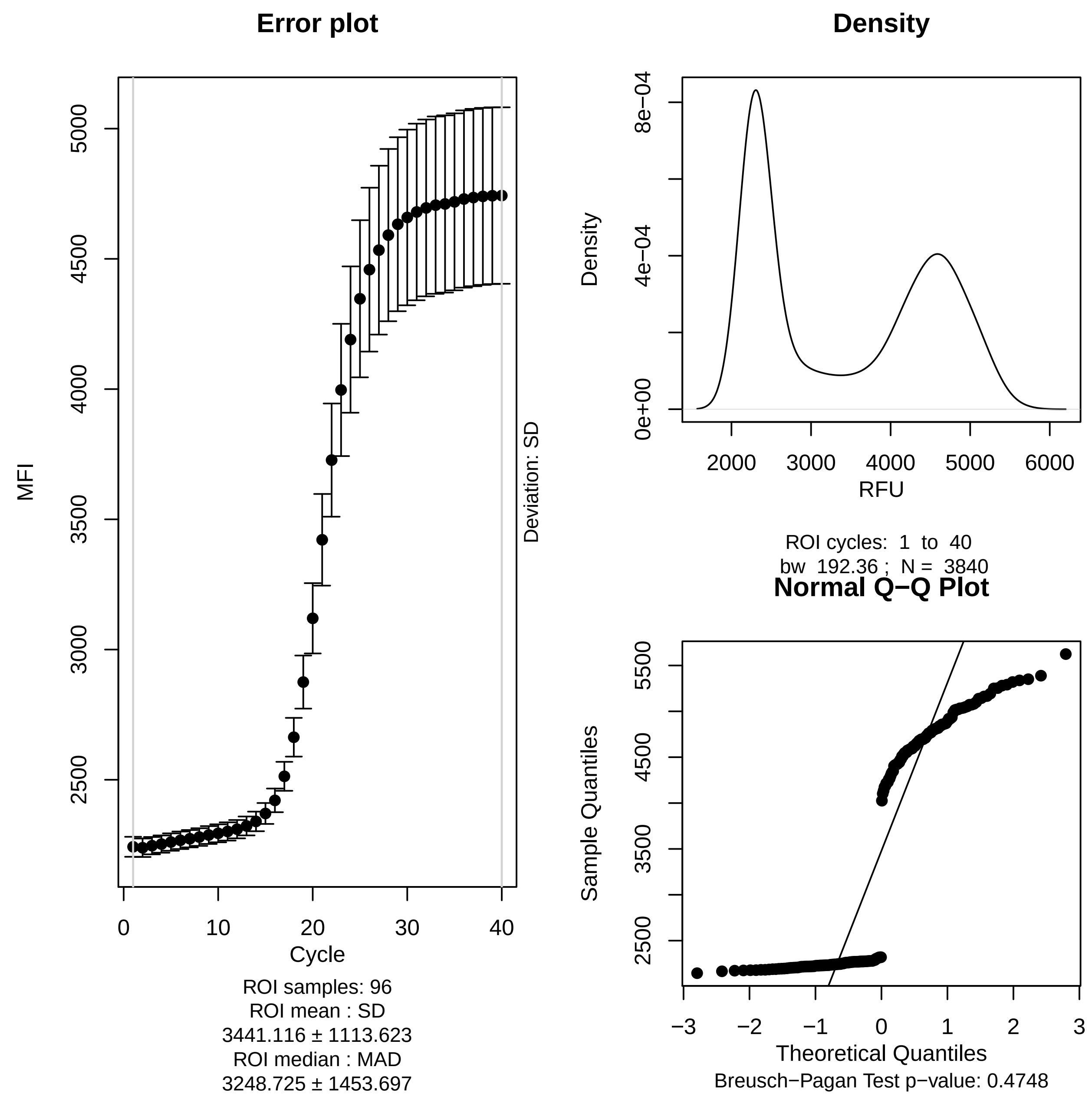


Figure S8: Signal analysis using the VIMCFX96_60 dataset (96-well plate cycler (Bio-Rad CFX96)). All cycles (ROI: 1 – 40) were analyzed by the *MFIaggr* function. The density plot (right upper panel) and quantile-quantile analysis (right lower panel) show no normal distribution. Owing to the sigmoidal structure of the curve, the density function can be considered bimodal.

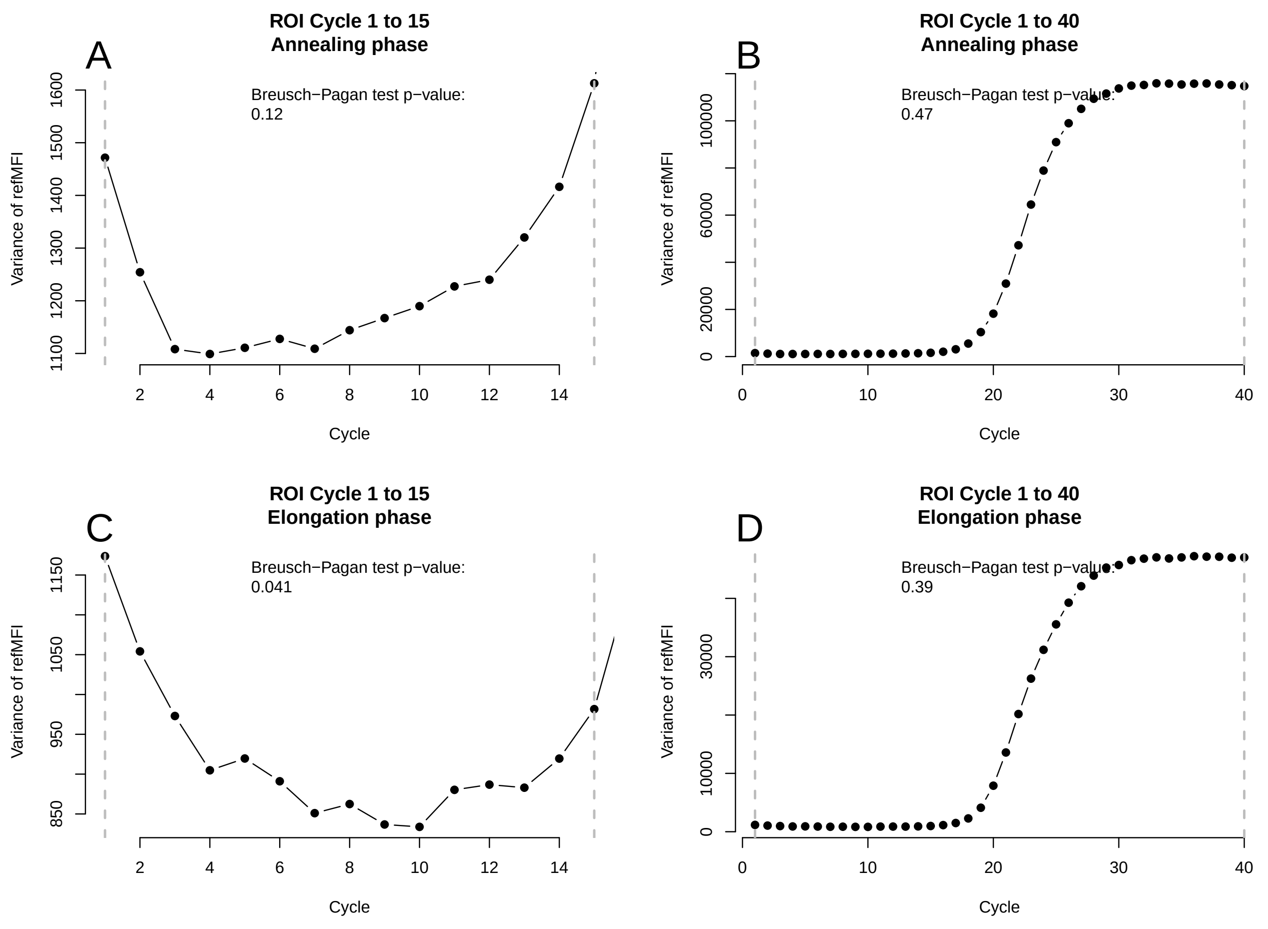


Figure S9: Use of *MFIaggr* to test for heteroskedacity. The data were aggregated with the *MFIaggr* function and assigned to the object *res*. The standard deviation was transformed to the variance. The plot shows the cycle dependent variance measured at 60 degrees Celsius (annealing phase; A, B) and 69 degrees Celsius (elongation phase, C, D) of 96 qPCR replicate amplification curves. The first cycles 1 to 10 and next the cycles 1 to 40 were analyzed. The Breusch-Pagan test of the *MFIaggr* confirmed the heteroskedasticity in the amplification curve data. The VIMCFX96_60 and VIMCFX96_69 datasets were used.

```
    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)
```

6.1 Data overview - *plotCurves*

plotCurves visualizes many curves on one plot in separate cells allowing quick assessment of experiments (Figure S10). In addition to this, *plotCurves* has an option to run an unsupervised *CPP* pre-processing step on the raw data. This smooths the data (Savitzky-Golay smoothing), removes missing values (spline interpolation by default) and performs a background subtraction (base-lining to zero). *plotCurves* has a colored indicator for rapid visualization of dataset with potentially problematic amplification curves. The plot output is arranged in

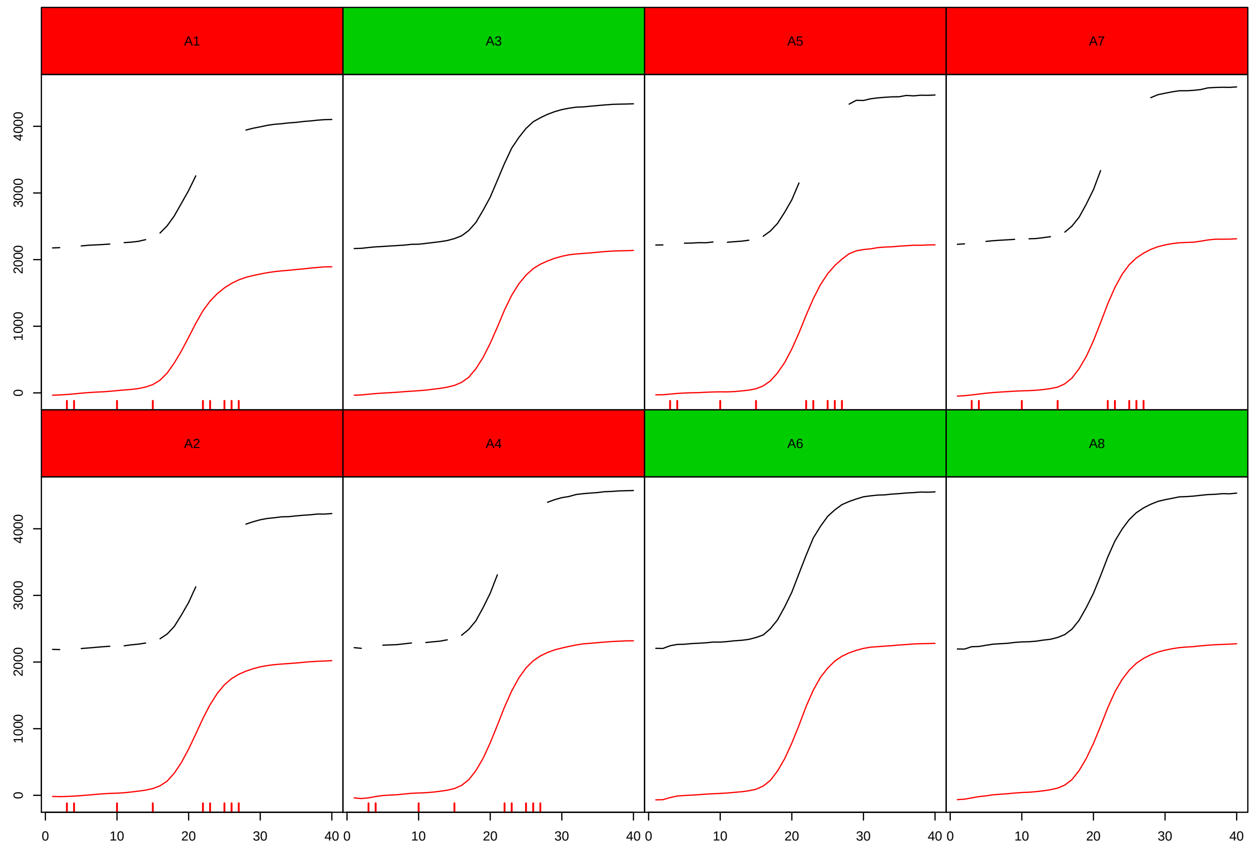


Figure S10: The *plotCurves* function. Notice: The function plots many curves on one plot in separate cells allowing for quick assessment. Missing values were artificially introduced at random positions to selected curves of the VIMCFX96_60 data set (solid black line). A colored box (topleft of each 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*).

a table-like fashion, where each curve is presented in a different cell.

```
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)
```

7 Proposed workflow

In the previous section we showed different methods for investigating specific properties of the measured data. Next, we focused on pre-processing methods of the *chipPCR* package. Here, we wish to state only function names and give some information on their working principle. Details will be explained in the subsequent sections in order to avoid confusion of the reader. We show the application of the *CPP* function, as a proposed workflow for customized pre-processor functions. Data were taken from the *VIMCFX96_60* data set. This dataset was measured with a Bio-Rad CFX96 thermo-cycler with 96 replicates (see *chipPCR* manual for experimental details).

The *CPP* function is a wrapper for pre-processing algorithms, which includes normalization, background subtraction, outlier removal using the (*fixNA* function) in the background range and tests for positive amplification reactions. *CPP* uses the *bg.max* function to automatically estimate the start of the amplification process. The background range is often noisy, which makes it hard 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 (Komsta, 2011). *rm.outlier* detects these outliers by a simple rule without statistical testing and replaces them by the sample mean. Outliers herein refer 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. This slope correction can either be done by a robust linear regression that computes MM-type regression estimators, or by 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 dataset. However, this step has to be performed with caution since this operation affects the AE. The background is assumed to be constant for the entire measurement. Caution is needed when using *trans* with time series (see *lm* from the *stats* package for details). Additionally, all data are normalized between the 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 S11). Note that 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)
```

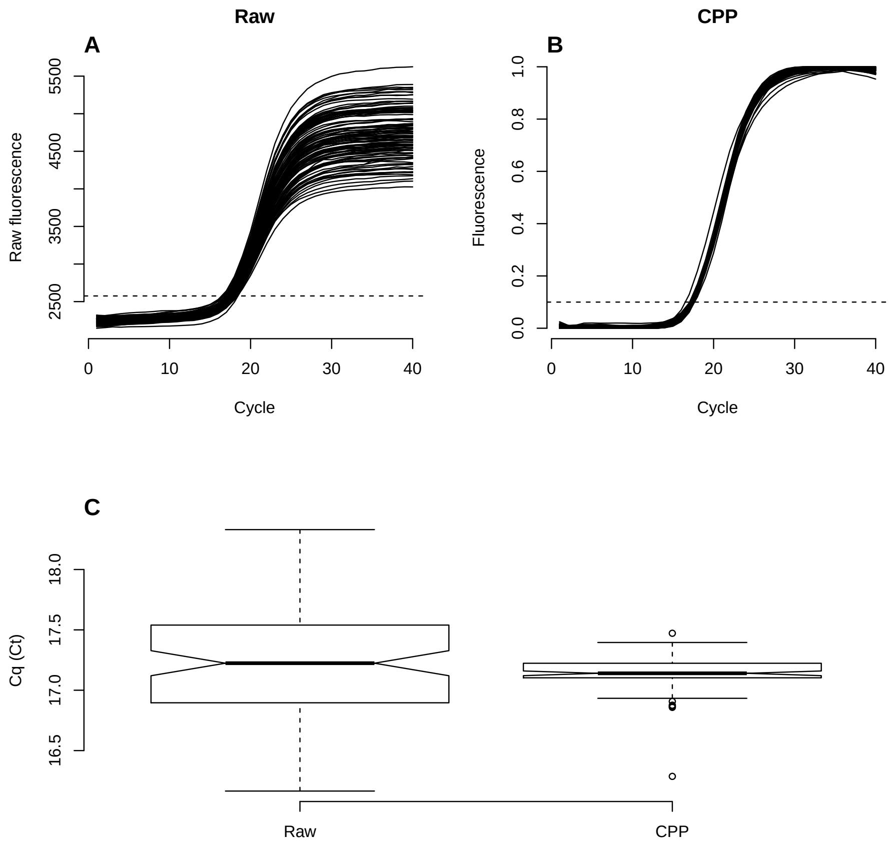


Figure S11: Application of the *CPP* and *th.cyc* functions. **A)** The raw data of the VIMCFX96_60 dataset were **B)** pre-processed with the *CPP* function and finally plotted. The parameter *trans* was set to *TRUE*, which leads to a linear trend correction and base-lining. By default a Savitzky-Golay 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 for the pre-processed data.

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. Upon ecnountering a missing value, many analytical functions stop to progress or discard entire datasets. Such a conduct is reasonable in cases where the data structure is unknown. However, in the case of data used for plotting amplification curves, 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 (Harrell, 2001). In the case of amplification curves other approaches are favorable. In particular, the transition phases (e.g., background phase to exponential phase) are 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 calculating 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) either by linear approximation or an approximation by cubic splines (default) (Figure S13). Other smoothing functions such as the Savitzky-Golay smoothing filter have the intrinsic capability to remove missing values (Savitzky and Golay, 1964; Eilers, 2003). However, such functionality is not yet implemented in this package. This linear approach is useful but may be problematic on the phases other than background or plateau phases of an amplification reaction. The parameter *spline* of *fixNA* enables a trend estimation on cubic splines and may be more appropriate in most scenarios.

The *reps384* dataset from the *qpcR* package (Spiess, 2014) was used to compare the influence of imputation on real-world data. The experimental details have been described in (Ruijter *et al.*, 2013). The dataset 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 added to each amplification curve at random positions. We separated the amplification curve into three different regions ("Linear phase" (cycle 1 – 10), "Exponential phase" (cycle 11 – 34) and "Plateau phase" (cycle 34 – 40), Figure S12) and investigated the impact on the qPCR parameters "Cq (SDM, Cy0)" and curve background parameter *bg*. The background was calculated as mean and standard deviation. The performance of the imputed models was analyzed with the goodness-of-fit in this region by the commonly used normalized root-mean-squared-error (NRMSE). We compared the imputation by "linear approximation" ($fixNA(x, y, spline = FALSE)$) and "spline approximation" ($fixNA(x, y, spline = TRUE)$). Our results show that imputation with the spline method worked reliably and introduced no significant bias to all the investigated parameters. The Kruskal-Wallis rank sum test (*kruskal.test*, *stats* package (R Core Team, 2013)) was used to compare the linear and spline-based imputation.

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) on the exponential phase and is therefore the recommended approach to remove missing values (Figure S12). We observed no significant bias using cubic spline interpolation (Table **??**). The performance of *fixNA* using cubic splines was better than a linear interpolation (Figure S13). However, the linear approximation might be applicable in measurements with high sampling 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 should not exceed 30 %. In a case where more than 30 % of all values are NAs, the *fixNA* function gives a warning.

Our results for the given experimental setting support the following statements. (I) The imputation of missing values by spline interpolation and linear methods introduce no significant bias on the tested parameters "Cq (SDM, Cy0)", "bg" and the accompanied quality measure NRMSE (Table **??**). (II) We found that the difference between the linear and spline imputation method 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])
```

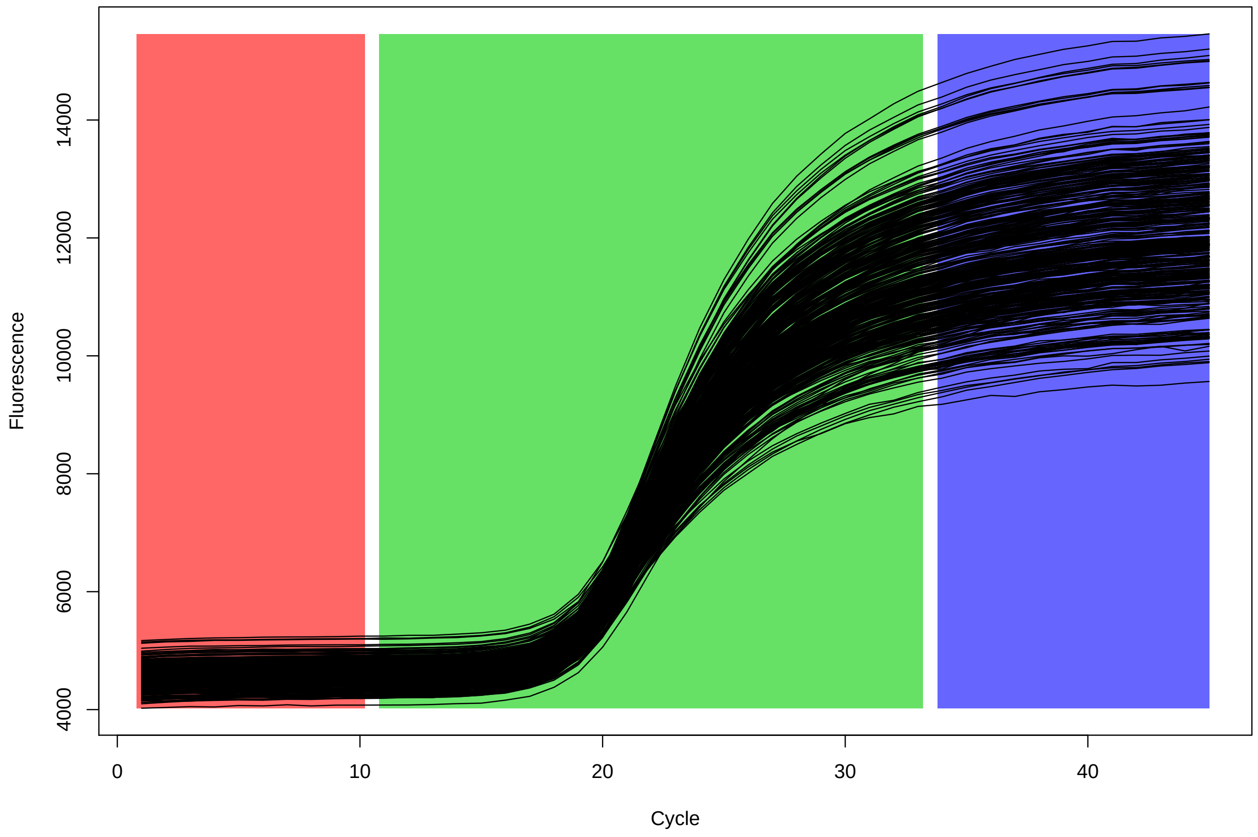


Figure S12: Inspection of the reps384 dataset. The reps384 dataset was used for the analysis of the impact of imputed missing values. Three areas of the curve data were defined as “Linear phase” (red, cycle 1 – 10), “Exponential phase” (blue, cycle 11 – 33), “Plateau phase” (green, cycle 34 – 40).

```r
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
```

```r
# 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)

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))
```

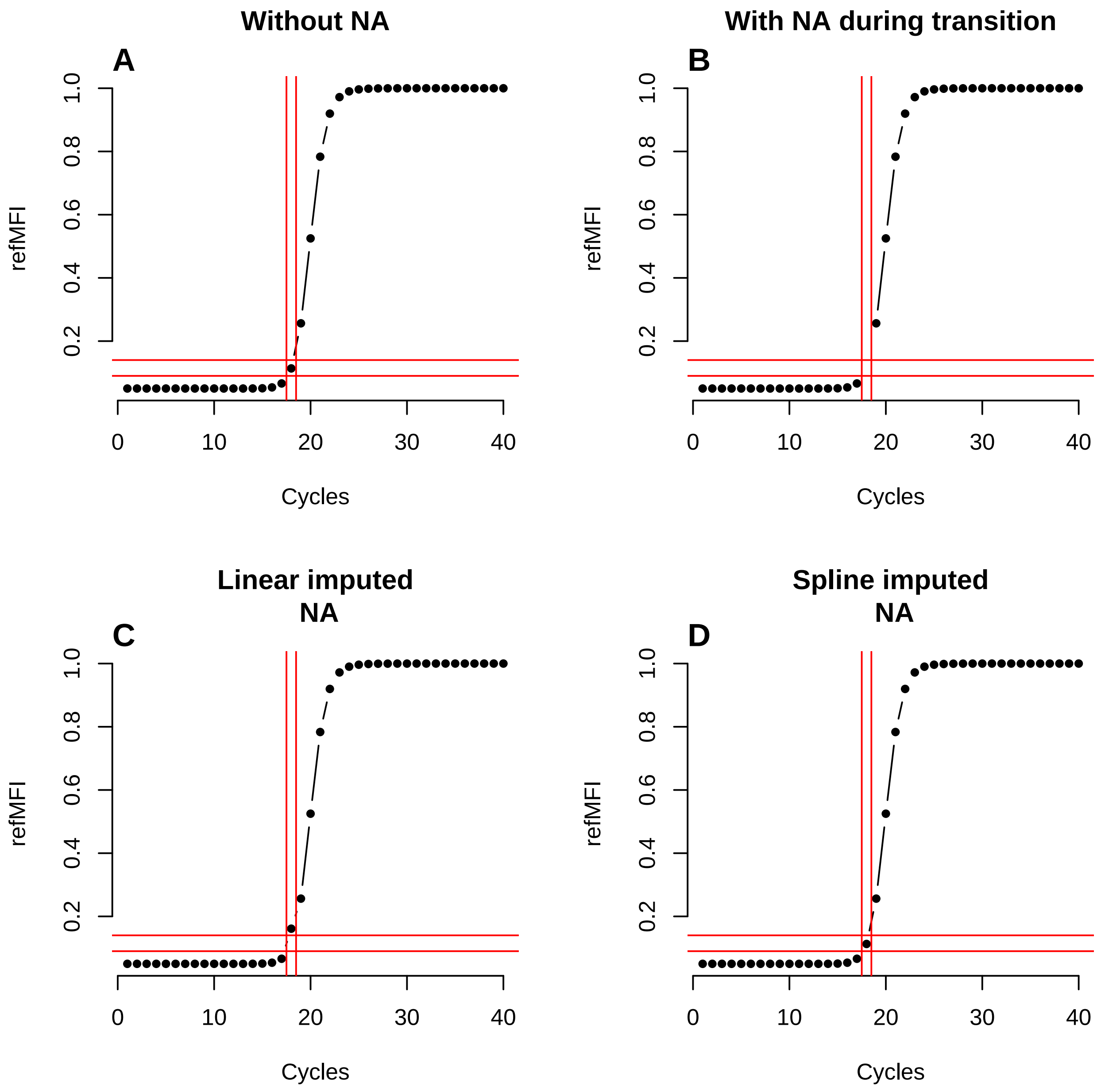


Figure S13: Imputation of missing values in the data for generating amplification curves. *(A)* Raw data were generated using the *AmpSim* simulation function. *(B)* A missing value was introduced in the transition phase. The missing value was imputed either by *(C)* linear approximation or *(D)* a cubic spline approximation. The spline approximation nearly reconstituted the original curve.

```r
# Evaluation of fixNA by introducing missing values and
# comparing efficiency measures for imputed and raw data

library(qpcR)
library(chipPCR)

linear_range <- 1L:10
exponential_range <- 11L:33
plateau_range <- 34L:45

raw.eff <- t(sapply(2L:ncol(reps384), function(i) {
    fit.raw <- pcrfit(reps384, cyc = 1, fluo = i)
    c(cpD2.raw = efficiency(fit.raw, type = "cpD2", plot = FALSE)[["cpD2"]],
        Cy0.raw = efficiency(fit.raw, type = "Cy0", plot = FALSE)[["Cy0"]],
        background = mean(reps384[1L:5, i]))
}))

# calculate raw efficiency measures
raw.df <- as.vector(apply(raw.eff, 2, function(i) c(mean(i),
    sd(i))))

# reformat raw data
raw.df <- cbind(data.frame("none", 0, "-"), t(raw.df))
colnames(raw.df) <- c("Imputation", "NA.number", "Phase", "cpD2.mean",
    "cpD2.sd", "Cy0.mean", "Cy0.sd", "background.mean", "background.sd")

# helper function introducing NA into data, fixing them and
# calculating efficiency
gen.fix <- function(number_points, phase, spline) {
    # copy raw data
    temp_dat <- reps384[, -1]

    # introduce NA(s)
    for (i in 1L:ncol(temp_dat)) temp_dat[sample(phase, 1), i] <- NA

    # impute missing values using fixNA
    fixed_dat <- sapply(1L:ncol(temp_dat), function(i) fixNA(reps384[,
        1], temp_dat[, i], spline = spline))

    # compute efficiency measures
    t(sapply(2L:ncol(fixed_dat), function(i) {
        fit.fix <- pcrfit(cbind(reps384[, 1], fixed_dat), cyc = 1,
            fluo = i)
        c(cpD2.fix = efficiency(fit.fix, type = "cpD2", plot = FALSE)[["cpD2"]],
            Cy0.fix = efficiency(fit.fix, type = "Cy0", plot = FALSE)[["Cy0"]],
            background = mean(fixed_dat[1L:5, i]))
    }))
}

# calculate results for linear imputation
res.linear <- lapply(c(1, 3), function(number_of_points) lapply(list(linear_range,
    exponential_range, plateau_range), function(phase) apply(gen.fix(number_of_points,
    phase, FALSE), 2, function(i) c(mean(i), sd(i)))))
linear.df <- data.frame(num = unlist(lapply(c(1, 3), rep, 3)),
    region = rep(c("linear", "exponential", "plateau"), 2), do.call(rbind,
        lapply(unlist(res.linear, recursive = FALSE), as.vector)))


# calculate results for spline imputation
res.spline <- lapply(c(1, 3), function(number_of_points) lapply(list(linear_range,
    exponential_range, plateau_range), function(phase) apply(gen.fix(number_of_points,
    phase, TRUE), 2, function(i) c(mean(i), sd(i)))))
spline.df <- data.frame(num = unlist(lapply(c(1, 3), rep, 3)),
    region = rep(c("linear", "exponential", "plateau"), 2), do.call(rbind,
        lapply(unlist(res.linear, recursive = FALSE), as.vector)))

# join and format results
sim.res <- cbind(unlist(lapply(c("linear", "spline"), rep, 6)),
    rbind(linear.df, spline.df))
colnames(sim.res) <- c("Imputation", "NA.number", "Phase", "cpD2.mean",
    "cpD2.sd", "Cy0.mean", "Cy0.sd", "background.mean", "background.sd")

# join simulation results with results for raw data
fixNA.evaluation <- rbind(raw.df, sim.res)
xtable(fixNA.evaluation[, c(1L:7)], digit = 4)

xtable(fixNA.evaluation[, c(1L:3, 8L:9)], digit = 4)
```

	Imputation	NA.number	phase	cpD2.mean	cpD2.sd	Cy0.mean	Cy0.sd
1	none	0.0000	-	19.2931	0.1464	11.1806	1.1326
2	linear	1.0000	linear	19.2932	0.1466	11.1801	1.1337
3	linear	1.0000	exponential	19.2875	0.1472	11.1631	1.1351
4	linear	1.0000	plateau	19.2932	0.1469	11.1814	1.1336
5	linear	3.0000	linear	19.2929	0.1468	11.1803	1.1339
6	linear	3.0000	exponential	19.2878	0.1465	11.1636	1.1351
7	linear	3.0000	plateau	19.2932	0.1466	11.1815	1.1335
8	spline	1.0000	linear	19.2932	0.1466	11.1801	1.1337
9	spline	1.0000	exponential	19.2875	0.1472	11.1631	1.1351
10	spline	1.0000	plateau	19.2932	0.1469	11.1814	1.1336
11	spline	3.0000	linear	19.2929	0.1468	11.1803	1.1339
12	spline	3.0000	exponential	19.2878	0.1465	11.1636	1.1351
13	spline	3.0000	plateau	19.2932	0.1466	11.1815	1.1335

Table S2: Results of fixNA data imputation, part 1: cpD2 and Cy0. NA column determines how many NA values were introduced in a given phase.

	Imputation	NA.number	phase	background.mean	background.sd
1	none	0.0000	-	4567.5648	184.7082
2	linear	1.0000	linear	4568.2296	184.7895
3	linear	1.0000	exponential	4568.0775	184.6828
4	linear	1.0000	plateau	4568.0775	184.6828
5	linear	3.0000	linear	4568.2859	184.5738
6	linear	3.0000	exponential	4568.0775	184.6828
7	linear	3.0000	plateau	4568.0775	184.6828
8	spline	1.0000	linear	4568.2296	184.7895
9	spline	1.0000	exponential	4568.0775	184.6828
10	spline	1.0000	plateau	4568.0775	184.6828
11	spline	3.0000	linear	4568.2859	184.5738
12	spline	3.0000	exponential	4568.0775	184.6828
13	spline	3.0000	plateau	4568.0775	184.6828

Table S3: Results of fixNA data imputation, part 2: background. NA column determines how many NA values were introduced in a given phase.

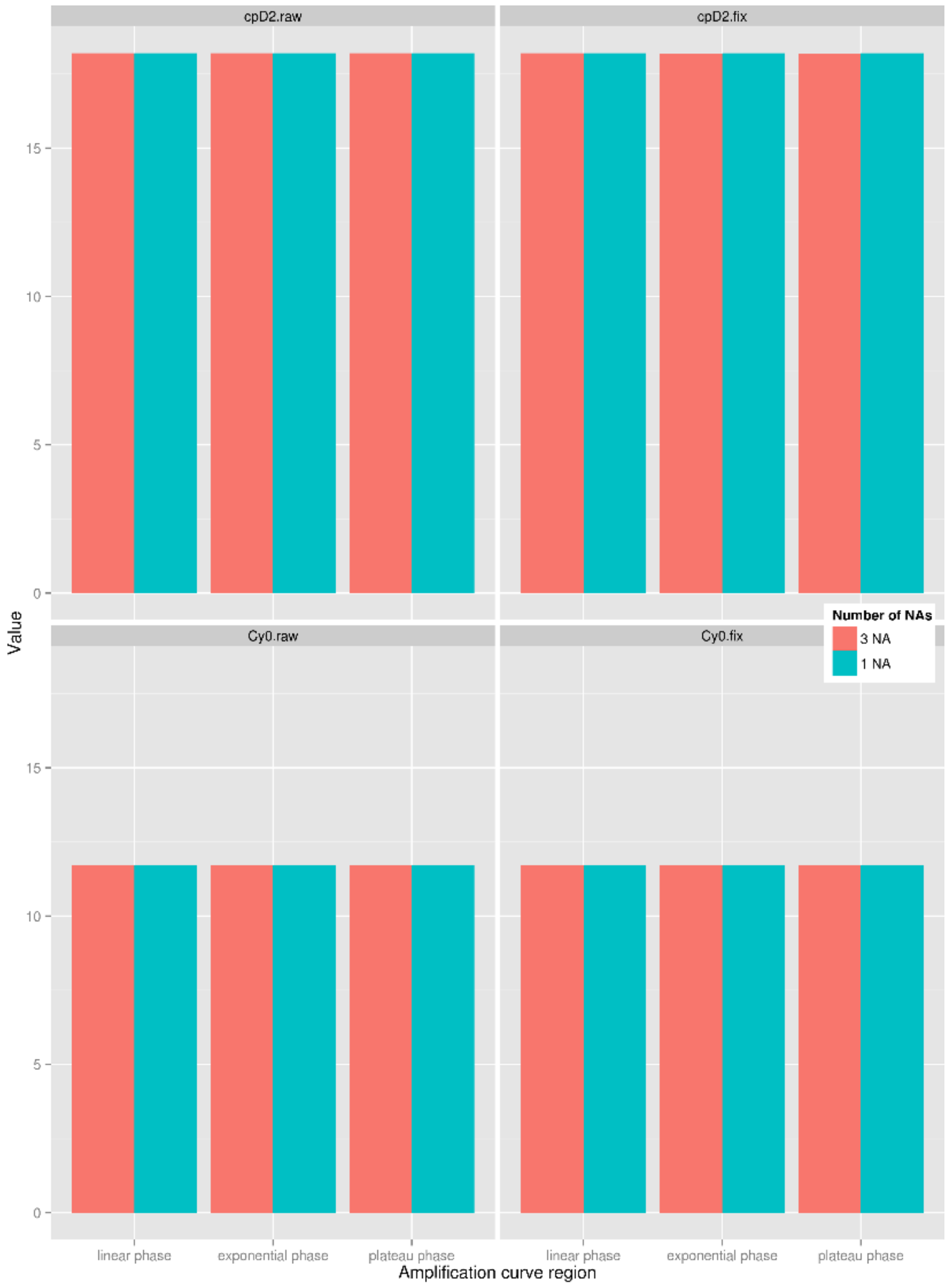


Figure S14: Imputation of missing values in data for plotting amplification curves by splines and linear trends. Missing values were artificially introduced into the *reps384* dataset from the *qpcR* package and imputed by the *fixNA* function. We found no significant difference between raw data and 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

For data presentation it is often useful to smooth or filter the data. Smoothing and filtering are different approaches with similar goals. Both pre-process an input signal as output for subsequent analysis steps. Filtering uses methods of signal processing and takes a data input and applies a function to form an output. Smoothing in contrast uses statistical approaches, like local regression models (e.g., least square estimate) or cubic splines. We developed the *smoother* function, which is a wrapper for smoother functions and filters commonly used to process data for amplification curves. *smoother* inherited traits (Table 9) of the parent functions (Spiess *et al.*, 2014). However, the functionality of *smoother* greatly outgrows 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 the syntax of specific functions. *smoother* was enhanced by the functionality of *fixNA* and *CPP*. Figure S15 shows results of the *smoother* function on data used for amplification curves.

```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)
})
```

The presence of noise may cause many false estimates for the FDM (first derivative maximum) and SDM (second derivative maximum). To reduce noise, it is possible to smooth the first derivative of the amplification curve. Many methods integrated the moving average as a first pre-processing step (e.g., (Shain and Clemens, 2008)). The moving average filter is a linear filter, which sequentially replaces 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

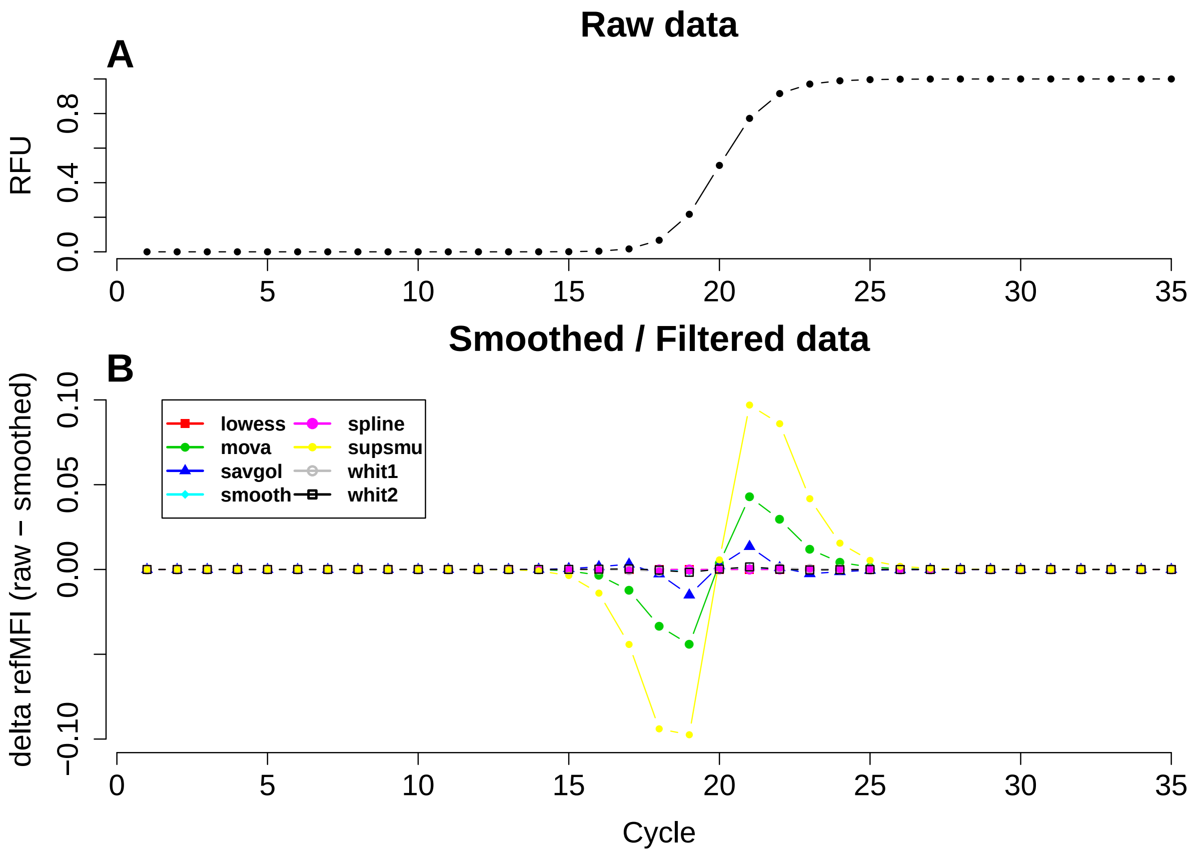


Figure S15: Smoother and filter methods of the *chipPCR* package. *(A)* Raw data were generated using the *AmpSim* simulation function. *(B)* The difference of the raw data to the smoothed data was plotted. “savgol” (Savitzky-Golay smoothing), “lowess” (locally-weighted polynomial regression), “mova3” (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 alterations in the transition phases of the amplification curve. Optimized time series smoother, like the Kalman filter (Tusell, 2011), are not yet integrated in this package.

smoother function uses exclusively the arithmetic mean. The moving average is intuitive and easy to implement. However, such processed data lags behind a trend and ignores rapid changes, leading to a forerun of few cycles (Spiess *et al.*, 2014). This is in particular problematic during the exponential phase. Splines apply non-parametric regression by local cubic polynomials between knot points (Nie and Racine, 2012). Other examples for smoothers include the Savitzky-Golay smoothing filter, Friedman's SuperSmoother, and the Weighted Whittaker smoother (see the *smoother* function for details).

Selected smoothing and filtering functions (e.g., Savitsky-Golay filter) assume uniform (equally spaced) sampling. The function *smoother* and *CPP* (inherited from *smoother*) give a warning in such cases. It is recommended to pre-process the data to have equally spaced values. The *smoother* function enables users to tune the behavior of the chosen smoothing algorithm by using nearly all parameters available in called subroutines and at the same time uniformizes input and output. It should be noted that smoothing may alter the curve shape and thus lead to artificial results. Smoothed data are easier to read but introduce a bias to the pre-processed data. Therefore, the prime use of smoothers is in processing data for visualization purposes. However, it is not recommended to smooth unsupervised signals 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 obtained after curve smoothing can be used to locate peaks, but such a procedure should be used cautiously for measuring peaks (Spiess *et al.*, 2014).

Table S4: Smoothing and filtering methods of the *chipPCR* package. The parameter *lowess* (locally-weighted polynomial regression, LOWESS) can be adjusted by the parameters *f* and *iter*. The parameter *mova* (moving average) can be adjusted by the windowsize parameter *movaww*. The parameter *savgol* (Savitzky-Golay smoothing filter) can be adjusted by the parameter *p*. The parameter *smooth* (cubic spline smooth) can be adjusted by the parameter *df.fact*. A *df.fact* value of 1 smooths the data least while a value of 0.5 smooths the curve most. The parameter *spline* (standard cubic spline smooth) has no additional parameter. The parameter *supsmu* (Friedman's SuperSmoother) can be adjusted by the parameter *span*. The parameter *whit*1 (first order finite difference penalty) and *whit*2 (second order finite difference penalty) for Weighted Whittaker smoothing filter can be adjusted by the parameter *lambda*. For further details on the smoothers refers to the documentation of their 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	*whit*1, *whit*2	*lambda*	*whit1*, *whit2*, *ptw*
All smoother	*all*	defaults	

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

The following paragraphs describe methods used in literature to detect the background range of amplification curves. Background range herein refers to a level of reporter fluorescence signal 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 in **R** function. The easiest way to classify them is by the extend of assumptions made before applying of a method.

The simplest approach is to treat the background fluorescence as a value constant during the whole amplification reaction. In this case the noise could be approximated as the mean or median of fluorescence values in the lag phase (Frank, 2009) or their standard deviations (Peirson *et al.*, 2003). 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 the log-linear phase (Frank, 2009). 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 invoke the assumption that background fluorescence is a constant value and instead describe it as a function of the cycle number. The algorithm in **SoFar** (Wilhelm *et al.*, 2003) fits a nonlinear saturation function to points measured 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 uses the premise that changes of fluorescence between subsequent cycles could exclusively be caused by the amplification of the product (Rao *et al.*, 2013). Thus, the corrected values are calculated by simply subtracting the fluorescence value in the former cycle from fluorescence in the latter. The real fluorescence value in the first cycle is unknown since the number of cycles used for the computation is reduced by one.

The **Real-Time PCR Miner** algorithm is nearly assumption-free (Zhao and Fernald, 2005). 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-elemental groups. For each set of points a pair of the estimate of the efficiency and the significance of model representing the relation between the fluorescence value and the cycle number is calculated. The estimates with the highest significance are the most influential in the computation of the final efficiency.

Finding 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 for the amplification reaction. To this group belongs **SoFar** (Wilhelm *et al.*, 2003). 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 a higher value than background noise. A very similar procedure is implemented in **Real-Time PCR Miner** (Zhao and Fernald, 2005), where background noise is used as parameter in implemented models to calculate a theoretical start of the amplification process. The end of amplification process is detected by calculating the third derivative of an 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) 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 the first three measurement points. If none of the residuals is statistically identified as an outlier, 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 defines the end of lag phase. It is worth noting that this algorithm is versatile enough to 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. The signal drastically changes during transition into the early exponential phase. First data are smoothed by the 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 overshoot 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 S16). See section *bg.max* "Details" of the *chipPCR* manual for further details. This function is a rational basis for trimming of unwanted data.

```r
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 the 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 the 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))
```

The *bg.max* algorithm was used to analyze amplification data from a capillary convective PCR (*capillaryPCR* dataset, *chipPCR* package). The raw data (Figure S17 A) and pre-processed data (Figure S17 B) using the *CPP* showed comparable curvatures. We observed no significant difference between the raw and pre-processed data.

```r
# 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 dataset 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)
for (i in c(1, 3, 5, 7)) {
    x <- capillaryPCR[1L:750, i]
```

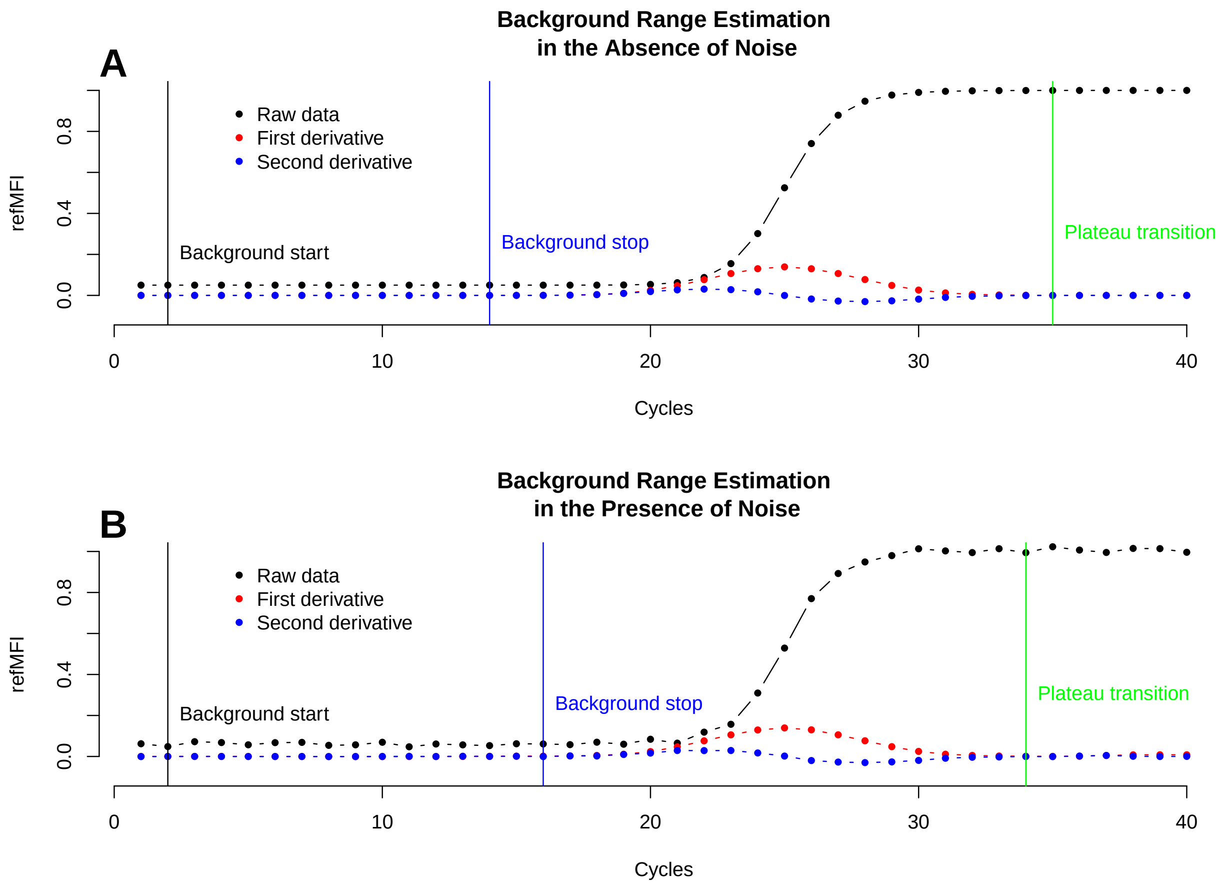


Figure S16: *bg.max* tries to estimate the range between the background and the plateau phase of an amplification reaction. *(A)* in absence and *(B)* presence of noise. The data were simulated with the *AmpSim* function.

```
    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)
}
```

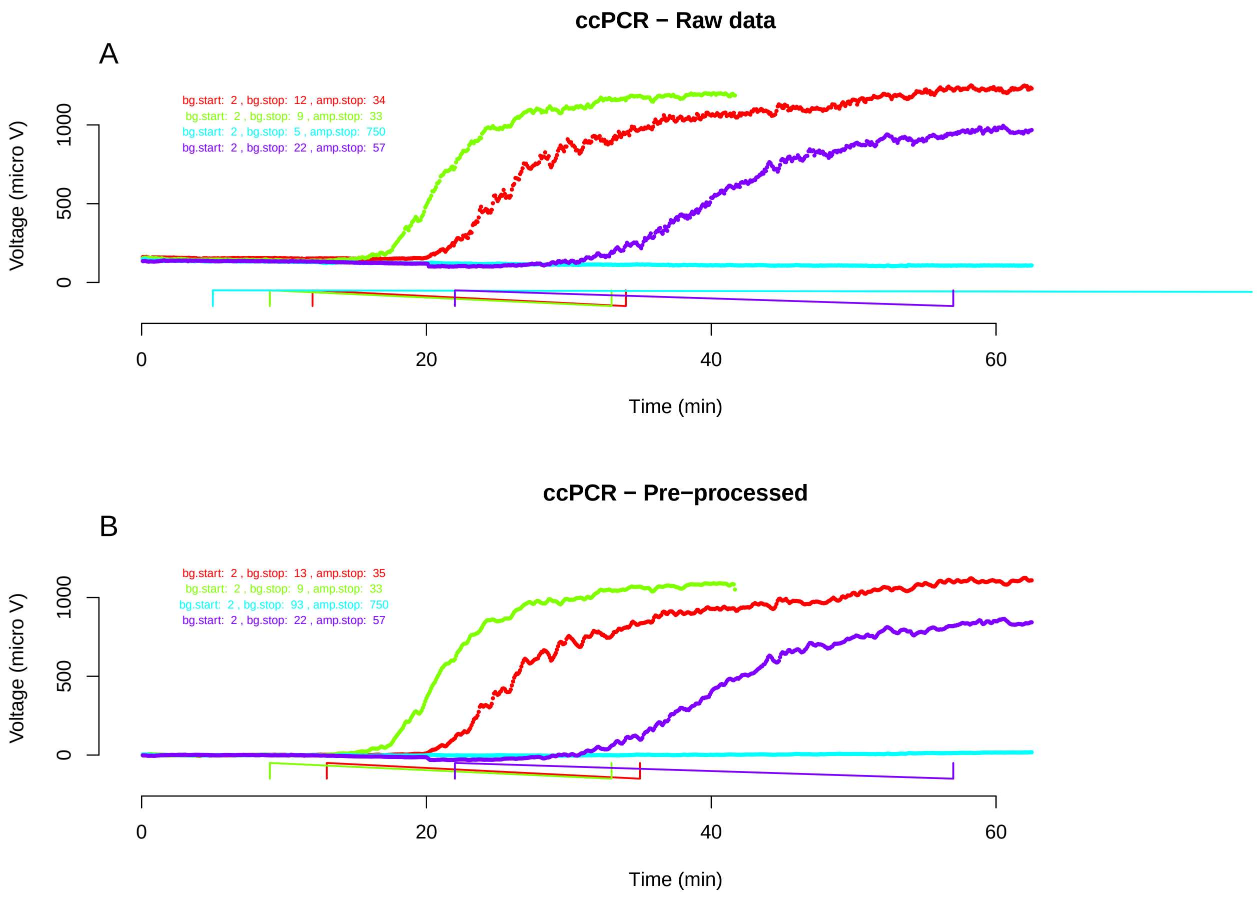


Figure S17: Application of the *bg.max* function. Amplification curve data from a capillary convective PCR were used *(A)* as raw data and *(B)* pre-processed (smoothed (moving average, window size 3), base-lined and trend corrected (robust MM-estimator)) with the *CPP* function. The output of the *CPP* function 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 and pre-processed data.

11 Normalization of amplification curve data

As illustrated in Figure S18 *A*, it is a common characteristic of data used for plotting amplification curves, that the fluorescence values in the baseline and plateau region vary between samples (e.g., due to high background, variances in dye quantities). Minimum and maximum values differ within an experiment. For visualization it is recommended to scale the data. This helps to grasp the data faster and alleviates the comparison of data from different measurements and/or scaling. *normalizer* is a function used to normalize any data by different methods (see details). To scale the fluorescence between 0 and 1 a *Min-Max normalization* (Equation S2) is recommended (Rödiger *et al.*, 2013b). We propose a quantile based normalization as alternative (Equation S4) since 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 quantiles. By default, the 3 % and 97 % quantiles are used for the normalization. In addition, a maximum normalization (Equation S3, Figure S18 *D*) and a standard score normalization (Equation S5, Figure S18 *F*) is implemented.

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

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

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

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

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 to the minimum and the maximum of an amplification plot.
- minm does a min-max normalization between 0 and 1 (see (Rödiger *et al.*, 2013b) 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)
lin <- apply(tmp[, -1], 2, function(x) lines(tmp[, 1], CPP(tmp[,
    1], x, method.norm = "minm")$y))
```

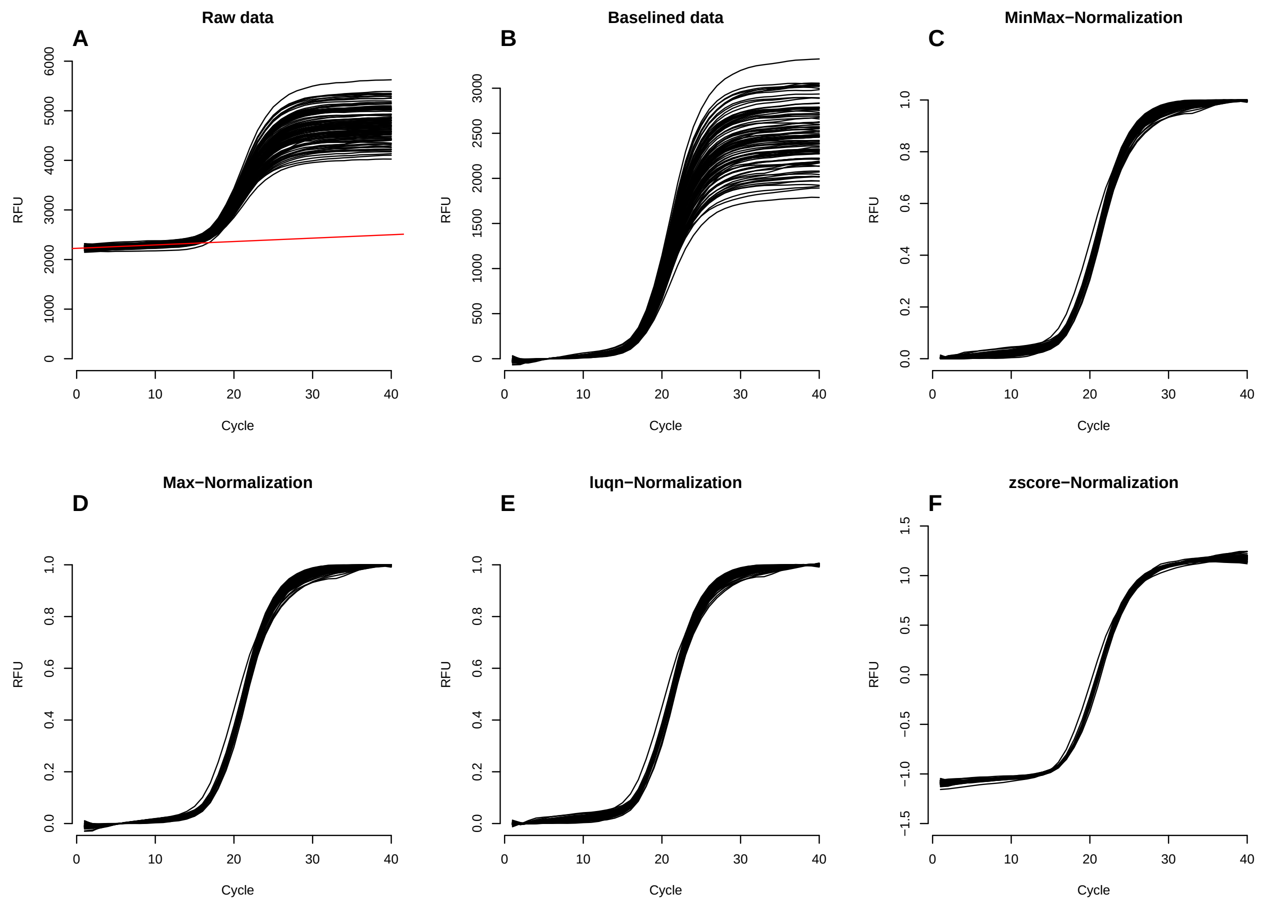


Figure S18: Comparison of the normalization methods with the *CPP* function. The VIMCFX96_60 dataset (96-well plate cycler, Bio-Rad CFX96, EvaGreen detection) was used. *(A)* Plot of raw data for all amplification curves. The signals are superimposed to circa 2200 RFU and the inter-sample baseline and 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 slope 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.*

```
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))
```

11.1 Computing linear model coefficients - Background subtraction based on linear models

The slope of the background range is often unequal to zero and in most cases and is accompanied by a positive or negative trend. It is however possible to correct the slope by a linear trend extrapolation. The functions *lm.coefs* and *CPP* are wrappers used by functions to perform normal (linear least squares) and robust linear regression. *lm.coefs* calculates the estimated background of an amplification curve. This includes a ordinary least squares method (*lm*, *stats*) and three other robust methods. Robust regression methods are less vulnerable to outliers. This feature is especially useful, when the background range contains noise. These robust methods are (I) a nonparametric rank-based estimator (Kloke and McKean, 2012), (II) quantile regression (Koenker, 2008) and (III) a MM-type estimators for linear regression (Todorov and Filzmoser, 2009). By default, the MM-type estimator is used. Under the assumption that the background is constant, *CPP* or *lm.coefs* uses a defined range of the amplification curve (e.g., background range) to extrapolate a linear trend over the entire data. The coefficients of the analysis can be used for a trend-based correction of the entire dataset (Figure S19). If the robust linear regression is impossible, *lm.coefs* performs a linear regression using the least squares method. However, this operation effects the AE. Caution should be exercised 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)
}
legend("right", c("Data", "lmrob", "rq", "least", "rfit"), lty = 1,
    col = 1:5, cex = 0.95)
```

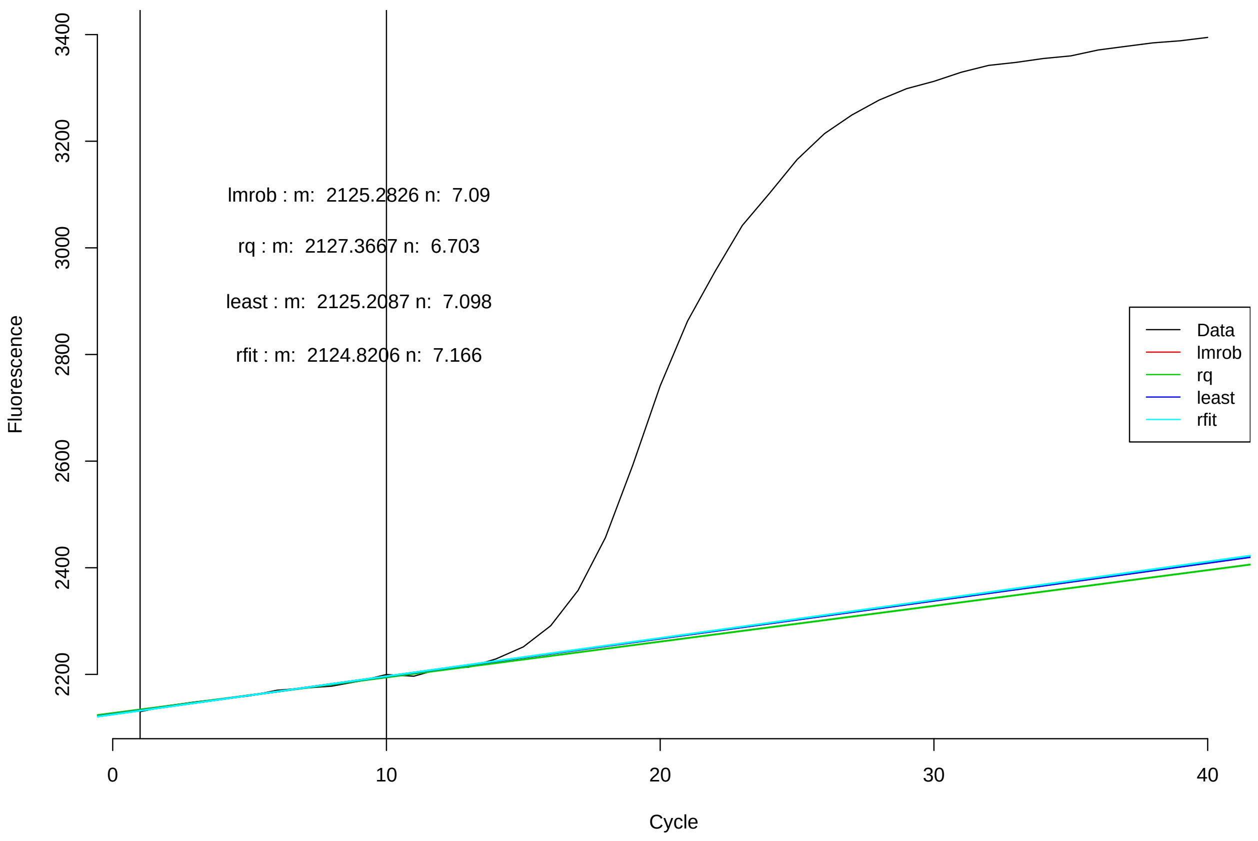


Figure S19: *lm.coefs* a function to compute coefficients for linear models. The function is a wrapper for functions to perform normal (least squares) and robust linear regression. If the computation of the robust linear regression fails, then *lm.coefs* performs a linear regression using the least squares method. lmrob, MM-type estimators for linear regression; rq, quantile regression fit; least, least squares linear regression; rfit, Rank-based estimates of regression coefficients. m, slope; n, asymmetry.

12 The *inder* function - an interpolating five-point stencil

Many methods for curve 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 fitting process and unnecessarily 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 (Dahlquist and Björck, 2008). This function can estimate the approximate quantification cycles (Cq). Differentiation is a method for background suppression and reduction of the inter sample background amplitude variations (Figure S22 A and B). Smoothing may enhance the calculation of derivatives and optimize the signal-to-noise ratio. Therefore, we implemented spline interpolation and Friedman's SuperSmoother. However, the use of this smoother is limited in use to 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, a Nip less than 2 and higher than 20 are not meaningful for conventional qPCR with 30 to 50 cycles. In the context of qIA, a higher Nip might be appropriate.

12.1 Quantitative description of amplification reactions

The Cq is a relative value, which depends on the template copy number, instrument, reagents, AE and probe technology. Low Cqs correlate with high quantities of template copy numbers. Real-time technologies enable the quantification of nucleic acids by calculation of specific curve parameters like the Cq and the 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 algorithms as described elsewhere (Ruijter *et al.*, 2009, 2013; Tellinghuisen and Spiess, 2014). The output of *inder* includes the first derivative maximum (FDM) and second derivative maximum (SDM), which are commonly used in qPCR experiments but might be useful for isothermal amplification processes, too. Figure S20 shows a typical result of the *inder* function. Following we show three examples that explain properties of *inder* and illustrate applications of the function in combination with other functions.

The *inder* function 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

```
# 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 S20 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 set to calculate the second derivative maximum (SDM) with *inder*. If *logy* is $TRUE$ then 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, *inder* is the first tool in **R**, which allows a user to numerically derive his data without fitting them to any function or a combination of functions. The universality of this stencil approach can find an application even in problems not related to the analysis of amplification curves.

```
# 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])
```

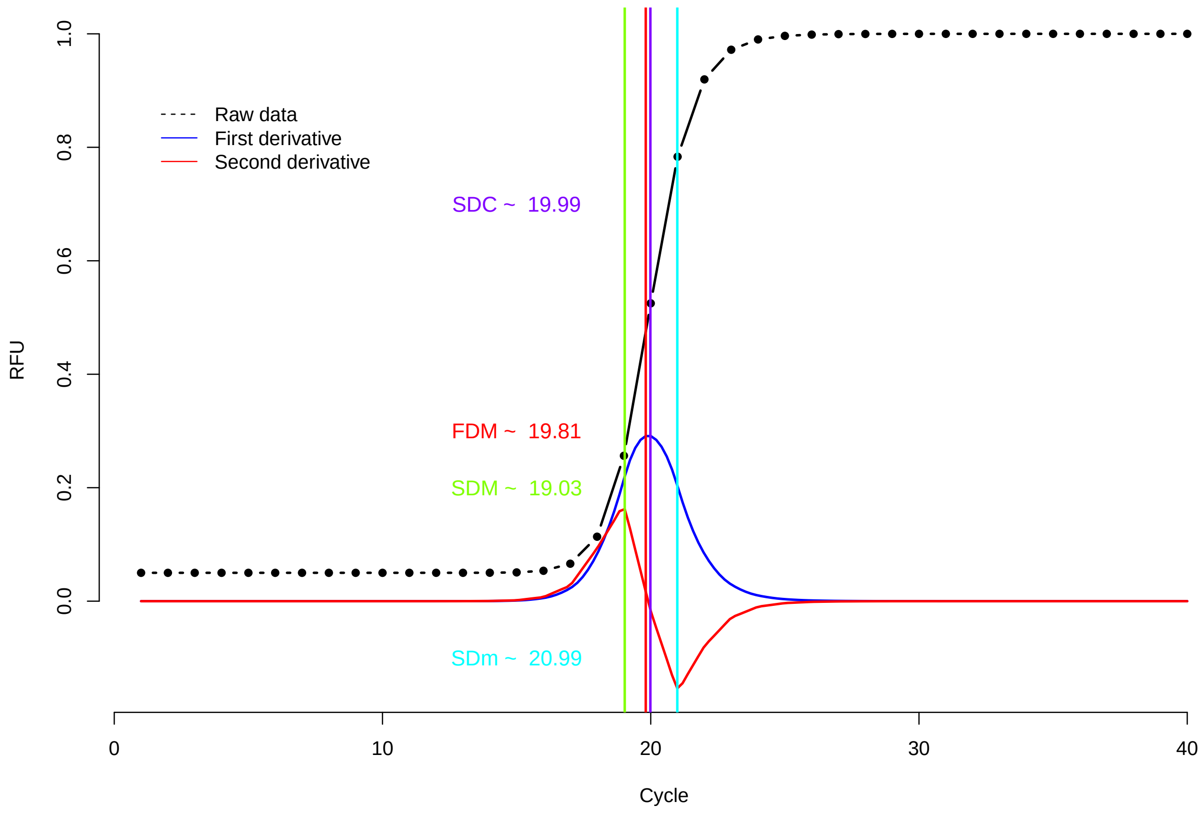


Figure S20: Quantification cycle (Cq) by the second derivative maximum method. Raw data (●) were generated by the *AmpSim* function. The inflection point is the point where the slope is maximum and the curvature is zero. The first derivative of the amplification curve has a first derivative maximum (FDM) 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 the five-point stencil. This method is assumption free regarding the function f. *inder* calculates the approximate SDM. 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.

```
legend(1.1, 0.9, c("Raw data", "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. The *diffQ* function is part of a routine to calculate the melting points of nucleic acids (Rödiger *et al.*, 2013b). 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 S21 A) than samples detected with the hydrolysis probe (Figure S21 B). The mean variation of the Cq was less in samples where EvaGreen was used for monitoring.

```r
# 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)

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)
```

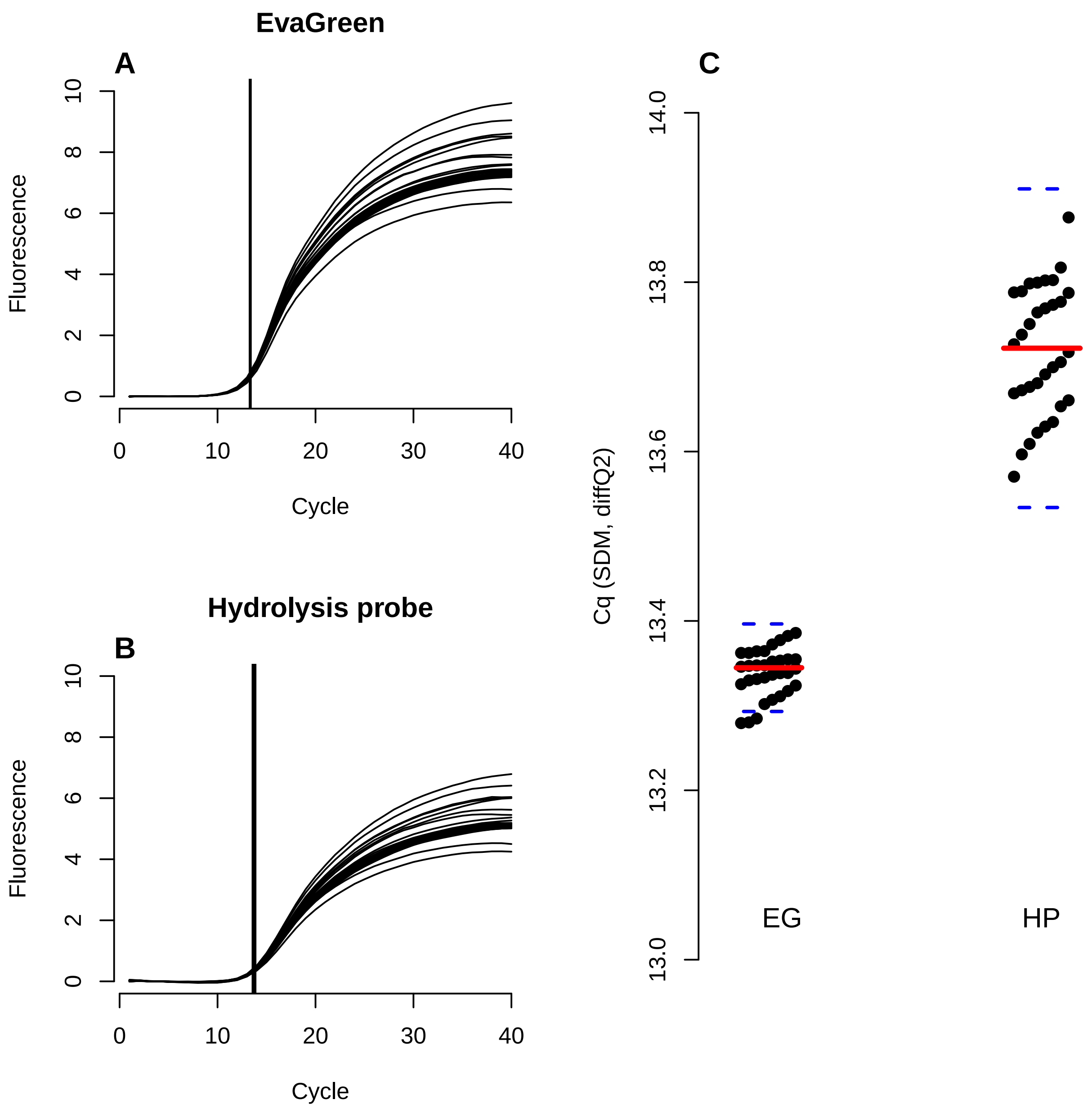


Figure S21: Plot of all data from C127EGHP and calculate the *SDM* (Second Derivative Maximum) values with the *diffQ2* function. *(A)* Plot the samples detected with EvaGreen and *(B)* shows the same samples detected with the Hydrolysis probe for *MLC-2v*. *(C)* Stripchart of the Cq values (●) with the median (–) and the median absolute deviation (– –). This result indicates, that the variance derived from detection with hydrolysis probes is higher than for the samples detected with EvaGreen. Note: the *inder* parameter is set as TRUE.

13 Quantification cycle calculation by the *inder* function

13.1 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*. Smoothing alters the peak signal (e.g., peak height reduction and peak width increase are commonly encountered problems). 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 (Ritz and Streibig, 2005) to fit a five-parameter log-logistic function (S-shaped). The *inder* function was used to calculate the *SDM* of the predicted models (Figure S22).

```
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)
```

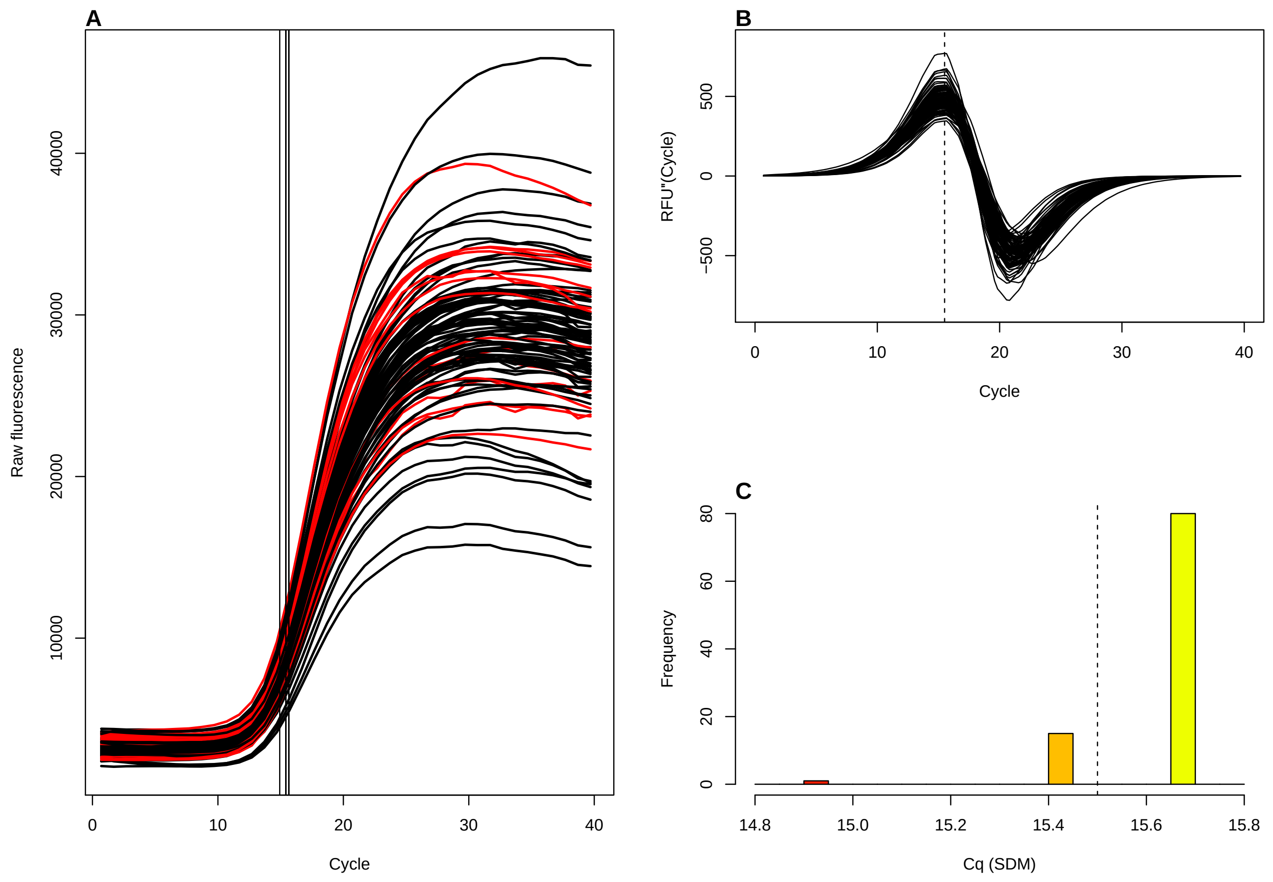


Figure S22: Amplification curve profiles from the Bio-Rad iQ5 thermo cycler for the human gene *HPRT1*. *(A)* The *C126EG595* dataset was used with 96 replicates of equal starting numbers of template molecules. Vertical lines represent the Cq (*SDM* method) determined by the *inder* method on amplification 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 calculates the Cq in qPCRs or cycle time in quantitative isothermal amplification (qIA) experiments. This method requires proper baselining. Note: more sophisticated quantification methods exist in qPCR analysis (Ruijter *et al.*, 2009, 2013; Tellinghuisen and Spiess, 2014). This function was implemented primarily for the analysis qIA experiments but is usable for qPCR too. We implemented a symmetric approximation algorithm based on linear and quadratic least squares regression. The Threshold Cycle (Ct) is the cycle number at which the reporter fluorescence signal significantly exceeds a point above the baseline and defined threshold in particular samples. 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. Therefore, *th.cyc* has no requirement to fit a "complex" non linear model to the entire dataset but rather focuses on a specific area. The polynomial is calculated from four neighbor values of the fluorescence threshold.

```
# Raw data from the VIMCFX96_69 dataset.  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* function on ccPCR data

In the following example we analyzed a continuous amplification reaction. The data were taken from the *capillaryPCR* dataset from the *chipPCR* package measured with capillary convective PCR (ccPCR) technology. We used a modified device of the ccPCR system as described by (Chou *et al.*, 2011). For technical details see the supplement of (Spiess *et al.*, 2014). The PCR was performed with SYBR(R) Green fluorescent intercalating dye. Thereof the ESE-Log has an excitation of 470 nm and a detection wavelenth of 520 nm in one channel. The data was recorded by the FL Digital Software (QIAGEN Lake Constance) and exported as text-based raw data. The raw data were noisy and showed an off-set of circa 150 micro Volt with a slightly negative trend (Figure S17 A). The data were first pre-processed (baselined and slightly smoothed) by the *CPP* function. The

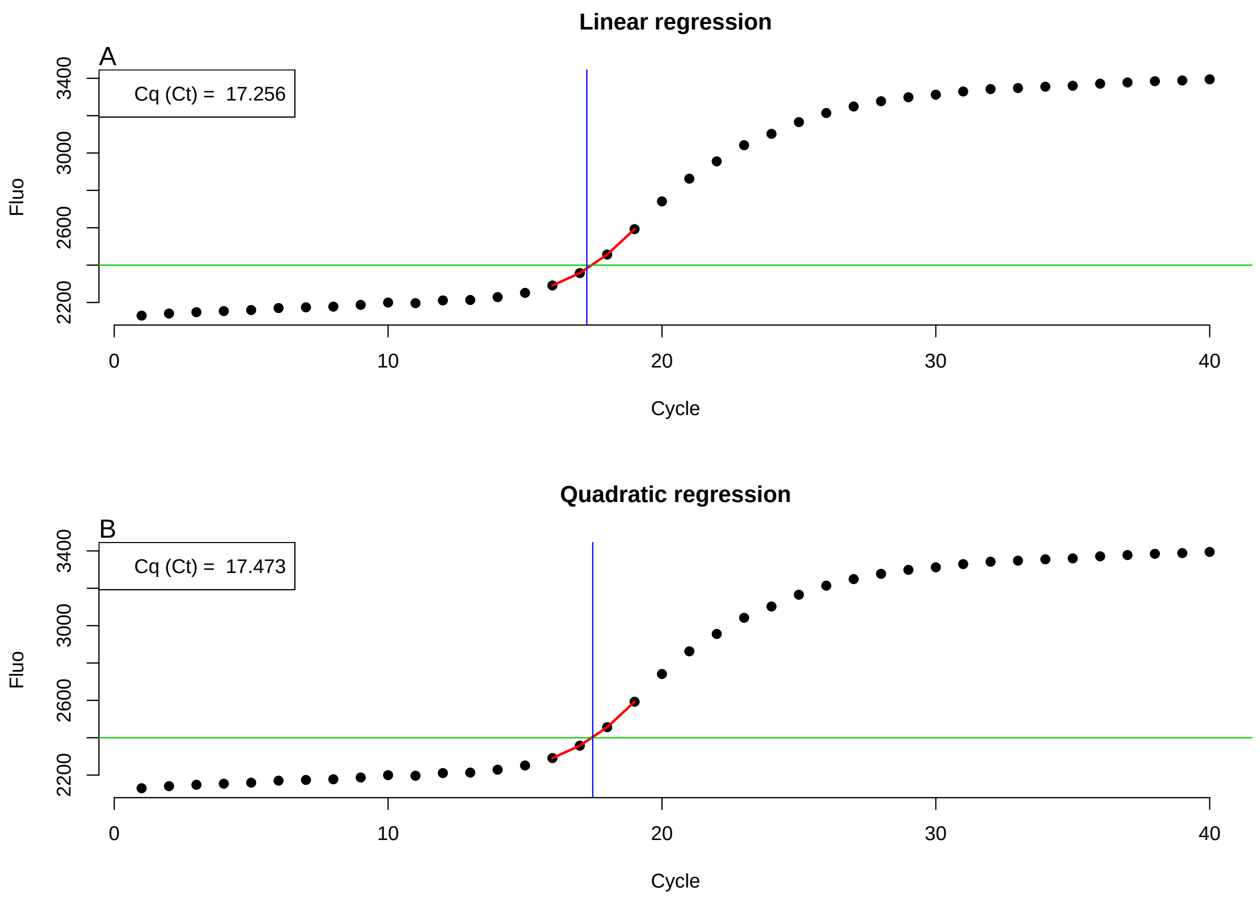


Figure S23: Working principle of the *th.cyc* function. The function provides two modes (**A**) is the linear regression. **B)** Quadratic regression for the calculation of the Cq. In both cases the highest R squared value determines how many left and right neighbors are used above and the below the defined threshold level .

th.cyc function was used to determine the time required to reach a certain threshold level above the defined value (Threshold level "r" (80 μVolts)) (Figure S24 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")
```

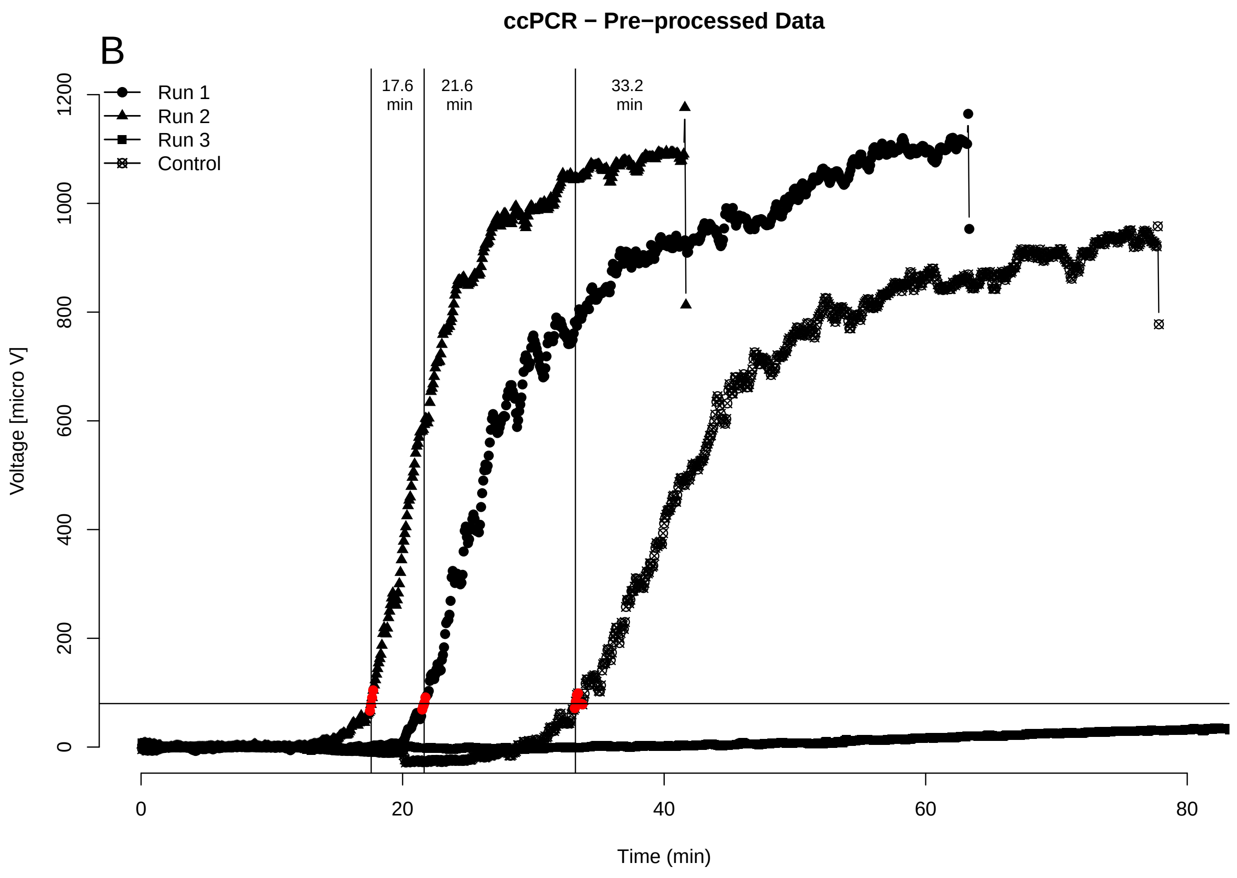


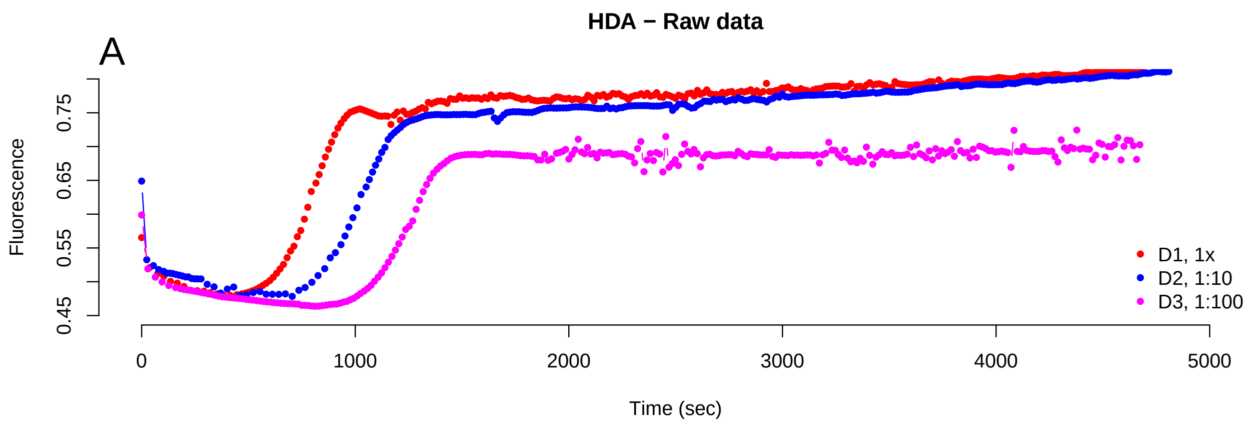
Figure S24: Application of the *th.cyc* function for the analysis of ccPCR data. The *CPP* function was used to pre-process the data. Subsequently, the data were analyzed using the *th.cyc* function in the linear regression mode. The threshold level ($r = 50$) was identical for all data. The Cq is shown 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* functions for helicase dependent Amplification

In this example for time-dependent reactions, we examined a helicase-dependent amplification (HDA). The target was human *vimentin* (*vim*). The VideoScan Platform (Rödiger *et al.*, 2013a) was used to monitor the amplification. The HDA was performed at 65 °C (Spiess *et al.*, 2014). Three concentrations of input DNA (D1, D2, D3) were used (Figure S25). In detail, the IsoAmp(R) III Universal tHDA Kit (Biohelix) was used. Primers and templates are described in (Rödiger *et al.*, 2013a). Reaction mix A was: 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. Measurement started immediately after after adding buffer B. Three dilutions of input DNA were used (1x (D1), a 1:10 (D2) and a 1:100 (D3) dilution). Temperature profile (after Preincubation):

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

```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 <- 0.025
# 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"))
}
```



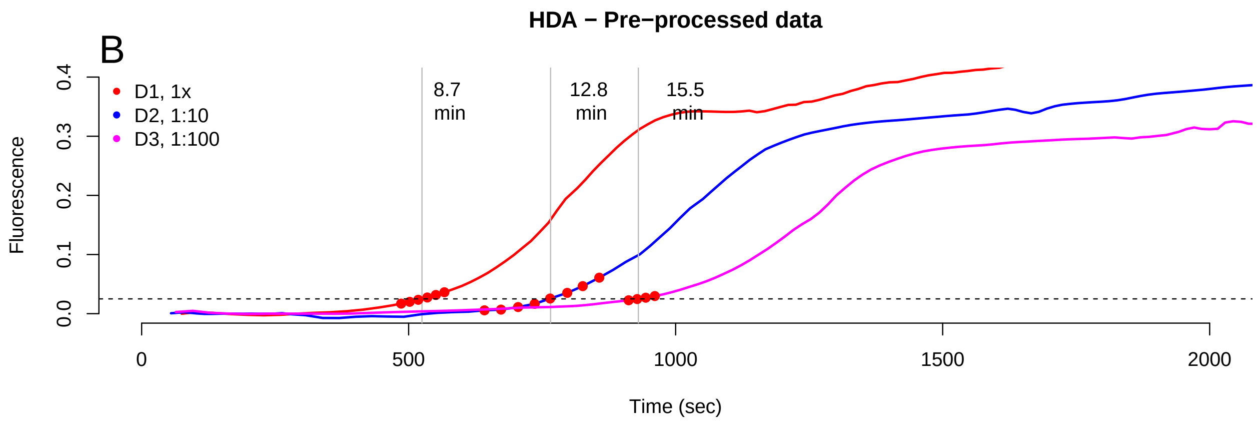


Figure S25: Helicase-dependent amplification (HDA) of *vimentin* (extitvim) in the VideoScan Platform. The HDA was performed at 65 degrees Celsius. Three concentrations of input DNA (D1, D2, D3) were used. The amplification curves were smoothed by a moving average (windowsize 3) and base-lined by a robust 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 (–) arbitrary fluorescence units.

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

15 Amplification efficiency

Various factors influence amplification reactions. The amplification efficiency (AE) 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 (Mehra and Hu, 2005). Some probe systems are perceived as bias-introducing. Therefore, qPCR reactions should be corrected based on the AE (Tuomi *et al.*, 2010; Ruijter *et al.*, 2014). The 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 (Liu and Saint, 2002; Tichopad *et al.*, 2003; Alvarez *et al.*, 2007; Smith *et al.*, 2007; Batsch *et al.*, 2008; Mallona *et al.*, 2011). The *qpcR* function has many functions for indirect estimation of the AE. However, the most commonly used is the "direct method" (Liu and Saint, 2002; Stååålberg *et al.*, 2003), whereby 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 S26). An object of the class list contains the "Concentration", Cqs, deviation of the Cqs, "Coefficient of Variance" sequentially in the columns, the AE (%) according to Equation S6, the results of the linear regression and the correlation test (Pearson's) (Table S5). 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 chipPCR manual.

Providing 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 S21 illustrates the use of the *diffQ* and *diffQ2* functions from the *MBmca* package (Rödiger *et al.*, 2013b) and the integration of the *inder* function. The *inder* function is used in *diffQ* and *diffQ2* for a precise peak location, while the approximate SDM is calculated from the derivative of a quadratic function at the approximate SDM.

```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()
```

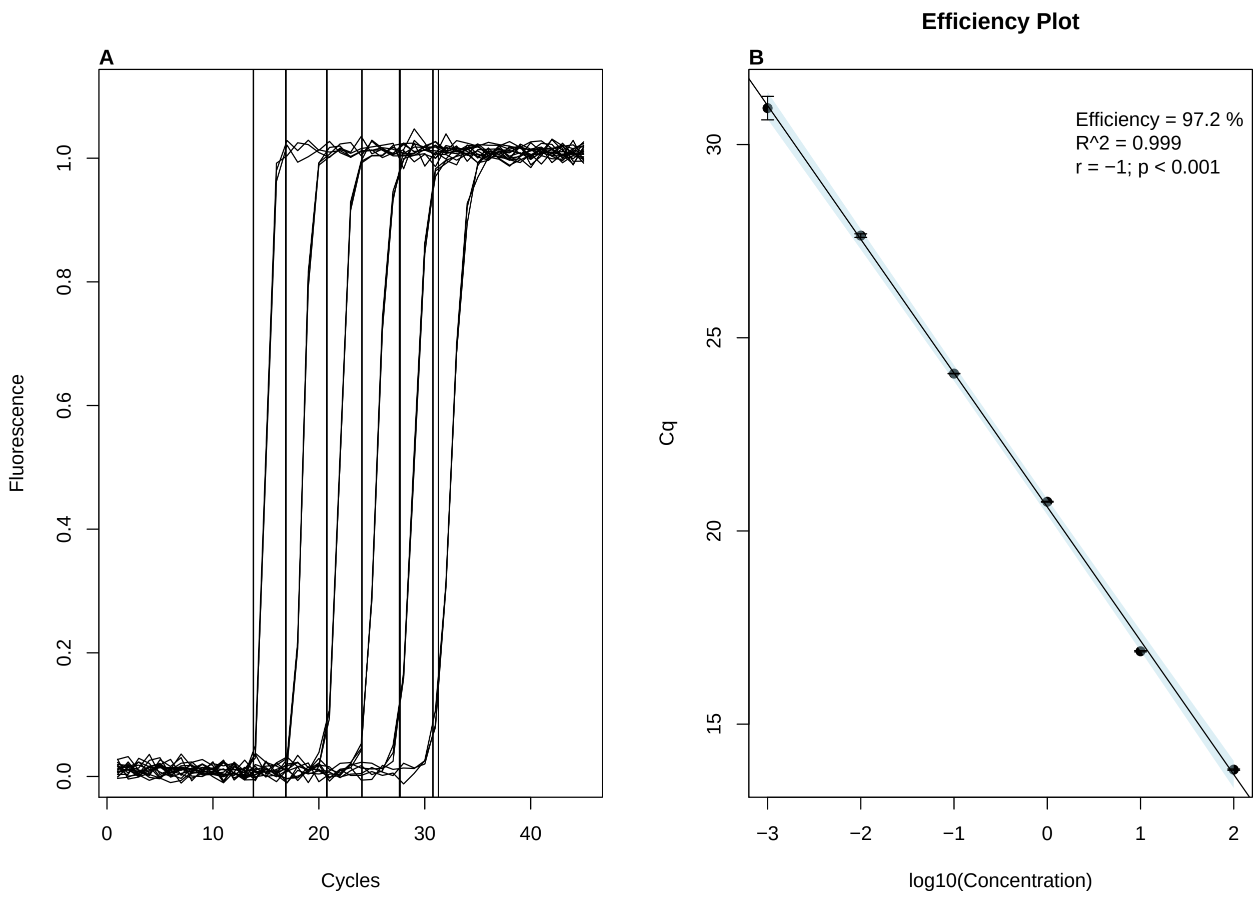


Figure S26: Amplification standard curve simulation and regression analysis. *(A) AmpSim* was used to synthesize a qPCR experiment of six dilutions (three replicates per dilution) standard samples. The Cqs were determined by the *SDM* method (solid black vertical lines). *(B) effcalc* was used to automatically perform a linear regression (decadic logarithm of input concentration versus the Cq). The 95% confidence interval is shown be the light-blue solid lines.

```
# 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)
```

Then we analyzed the *C54* dataset from the *chipPCR* package with the *effcalc* function. Herein, a qPCR experiment for the amplification of *MLC-2v* was performed using the VideoScan heating/cooling-unit. Calculation of the Cq required pre-processing of the data. One amplification curve contained a missing value (Figure S27 A) which was removed by the spline method of *CPP*. Additionally, the data were baselined (linear model, robust MM-estimator) and smoothed by Savitzky-Golay smoothing (Figure S27 B). A final analysis with the *effcalc* function showed that the AE was circa 87.3 % for the gene *MLC-2v* using the VideoScan HCU (Figure S27 C). However, since only few measurement points were tested for this experiment, it just safe to say that the hardware of the HCU works reliably under the experimental conditions.

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

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]), res.fit = NULL)
```

	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 S5: 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 S28 A) and *MLC-2v* (Figure S28 B) were amplified in a Roche Light Cycler 1.5 and detected by the HRM dye EvaGreen in independent experiments. As shown in the code and Figure S28 it is possible to obtain a complete analysis with few commands. The amplification efficiencies for both qPCRs was higher than 94 % (Figure S28 C and D, Table S5).

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

plot(NA, NA, xlim = c(0, 44), ylim = c(0, 6), xlab = "Cycles",
    ylab = "RFU")
legend(0, 6, colnames(C60.amp[, 4L:17]), ncol = 2, col = colors[1:14],
    pch = 19, bty = "n")
```

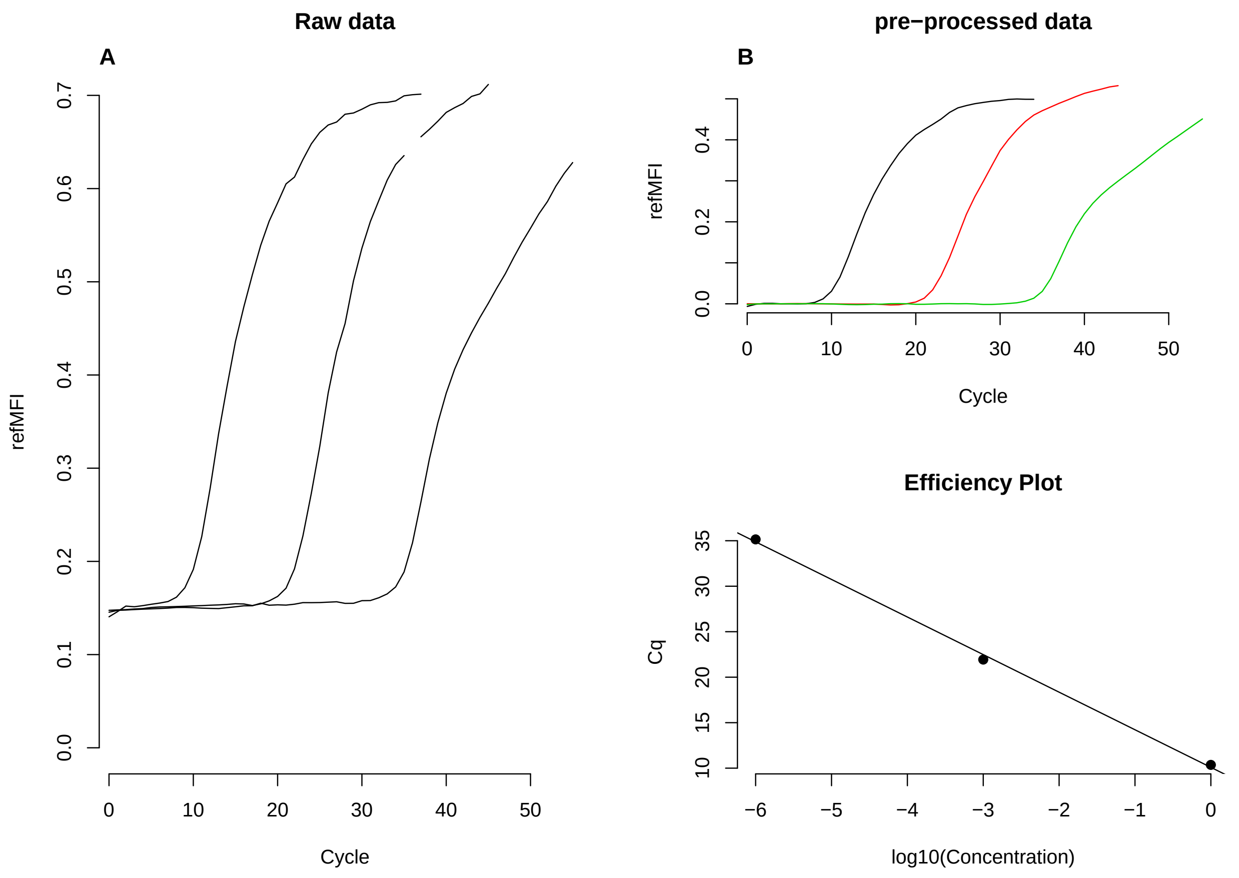


Figure S27: Calculation of the amplification efficiency. Data of a VideoScan HCU dilution experiment (C54 dataset) were analyzed. *(A)* Visualization of the raw data. One of the three dilutions contains a missing value due to a sensor error. *(B, top panel)* The *CPP* function was used to baseline, remove the missing value (–) and smooth (–, –, –) the raw data. *(B, bottom panel).* The Cqs (*SDM* by the *diffQ* function) were then determined. An amplification efficiency of 87.3 % was calculated with the *effcalc* function.

```r
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)
```

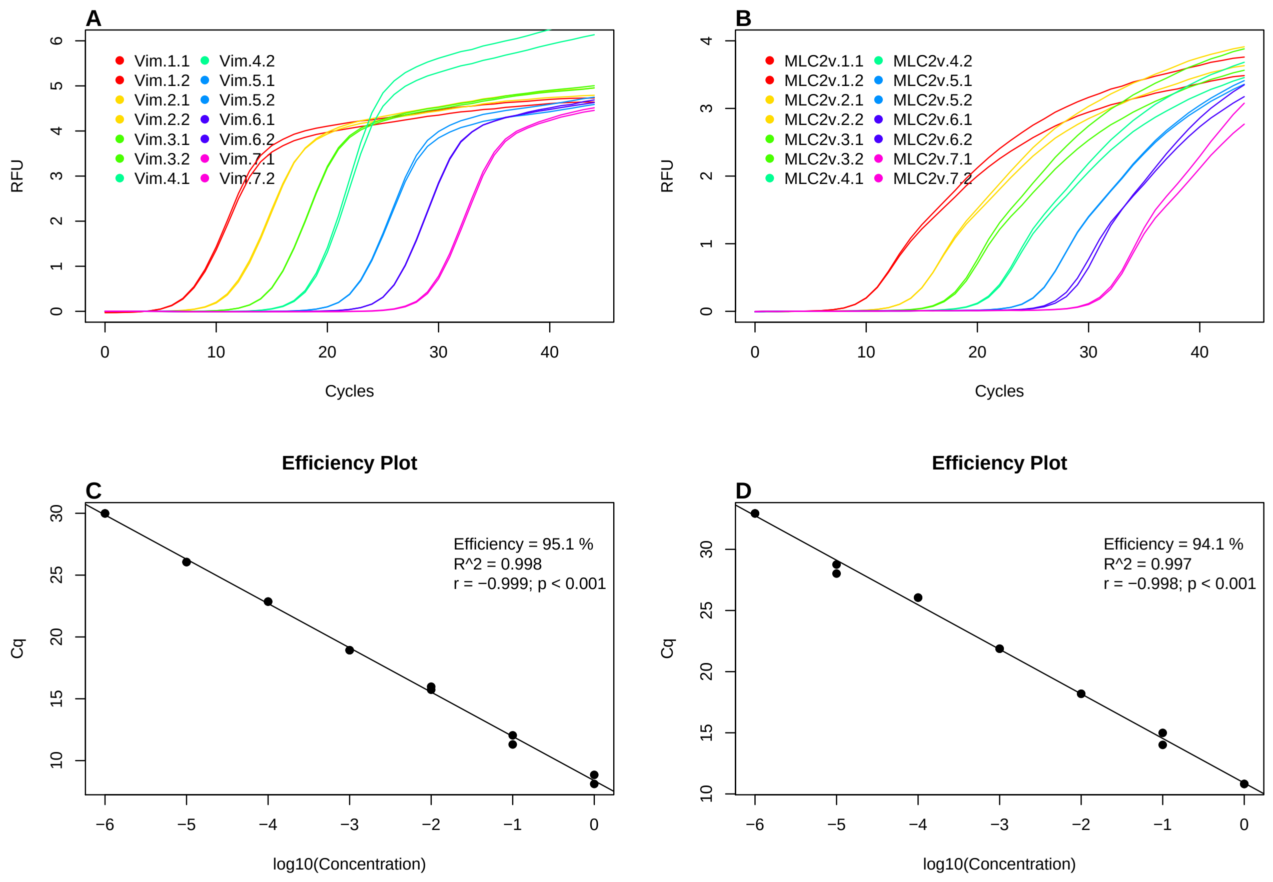


Figure S28: Calculation of the amplification efficiency (AE) from a qPCR experiment. DNA of *vimentin* (A) and *MLC-2v* (B) was diluted and amplified in a Roche Light Cycler 1.5 (C60.amp dataset). Cqs (SDM) were calculated by the *inder* function and analyzed with *effcalc*. The AE was (C) 95.1 % for *vimentin* and (D) 94.1 % for *MLC-2v*.

16 Datasets

1. Dataset: capillaryPCR:
   - Dataset 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. Dataset: C60.amp:
   - Dataset 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. Dataset: C60.melt:
   - Dataset 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. Dataset: C126EG595:
   - Dataset 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. Dataset: C126EG685:
   - Dataset 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. Dataset: C127EGHP:
   - Dataset 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. Dataset: 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
   - Dataset 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. Dataset: 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
   - Dataset 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. Dataset: 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
   - Dataset 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. Dataset: 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
    - Dataset 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. Dataset: 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
    - Dataset 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. Dataset: 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
    - Dataset 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. Dataset: C54:
    - Dataset 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. Dataset: CD74:
    - Dataset 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. Dataset: Eff625:
    - Dataset 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. Dataset: Eff750:
    - Dataset 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. Dataset: Eff875:
    - Dataset 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. Dataset: Eff1000:
    - Dataset 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. Dataset: C67:
    - Dataset type: Isothermal Amplification - 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. Dataset: CD75:
    - Dataset type: Isothermal Amplification - 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. Dataset: C81:

- Dataset type: Isothermal Amplification - 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. Dataset: C85:

    - Dataset type: Isothermal Amplification - 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 InnoProfileTransfer-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 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 version of the manuscript.

List of Figures

References

Alvarez, M. J., Vila-Ortiz, G. J., Salibe, M. C., Podhajcer, O. L., and Pitossi, F. J. (2007). Model based analysis of real-time PCR data from DNA binding dye protocols. *BMC Bioinformatics*, **8**(1), 85.

Batsch, A., Noetel, A., Fork, C., Urban, A., Lazic, D., Lucas, T., Pietsch, J., Lazar, A., Schömig, E., and Gründemann, D. (2008). Simultaneous fitting of real-time PCR data with efficiency of amplification modeled as Gaussian function of target fluorescence. *BMC Bioinformatics*, **9**(1), 95.

Blagodatskikh, K. A., Roediger, S., and Burdukiewicz, M. (2014). *Importing real-time thermo cycler (qPCR) data from RDML format files*.

Blanton, B. and Lenhardt, C. (2014). A scientist's perspective on sustainable scientific software. *Journal of Open Research Software*, **2**(1).

Bustin, S. A., Benes, V., Garson, J. A., Hellemans, J., Huggett, J., Kubista, M., Mueller, R., Nolan, T., Pfaffl, M. W., Shipley, G. L., Vandesompele, J., and Wittwer, C. T. (2009). The MIQE guidelines: minimum information for publication of quantitative real-time PCR experiments. *Clinical Chemistry*, **55**(4), 611–622.

Chang, C.-C., Chen, C.-C., Wei, S.-C., Lu, H.-H., Liang, Y.-H., and Lin, C.-W. (2012). Diagnostic devices for isothermal nucleic acid amplification. *Sensors (Basel, Switzerland)*, **12**(6), 8319–8337. PMID: 22969402 PMCID: PMC3436031.

Chou, W. P., Chen, P. H., Miao, M., Kuo, L. S., Yeh, S. H., and Chen, P. J. (2011). Rapid DNA amplification in a capillary tube by natural convection with a single isothermal heater. *BioTechniques*, **50**(1), 52–57.

Cobb, B. D. and Clarkson, J. M. (1994). A simple procedure for optimising the polymerase chain reaction (PCR) using modified Taguchi methods. *Nucleic Acids Research*, **22**(18), 3801–3805. 00249 PMID: 7937094.

Dahlquist, G. and Björck, Å. (2008). Numerical Methods in Scientific Computing. volume II. Society for Industrial and Applied Mathematics.

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

Eilers, P. H. C. (2003). A Perfect Smoother. *Analytical Chemistry*, **75**(14), 3631–3636.

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

Gehlenborg, N., Noble, M. S., Getz, G., Chin, L., and Park, P. J. (2013). Nozzle: a report generation toolkit for data analysis pipelines. *Bioinformatics*, **29**(8), 1089–1091.

Gerhard, D., Bremer, M., and Ritz, C. (2014). Estimating marginal properties of quantitative real-time PCR data using nonlinear mixed models. *Biometrics*, **70**(1), 247–254.

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

Heckmann, L.-H., Sørensen, P. B., Krogh, P. H., and Sørensen, J. G. (2011). NORMA-Gene: A simple and robust method for qPCR normalization based on target gene data. *BMC Bioinformatics*, **12**(1), 250.

Huntley, M. A., Larson, J. L., Chaivorapol, C., Becker, G., Lawrence, M., Hackney, J. A., and Kaminker, J. S. (2013). ReportingTools: an automated result processing and presentation toolkit for high-throughput genomic analyses. *Bioinformatics*, **29**(24), 3220–3221.

Jacobs, C., Avdis, A., Gorman, G., and Piggott, M. (2014). Pyrdm: A python-based library for automating the management and online publication of scientific software and data. *Journal of Open Research Software*, **2**(1).

Karatzoglou, A., Smola, A., Hornik, K., and Zeileis, A. (2004). kernlab - An S4 Package for Kernel Methods in R. *Journal of Statistical Software*, **11**(9), 1–20.

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

Koenker, R. (2008). Censored Quantile Regression Redux. *Journal of Statistical Software*, **27**(6), 1–25.

Komsta, L. (2011). *outliers: Tests for outliers*. R package version 0.14.

Larionov, A., Krause, A., and Miller, W. (2005). A standard curve based method for relative real time PCR data processing. *BMC Bioinformatics*, **6**(1), 62. PMID: 15780134.

Liu, W. and Saint, D. A. (2002). 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. PMID: 11846375.

Mallona, I., Weiss, J., and Egea-Cortines, M. (2011). pcrEfficiency: a Web tool for PCR amplification efficiency prediction. *BMC Bioinformatics*, **12**(1), 404.

McCall, M. N., McMurray, H. R., Land, H., and Almudevar, A. (2014). On non-detects in qPCR data. *Bioinformatics (Oxford, England)*.

Mehra, S. and Hu, W.-S. (2005). A kinetic model of quantitative real-time polymerase chain reaction. *Biotechnology and bioengineering*, **91**(7), 848–860. PMID: 15986490.

Nie, Z. and Racine, J. S. (2012). The crs Package: Nonparametric Regression Splines for Continuous and Categorical Predictors. *The R Journal*, **4**(2), 48–56.

Pabinger, S., Rödiger, S., Kriegner, A., Vierlinger, K., and Weinhäusel, A. (2014). A survey of tools for the analysis of quantitative PCR (qPCR) data. *Biomolecular Detection and Quantification*, **1**(1), 23–33. 00000.

Peirson, S. N., Butler, J. N., and Foster, R. G. (2003). Experimental validation of novel and conventional approaches to quantitative real-time PCR data analysis. *Nucleic Acids Research*, **31**(14), e73. PMID: 12853650 PMCID: PMC167648.

Perkins, J. R., Dawes, J. M., McMahon, S. B., Bennett, D. L., Orengo, C., and Kohl, M. (2012). ReadqPCR and NormqPCR: R packages for the reading, quality checking and normalisation of RT-qPCR quantification cycle (Cq) data. *BMC Genomics*, **13**(1), 296. PMID: 22748112.

R Core Team (2013). R: A Language and Environment for Statistical Computing. Vienna, Austria.

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

Rao, X., Lai, D., and Huang, X. (2013). 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. PMID: 23841653 PMCID: PMC3762066.

Ritz, C. and Spiess, A.-N. (2008). qpcR: an R package for sigmoidal model selection in quantitative real-time polymerase chain reaction analysis. *Bioinformatics*, **24**(13), 1549–1551. PMID: 18482995.

Ritz, C. and Streibig, J. C. (2005). Bioassay Analysis using R. *Journal of Statistical Software*, **12**.

Rödiger, S., Friedrichsmeier, T., Kapat, P., and Michalke, M. (2012). RKWard: A Comprehensive Graphical User Interface and Integrated Development Environment for Statistical Analysis with R. *Journal of Statistical Software*, **49**(9), 1–34.

Rödiger, S., Schierack, P., Böhm, A., Nitschke, J., Berger, I., Frömmel, U., Schmidt, C., Ruhland, M., Schimke, I., Roggenbuck, D., Lehmann, W., and Schröder, C. (2013a). 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.

Rödiger, S., Böhm, A., and Schimke, I. (2013b). Surface Melting Curve Analysis with R. *The R Journal*, **5**(2), 37–53.

Rödiger, S., Schierack, P., Böhm, A., Nitschke, J., Lehmann, W., and Schröder, C. (2013c). VideoScan - A Microscope Imaging Technology Platform for the Multiplex Real-Time PCR. *qPCR & NGS 2013 Prodceedings; ISBN 9783000410246*.

Rödiger, S., Liebsch, C., Schmidt, C., Lehmann, W., Resch-Genger, U., Schedler, U., and Schierack, P. (2014). Nucleic acid detection based on the use of microbeads: a review. *Microchimica Acta*, **181**(11-12), 1151–1168.

Roth, T. (2012). *qualityTools: Statistics in Quality Science.* R package version 1.54 http://www.r-qualitytools.org.

RStudio and Inc. (2014). *shiny: Web Application Framework for R.* R package version 0.10.1.

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

Ruijter, J. M., Pfaffl, M. W., Zhao, S., Spiess, A. N., Boggy, G., Blom, J., Rutledge, R. G., Sisti, D., Lievens, A., De Preter, K., Derveaux, S., Hellemans, J., and Vandesompele, J. (2013). Evaluation of qPCR curve analysis methods for reliable biomarker discovery: bias, resolution, precision, and implications. *Methods (San Diego, Calif.)*, **59**(1), 32–46. PMID: 22975077.

Ruijter, J. M., Lorenz, P., Tuomi, J. M., Hecker, M., and van den Hoff, M. J. B. (2014). 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.

Savitzky, A. and Golay, M. J. E. (1964). Smoothing and Differentiation of Data by Simplified Least Squares Procedures. *Analytical Chemistry*, **36**(8), 1627–1639.

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

Smith, M. V., Miller, C. R., Kohn, M., Walker, N. J., and Portier, C. J. (2007). Absolute estimation of initial concentrations of amplicon in a real-time RT-PCR process. *BMC Bioinformatics*, **8**(1), 409.

Spiess, A.-N. (2014). *qpcR: Modelling and analysis of real-time PCR data.* R package version 1.4-0.

Spiess, A.-N., Feig, C., and Ritz, C. (2008). Highly accurate sigmoidal fitting of real-time PCR data by introducing a parameter for asymmetry. *BMC Bioinformatics*, **9**(1), 221.

Spiess, A.-N., Deutschmann, C., Burdukiewicz, M., Himmelreich, R., Klat, K., Schierack, P., and Rödiger, S. (2014). Impact of Smoothing on Parameter Estimation in Quantitative DNA Amplification Experiments. *Clinical Chemistry*, page clinchem.2014.230656. 00000.

Ståålberg, A., Aman, P., Ridell, B., Mostad, P., and Kubista, M. (2003). 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. PMID: 12507960.

Stodden, V. and Miguez, S. (2014). Best practices for computational science: Software infrastructure and environments for reproducible and extensible research. *Journal of Open Research Software*, **2**(1).

Tellinghuisen, J. and Spiess, A.-N. (2014). Comparing real-time quantitative polymerase chain reaction analysis methods for precision, linearity, and accuracy of estimating amplification efficiency. *Analytical Biochemistry*, **449**, 76–82. PMID: 24365068.

Thanakiatkrai, P. and Welch, L. (2012). Using the taguchi method for rapid quantitative PCR optimization with SYBR green i. *International Journal of Legal Medicine*, **126**(1), 161–165. 00007.

Tichopad, A., Dilger, M., Schwarz, G., and Pfaffl, M. W. (2003). Standardized determination of real-time PCR efficiency from a single reaction set-up. *Nucleic Acids Research*, **31**(20), e122. PMID: 14530455 PMCID: PMC219490.

Todorov, V. and Filzmoser, P. (2009). An Object-Oriented Framework for Robust Multivariate Analysis. *Journal of Statistical Software*, **32**(3), 1–47.

Tuomi, J. M., Voorbraak, F., Jones, D. L., and Ruijter, J. M. (2010). 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. PMID: 20138998.

Tusell, F. (2011). Kalman Filtering in R. *Journal of Statistical Software*, **39**(2), 1–27.

Wilhelm, J., Pingoud, A., and Hahn, M. (2003). Real-time PCR-based method for the estimation of genome sizes. *Nucleic Acids Research*, **31**(10), e56. PMID: 12736322 PMCID: PMC156059.

Zhang, J. D., Biczok, R., and Ruschhaupt, M. (2010). *ddCt: The ddCt Algorithm for the Analysis of Quantitative Real-Time PCR (qRT-PCR)*.

Zhang, X. D. and Zhang, Z. (2013). displayHTS: a R package for displaying data and results from high-throughput screening experiments. *Bioinformatics*, **29**(6), 794–796.

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