Stefan Rödiger* and Michał Burdukiewicz and Peter Schierack

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

Contents

Abstract

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

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

1 Setting up a work environment

The vignette can be viewed from **R** using command: vignette("chipPCR").

Further details of the experimental set-up for the data sets are described in the manual of the *chipPCR* package.

Before the start of any analysis, a user must must choose data set, as shown in example below.

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

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

Table S1: First five cycles of imported data.

All datasets used in following examples can be loaded in the same manner. They atre also automatically avaible after loading whole package.

2 Inspection and analysis of amplification curve data

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

2.1 MFIaggr

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

In our example we analyzed the raw fluorescence from 96 replicates ("VIMCFX96_60" data set) of a qPCR experiment for the human gene *Vimentin*. The *MFIaggr* plot shows that the analysis of all cycles is non-normal distributed (Figure S1).

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

MFIaggr can be used to analyze the heteroskedasticity in a given data set. Heteroskedasticity ("hetero" = different, "skedasis" = dispersion) is present if the variance (error term) is not constant. In case the variance is constant data are considered to be homoskedastic. If the error terms do not have constant variance, they are said to be homoskedastic. Analysis of the heteroskedasticity can give some insight into the characteristics of a system. In the following example we compared the VIMCFX96_60 and VIMCFX96_69 data sets. Both data set were obtained from the same qPCR run in a Bio-Rad CFX96. However, data from the VIMCFX96_60 data set were obtained during the annealing phase at 60 degree Celsius and data from the VIMCFX96_60 data set were measured during the elongation phase at 69 degree Celsius. The heteroskedasticity increased expectedly during the amplification reaction. The variance in the elongation phase (Figure S1C and D) was lower than in the annealing phase (Figure S1A and B). The heteroskedasticity was significant in during the (Figure S1A) first 15 cycles at 60 degree Celsius.

```
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)],
    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)
```

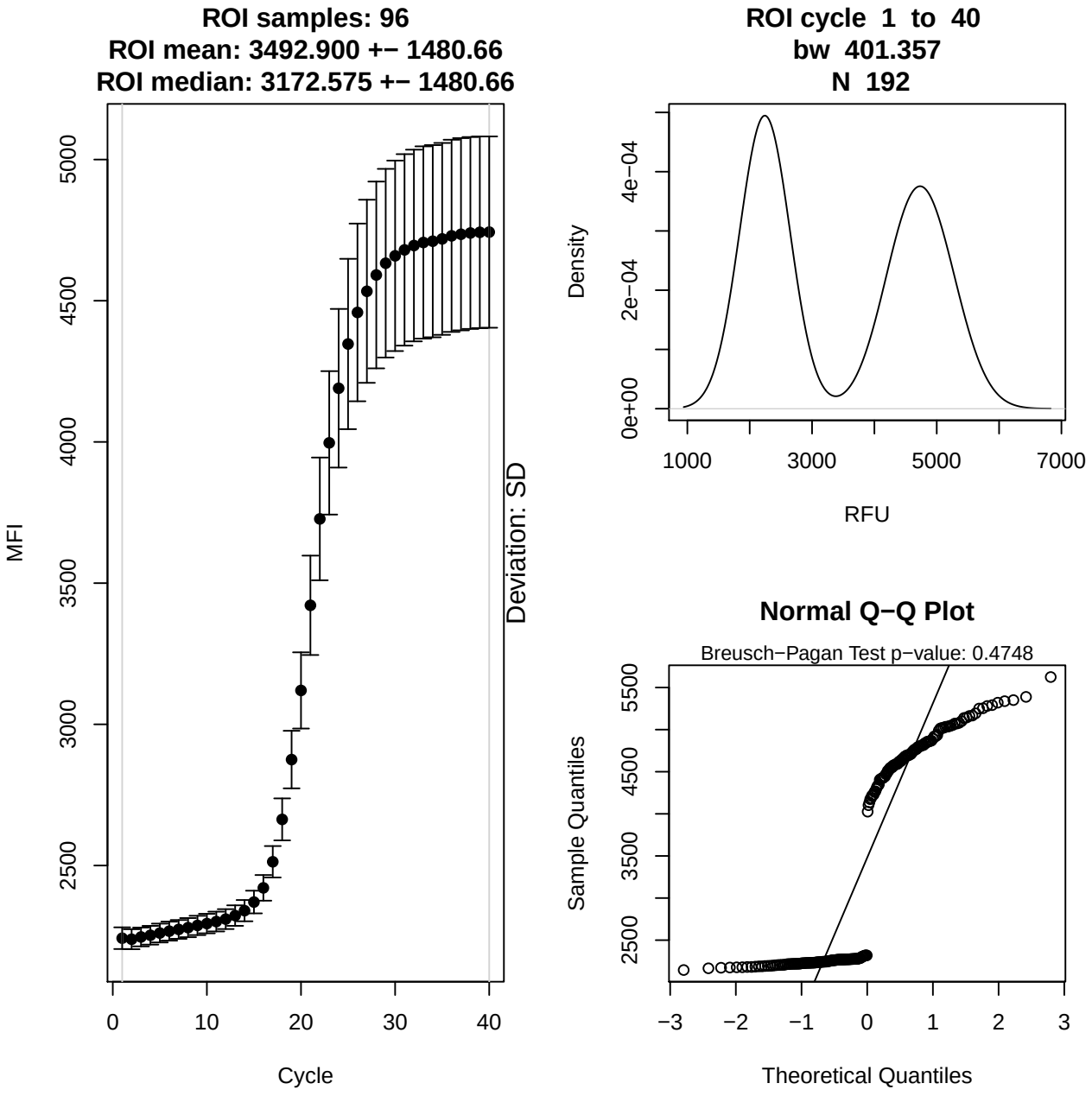

Figure S1: Signal analysis using the VIMCFX96_60 data set (96-well plate cycler (Bio-Rad CFX96)). All cycles (ROI: 1 – 40) were analyzed by the MFIaggr function. The density plot (right upper panel) and quantile-quantile analysis (right lower panel) show no normal distribution. Due to the sigmoidal curve structure is the density function bimodal.

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

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

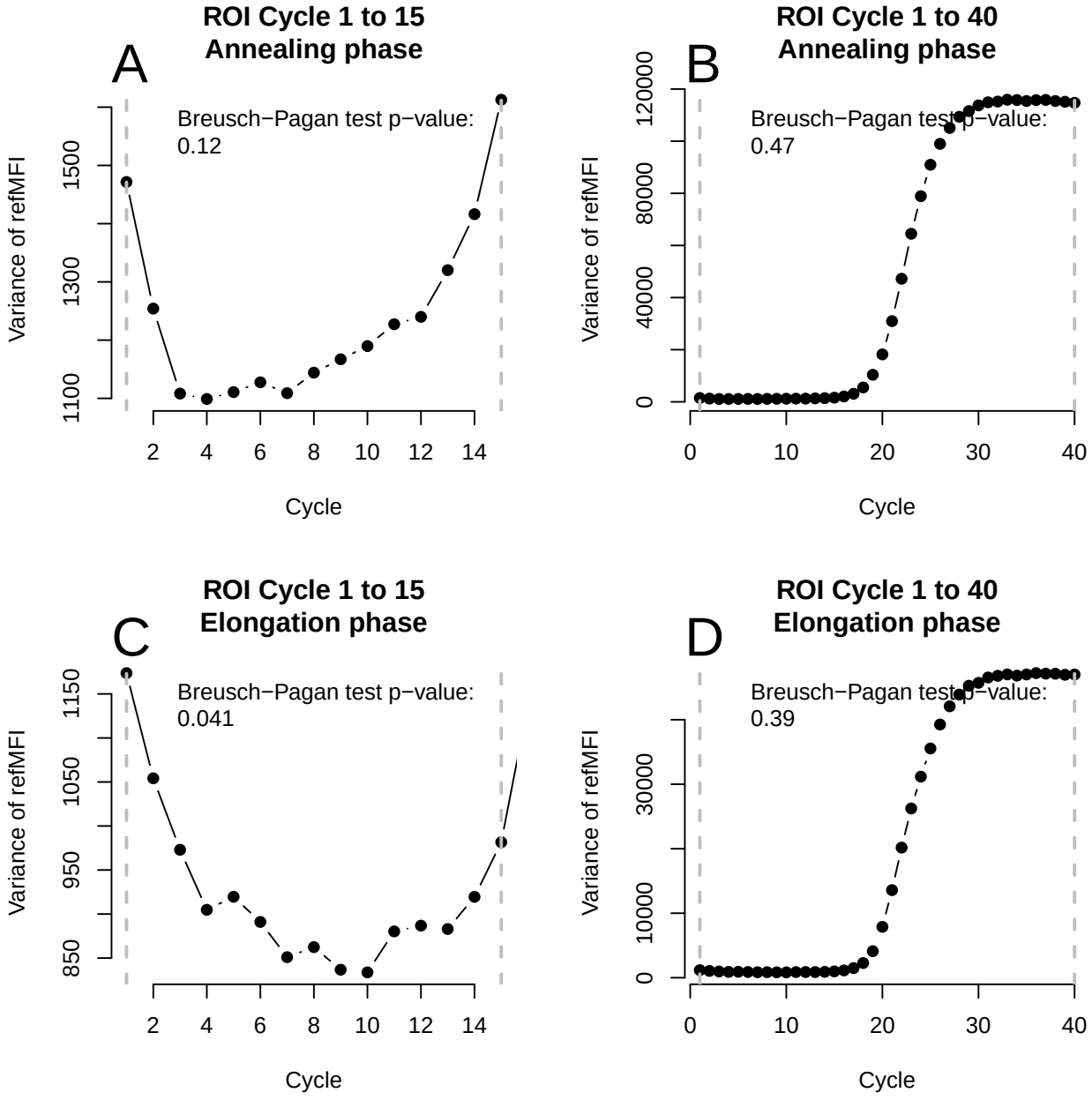

Figure S2: Use of MFIaggr to test for heteroskedasticity using the Breusch-Pagan test. The data were aggregated with the MFIaggr function and assigned to the object res. The standard deviation was transformed to the variance. The plot shows the cycle dependent variance measured at 60 degree Celsius (annealing phase; A, B) and 69 degree Celsius (elongation phase, C, D). First cycles 1 to 10 of 96 qPCR replicate amplification curves were analyzed. Next the cycles 1 to 40 of the same amplification curve data were analyzed. The Breusch-Pagan confirmed the heteroskedasticity in the amplification curve data. The VIMCFX96_60 and VIMCFX96_69 data sets were used.

2.2 Data overview - plotCurves

plotCurves visualizes many curves on one plot in separate cells allowing quick experiment assessment (Figure S3). In addition, *plotCurves* has an option to run an unsupervised *CPP* pre-processing step on the raw data. This will smooth the data (Savitzky-Golay Smoothing), remove missing values (spline interpolation by default) and perform a background subtraction (base-lining to zero).

Warnings in following code chunks were supressed.

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

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

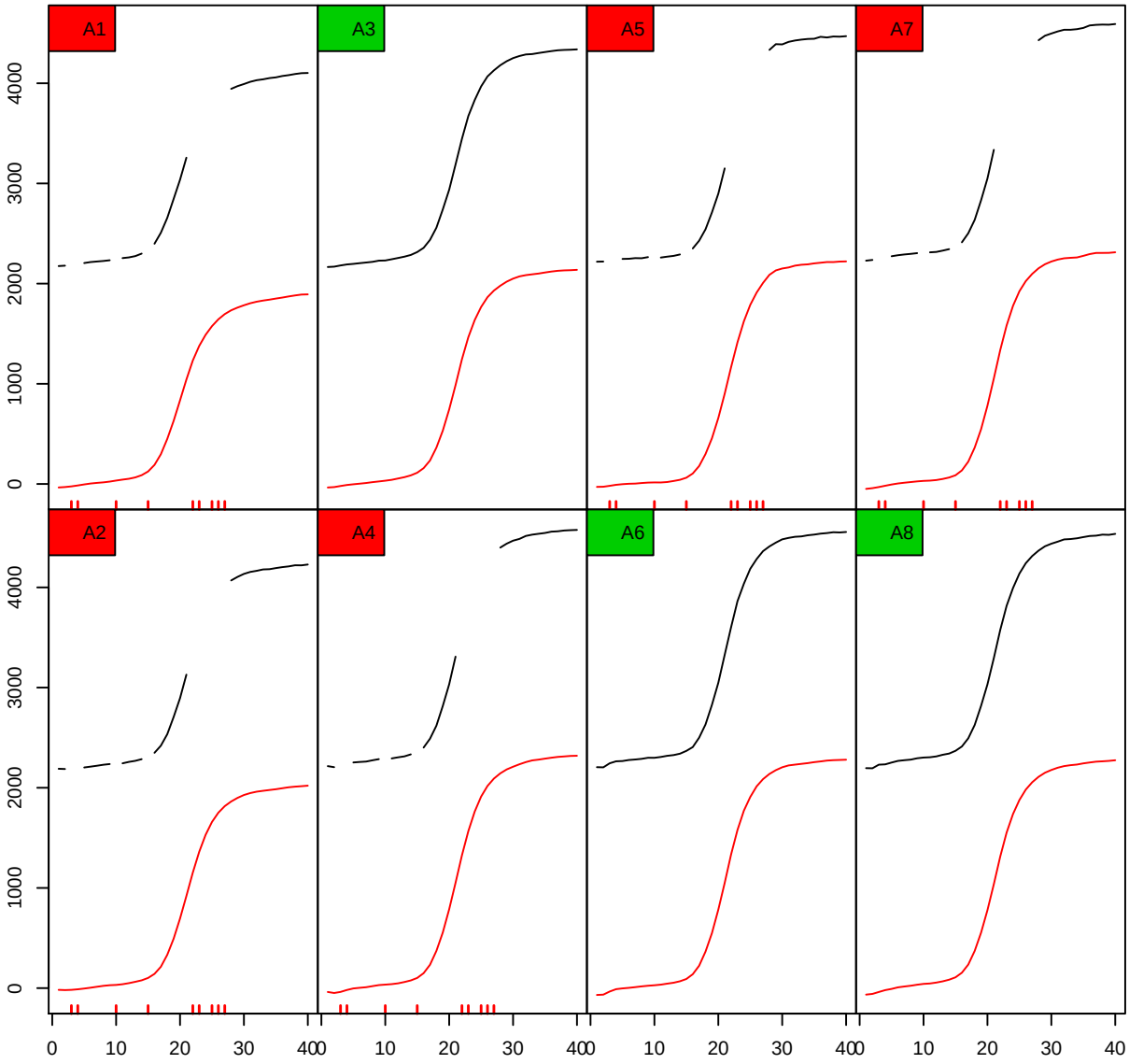

Figure S3: The plotCurves function. Plots many curves on one plot in separate cells allowing quick assessment. Missing values were artificially introduced at random position to selected curves of the VIMCFX96_60 data set (solid black line). A colored box (topleft of each plot) indicates the sample name and if the data contain missing values. The red rug indicates the position of the missing values. The red lined shows the amplification curve after unsupervised pre-processesing (using an instance of CPP).

2.3 Imputation of missing values in amplification curve data - fixNA

Amplification data of experimental systems may contain missing values (NA). 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. Imputation encompasses various approaches. This includes to calculate a location parameter (e.g., mean, median) or other significant values (e.g., minimum, maximum, modus) of a data column. However, in non-linear processes such as amplification processes its is better to estimate the missing values from a trend. The function *fixNA* was empirically tested and relies on a linear trend estimation based on the approx function. This approach is useful but may be problematic on the phases other then background or plateau phases of an amplification reaction. The parameter spline on *fixNA* enables a trend estimation on splines and may be more appropriate in most scenarios. Other smoothing functions such as the Savitzky-Golay smoothing filter have the intrinsic capability to remove missing values [4, 18]. The function *fixNA* imputes missing values in a single column of data. The imputation is based on a linear approximation by default. However, the data can also be estimated from an approximation by splines.

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

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

Table S2: Results of fixNA data imputation.

```
# 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)
```

```r
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))
```

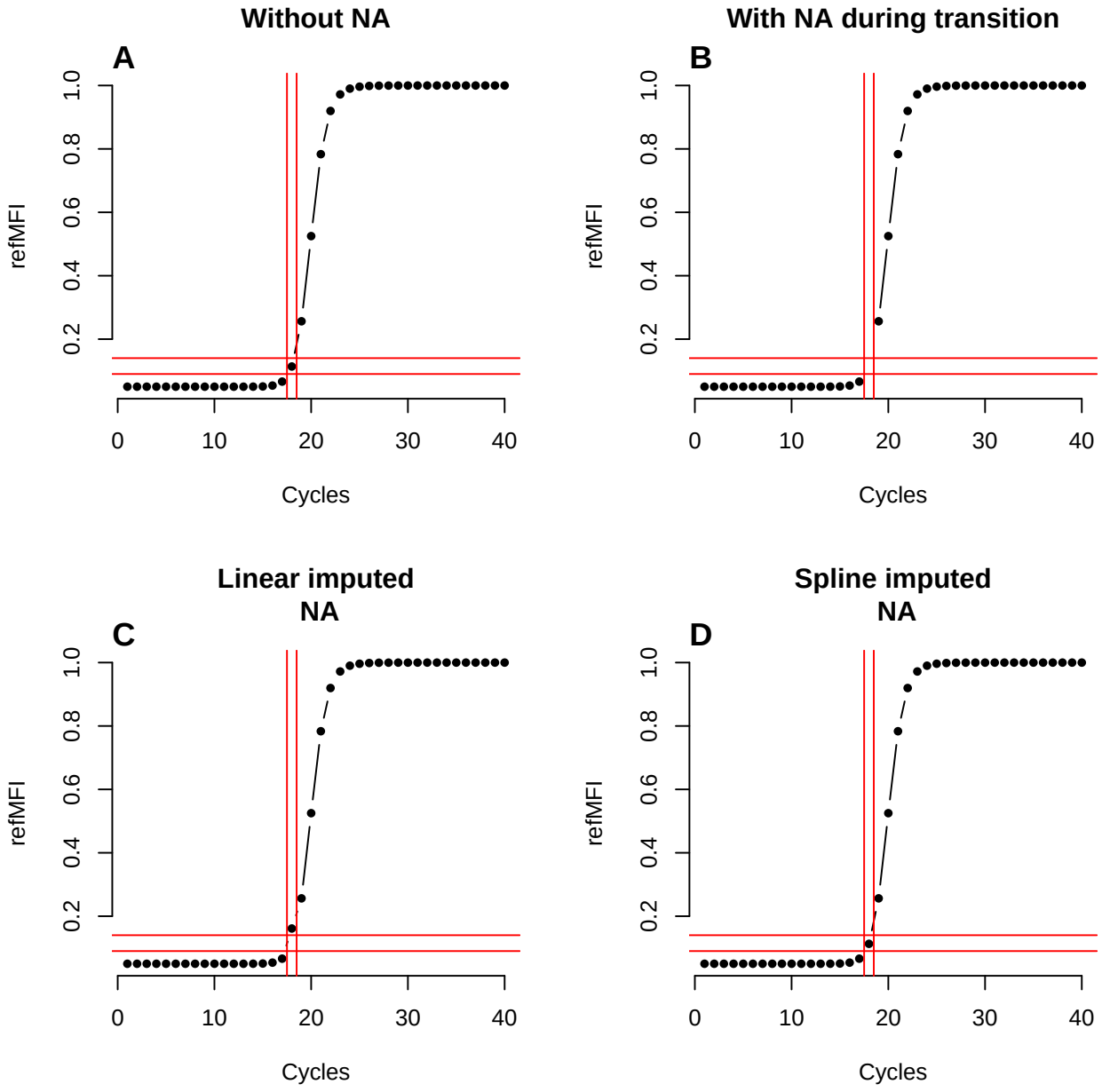

Figure S4: Imputation of missing values in amplification curve data. *(A)* Raw data were generated using the AmpSim simulation function. *(B)* A missing value was introduced in the transition phase. The missing value was imputed either by *(C)* linear approximation or *(D)* a cubic spline approximation. The spline approximation nearly reconstituted the original curve.

3 Normalization of amplification curve data

normalizer is a function to normalize any data set. It is possible to chose from different methods (see Details). The function is useful if the data from an experiment have considerable variation regarding the background and plateau signal.

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

- A quantile normalization herein refers to an approach, which is less prone to outliers than a normalization based on the minimum and the maximum of an amplification curve.
- minmax does a normalization between 0 and 1 (see [13] 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.

```
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 = "minmax")$y))

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

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

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

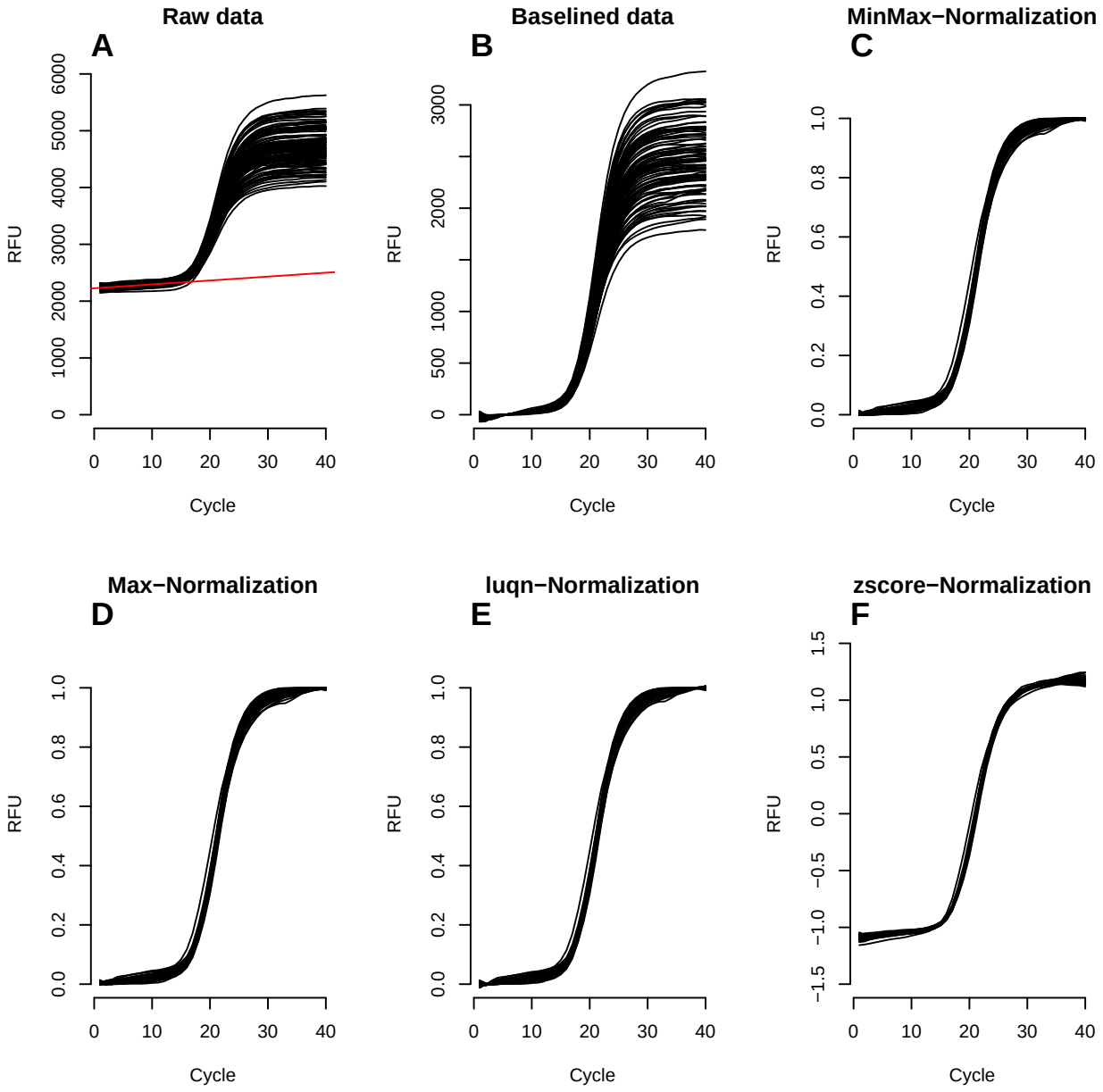

Figure S5: Comparison of the normalization functions from *CPP*. The VIMCFX96_60 data set (96-well plate cycler, Bio-Rad CFX96, EvaGreen detection) was used. *(A)* Raw data of all amplification curves. The signals are superimposed to circa 2200 RFU and the inter-sample baseline and plateau shift is high. Note the positive trend (–, fitted with an ordinary least squares method) in the background range of cycles 1 to 15. All subsequent plots were processed with the CPP function. By default, the curves are base-lined, smoothed (Savitzky-Golay smoother) and the slop corrected by a linear regression ($trans = TRUE$). *(B)* base-lined raw data, *(C) Min-Max normalization*, *(D) Max normalization*, *(E) lugn-normalization* with a cut off 3% and *(F) zscore-normalization*.

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

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

```
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)
```

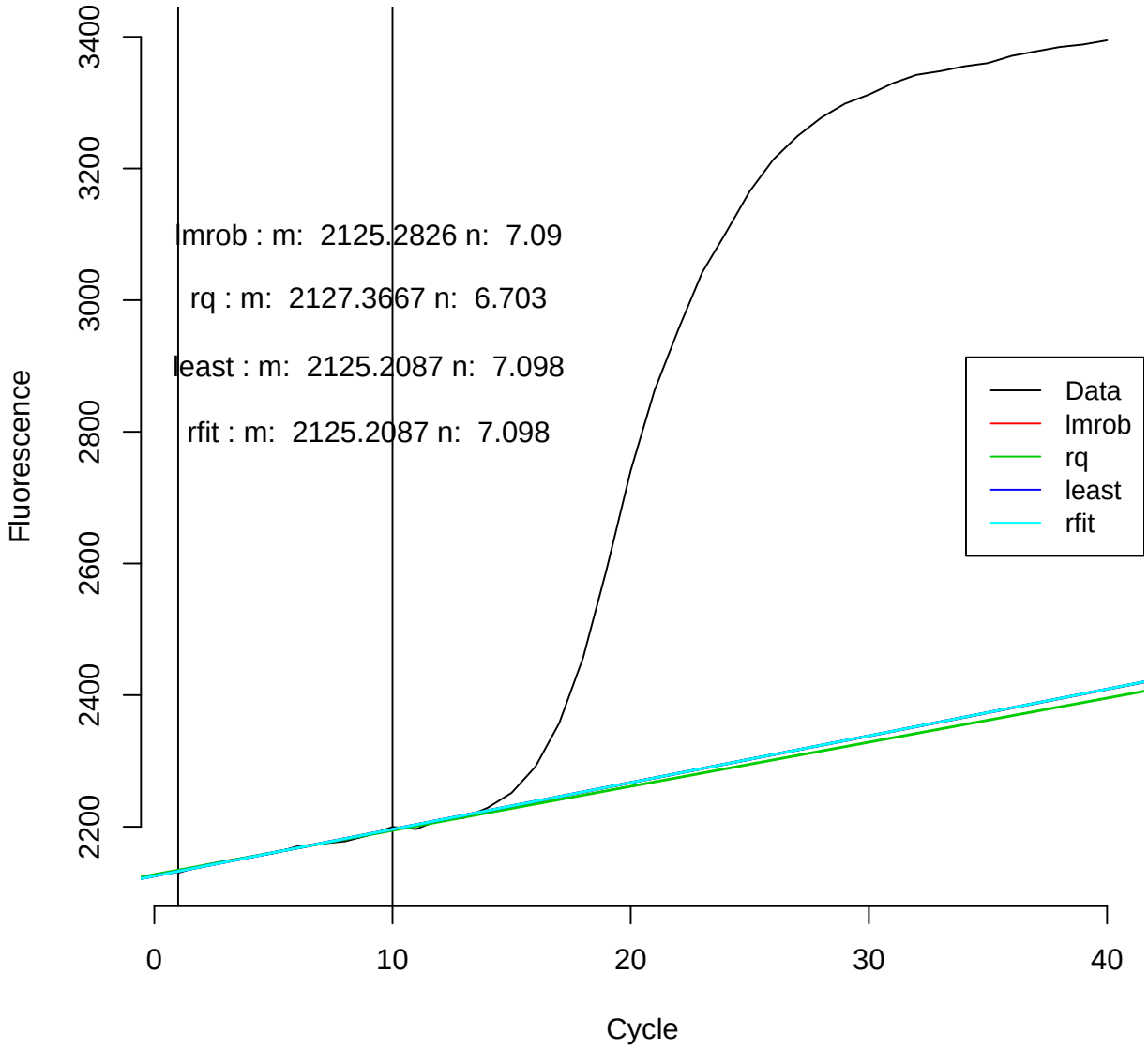

Figure S6: FILL ME.

5 Quantitative description of amplification reactions

5.1 The inder function - an interpolating five-point stencil

The output of *inder* includes the first derivative maximum (FDM) and second derivative maximum (SDM), which are commonly used in qPCR experiments. Figure S15 shows a typical result of the *inder* function. Following we show three examples explain properties of *inder* and to illustrate applications of the function in combination with other functions.

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

Warnings in following code chunks were supressed.

```
# 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,] 1.000 0.05 -4.351e-13  8.698e-13
## [2,] 1.245 0.05 -2.217e-13  8.704e-13
## [3,] 1.491 0.05 -8.206e-15  8.702e-13
## [4,] 1.736 0.05  2.087e-13  8.842e-13
## [5,] 1.981 0.05  4.223e-13  7.731e-13
## [6,] 2.226 0.05  4.929e-13 -2.901e-13
```

```
summary(res)
```

```
## Smoothing method: spline
## First derivative maximum: 20
## Second derivative maximum: 19
## Second derivative minimum: 21
## Second derivative center: 20
```

```
head(rd)
```

```
##      cyc    y        d1y        d2y
## [1,]   1 0.05 -2.217e-13  8.701e-13
## [2,]   2 0.05  3.472e-13 -5.632e-14
## [3,]   3 0.05 -1.083e-12 -4.553e-13
## [4,]   4 0.05  3.945e-12  2.505e-12
## [5,]   5 0.05 -1.449e-11 -1.173e-11
## [6,]   6 0.05  5.390e-11  5.562e-11
```

```
# summary(rd)
```

Figure S15 illustrates the most important parameters of the *inder* function. We used the *AmpSim* function to simulate an ideal "noise-free" amplification curve with the default setting to calculated the second derivative maximum (SDM) with *inder*. If *logy* is $TRUE$ than a semi-decadic log scale graph (corresponds to the linear phase) to illustrate the exponential dynamic of the qPCR amplification is used. The parameter *logy* is $FALSE$ by default. To the best of our knowledge, is *inder* the first tool in **R**, which allows user to numerically derive his

data without fitting them to any function or combination of functions. The universality of stencil approach can find an application even in problems not related to the analysis of amplification curve.

inder is a helper function, which can be part of other routines. Recently, we added this approach to the *diffQ* function of the *MBmca* for improved predictions. *diffQ* function is part of a routine to calculate the melting points of nucleic acids [13].The FDM and SDM are peak values to determine the Cq.

Quantification cycle calculation by the inder function

The presence of noise may cause many false estimates for the FDM and SDM. To minimize this problem, it is possible to smooth the first derivative of the amplification curve. Many methods integrated the moving average as first pre-processing step (e.g., [19]). The moving average filter is linear filter, which replaces sequentially data points with the average of the neighbor data points. The average is calculated from a defined span ("window") of odd count (e.g., 3, 5). The "average" herein may refer to the arithmetic mean, the median, the geometric or the exponential mean. The *smoother* function uses exclusively the arithmetic mean. Moving average is intuitive and easy to implement but it lags behind a trend and ignores rapid changes. For example, the 3- and 5-window moving average (running mean) filters are useful to pre-process data, but always leads to a forerun of few cycles. This is in particular problematic in the exponential phase. Splines apply non-parametric regression by local cubic polynomials between knot points [9]. Other examples for smoothers include Savitzky-Golay smoothing filter, Friedman's SuperSmoother, and the Weighted Whittaker smoother (see the *smoother* function for details).

Provided that the smoother is properly adjusted, it is possible to detect only the significant peaks while small or to narrow peaks are ignored. *smoother* is used by other functions of *chipPCR* like *CPP*. The example for Figure S16 illustrates the use of the *diffQ* and *diffQ2* function from the *MBmca* and the integration of the *inder* function. In contrast to the original publication [13] is the *inder* function in *diffQ* and *diffQ2* used for a precise peak location while the approximate SDM is calculated from the derivative of a quadratic function at the approximate SDM.

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

In the previous example we used smoothing and the *inder* method to calculate the SDM. But, smoothing may alter peak signal considerably. For example, peak height reduction and peak width increase are a common problem. An alternative technique to determine the FDM of SDM is by fitting the raw data. In the next example we used the *drm* function from the *drc* package [12] to fit a five-parameter log-logistic function (S-shaped). The *inder* function was used to calculate the SDM of the predicted models (Figure S17).

5.2 Threshold cycle method

th.cyc function can also be used to calculate the quantification cycle. This function was implemented primarily for the analysis of amplification from qIA but also for qPCR. We implement a symmetrically approximation algorithm based on linear and quadratic least squares regression.

th.cyc is a function to calculate the number of cycles at which the fluorescence exceeds a defined threshold, called the threshold cycle (Ct) (Figure S7). According to the MIQE guidelines the Ct is referred to as quantification cycle (Cq). The calculated Cq is a relative value, which depends on the template copy number, instrument, reagents, amplification efficiency and probe technology. Low Cqs correlate with high quantities template copy numbers. Real-time technologies enable the quantification of nucleic acids by calculation of specific curve parameters like the quantification point (Cq) and the amplification efficiency (AE) based on the kinetics of the amplification curve. The Cq represents the number of cycles (time for qIA) needed to reach a defined fluorescence signal level in the exponential phase of the amplification curve. The Cq can be determined from a fixed threshold value or by various analytical algorithm as described elsewhere [14, 16, 22].

The Threshold Cycle (Ct) (Cq according to MIQE, see [3]) is the cycle number at which the fluorescence exceeds significantly a point above the baseline and defined threshold in a particular samples. Thus the Ct is the cycle when sufficient numbers of amplicons have accumulated. The *th.cyc* calculates the intersection of the user defined Ct value (r) and a linear regression or quadratic polynomial in the range of the user defined Ct value. In contrast to other methods is does *th.cyc* have no requirement to fit a "complex" non linear model to the entire data set but rather focuses on the specific area. The polynomial is calculated from four neighbor values at the fluorescence threshold.

Warnings in following code chunks were supressed.

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

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

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

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

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

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

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

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

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

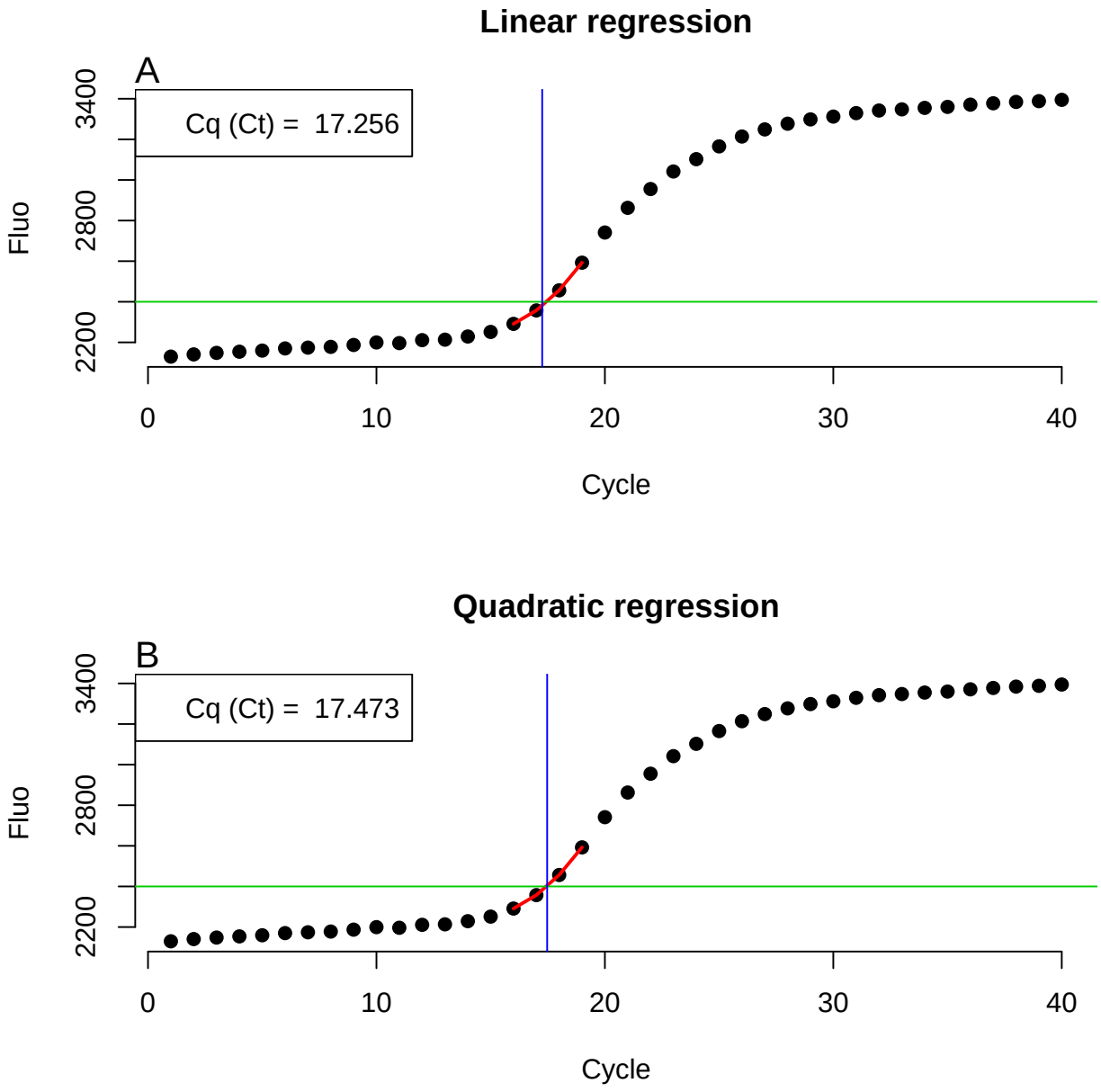

Figure S7: Working principle of *th.cyc*. The function provides two modes (**A)** is the linear regression. **B)** Quadratic regression) for the calculation of the Cq. In both cases is the highest R squared value determining how many left and right neighbors above and the below the used defined threshold level are use.

5.2.1 Application of the th.cyc and CPP function on ccPCR data

```
# 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 - Raw Data")

# Threshold level 'r' (50 micro Volts)
for (i in c(1, 3, 5, 7)) {
    y.tmp <- capillaryPCR[, i + 1] - mean(capillaryPCR[1L:150,
        i + 1])
    Ct.tmp <- th.cyc(capillaryPCR[, i], y.tmp, r = 50, linear = FALSE)
    abline(v = Ct.tmp[1])
    text(Ct.tmp[1] * 1.1, 1200, paste(round(Ct.tmp[1], 1), "\nmin"))
    lines(capillaryPCR[, i], y.tmp, type = "b", pch = 20 - i)
    points(Ct.tmp@input, col = "red", pch = 19)
}
abline(h = 50)
legend("topleft", c("Run 1", "Run 2", "Run 3", "Control"), pch = c(19,
    17, 15, 13), lwd = 1.3, bty = "n")
```

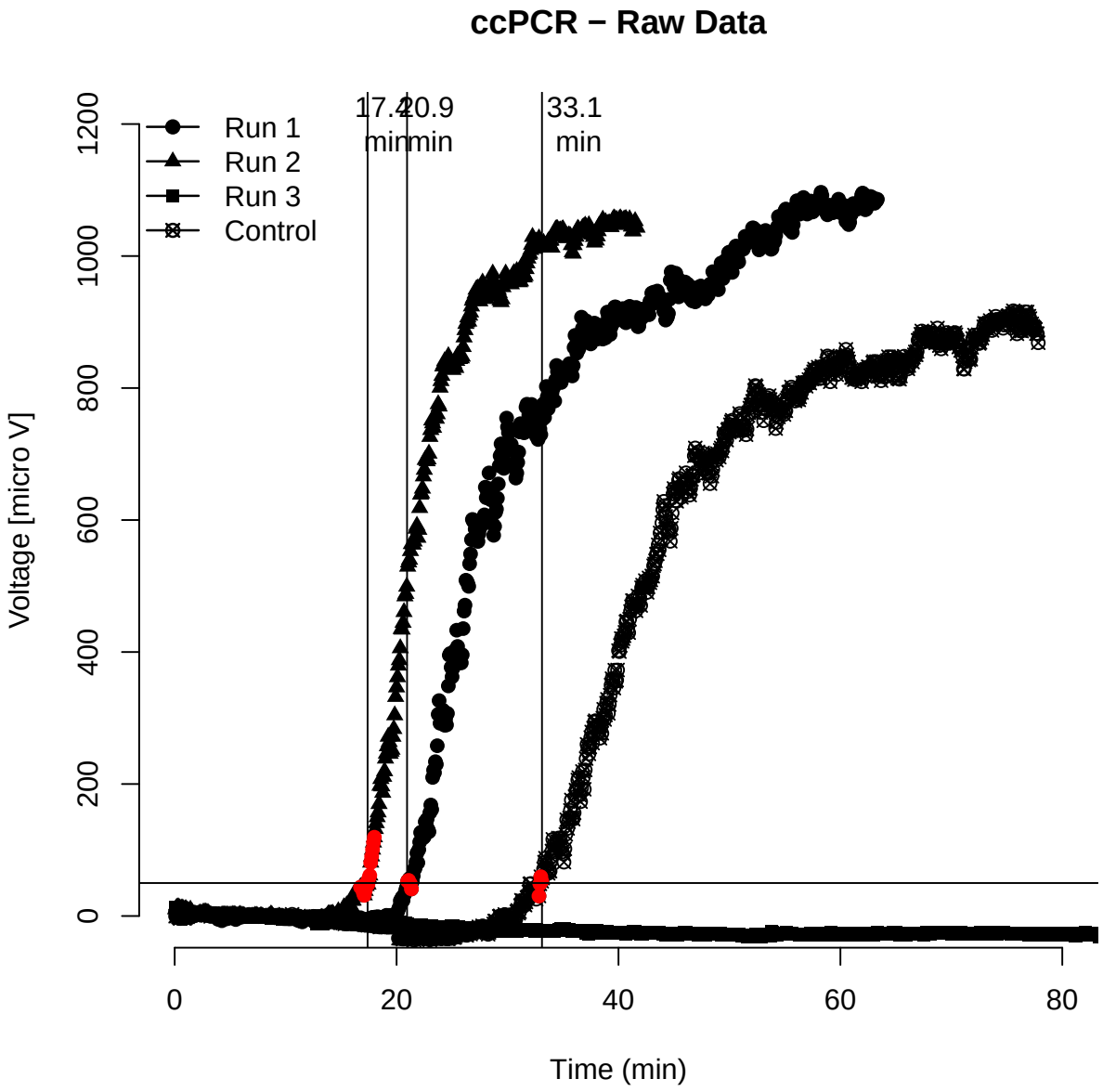

Figure S8: Application of *th.cyc* for the analysis of ccPCR data. Data from a ccPCR were analyzed using the *th.cyc* function using the linear regression mode. The threshold level ($r = 50$) was identical for all data. The Cq (Ct) are given in minutes. The range used for the calculation of the Cq is indicated in red. Negative curves are automatically excluded from the analysis if the 90% percentile is lower or equal to the threshold level (r).

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

A Helicase Dependent Amplification (HDA) of Vimentin (Vim) was performed. The VideoScan Platform [17] was used to monitor the amplification. The HDA was performed at 65 degree Celsius. Three concentrations of input DNA (D1, D2, D3) were used (Figure S9).

To perform an isothermal amplification in VideoScan, standard conditions for the IsoAmp(R) III Universal tHDA Kit (Biohelix) were used. Primers and templates are described in [17]. The reaction was composed of reaction mix A) 10 μL A. bidest, 1.25 μL 10xbuffer, 0.75 μL primer(150 nM final), 0.5 μL template plasmid. Preincubation: This mixture was incubated for 2 min at 95 degree. Celsius and immediately placed on ice. Reaction mix B) 5 μL A. bidest., 1.25 μL 10x buffer, 2 μL NaCl, 1.25 μL MgSO4, 1.75 μL dNTPs, 0.25 μL EvaGreen, 1 μL enzyme mix. The mix was covered with 50 μL mineral oil. The fluorescence measurement in VideoScan HCU started directly after adding buffer B at 65 degree Celsius. A 1x (D1), a 1:10 dilution (D2) and a 1:100 (D3) dilution were tested. Temperature profile (after Preincubation): - 60 seconds at 65 degree Celsius - 11 seconds at 55 degree Celsius & Measurement

Warnings in following code chunks were supressed.

```
par(mfrow = c(2, 1), bty = "n")
plot(NA, NA, xlim = c(0, 5000), ylim = c(0, 1), 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("topleft", c("D1, 1x", "D2, 1:10", "D3, 1:100"), col = c(2,
    4, 6), pch = rep(20, 3))

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))

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

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

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

# method for baseline normalization
lrg <- "least"

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

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

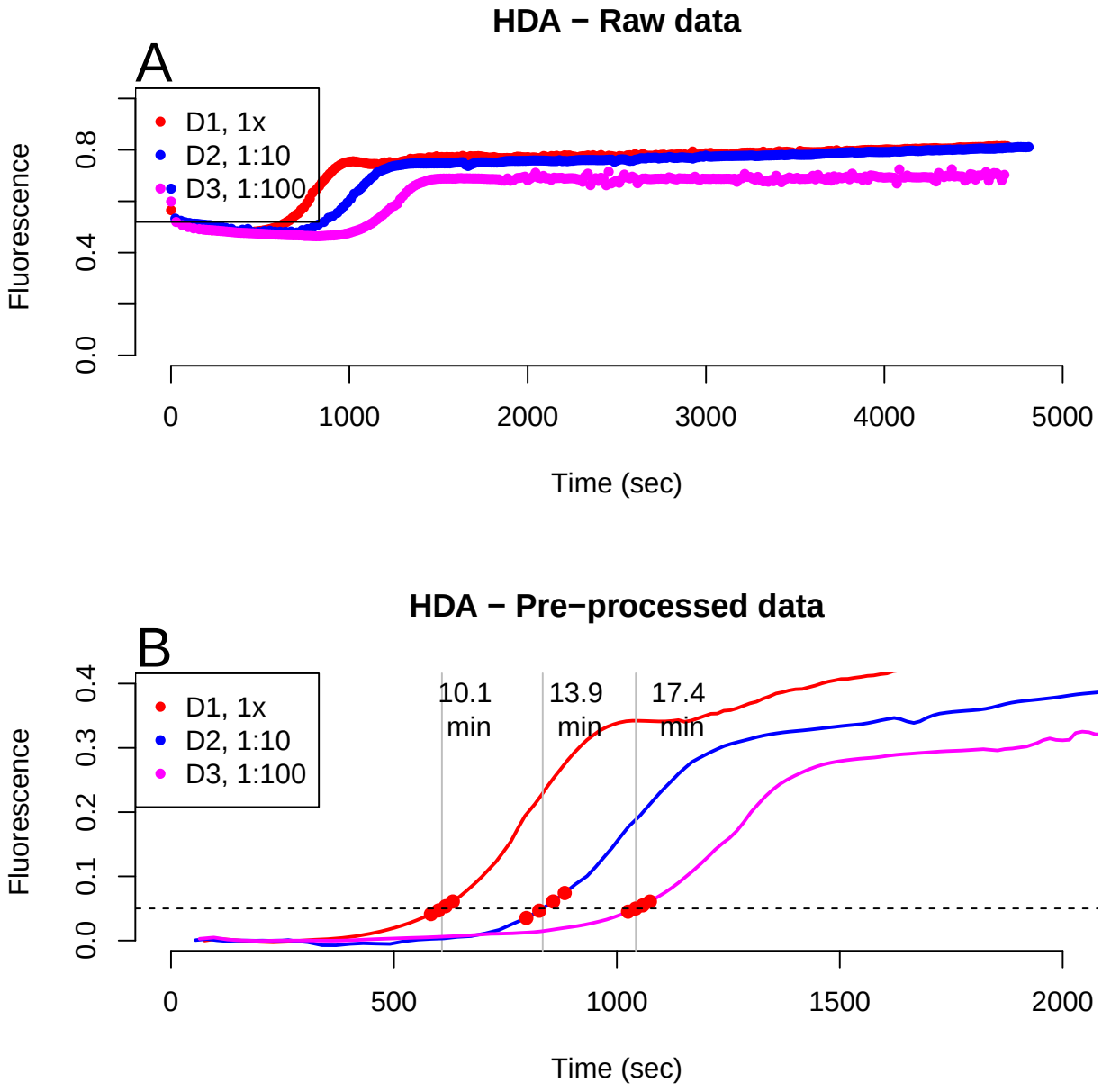

Figure S9: Helicase Dependent Amplification (HDA) of Vimentin (Vim). The VideoScan Platform was used to monitor the amplification. The HDA was performed at 65 degree Celsius. Three concentrations of input DNA (D1, D2, D3) were used. The amplification curves were smoothed by a moving average (windowsize 3) and base-lined by a a robust linear regression by computing MM-type regression estimator. The *th.cyc* function was used to determine the time required to reach the threshold level of 0.05 (–).

5.3 Amplification efficiency

Various influences alter amplification reactions. The amplification efficiency is controlled by a complex interaction of the intrinsic and extrinsic factors like reaction conditions, substrate consumption, primer dimmer formation and molecule specific reaction rates [8]. Some probe systems are percevied as bias-introducing. Therefore, qPCR reaction should be corrected based on the amplification efficiency [24, 15]. The amplification efficiency (AE) can be estimated from individual samples or a set of samples to compensate the presence of inhibitors and noise. Indirect methods use fitted mathematical models or estimate the AE from absolute fluorescence values [23, 6, 1, 20, 2, 7]. The *qpcR* has many functions included, which can be used for the indirect estimation of the AE. However, most commonly used is the "direct method" [6, 21]. Herein, the AE is estimated from dilution series of a template. The AE of a qPCR reaction is calculated from the slope of the standard curve (Equation S1).

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

The function *effcalc* is used for the automatic calculation of the AE of a dilution series (Figure S10). An object of the class list contains the "Concentration", Cqs, deviation of the Cqs, "Coefficient of Variance" sequentially in the columns, the amplification efficiency (%) according to Equation S1, the results of the linear regression and the correlation test (Pearson) (Table SS3). The *effcalc* has several options to enhance the plot. For example, it is possible to indicate the confidence interval (default CI = 95 %). Further options are described in the manual.

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

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

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

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

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

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

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

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

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

Cq.out <- vector()

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

for (i in 1L:18) {
    ma.out[1:45, i] <- AmpSim(cyc = c(1:45), b.eff = -50, bl = 0.001,
        ampl = 1, Cq = Cq.t[i], noise = TRUE, nnl = 0.02)[, 2]
    lines(1:45, ma.out[, i])
    tmpP <- mcaSmoother(1:45, ma.out[, i])
```

```
    # Calculate the pseudo Second Derivative Maximum (SDM) (Cq)
    # using the diffQ2 function from the MBmca package.
    Cq.tmp <- diffQ2(tmpP, inder = TRUE)$xTm1.2.D2[1]
    abline(v = Cq.tmp)
    Cq.out <- c(Cq.out, Cq.tmp)
}
```

```
##   Relative Deviation (%) Approximate Tm Calculated Tm
## 1                   6.11             13         11.92
##   Relative Deviation (%) Approximate Tm Calculated Tm
## 1                 8.478          15.62         13.85
##   Relative Deviation (%) Approximate Tm Calculated Tm
## 1                 5.237             23         24.77
##   Relative Deviation (%) Approximate Tm Calculated Tm
## 1                 10.12          19.46         16.86
##   Relative Deviation (%) Approximate Tm Calculated Tm
## 1                 5.388             31         33.46
##   Relative Deviation (%) Approximate Tm Calculated Tm
## 1                 5.844          22.54         20.75
##   Relative Deviation (%) Approximate Tm Calculated Tm
## 1                 9.405             15         13.13
##   Relative Deviation (%) Approximate Tm Calculated Tm
## 1                 5.696             27         24.91
##   Relative Deviation (%) Approximate Tm Calculated Tm
## 1                 8.492          15.62         13.85
##   Relative Deviation (%) Approximate Tm Calculated Tm
## 1                 5.965          18.69         17.18
##   Relative Deviation (%) Approximate Tm Calculated Tm
## 1                 5.778          22.54         20.77
##   Relative Deviation (%) Approximate Tm Calculated Tm
## 1                 10.59             21         18.07
##   Relative Deviation (%) Approximate Tm Calculated Tm
## 1                 5.691          33.31         30.73
##   Relative Deviation (%) Approximate Tm Calculated Tm
## 1                 6.512             13         11.86
##   Relative Deviation (%) Approximate Tm Calculated Tm
## 1                 8.611          15.62         13.82
##   Relative Deviation (%) Approximate Tm Calculated Tm
## 1                 5.079             22         23.64
##   Relative Deviation (%) Approximate Tm Calculated Tm
## 1                 10.01          19.46         16.89
##   Relative Deviation (%) Approximate Tm Calculated Tm
## 1                 9.649             15         13.08
##   Relative Deviation (%) Approximate Tm Calculated Tm
## 1                 5.828          22.54         20.75
##   Relative Deviation (%) Approximate Tm Calculated Tm
## 1                 7.253          26.38         23.81
##   Relative Deviation (%) Approximate Tm Calculated Tm
## 1                 5.182             36         38.74
```

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

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

Next we used *effcalc* to analyze the C54 data set from the *chipPCR* package. Herein, a qPCR Experiment for the amplification of MLC-2v using the VideoScan heating/cooling-unit was performed. To calculate the Cq

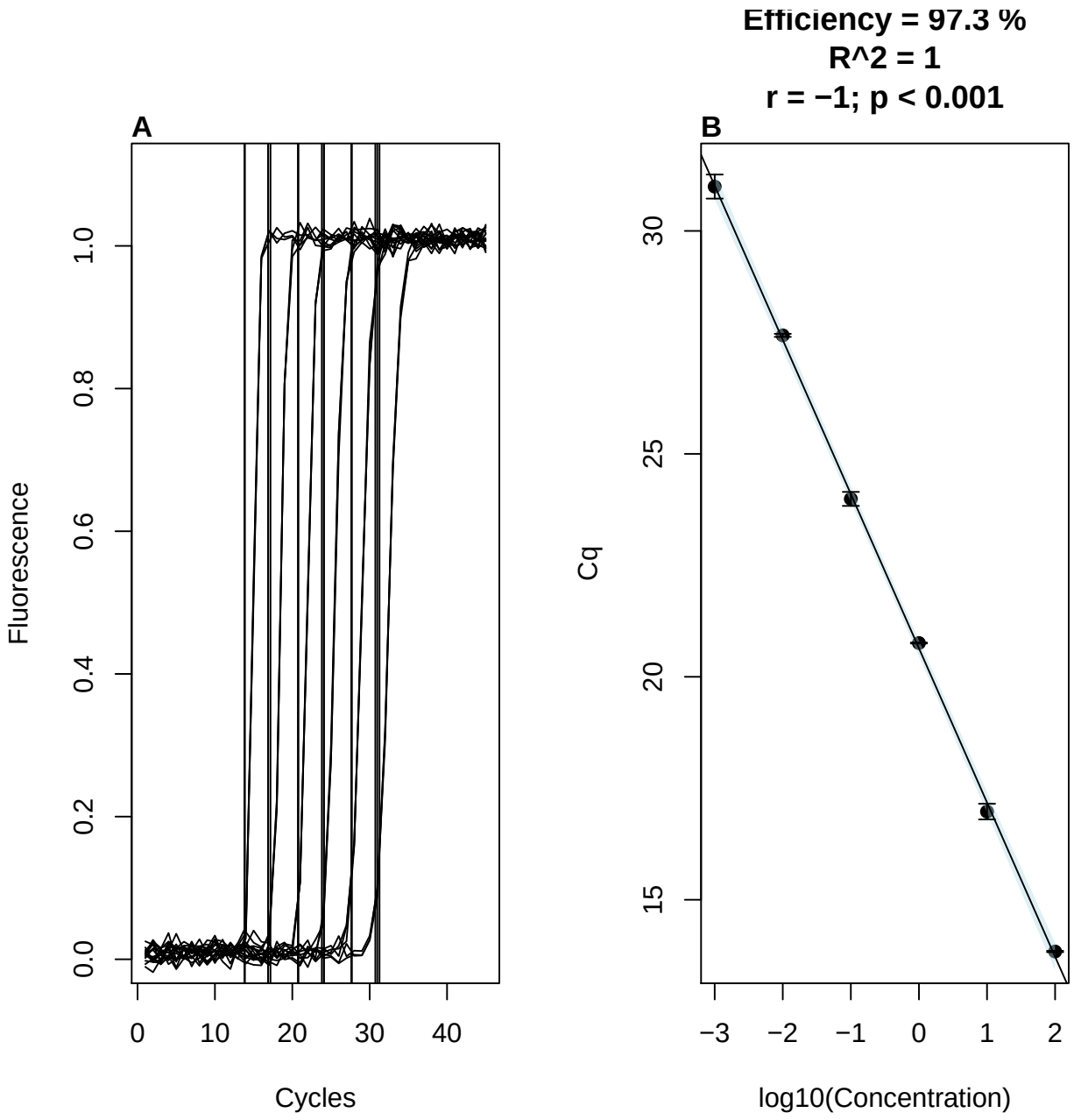

Figure S10: Amplification standard curve simulation and regression analysis. *(A)* AmpSim was used to synthesize a qPCR experiment of six dilutions (three replicates per dilution) standard samples. The Cqs were determined by the SDM method (solid black vertical lines). *(B)* effcalc was used to automatically perform a linear regression. The regression curve (–) was plotted as the decadic logarithm of input concentration versus the Cq. The 95% confidence interval is shown be the light-blue solid lines.

it was necessary to pre-process the amplifcation curve data. One amplifcation curve contained a missing value (Figure S11 A) wich was remove by the spline method in *CPP*. In addition, the data were baselined (linear model, robus MM-estimator) and smoothed by Savitzky-Golay Smoothing (Figure S11 B). The final analysis with the *effcalc* function showed that the amplification efficiency is circa 87.3 % for the gene MLC-2v in the VideoScan HCU (Figure S11 C). However, since only few measure points were tested in this experiment it just save to say that the hardware of the HCU works reliably.

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

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

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

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

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

## Approximate and calculated Tm varri.  This is an expected behaviour
## but the calculation should be confirmed with a plot (see examples of diffQ).

##   Relative Deviation (%) Approximate Tm Calculated Tm
## 1                  15.64              3         3.746

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

##   Relative Deviation (%) Approximate Tm Calculated Tm
## 1                  12.14          12.31         10.36

## The distribution of the curve data indicates noise.
## The data should be visually inspected with a plot (see examples of diffQ).
## The distribution of the curve data indicates noise.
## The data should be visually inspected with a plot (see examples of diffQ).

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

## Approximate and calculated Tm varri.  This is an expected behaviour
## but the calculation should be confirmed with a plot (see examples of diffQ).

##   Relative Deviation (%) Approximate Tm Calculated Tm
## 1                  16.42             16         12.67

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

```
##   Relative Deviation (%) Approximate Tm Calculated Tm
## 1                  6.534             20         21.94

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

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

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

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

Table S3: 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 S12 A) and MLC-2v (Figure S12 B) were amplified in an Roche Light Cycler 1.5 and detected by the HRM dye EvaGreen in independent experiments. As shown in the code and Figure S12 it is possible to obtain a complete analysis with few commands. The amplification efficiencies for both qPCRs was higher than 94 % (Figure S12 C and D, Table S3).

```
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")
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 dillutions
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)
```

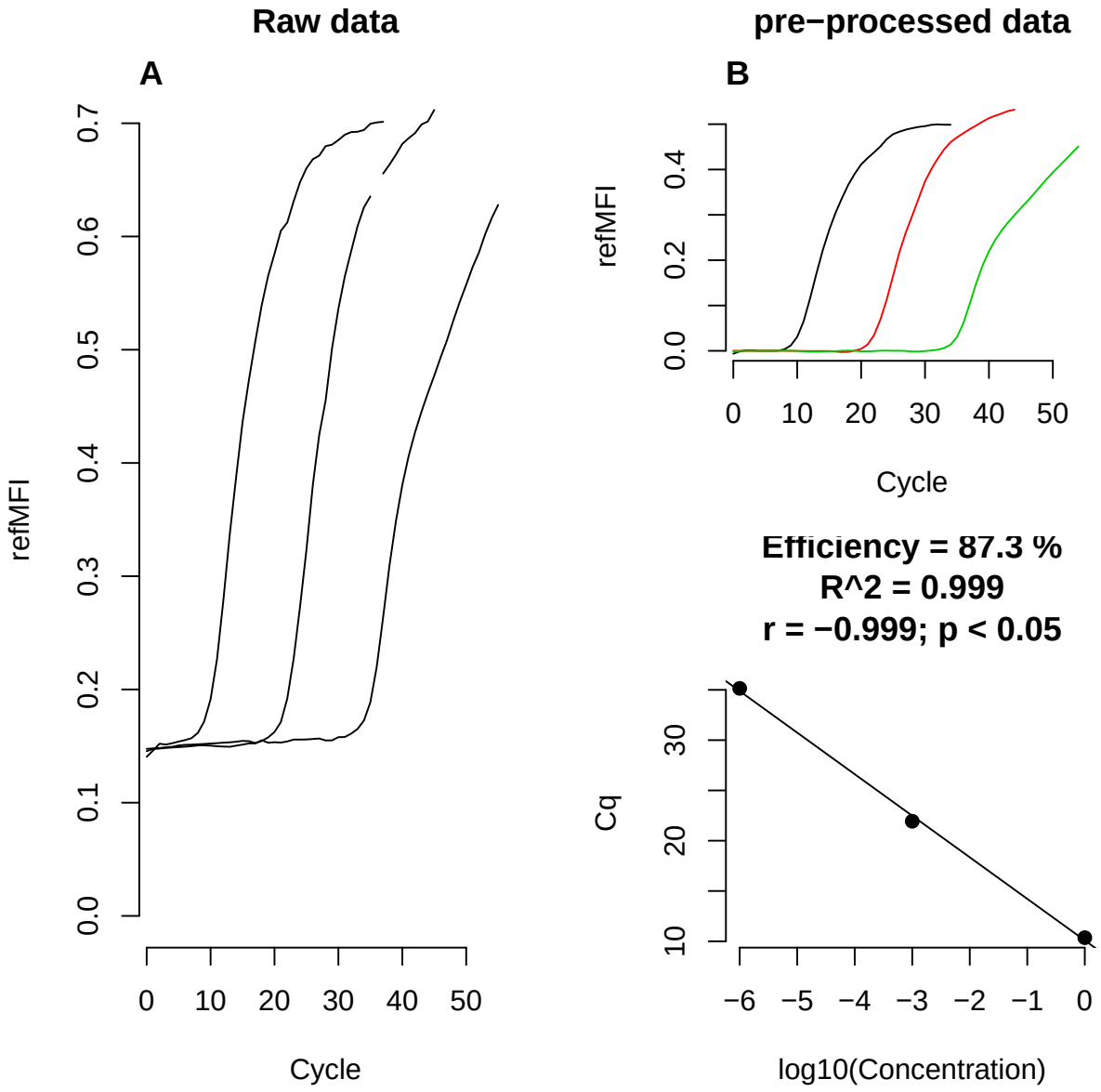

Figure S11: Calculation of the amplification efficiency. Data of a VideoScan HCU dilution experiment (C54 data set) were analyzed. *(A)* Visualization of the raw data. One of the three dilutions contains a missing value due to a sensor error. *(B, top panel)* The CPP function was used to baseline, to remove the missing value (–) and to smooth (–, –, –) the raw data. *(B, bottom panel).* The Cqs (SDM) of the pre-processed data were calculated by diffQ2 (see main text) and analyzed with *effcalc*. The amplification efficiency approximately at 87.3 %.

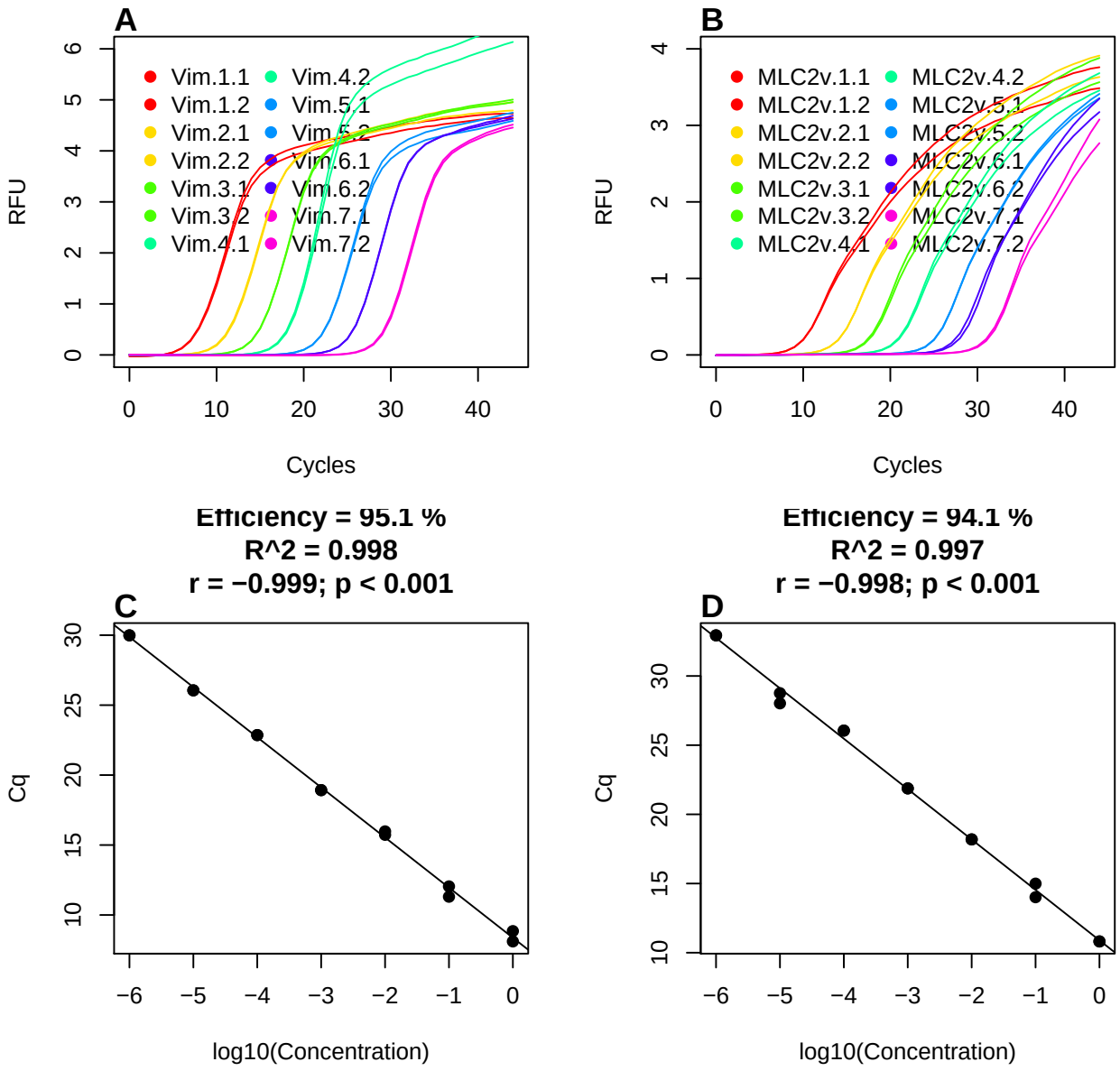

Figure S12: FILL ME.

6 AmpSim - a function to simulate amplification curves

The function *AmpSim* is a simulator for amplification reactions. Use cases include teaching, algorithm testing or the comparison of an experimental system to the predicted ("optimal") model. *AmpSim* uses a 5-parameter model (Equation S2).

$$fluo = bl + \frac{ampl - bl}{1 + \exp\left(b.eff * (\log cyc - \log Cq)\right)} \tag{2}$$

It is an intrinsic property of *AmpSim* to generate unique results if the *noise* parameter is set $TRUE$. This is due the use of the *rnorm* (*stats*) function to simulate noise. If data need to be replicated identically use *set.seed(123)* to alter the random number generator (RNG) state. For example, the amplification curves of Figure S10 *A* are generated with the same starting parameter of *AmpSim* but noise was added. *AmpSim.gui* is a *shiny* GUI (graphical user interface) implementation for *AmpSim*. The code example below is an exmple how-to invoke the *AmpSim.gui*. Further details on *shiny* are described in in the main document. *AmpSim* was also used to illustrate the *inder* function (Figure S15), the *fixNA* function (Figure S4) and the use of the *smoother* (Figure **??**) function.

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

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

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

AmpSim has several parameters, which can be used to simulate an amplification curve. $b.eff$ and Cq are most connected with another. Thus changing one of them will change both values. Cq can be used to define an approximate Cq value. The expression "approximate Cq value" is used because the calculated Cq value will vary depending on the preferred Cq quantification method (e.g., Second Derivative Maximum (SDM) method, threshold method). *AmpSim* can be used to simulate data with noise (based on *rnorm*, *stats*), signal-to-noise ratios, photo-bleaching and other influences on a qPCR reaction. The following example illustrates the use of *AmpSim* (Figure S13).

Warnings in following code chunks were supressed.

```
# 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])
})
```

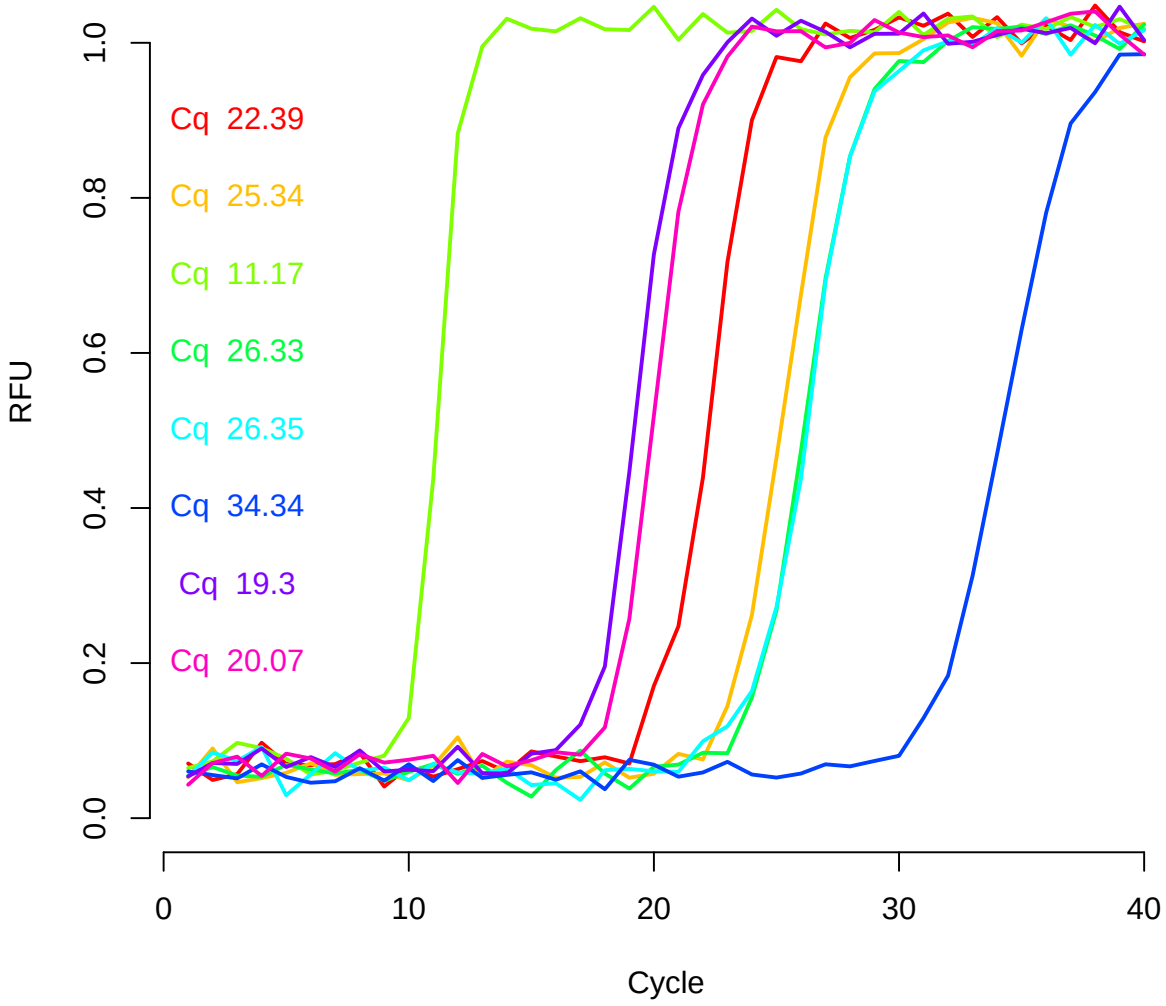

Figure S13: The amplification curves were generated with the *AmpSim* function. All Cqs are unique due to the use of random value, which were added to the starting Cq of 25. The parameter $noise = 0.03$ adds some scatter to the amplification curve data.

7 Proposed workflow

Warnings in following code chunks were supressed.

```
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 = "minmax")[["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)
```

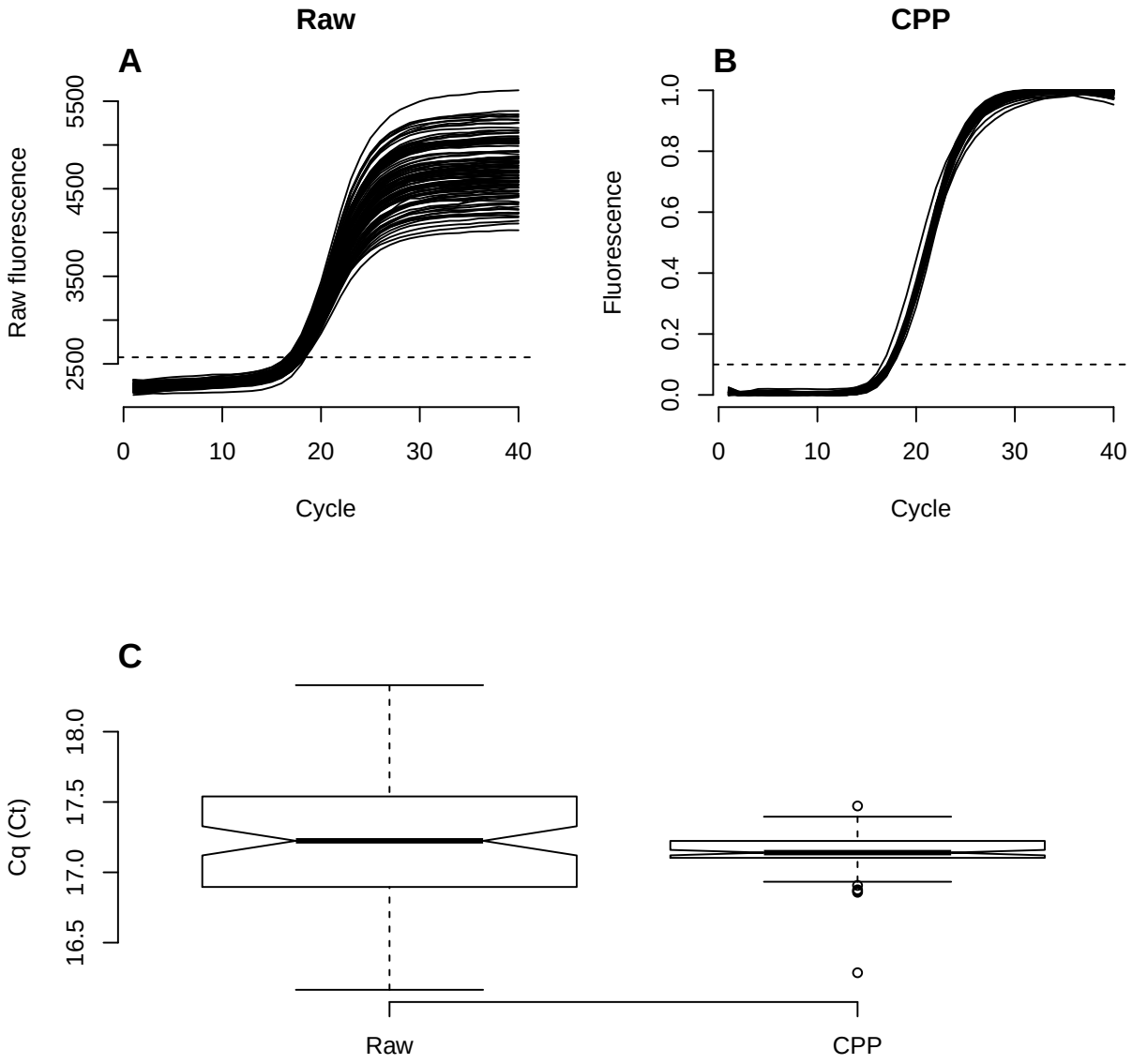

Figure S14: Application of the *CPP* and *th.cyc* functions. **A)** The raw data of the VIMCFX96_60 data set were plotted without pre-processing. **B)** All amplification curve data were pre-processed with the CPP function. The parameter *trans* was set to $TRUE$, which lead to a linear trend correction and base-lining. By default a Savitsky-Golay filter was used to smooth the data. The data were normalized between 0 and 1 ($method.norm =' minmax'$). **C)** All Cqs were calculated with *th.cyc* function. The Cq for the raw data was 17.25 ± 0.5 (at $r = 2575$) and 17.1 ± 0.1 (at $r = 0.1$) for the pre-processed data. Our results indicate that the dispersion of the Cq values was slightly lower.

8 Auxillary functions

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

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

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

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

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

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

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

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

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

# Summary of the object res.
summ

##   FDM   SDM   SDm   SDC
## 19.81 19.03 20.99 19.99
```

```
# 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

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)
```

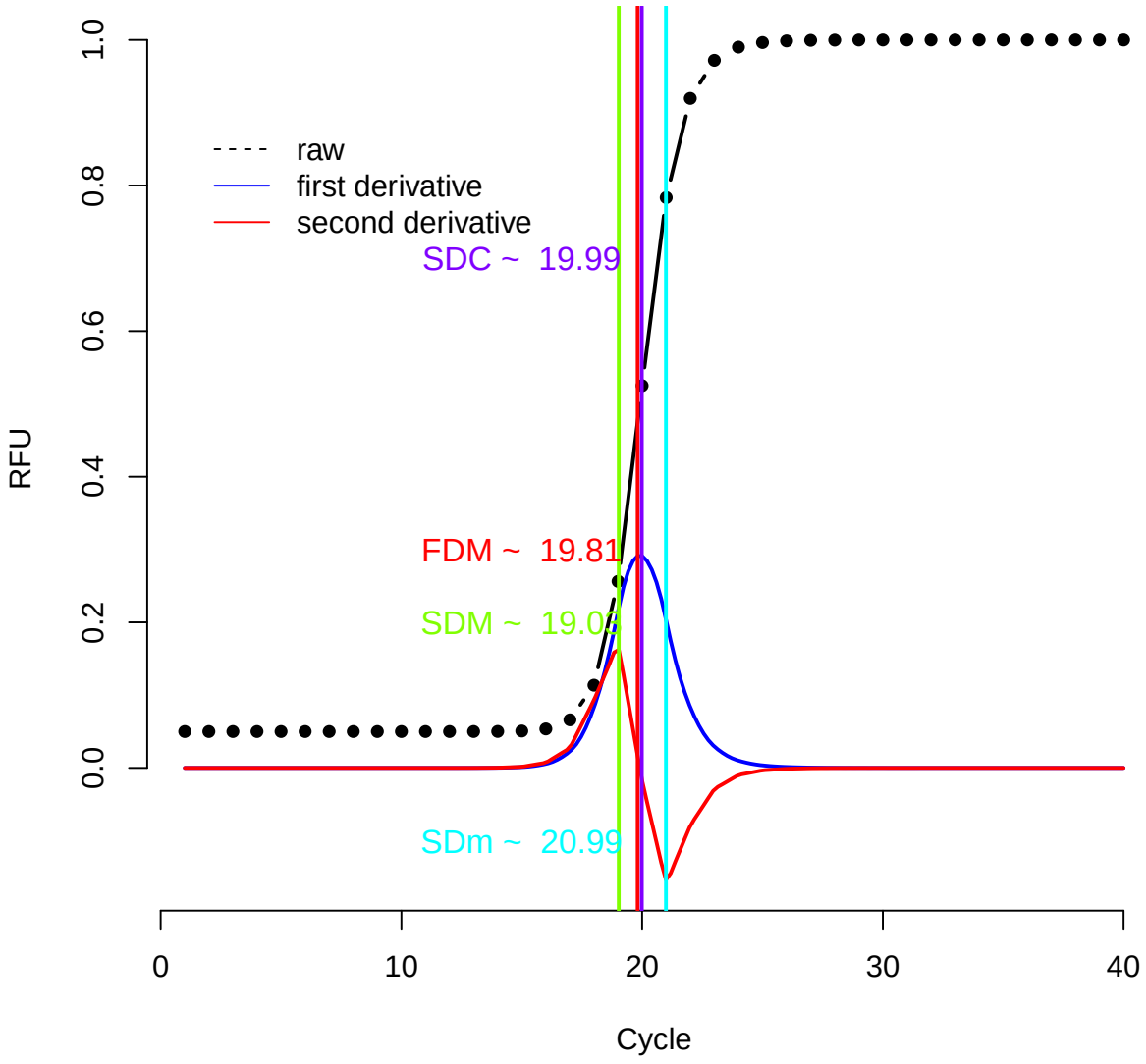

Figure S15: Cycle of quantification by the second derivative maximum method. Raw data (•) were generated using the AmpSim simulation function (see example main text). The inflection point is the point where the slope is maximum and the curvature is zero. The first derivative of the amplification curve has a first derivative maximum (FDM) at the inflection point. The second derivative maximum method (SDM) needs to differentiate a curve to the second order prior to quantification. The second derivative exhibits a zero-crossing at the FDM. The function $y = f(x)$ is numerically derived by five-point stencil. This method do not require any assumptions regarding the function f. The function inder calculates the approximate SDM. The SDM might in addition be useful for isothermal amplification processes. The SDM is calculated from a derived cubic spline. Similarly the first approximate derivative maximum (FDM), second derivative minimum (SDm), and approximate second derivative center (SDC, geometric mean of SDM and SDm) are available. FDM, SDm and SDC values can be used to further characterize the amplification process.

```
    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)
    arrows(xx[1] * 1.01, x.median + x.mad, tail(xx, 1) * 0.99,
        x.median + x.mad, param, lwd = 2, lty = 2)
    arrows(xx[1] * 1.01, x.median - x.mad, tail(xx, 1) * 0.99,
        x.median - x.mad, param, lwd = 2, lty = 2)
}

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)
```

```
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]
```

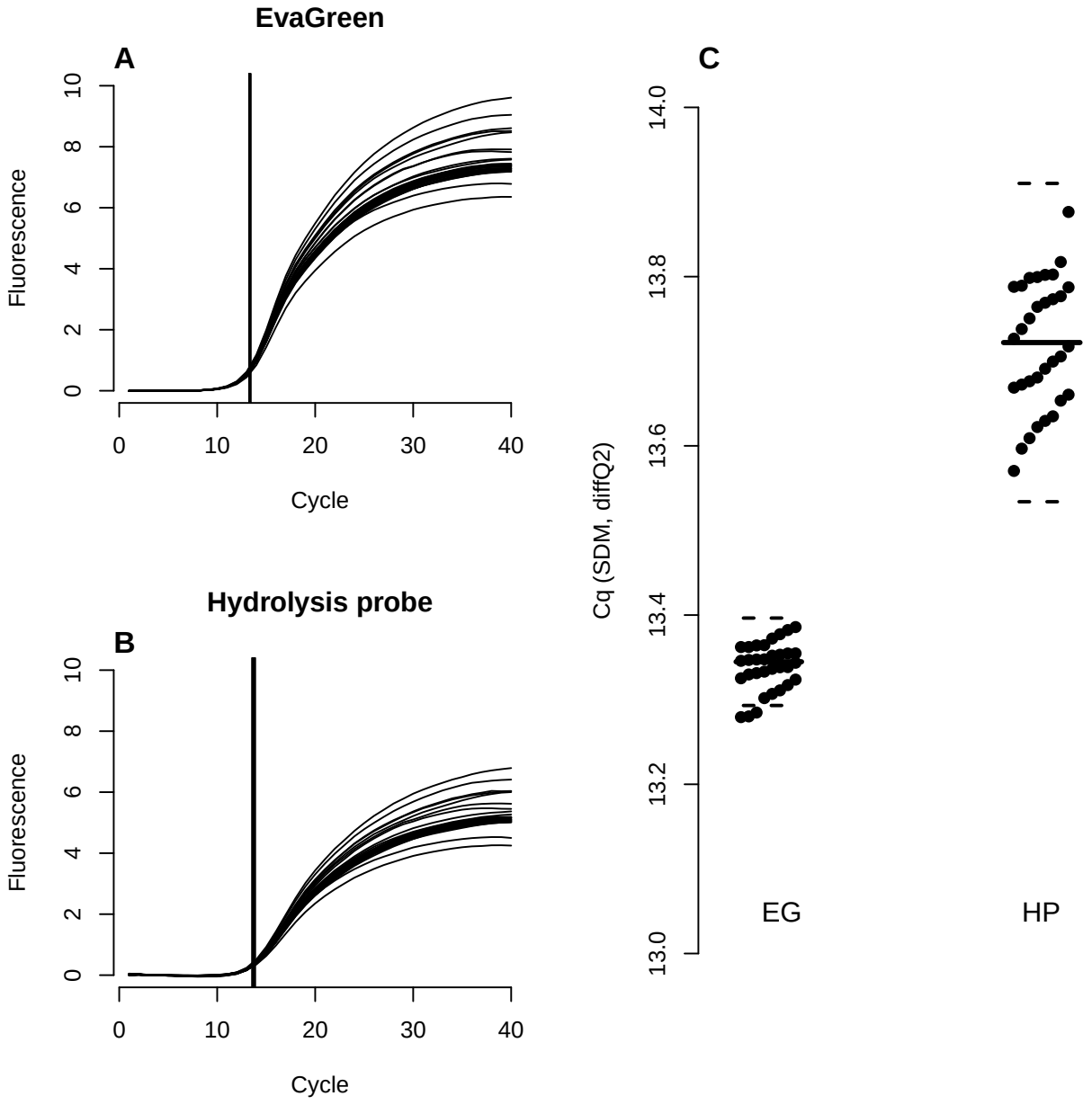

Figure S16: Plot all data from C127EGHP and calculate the SDM (Second Derivative Maximum) values with the *diffQ2* function. *(A)* Plot the samples detected with EvaGreen and *(B)* shows the same samples detected with the Hydrolysis probe for MLC-2v. *(C)* Stripchart of the Cq values (•) with the median (–) and the median absolute deviation (– –). This result indicates, that the variance of the derived from the detection with hydrolysis probes is higher than the samples detected with EvaGreen. Note: the *inder* parameter is set as TRUE.

```
}

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))

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)
```

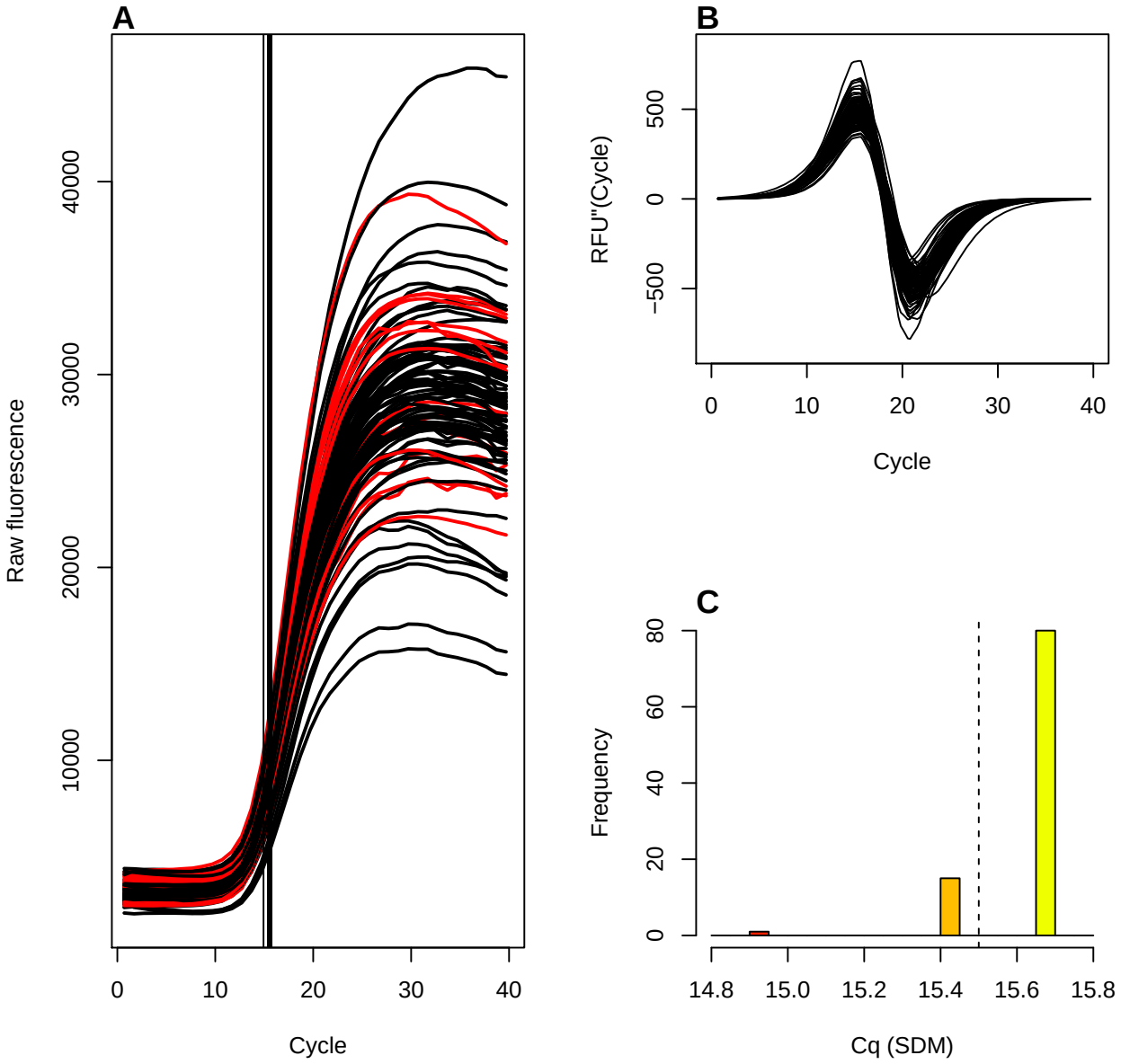

Figure S17: Amplification curve profiles from the Bio-Rad iQ5 thermo cycler for the human gene *HPRT1*. *(A)* The *C126EG595* data set was used with 96 replicates of equal starting numbers of template molecules. Vertical lines represent the Cq (*SDM* method) determined with inder method on amplification curves fitted with a 5-parameter curve function. Curves with Cqs less than 14.5 are indicated in red (–). *(B)* Second derivatives of the amplification curves. Note that after differentiation all inter sample baseline and plateau shifts are similar. *(C)* Histogram (class width = 0.05 Cq) of the Cq values (*SDM*). Cqs were mainly at circa 15.7 (N = 80) while some amplification curves had a Cq less than 15.5 (N = 16).

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

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

The simplest approach is to treat the background fluorescence as a value constant during whole amplification reaction. In this case the noise could be approximated as the mean or median of fluorescence values in lag phase [5] or their standard deviations [10]. The more sophisticated way of approximating constant background fluorescence requires optimizing its value to achieve linearity of the model fit on the semi logarithmic plot in log-linear phase [5]. The later procedure is greatly enhanced by performing further computations only on a subset of consecutive measurements for which calculated efficiencies have the lowest variance. Other methods loosen the assumption that background fluorescence is a constant value and instead describe it as a function of the cycle number. For example the algorithm used in SoFar [25] fits a nonlinear saturation function to measurement points before the start of the exponential growth phase. Parameters of the saturation function are chosen to minimize the sum of squared residuals of the fitted function. Then the value of saturation function is calculated for all data points and subtracted from measured values giving corrected values of fluorescence, which are used in next calculations.

Some approaches make even less assumptions regarding the form of the background noise. The taking-difference linear regression method has a premise that changes of fluorescence between subsequent cycles are exclusively caused by the amplification of the product [11]. The corrected values are calculated by simply subtracting the fluorescence value in the former cycle from fluorescence in the latter. Of course in this case the real fluorescence value in first cycle is unknown, so the number of cycles that can be used in following computations is reduced by one.

The Real-Time PCR Miner algorithm is also nearly assumption-free ([26]). The main principle is that background fluorescence is similar in the small groups of subsequent measurements. So the first step of the algorithm is division of subsequent measurement points belonging to the exponential phase of amplification in at least four-element groups. For each set of points is calculated a pair of the estimate of the efficiency and the significance of model representing relation between the fluorescence value and the cycle number. The estimates paired with the highest significance are the most influential in the computation of the final efficiency.

To find the beginning of the lag phase and end of plateau phase is important for the goodness-of-fit for both exponential-phase-only and S-shaped models. There are two strategies. The first narrows the area of the search to the neighborhood of their theoretical values determined by a fitted model of the amplification reaction. To this group belongs SoFar (Wilhelm et al. (2003) [25]). The algorithm looks for the start and the end of the exponential phase near the second derivatives of the function representing the relation between logarithm of the fluorescence and the cycle number. The available correction guarantees that the start of amplification has higher value than background noise. The very similar procedure is implemented in Real-Time PCR Miner [26], where background noise is also used as parameter in implemented models to calculate theoretical the start of the amplification process. The end of amplification process is detected by calculating the third derivative of implemented S-shaped model. The second approach does not require theoretical values. A very intuitive solution, designated take-off point, by Tichopad et al. (2003) [23] describes the lag phase using a linear function. Random deviations are taken into account as standardized residuals. The method starts with a fitting of a linear function to first three measurement points. If none of residuals is considered an outlier with a statistical test, the algorithm fits a new linear model to the first four measurement points and so on. The procedure stops when two last points are designated as outliers. The first of aforementioned outliers is considered the end of lag phase. It is worth noting that this algorithm is versatile enough to also detect the beginning of the plateau phase.

The algorithm of *bg.max* is based on the assumption that the signal difference of successive cycles in the linear ground phase is approximately constant. After transition in the early exponential phase the signal changes drastically. First data are smoothed by Friedman's 'super smoother' (as found in supsmu. Thereof the approximate first and second derivative are calculated by a five-point stencil *inder*. The difference of cycles at the maxima of the first and second approximate derivative and a correction factor are used to estimate the range before the exponential phase. This simple function finds the background range without modeling the function. The start of the background range is defined be a "fixed" value. Since many signals tend to overshot in the first cycles a default value of 2 (for qPCR) is chosen. *bg.max* tries also to estimate the end of an amplification reaction (Figure S18). Application of this function is for example a rational basis for trimming of unneeded data.

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

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

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

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

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

We used to the *bg.max* algorithm to analyze amplification curve data from an capillary convective PCR (capillaryPCR *chipPCR* data set). The data were used as raw data (Figure S19 A) and pre-processed data (Figure S19 B) using the *CPP* function. For both cases it was possible to recieve results, which can be used for further processing. We observed no significant difference between the raw and pre-processed data.

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

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

colors <- rainbow(8)

plot(NA, NA, xlim = c(0, 75), ylim = c(-200, 1300), xlab = "Time (min)",
    ylab = "Voltage (micro V)", main = "ccPCR - Raw data")
mtext("A", cex = 1.5, side = 3, adj = 0)
for (i in c(1, 3, 5, 7)) {
    x <- capillaryPCR[1L:750, i]
    y <- capillaryPCR[1:750, i + 1]
    res.bg <- summary(bg.max(x, y))
```

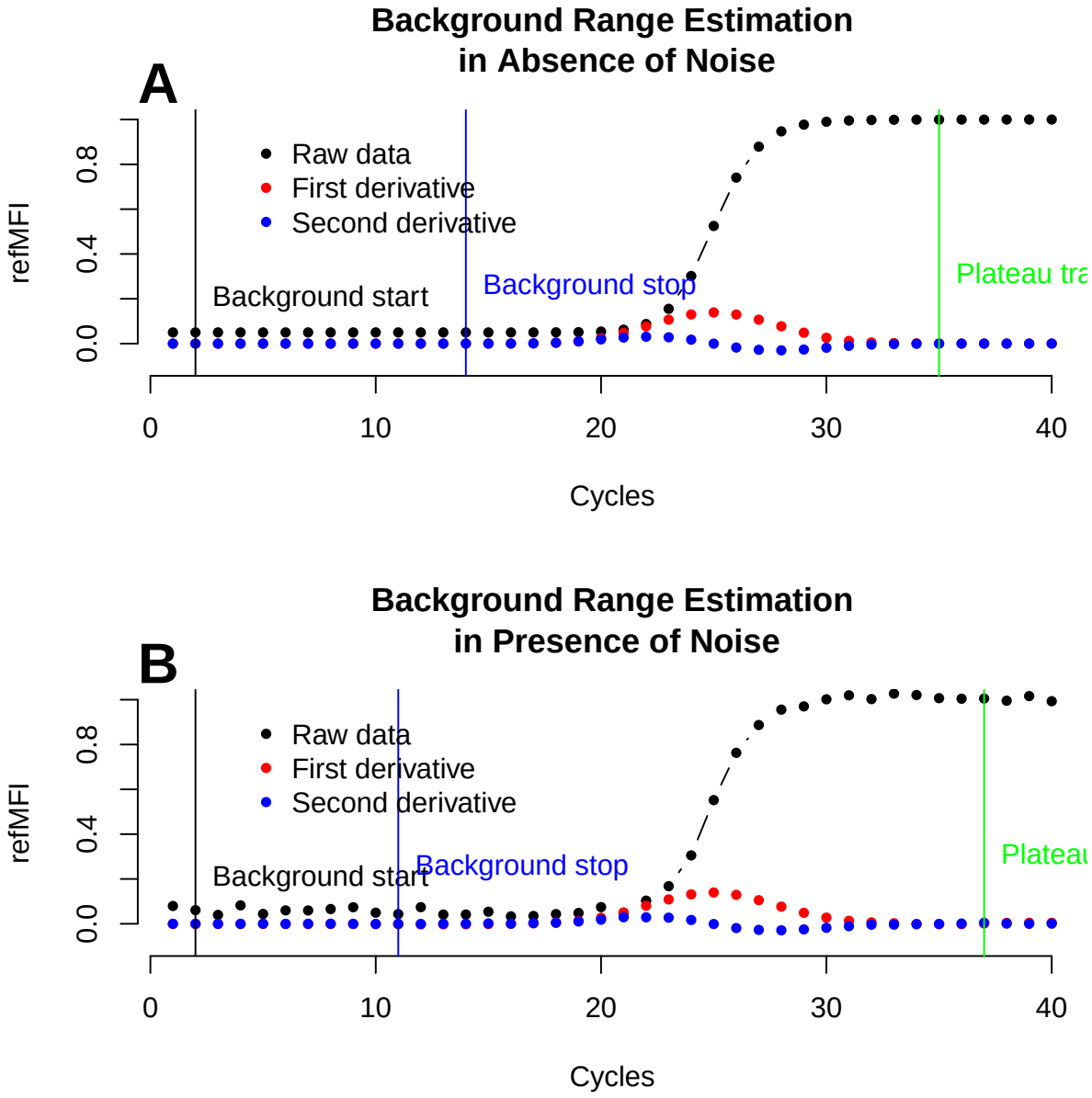

Figure S18: *bg.max* tries to estimate the range between the background and the plateau phase of an amplification reaction. *(A)* in absence and *(B)* presence of noise. The data were simulated with the *AmpSim* function.

```
    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)
}
```

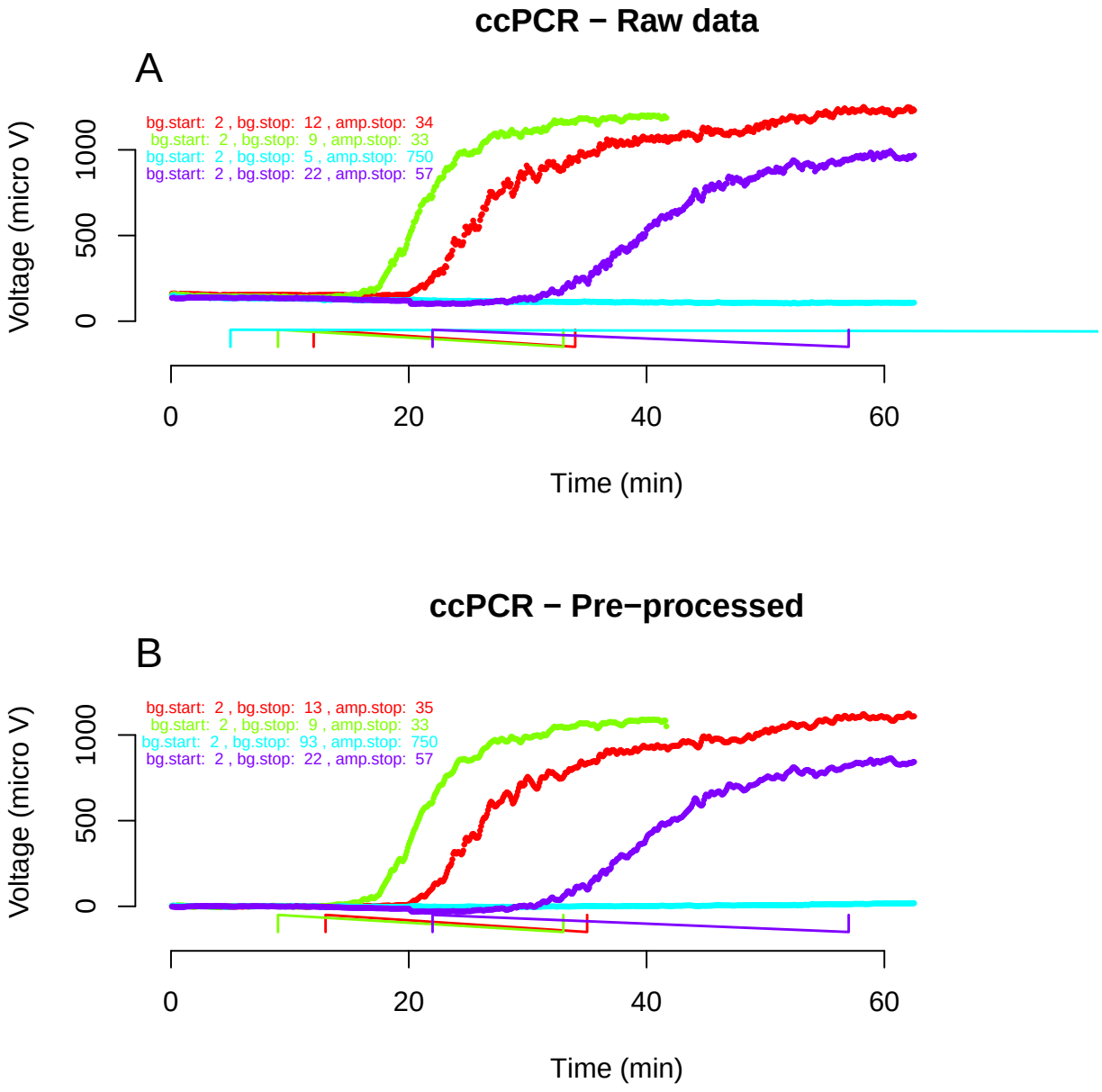

Figure S19: Application of the *bg.max* function. Amplification curve data from a capillary convective PCR were used *(A)* as raw data and *(B)* pre-processed (smoothed (moving average, window size 3), base-lined and trend corrected (robust MM-estimator)) with the CPP function. The output of the was used by *bg.max* to detected the start and the end of the amplification reaction. The start and end were reliably estimated (range between "bg.stop" and "amp.stop"). There was no significant difference between raw data and pre-processed data.

10 humanrater

The function *humanrater* is an interactive function, which can be used to rate a curve for a certain characteristic. *humanrater* draws individual graphs of a curve and prompts an input field for the user. This function can be used to compare the human rating and the rating of a machine.

11 Data sets

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

- Number of measurements: 97

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

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

15. Data set: Eff625:

    - Data set type: simulations
    - Description: Highly replicate number amplification curves with an approximate amplification efficiency of 62.5 percent at cycle number 18. The data were derived from a simulation such as the AmpSim function.
    - Number of variables: 40
    - Number of measurements: 1000

16. Data set: Eff750:

    - Data set type: simulations
    - Description: Highly replicate number amplification curves with an approximate amplification efficiency of 75 percent at cycle number 18. The data were derived from a simulation such as the AmpSim function.
    - Number of variables: 40
    - Number of measurements: 1000

17. Data set: Eff875:

    - Data set type: simulations
    - Description: Highly replicate number amplification curves with an approximate amplification efficiency of 87.5 percent at cycle number 18. The data were derived from a simulation such as the AmpSim function.
    - Number of variables: 40
    - Number of measurements: 1000

18. Data set: Eff1000:

    - Data set type: simulations
    - Description: Highly replicate number amplification curves with an approximate amplification efficiency of 100 percent at cycle number 18. The data were derived from a simulation such as the AmpSim function.
    - Number of variables: 40
    - Number of measurements: 1000

19. Data set: C67:

    - Data set type: Isothermal Amplifcation - Helicase Dependent Amplification
    - Description: A Helicase Dependent Amplification (HDA) of HRPT1 (Homo sapiens hypoxanthine phosphoribosyltransferase 1), performed at different input DNA quantities using the Bio-Rad iQ5 thermo cycler.
    - Number of variables: 43
    - Number of measurements: 6

20. Data set: CD75:

    - Data set type: Isothermal Amplifcation - Helicase Dependent Amplification
    - Description: Helicase Dependent Amplification in the VideoScan HCU of HRPT1 (Homo sapiens hypoxanthine phosphoribosyltransferase 1) measured at at 55, 60 or 65 degree Celsius.
    - Number of variables: 93
    - Number of measurements: 6

21. Data set: C81:

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

22. Data set: C85:

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

12 Acknowledgment

Part of this work was funded by the BMBF InnoProfile-Projekt 03 IPT 611X. Grateful thanks belong to all authors of the cited **R** packages, the **R** community and RKWard developers.

List of Figures

References

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[17] Stefan Rödiger, Peter Schierack, Alexander Böhm, Jörg Nitschke, Ingo Berger, Ulrike Frömmel, Carsten Schmidt, Mirko Ruhland, Ingolf Schimke, Dirk Roggenbuck, Werner Lehmann, and Christian Schröder. A highly versatile microscope imaging technology platform for the multiplex real-time detection of biomolecules and autoimmune antibodies. *Advances in Biochemical Engineering/Biotechnology*, 133:35–74, 2013.

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

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

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

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

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

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

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

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

[26] Sheng Zhao and Russell D. Fernald. Comprehensive algorithm for quantitative real-time polymerase chain reaction. *Journal of Computational Biology : a Journal of Computational Molecular Cell Biology*, 12(8):1047–1064, October 2005. PMID: 16241897 PMCID: PMC2716216.