Package: qpcR (via r-universe)

October 4, 2026

Type Package

LazyLoad yes

LazyData yes

Title Modelling and Analysis of Real-Time PCR Data

Version 1.4-2

Date 2025-06-05

Description Sigmoidal model fitting, optimal model selection and calculation of various features that are essential in the analysis of quantitative real-time polymerase chain reaction (qPCR), according to Ritz & Spiess (2008) <doi:10.1093/bioinformatics/btn227> and Spiess et al. (2008) <doi:10.1186/1471-2105-9-221>.

License GPL (>= 2)

Depends R (>= 2.13.0), MASS, minpack.lm, rgl, robustbase, Matrix

NeedsCompilation yes

Imports methods

Author Andrej-Nikolai Spiess [aut, cre]

Maintainer Andrej-Nikolai Spiess <draspiess@gmail.com>

Config/pak/sysreqs cmake libfreetype6-dev libglu1-mesa-dev make texlive libpng-dev libuv1-dev libgl1-mesa-dev zlib1g-dev

Repository https://anspiess.r-universe.dev

Date/Publication 2025-06-10 13:00:09 UTC

RemoteUrl https://github.com/cran/qpcR

RemoteRef HEAD

RemoteSha 0ad07f2036db03add33a475964522f357a4ab8df

Contents

AICc 3
akaike.weights 4

calib . . . 5
Cy0 . . . 7
eff . . . 8
efficiency . . . 10
evidence . . . 13
expcomp . . . 14
expfit . . . 15
fitchisq . . . 17
getPar . . . 18
is.outlier . . . 20
KOD . . . 21
llratio . . . 23
LOF.test . . . 24
LRE . . . 26
maxRatio . . . 28
meltcurve . . . 29
midpoint . . . 32
modlist . . . 33
mselect . . . 37
parKOD . . . 39
pcrbatch . . . 40
pcrboot . . . 42
pcrfit . . . 44
pcrGOF . . . 47
pcropt1 . . . 48
pcrsim . . . 49
plot.pcrfit . . . 51
predict.pcrfit . . . 53
PRESS . . . 54
propagate . . . 56
qpcR.news . . . 62
qpcR_datasets . . . 62
qpcR_functions . . . 67
ratiobatch . . . 71
ratiocalc . . . 76
ratioPar . . . 81
refmean . . . 85
replist . . . 88
resplot . . . 90
resVar . . . 91
RMSE . . . 92
Rsq . . . 93
Rsq.ad . . . 94
RSS . . . 95
sliwin . . . 96
takeoff . . . 98
update.pcrfit . . . 99

Index **101**

AICc *Akaike's second-order corrected Information Criterion*

Description

Calculates the second-order corrected Akaike Information Criterion for objects of class `pcrfit`, `nls`, `lm`, `glm` or any other models from which `coefficients` and `residuals` can be extracted. This is a modified version of the original AIC which compensates for bias with small n. As qPCR data usually has $\frac{n}{k} < 40$ (see original reference), AICc was implemented to correct for this.

Usage

```
AICc(object)
```

Arguments

`object`	a fitted model.

Details

Extends the AIC such that

$$AICc = AIC + \frac{2k(k+1)}{n-k-1}$$

with k = number of parameters, and n = number of observations. For large n, AICc converges to AIC.

Value

The second-order corrected AIC value.

Author(s)

Andrej-Nikolai Spiess

References

Akaike Information Criterion Statistics.
Sakamoto Y, Ishiguro M and Kitagawa G.
D. Reidel Publishing Company (1986).

Regression and Time Series Model Selection in Small Samples.
Hurvich CM & Tsai CL.
Biometrika (1989), **76**: 297-307.

See Also

`AIC`, `logLik`.

Examples

```
m1 <- pcrfit(reps, 1, 2, l5)
AICc(m1)
```

`akaike.weights` *Calculation of Akaike weights/relative likelihoods/delta-AICs*

Description

Calculates Akaike weights from a vector of AIC values.

Usage

```
akaike.weights(x)
```

Arguments

`x`	a vector containing the AIC values.

Details

Although Akaike's Information Criterion is recognized as a major measure for selecting models, it has one major drawback: The AIC values lack intuitivity despite higher values meaning less goodness-of-fit. For this purpose, Akaike weights come to hand for calculating the weights in a regime of several models. Additional measures can be derived, such as $\Delta(AIC)$ and relative likelihoods that demonstrate the probability of one model being in favor over the other. This is done by using the following formulas:

delta AICs:

$$\Delta_i(AIC) = AIC_i - min(AIC)$$

relative likelihood:

$$L \propto exp\left(-\frac{1}{2}\Delta_i(AIC)\right)$$

Akaike weights:

$$w_i(AIC) = \frac{exp\left(-\frac{1}{2}\Delta_i(AIC)\right)}{\sum_{k=1}^{K} exp\left(-\frac{1}{2}\Delta_k(AIC)\right)}$$

Value

A list containing the following items:

`deltaAIC`	the $\Delta(AIC)$ values.
`rel.LL`	the relative likelihoods.
`weights`	the Akaike weights.

Author(s)

Andrej-Nikolai Spiess

References

Classical literature:
Akaike Information Criterion Statistics.
Sakamoto Y, Ishiguro M and Kitagawa G.
D. Reidel Publishing Company (1986).

Model selection and inference: a practical information-theoretic approach.
Burnham KP & Anderson DR.
Springer Verlag, New York, USA (2002).

A good summary:
AIC model selection using Akaike weights. Wagenmakers EJ & Farrell S.
Psychonomic Bull Review (2004), **11**: 192-196.

See Also

AIC, logLik.

Examples

```
## Apply a list of different sigmoidal models to data
## and analyze GOF statistics with Akaike weights
## on 8 different sigmoidal models.
modList <- list(l7, l6, l5, l4, b7, b6, b5, b4)
aics <- sapply(modList, function(x) AIC(pcrfit(reps, 1, 2, x)))
akaike.weights(aics)$weights
```

`calib` *Calculation of qPCR efficiency using dilution curves and replicate bootstrapping*

Description

This function calculates the PCR efficiency from a classical qPCR dilution experiment. The threshold cycles are plotted against the logarithmized concentration (or dilution) values, a linear regression line is fit and the efficiency calculated by $E = 10^{\frac{-1}{slope}}$. A graph is displayed with the raw values plotted with the threshold cycle and the linear regression curve. The threshold cycles are calculated either by some arbitrary fluorescence value (i.e. as given by the qPCR software) or calculated from the second derivative maximum of the dilution curves. If values to be predicted are given, they are calculated from the curve and also displayed within. `calib2` uses a bootstrap approach if replicates for the dilutions are supplied. See 'Details'.

Usage

```
calib(refcurve, predcurve = NULL, thresh = "cpD2", dil = NULL,
       group = NULL, plot = TRUE, conf = 0.95, B = 200)
```

Arguments

refcurve	a 'modlist' containing the curves for calibration.
predcurve	an (optional) 'modlist' containing the curves for prediction.
thresh	the fluorescence value from which the threshold cycles are defined. Either "cpD2" or a numeric value.
dil	a vector with the concentration (or dilution) values corresponding to the calibration curves.
group	a factor defining the group membership for the replicates. See 'Examples'.
plot	logical. Should the fitting (bootstrapping) be displayed? If FALSE, only values are returned.
conf	the confidence interval. Defaults to 95%, can be omitted with NULL.
B	the number of bootstraps.

Details

calib calculates confidence intervals for efficiency, AICc, adjusted R^2_{adj} and the prediction curve concentrations. If single replicates per dilution are supplied by the user, confidence intervals for the prediction curves are calculated based on asymptotic normality. If multiple replicates are supplied, the regression curves are calculated by randomly sampling one of the replicates from each dilution group. The confidence intervals are then calculated from the bootstraped results.

Value

A list with the following components:

eff	the efficiency.
AICc	the second-order corrected AIC.
Rsq.ad	the adjusted R^2_{adj}.
predconc	the (log) concentration of the predicted curves.
conf.boot	a list containing the confidence intervals for the efficiency, the AICc, Rsq.ad and the predicted concentrations.

A plot is also supplied for efficiency, AICc, Rsq.ad and predicted concentrations including confidence intervals in red.

Author(s)

Andrej-Nikolai Spiess

Examples

```
## Define calibration curves,
## dilutions (or copy numbers)
## and curves to be predicted.
## Do background subtraction using
## average of first 8 cycles. No replicates.
CAL <- modlist(reps, fluo = c(2, 6, 10, 14, 18, 22),
               baseline = "mean", basecyc = 1:8)
COPIES <- c(100000, 10000, 1000, 100, 10, 1)
PRED <- modlist(reps, fluo = c(3, 7, 11),
                baseline = "mean", basecyc = 1:8)

## Conduct normal quantification using
## the second derivative maximum of first curve.
calib(refcurve = CAL, predcurve = PRED, thresh = "cpD2",
       dil = COPIES, plot = FALSE)
```

`Cy0` *Cy0 alternative to threshold cycles as in Guescini et al. (2008)*

Description

An alternative to the classical crossing point/threshold cycle estimation as described in Guescini *et al* (2002). A tangent is fit to the first derivative maximum (point of inflection) of the modeled curve and the intersection with the x-axis is calculated.

Usage

```
Cy0(object, plot = FALSE, add = FALSE, ...)
```

Arguments

`object`	a fitted object of class 'pcrfit'.
`plot`	if `TRUE`, displays a plot of Cy0.
`add`	if `TRUE`, a plot is added to any other existing plot, i.e. as from `plot.pcrfit`.
`...`	other parameters to be passed to `plot.pcrfit` or `points`.

Details

The function calculates the first derivative maximum (cpD1) of the curve and the slope and fluorescence F_{cpD2} at that point. Cy0 is then calculated by $Cy_0 = cpD1 - \frac{F_{cpD2}}{slope}$.

Value

The Cy_0 value.

Author(s)

Andrej-Nikolai Spiess

References

A new real-time PCR method to overcome significant quantitative inaccuracy due to slight amplification inhibition.
Guescini M, Sisti D, Rocchi MB, Stocchi L & Stocchi V.
BMC Bioinformatics (2008), **9**: 326.

Examples

```
## Single curve with plot.
m1 <- pcrfit(reps, 1, 2, l5)
Cy0(m1, plot = TRUE)

## Add to 'efficiency' plot.
efficiency(m1)
Cy0(m1, add = TRUE)

## Compare s.d. of replicates between
## Cy0 and cpD2 method. cpD2 wins!
ml1 <- modlist(reps, model = l4)
cy0 <- sapply(ml1, function(x) Cy0(x))
cpd2 <- sapply(ml1, function(x) efficiency(x, plot = FALSE)$cpD2)
tapply(cy0, gl(7, 4), function(x) sd(x))
tapply(cpd2, gl(7, 4), function(x) sd(x))
```

---

eff *The amplification efficiency curve of a fitted object*

Description

Calculates the efficiency curve from the fitted object by $E_n = \frac{F_n}{F_{n-1}}$, with E = efficiency, F = raw fluorescence, n = Cycle number. Alternatively, a cubic spline interpolation can be used on the raw data as in Shain *et al*. (2008).

Usage

```
eff(object, method = c("sigfit", "spline"), sequence = NULL, baseshift = NULL,
    smooth = FALSE, plot = FALSE)
```

Arguments

`object`	an object of class 'pcrfit'.
`method`	the efficiency curve is either calculated from the sigmoidal fit (default) or a cubic spline interpolation.

sequence	a 3-element vector (from, to, by) defining the sequence for the efficiency curve. Defaults to [min(Cycles), max(Cycles)] with 100 points per cycle.
baseshift	baseline shift value in case of type = "spline". See documentation to maxRatio.
smooth	logical. If TRUE and type = "spline", invokes a 5-point convolution filter (filter). See documentation to maxRatio.
plot	should the efficiency be plotted?

Details

For more information about the curve smoothing, baseline shifting and cubic spline interpolation for the method as in Shain *et al*. (2008), see 'Details' in maxRatio.

Value

A list with the following components:

eff.x	the cycle points.
eff.y	the efficiency values at eff.x.
effmax.x	the cycle number with the highest efficiency.
effmax.y	the maximum efficiency.

Author(s)

Andrej-Nikolai Spiess

References

A new method for robust quantitative and qualitative analysis of real-time PCR.
Shain EB & Clemens JM.
Nucleic Acids Research (2008), **36**, e91.

Examples

```
## With default 100 points per cycle.
m1 <- pcrfit(reps, 1, 7, l5)
eff(m1, plot = TRUE)

## Not all data and only 10 points per cycle.
eff(m1, sequence = c(5, 35, 0.1), plot = TRUE)

## When using cubic splines it is preferred
## to use the smoothing option.
eff(m1, method = "spline", plot = TRUE, smooth = TRUE, baseshift = 0.3)
```

`efficiency` *Calculation of qPCR efficiency and other important qPCR parameters*

Description

This function calculates the PCR efficiency of a model of class 'pcrfit', including several other important values for qPCR quantification like the first and second derivatives and the corresponding maxima thereof (i.e. threshold cycles). These values can subsequently be used for the calculation of PCR kinetics, fold induction etc. All values are included in a graphical output of the fit. Additionally, several measures of goodness-of-fit are calculated, i.e. the Akaike Information Criterion (AIC), the residual variance and the R^2 value.

Usage

```
efficiency(object, plot = TRUE, type = "cpD2", thresh = NULL,
           shift = 0, amount = NULL, ...)
```

Arguments

`object`	an object of class 'pcrfit'.
`plot`	logical. If TRUE, a graph is displayed. If FALSE, values are printed out.
`type`	the method of efficiency estimation. See 'Details'.
`thresh`	an (optional) numeric value for a fluorescence threshold border. Overrides `type`.
`shift`	a user defined shift in cycles from the values defined by `type`. See 'Examples'.
`amount`	the template amount or molecule number for quantitative calibration.
`...`	other parameters to be passed to `eff` or `plot.pcrfit`.

Details

The efficiency is always (with the exception of `type = "maxRatio"`) calculated from the efficiency curve (in blue), which is calculated according to $E_n = \frac{F_n}{F_{n-1}}$ from the fitted curve, but taken from different points at the curve, as to be defined in `type`:

"cpD2" taken from the maximum of the second derivative curve,
"cpD1" taken from the maximum of the first derivative curve,
"maxE" taken from the maximum of the efficiency curve,
"expR" taken from the exponential region by $expR = cpD2 - (cpD1 - cpD2)$,
"CQ" taken from the 20% value of the fluorescence at "cpD2" as developed by Corbett Research (comparative quantification),
"Cy0" the intersection of a tangent on the first derivative maximum with the abscissa as calculated according to Guescini et al. or
a numeric value taken from the threshold cycle output of the PCR software, i.e. 15.24 as defined in `type` or
a numeric value taken from the fluorescence threshold output of the PCR software as defined in `thresh`.

The initial fluorescence F_0 for relative or absolute quantification is either calculated by setting $x = 0$ in the sigmoidal model of `object` giving `init1` or by calculating an exponential model down (`init2`) with $F_0 = \frac{F_n}{E^n}$, with F_n = raw fluorescence, E = PCR efficiency and n = the cycle number defined by `type`. If a template amount is defined, a conversion factor $cf = \frac{amount}{F_0}$ is given. The different measures for goodness-of-fit give an overview for the validity of the efficiency estimation. First and second derivatives are calculated from the fitted function and the maxima of the derivatives curve and the efficiency curve are obtained.

If `type = "maxRatio"`, the maximum efficiency is calculated from the cubic spline interpolated raw fluorescence values and therefore NOT from the sigmoidal fit. This is a different paradigm and will usually result in fairly the same threshold cycles as with `type = "cpD2"`, but the efficiencies are generally lower. See documentation to `maxRatio`. This method is usually not applied for calculating efficiencies that are to be used for relative quantification, but one might try...

Value

A list with the following components:

`eff`	the PCR efficiency.
`resVar`	the residual variance.
`AICc`	the bias-corrected Akaike Information Criterion.
`AIC`	the Akaike Information Criterion.
`Rsq`	the R^2 value.
`Rsq.ad`	the adjusted R^2_{adj} value.
`cpD1`	the first derivative maximum (point of inflection in 'l4' or 'b4' models, can be used for defining the threshold cycle).
`cpD2`	the second derivative maximum (turning point of cpD1, more often used for defining the threshold cycle).
`cpE`	the PCR cycle with the highest efficiency.
`cpR`	the PCR cycle within the exponential region calculated as under 'Details'.
`cpT`	the PCR cycle corresponding to the fluorescence threshold as defined in `thresh`.
`Cy0`	the PCR threshold cycle 'Cy0' according to Guescini et al. See 'Details'.
`cpCQ`	the PCR cycle corresponding to the 20% fluorescence value at 'cpD2'.
`cpMR`	the PCR cycle corresponding to the 'maxRatio', if this was selected.
`fluo`	the raw fluorescence value at the point defined by `type` or `thresh`.
`init1`	the initial template fluorescence from the sigmoidal model, calculated as under 'Details'.
`init2`	the initial template fluorescence from an exponential model, calculated as under 'Details'.
`cf`	the conversion factor between raw fluorescence and template amount, if the latter is defined.

If `object` was of type 'modlist', the results are given as a matrix, with samples in columns.

Note

In some curves that are fitted with the 'b5'/'l5' models, the 'f' (asymmetry) parameter can be extremely high due to severe asymmetric structure. The efficiency curve deduced from the coefficients of the fit can then be very extreme in the exponential region. It is strongly advised to use `efficiency(object, method = "spline")` so that `eff` calculates the curve from a cubic spline of the original data points (see 'Examples').

Three parameter models ('b3' or 'l3') do not work very well in calculating the PCR efficiency. It is advisable not to take too many cycles of the plateau phase prior to fitting the model as this has a strong effect on the validity of the efficiency estimates.

Author(s)

Andrej-Nikolai Spiess

References

Validation of a quantitative method for real time PCR kinetics.
Liu W & Saint DA.
BBRC (2002), **294**: 347-353.

A new real-time PCR method to overcome significant quantitative inaccuracy due to slight amplification inhibition.
Guescini M, Sisti D, Rocchi MB, Stocchi L & Stocchi V.
BMC Bioinformatics (2008), **9**: 326.

Examples

```
## Fitting initial model.
m1 <-  pcrfit(reps, 1, 2, l4)
efficiency(m1)

## Using one cycle 'downstream'
## of second derivative max.
efficiency(m1, type = "cpD2", shift = -1)

## Using "maxE" method, with calculation of PCR efficiency
## 2 cycles 'upstream' from the cycle of max efficiency.
efficiency(m1, type = "maxE", shift = 2)

## Using the exponential region.
efficiency(m1, type = "expR")

## Using threshold cycle (i.e. 15.32)
## from PCR software.
efficiency(m1, type = 15.32)

## Using Cy0 method from Guescini et al. (2008)
## add Cy0 tangent.
efficiency(m1, type = "Cy0")
Cy0(m1, add = TRUE)
```

```
## Using a defined fluorescence
## threshold value from PCR software.
efficiency(m1, thresh = 1)

## Using the first 30 cycles and a template amount
## (optical calibration).
m2 <-  pcrfit(reps[1:30, ], 1, 2, l5)
efficiency(m2, amount = 1E3)

## Using 'maxRatio' method from Shain et al. (2008)
## baseshifting essential!
efficiency(m1, type = "maxRatio", baseshift = 0.2)

## Using the efficiency from a cubic spline fit
## of the 'eff' function.
efficiency(m1, method = "spline")


## On a modlist with plotting
## of the efficiencies.
ml1 <- modlist(reps, model = l5)
res <- sapply(ml1, function(x) efficiency(x)$eff)
barplot(as.numeric(res))
```

`evidence` *Evidence ratio for model comparisons with AIC, AICc or BIC*

Description

The evidence ratio

$$\frac{1}{exp(-0.5 \cdot (IC2 - IC1))}$$

is calculated for one of the information criteria $IC = AIC, AICc, BIC$ either from two fitted models or two numerical values. Models can be compared that are not nested and where the f-test on residual-sum-of-squares is not applicable.

Usage

```
evidence(x, y, type = c("AIC", "AICc", "BIC"))
```

Arguments

`x`	a fitted object or numerical value.
`y`	a fitted object or numerical value.
`type`	any of the three Information Criteria `AIC`, `AICc` or `BIC`.

Details

Small differences in values can mean substantial more 'likelihood' of one model over the other. For example, a model with AIC = -130 is nearly 150 times more likely than a model with AIC = -120.

Value

A value of the first model `x` being more likely than the second model `y`. If large, first model is better. If small, second model is better.

Author(s)

Andrej-Nikolai Spiess

Examples

```
## Compare two four-parameter and five-parameter
## log-logistic models.
m1 <- pcrfit(reps, 1, 2, l4)
m2 <- pcrfit(reps, 1, 2, l5)
evidence(m2, m1)

## Ratio of two AIC's.
evidence(-120, -123)
```

---

expcomp *Comparison of all sigmodal models within the exponential region*

Description

The exponential region of the qPCR data is identified by the studentized outlier method, as in `expfit`. The root-mean-squared-error (RMSE) of all available sigmoidal models within this region is then calculated. The result of the fits are plotted and models returned in order of ascending RMSE.

Usage

```
expcomp(object, ...)
```

Arguments

`object`	an object of class 'pcrfit'.
`...`	other parameters to be passed to `expfit`.

Details

The following sigmoidal models are fitted: b4, b5, b6, b7, l4, l5, l6, l7

Value

A dataframe with names of the models, in ascending order of RMSE.

Author(s)

Andrej-Nikolai Spiess

Examples

```
m1 <- pcrfit(reps, 1, 2, l4)
expcomp(m1)
```

`expfit` *Calculation of PCR efficiency by fitting an exponential model*

Description

An exponential model is fit to a window of defined size on the qPCR raw data. The window is identified either by the second derivative maximum 'cpD2' (default), 'studentized outlier' method as described in Tichopad *et al.* (2003), the 'midpoint' method (Peirson *et al.*, 2003) or by subtracting the difference of cpD1 and cpD2 from cpD2 ('ERBCP', unpublished).

Usage

```
expfit(object, method = c("cpD2", "outlier", "midpoint", "ERBCP"),
       model = c("exp", "linexp"), offset = 0, pval = 0.05, n.outl = 3,
       n.ground = 1:5, corfact = 1, fix = c("top", "bottom", "middle"),
       nfit = 5, plot = TRUE, ...)
```

Arguments

`object`	an object of class 'pcrfit'.
`method`	one of the four possible methods to be used for defining the position of the fitting window.
`model`	which exponential model to use. `expGrowth` is default, but the linear-exponential model `linexp` can also be chosen.
`offset`	for `method = "cpD2"`, the cycle offset from second derivative maximum.
`pval`	for `method = "outlier"`, the p-value for the outlier test.
`n.outl`	for `method = "outlier"`, the number of successive outlier cycles.
`n.ground`	for `method = "midpoint"`, the number of cycles in the noisy ground phase to calculate the standard deviation from.
`corfact`	for `method = "ERBCP"`, the correction factor for finding the exponential region. See 'Details'.
`fix`	for methods "midpoint" and "ERBCP", the orientation of the fitting window based on the identified point. See 'Details'.

nfit	the size of the fitting window.
plot	logical. If TRUE, a graphical display of the curve and the fitted region is shown.
...	other parameters to be passed to the plotting function.

Details

The exponential growth function $f(x) = a \cdot exp(b \cdot x) + c$ is fit to a subset of the data. Calls efficiency for calculation of the second derivative maximum, takeoff for calculation of the studentized residuals and 'outlier' cycle, and midpoint for calculation of the exponential phase 'midpoint'. For method 'ERBCP' (Exponential Region By Crossing Points), the exponential region is calculated by $expR = cpD2 - \texttt{corfact} \cdot (cpD1 - cpD2)$. The efficiency is calculated from the exponential fit with $E = exp(b)$ and the inital template fluorescence $F_0 = a$.

Value

A list with the following components:

point	the point within the exponential region as identified by one of the three methods.
cycles	the cycles of the identified region.
eff	the efficiency calculated from the exponential fit.
AIC	the Akaike Information Criterion of the fit.
resVar	the residual variance of the fit.
RMSE	the root-mean-squared-error of the fit.
init	the initial template fluorescence.
mod	the exponential model of class 'nls'.

Author(s)

Andrej-Nikolai Spiess

References

Standardized determination of real-time PCR efficiency from a single reaction set-up.
Tichopad A, Dilger M, Schwarz G & Pfaffl MW.
Nucleic Acids Research (2003), **31**:e122.

Comprehensive algorithm for quantitative real-time polymerase chain reaction.
Zhao S & Fernald RD.
J Comput Biol (2005), **12**:1047-64.

Examples

```
## Using default SDM method.
m1 <- pcrfit(reps, 1, 2, l5)
expfit(m1)

## Using 'outlier' method.
expfit(m1, method = "outlier")
```

```
## Linear exponential model.
expfit(m1, model = "linexp")
```

`fitchisq` *The chi-square goodness-of-fit*

Description

Calculates χ^2, reduced χ^2_ν and the χ^2 fit probability for objects of class `pcrfit`, `lm`, `glm`, `nls` or any other object with a `call` component that includes `formula` and `data`. The function checks for replicated data (i.e. multiple same predictor values). If replicates are not given, the function needs error values, otherwise `NA`'s are returned.

Usage

```
fitchisq(object, error = NULL)
```

Arguments

`object`	a single model of class 'pcrfit', a 'replist' or any fitted model of the above.
`error`	in case of a model without replicates, a single error for all response values or a vector of errors for each response value.

Details

The variance of a fit s^2 is also characterized by the statistic χ^2 defined as followed:

$$\chi^2 \equiv \sum_{i=1}^{n} \frac{(y_i - f(x_i))^2}{\sigma_i^2}$$

The relationship between s^2 and χ^2 can be seen most easily by comparison with the reduced χ^2:

$$\chi^2_\nu = \frac{\chi^2}{\nu} = \frac{s^2}{\langle \sigma_i^2 \rangle}$$

whereas ν = degrees of freedom (N - p), and $\langle \sigma_i^2 \rangle$ is the weighted average of the individual variances. If the fitting function is a good approximation to the parent function, the value of the reduced chi-square should be approximately unity, $\chi^2_\nu = 1$. If the fitting function is not appropriate for describing the data, the deviations will be larger and the estimated variance will be too large, yielding a value greater than 1. A value less than 1 can be a consequence of the fact that there exists an uncertainty in the determination of s^2, and the observed values of χ^2_ν will fluctuate from experiment to experiment. To assign significance to the χ^2 value, we can use the integral probability

$$P_\chi(\chi^2; \nu) = \int_{\chi^2}^{\infty} P_\chi(x^2, \nu) dx^2$$

which describes the probability that a random set of *n* data points sampled from the parent distribution would yield a value of χ^2 equal to or greater than the calculated one. This is calculated by $1 - pchisq(\chi^2, \nu)$.

Value

A list with the following items:

`chi2`	the χ^2 value.
`chi2.red`	the reduced χ^2_ν.
`p.value`	the fit probability as described above.

Author(s)

Andrej-Nikolai Spiess

References

Data Reduction and Error Analysis for the Physical Sciences.
Bevington PR & Robinson DK.
McGraw-Hill, New York (2003).

Applied Regression Analysis.
Draper NR & Smith H.
Wiley, New York, 1998.

Examples

```
## Using replicates by making a 'replist'.
ml1 <- modlist(reps, fluo = 2:5)
rl1 <- replist(ml1, group = c(1, 1, 1, 1))
fitchisq(rl1[[1]])

## Using single model with added error.
m1 <- pcrfit(reps, 1, 2, l5)
fitchisq(m1, 0.1)
```

`getPar` *Batch calculation of qPCR fit parameters/efficiencies/threshold cycles with simple output, especially tailored to high-throughput data*

Description

This is a cut-down version of `pcrbatch`, starting with data of class 'modlist', which delivers a simple dataframe output, with either the parameters of the fit or calculated threshold cycles/efficiencies. The column names are deduced from the run names. All calculations have been error-protected through `tryCatch`, so whenever there is any kind of error (parameter extraction, efficiency estimation etc), `NA` is returned. This function can be used with high throughput data quite conveniently. All methods as in `pcrbatch` are available. The results are automatically copied to the clipboard.

Usage

```
getPar(x, type = c("fit", "curve"), cp = "cpD2", eff = "sigfit", ...)
```

Arguments

x	an object of class 'pcrfit' or 'modlist'.
type	fit will extract the fit parameters, curve will invoke efficiency and return threshold cycles/efficiencies.
cp	which method for threshold cycle estimation. Any of the methods in efficiency, i.e. "cpD2" (default), "cpD1", "maxE", "expR", "Cy0", "CQ", "maxRatio".
eff	which method for efficiency estimation. Either "sigfit" (default), "sliwin" or "expfit".
...	other parameters to be passed to efficiency, sliwin or expfit.

Details

Takes about 4 sec for 100 runs on a Pentium 4 Quad-Core (3 Ghz) when using type = "curve". When using type = "fit", the fitted model parameters are returned. If type = "curve", threshold cycles and efficiencies are calculated by efficiency based on the parameters supplied in ... (default cpD2).

Value

A dataframe, which is automatically copied to the clipboard.

Author(s)

Andrej-Nikolai Spiess.

Examples

```
## Simple example with fit parameters.
ml1 <- modlist(rutledge, model = l5)
getPar(ml1, type = "fit")

## Using a mechanistic model such as
## 'mak3' and extracting D0 values
## => initial template fluorescence.
ml2 <- modlist(rutledge, 1, 2:41, model = mak3)
res <- getPar(ml2, type = "fit")
barplot(log10(res[1, ]), las = 2)
```

`is.outlier` *Outlier summary for objects of class 'modlist' or 'replist'*

Description

For model lists of class 'modlist' or 'replist', `is.outlier` returns a vector of logicals for each run if they are outliers (i.e. sigmoidal or kinetic) or not.

Usage

```
is.outlier(object)
```

Arguments

`object`	an object of class 'modlist' or 'replist'.

Value

A vector of logicals with run names.

Author(s)

Andrej-Nikolai Spiess

See Also

`KOD`.

Examples

```
## Analyze in respect to amplification
## efficiency outliers.
ml1 <- modlist(reps, 1, 2:5)
res1 <- KOD(ml1, check = "uni2")

## Which runs are outliers?
outl <- is.outlier(res1)
outl
which(outl)


## Test for sigmoidal outliers
## with the 'testdat' dataset.
ml2 <- modlist(testdat, model = l5, check = "uni2")
is.outlier(ml2)
```

KOD *(K)inetic (O)utlier (D)etection using several methods*

Description

Identifies and/or removes qPCR runs according to several published methods or own ideas. The univariate measures are based on efficiency or difference in first/second derivative maxima. Multivariate methods are implemented that describe the structure of the curves according to several fixpoints such as first/second derivative maximum, slope at first derivative maximum or plateau fluorescence. These measures are compared with a set of curves using the `mahalanobis` distance with a robust covariance matrix and calculation of statistics by a χ^2 distribution. See 'Details'.

Usage

```
KOD(object, method = c("uni1", "uni2", "multi1", "multi2", "multi3"),
    par = parKOD(), remove = FALSE, verbose = TRUE, plot = TRUE,  ...)
```

Arguments

`object`	an object of class 'modlist' or 'replist'.
`method`	which method to use for kinetic outlier identification. Method `"uni1"` is default. See 'Details' for all methods.
`par`	parameters for the different `methods`. See `parKOD`.
`remove`	logical. If `TRUE`, outlier runs are removed and the object is updated. If `FALSE`, the individual qPCR runs are tagged as 'outliers' or not. See 'Details'.
`verbose`	logical. If `TRUE`, all calculation steps and results are displayed on the console.
`plot`	logical. If `TRUE`, a multivariate plot is displayed.
`...`	any other parameters to be passed to `sliwin`, `efficiency` or `expfit`.

Details

The following methods for the detection of kinetic outliers are implemented

`uni1`: KOD method according to Bar et al. (2003). Outliers are defined by removing the sample efficiency from the replicate group and testing it against the remaining samples' efficiencies using a Z-test:

$$P = 2 \cdot \left[1 - \Phi\left(\frac{e_i - \mu_{train}}{\sigma_{train}}\right)\right] < 0.05$$

`uni2`: This method from the package author is more or less a test on sigmoidal structure for the individual curves. It is different in that there is no comparison against other curves from a replicate set. The test is simple: The difference between first and second derivative maxima should be less than 10 cycles:

$$\left(\frac{\partial^3 F(x; a, b, ...)}{\partial x^3} = 0\right) - \left(\frac{\partial^2 F(x; a, b...)}{\partial x^2} = 0\right) < 10$$

Sounds astonishingly simple, but works: Runs are defines as 'outliers' that really failed to amplify, i.e. have no sigmoidal structure or are very shallow. It is the default setting in `modlist`.

`multi1`: KOD method according to Tichopad et al. (2010). Assuming two vectors with first and second derivative maxima t_1 and t_2 from a 4-parameter sigmoidal fit within a window of points around the first derivative maximum, a linear model $t_2 = t_1 \cdot b + a + \tau$ is made. Both t_1 and the residuals from the fit $\tau = t_2 - \hat{t_2}$ are Z-transformed:

$$t_1(norm) = \frac{t_1 - \bar{t}_1}{\sigma_{t1}},\ \tau_{1norm} = \frac{\tau_1 - \bar{\tau}_1}{\sigma_{\tau 1}}$$

Both t_1 and τ are used for making a robust covariance matrix. The outcome is plugged into a `mahalanobis` distance analysis using the 'adaptive reweighted estimator' from package 'mvoutlier' and p-values for significance of being an 'outlier' are deduced from a χ^2 distribution. If more than two parameters are supplied, `princomp` is used instead.

`multi2`: Second KOD method according to Tichopad et al. (2010), mentioned in the paper. Uses the same pipeline as `multi1`, but with the slope at the first derivative maximum and maximum fluorescence as parameters:

$$\frac{\partial F(x; a, b, ...)}{\partial x}, F_{max}$$

`multi3`: KOD method according to Sisti et al. (2010). Similar to `multi2`, but uses maximum fluorescence, slope at first derivative maximum and y-value at first derivative maximum as fixpoints:

$$\frac{\partial F(x; a, b, ...)}{\partial x}, F\left(\frac{\partial^2 F(x; a, b, ...)}{\partial x^2} = 0\right), F_{max}$$

All essential parameters for the methods can be tweaked by `parKOD`. See there and in 'Examples'.

Value

An object of the same class as in `object` that is 'tagged' in its name (**name**) if it is an outlier and also with an item `$isOutlier` with outlier information (see `is.outlier`). If `remove = TRUE`, the outlier runs are removed (and the fitting updated in case of a 'replist').

Author(s)

Andrej-Nikolai Spiess

References

Kinetic Outlier Detection (KOD) in real-time PCR.
Bar T, Stahlberg A, Muszta A & Kubista M.
Nucl Acid Res (2003), **31**: e105.

Quality control for quantitative PCR based on amplification compatibility test.
Tichopad A, Bar T, Pecen L, Kitchen RR, Kubista M &, Pfaffl MW.
Methods (2010), **50**: 308-312.

Shape based kinetic outlier detection in real-time PCR.
Sisti D, Guescini M, Rocchi MBL, Tibollo P, D'Atri M & Stocchi V.
BMC Bioinformatics (2010), **11**: 186.

See Also

Function is.outlier to get an outlier summary.

Examples

```
## kinetic outliers:
## on a 'modlist', using efficiency from sigmoidal fit
## and alpha = 0.01.
## F7.3 detected as outlier (shallower => low efficiency)
ml1 <- modlist(reps, 1, c(2:5, 28), model = l5)
res1 <- KOD(ml1, method = "uni1", par = parKOD(eff = "sliwin", alpha = 0.01))
plot(res1)

## Sigmoidal outliers:
## remove runs without sigmoidal structure.
ml2 <- modlist(testdat, model = l5)
res2 <- KOD(ml2, method = "uni2", remove = TRUE)
plot(res2, which = "single")


## Multivariate outliers:
## a few runs are identified.
ml3 <- modlist(reps, model = l5)
res3 <- KOD(ml3, method = "multi1")

## On a 'replist', several outliers identified.
rl3 <- replist(ml3, group = gl(7, 4))
res4 <- KOD(rl3, method = "uni1")
```

---

llratio *Calculation of likelihood ratios for nested models*

Description

Calculates the likelihood ratio and p-value from a chi-square distribution for two nested models.

Usage

```
llratio(objX, objY)
```

Arguments

objX	Either a value of class logLik or a model for which logLik can be applied.
objY	Either a value of class logLik or a model for which logLik can be applied.

Details

The likelihood ratio statistic is

$$LR = \frac{f(X, \hat{\phi}, \hat{\psi})}{f(X, \phi, \hat{\psi}_0)}$$

The usual test statistic is

$$\Lambda = 2 \cdot (l(\hat{\phi}, \hat{\psi}) - l(\phi, \hat{\psi}_0))$$

Following the large sample theory, if H_0 is true, then

$$\Lambda \sim \chi^2_p$$

Value

A list containing the following items:

`ratio`	the likelihood ratio statistic.
`df`	the change in parameters.
`p.value`	the p-value from a χ^2 distribution. See Details.

Author(s)

Andrej-Nikolai Spiess

See Also

`AIC`, `logLik`.

Examples

```
## Compare l5 and l4 model.
m1 <- pcrfit(reps, 1, 2, l5)
m2 <- pcrfit(reps, 1, 2, l4)
llratio(m1, m2)
```

`LOF.test` *Formal lack-Of-Fit test of a nonlinear model against a one-way ANOVA model*

Description

Tests the nonlinear model against a more general one-way ANOVA model and from a likelihood ratio test. P-values are derived from the F- and χ^2 distribution, respectively.

Usage

```
LOF.test(object)
```

Arguments

object	an object of class 'replist', 'pcrfit' or 'nls', which was fit with replicate response values.

Details

The one-way ANOVA model is constructed from the data component of the nonlinear model by factorizing each of the predictor values. Hence, the nonlinear model becomes a submodel of the one-way ANOVA model and we test both models with the null hypothesis that the ANOVA model can be simplified to the nonlinear model (Lack-of-fit test). This is done by two approaches:

1) an F-test (Bates & Watts, 1988).
2) a likelihood ratio test (Huet *et al*, 2004).

P-values are derived from an F-distribution (1) and a χ^2 distribution (2).

Value

A list with the following components:

pF	the p-value from the F-test against the one-way ANOVA model.
pLR	the p-value from the likelihood ratio test against the one-way ANOVA model.

Author(s)

Andrej-Nikolai Spiess

References

Nonlinear Regression Analysis and its Applications.
Bates DM & Watts DG.
John Wiley & Sons (1988), New York.

Statistical Tools for Nonlinear Regression: A Practical Guide with S-PLUS and R Examples.
Huet S, Bouvier A, Poursat MA & Jolivet E.
Springer Verlag (2004), New York, 2nd Ed.

Examples

```
## Example with a 'replist'
## no lack-of-fit.
ml1 <- modlist(reps, fluo = 2:5, model = l5)
rl1 <- replist(ml1, group = c(1, 1, 1, 1))
LOF.test(rl1)

## Example with a 'nls' fit
## => there is a lack-of-fit.
DNase1 <- subset(DNase, Run == 1)
fm1DNase1 <- nls(density ~ SSlogis(log(conc), Asym, xmid, scal), DNase1)
LOF.test(fm1DNase1)
```

LRE *Calculation of qPCR efficiency by the 'linear regression of efficiency' method*

Description

The LRE method is based on a linear regression of raw fluorescence versus efficiency, with the final aim to obtain cycle dependent individual efficiencies E_n. A linear model is then fit to a sliding window of defined size(s) and within a defined border. Regression coefficients are calculated for each window, and from the window of maximum regression, parameters such as PCR efficiency and initial template fluorescence are calculated. See 'Details' for more information. This approach is quite similar to the one in `sliwin`, but while `sliwin` regresses cycle number versus log(fluorescence), `LRE` regresses raw fluorescence versus efficiency. Hence, the former is based on assuming a constant efficiency for all cycles while the latter is based on a per-cycle individual efficiency.

Usage

```
LRE(object, wsize = 6, basecyc = 1:6, base = 0, border = NULL,
    plot = TRUE, verbose = TRUE, ...)
```

Arguments

`object`	an object of class 'pcrfit'.
`wsize`	the size(s) of the sliding window(s), default is `6`. A sequence such as `4:6` can be used to optimize the window size.
`basecyc`	if `base != 0`, which cycles to use for an initial baseline estimation based on the averaged fluorescence values.
`base`	either `0` for no baseline optimization, or a scalar defining multiples of the standard deviation of all baseline points obtained from `basecyc`. These are iteratively subtracted from the raw data. See 'Details' and 'Examples'.
`border`	either `NULL` (default) or a two-element vector which defines the border from the take-off point to points nearby the upper asymptote (saturation phase). See 'Details'.
`plot`	if `TRUE`, the result is plotted with the fluorescence/efficiency curve, sliding window, regression line and baseline.
`verbose`	logical. If `TRUE`, more information is displayed in the console window.
`...`	only used internally for passing the parameter matrix.

Details

To avoid fits with a high R^2 in the baseline region, some border in the data must be defined. In `LRE`, this is by default (`base = NULL`) the region in the curve starting at the take-off cycle (top) as calculated from `takeoff` and ending at the transition region to the upper asymptote (saturation region). The latter is calculated from the first and second derivative maxima: $asympt = cpD1 + (cpD1 - cpD2)$. If the border is to be set by the user, `border` values such as `c(-2, 4)` extend these

values by $top + border[1]$ and $asympt + border[2]$. The efficiency is calculated by $E_n = \frac{F_n}{F_{n-1}}$ and regressed against the raw fluorescence values F: $E = F\beta + \epsilon$. For the baseline optimization, 100 baseline values Fb_i are interpolated in the range of the data:

$$F_{min} \leq Fb_i \leq base \cdot \sigma(F_{basecyc[1]}...F_{basecyc[2]})$$

and subtracted from F_n. For all iterations, the best regression window in terms of R^2 is found and its parameters returned. Two different initial template fluorescence values F_0 are calculated in `LRE`:

`init1`: Using the single maximum efficiency E_{max} (the intercept of the best fit) and the fluorescence at second derivative maximum F_{cpD2}, by

$$F_0 = \frac{F_{cpD2}}{E_{max}^{cpD2}}$$

`init2`: Using the cycle dependent efficiencies E_n from $n = 1$ to the near-lowest integer (floor) cycle of the second derivative maximum $n = \lfloor cpD2 \rfloor$, and the fluorescence at the floor of the second derivative maximum $F_{\lfloor cpD2 \rfloor}$, by

$$F_0 = \frac{F_{\lfloor cpD2 \rfloor}}{\prod E_n}$$

This approach corresponds to the paradigm described in Rutledge & Stewart (2008), by using cycle-dependent and decreasing efficiencies Δ_E to calculate F_0.

Value

A list with the following components:

`eff`	the maximum PCR efficiency E_{max} calculated from the best window.
`rsq`	the maximum R^2.
`base`	the optimized baseline value.
`window`	the best window found within the `borders`.
`parMat`	a matrix containing the parameters as above for each iteration.
`init1`	the initial template fluorescence F_0 assuming constant efficiency E_{max} as described under 'Details'.
`init2`	the initial template fluorescence F_0, assuming cycle-dependent efficiency E_n as described under 'Details'.

Author(s)

Andrej-Nikolai Spiess

References

A kinetic-based sigmoidal model for the polymerase chain reaction and its application to high-capacity absolute quantitative real-time PCR.
Rutledge RG & Stewart D.
BMC Biotech (2008), **8**: 47.

Examples

```
## Sliding window of size 5 between take-off point
## and 3 cycles upstream of the upper asymptote
## turning point, one standard deviation baseline optimization.
m1 <- pcrfit(reps, 1, 2, l4)
LRE(m1, wsize = 5, border = c(0, 3), base = 1)
```

`maxRatio` *The maxRatio method as in Shain et al. (2008)*

Description

The maximum ratio (MR) is determined along the cubic spline interpolated curve of $\frac{F_n}{F_{n-1}}$ and the corresponding cycle numbers FCN and its adjusted version FCNA are then calculated for MR.

Usage

```
maxRatio(x, method = c("spline", "sigfit"), baseshift = NULL,
         smooth = TRUE, plot = TRUE, ...)
```

Arguments

`x`	an object of class 'pcrfit' (single run) or 'modlist' (multiple runs).
`method`	the parameters are either calculated from the cubic spline interpolation (default) or a sigmoidal fit.
`baseshift`	numerical. Shift value in case of `type = "spline"`. See 'Details'.
`smooth`	logical. If `TRUE` and `type = "spline"`, invokes a 5-point convolution filter (`filter`). See 'Details'.
`plot`	Should diagnostic plots be displayed?
`...`	other parameters to be passed to `eff` or `plot`.

Details

In a first step, the raw fluorescence data can be smoothed by a 5-point convolution filter. This is optional but feasible for many qPCR setups with significant noise in the baseline region, and therefore set to `TRUE` as default. If `baseshift` is a numeric value, this is added to each response value $F_n = F_n + baseshift$ (baseline shifting). Finally, a cubic spline is fit with a resolution of 0.01 cycles and the maximum ratio (efficiency) is calculated by $MR = max(\frac{F_n}{F_{n-1}} - 1)$. FCN is then calculated as the cycle number at MR and from this the adjusted $FCNA = FCN - log_2(MR)$. Sometimes problems are encountered in which, due to high noise in the background region, randomly high efficiency ratios are calculated. This must be resolved by tweaking the `baseshift` value.

Value

A list with the following components:

`eff`	the maximum efficiency. Equals to `mr` + 1.
`mr`	the maximum ratio.
`fcn`	the cycle number at `mr`.
`fcna`	an adjusted `fcn`, as described in Shain et al.
`names`	the names of the runs as taken from the original dataframe.

Note

This function has been approved by the original author (Eric Shain).

Author(s)

Andrej-Nikolai Spiess

References

A new method for robust quantitative and qualitative analysis of real-time PCR.
Shain EB & Clemens JM.
Nucleic Acids Research (2008), **36**: e91.

Examples

```
## On single curve using baseline shifting.
m1 <- pcrfit(reps, 1, 2, l5)
maxRatio(m1, baseshift = 0.3)

## On a 'modlist' using baseline shifting.
ml1 <- modlist(reps, model = l5)
maxRatio(ml1, baseshift = 0.5)
```

`meltcurve` *Melting curve analysis with (iterative) Tm identification and peak area calculation/cutoff*

Description

This function conducts a melting curve analysis from the melting curve data of a real-time qPCR instrument. The data has to be preformatted in a way that for each column of temperature values there exists a corresponding fluorescence value column. See `edit(dyemelt)` for a proper format. The output is a graph displaying the raw fluorescence curve (black), the first derivative curve (red) and the identified melting peaks. The original data together with the results ($-\frac{\partial F}{\partial T}$ values, T_m values) are returned as a list. An automatic optimization procedure is also implemented which iterates over `span.smooth` and `span.peaks` values and finds the optimal parameter combination that delivers minimum residual sum-of-squares of the identified T_m values to known T_m values.

For all peaks, the areas can be calculated and only those included which have areas higher than a given cutoff (`cut.Area`). If no peak was identified meeting the cutoff values, the melting curves are flagged with a 'bad' attribute. See 'Details'.

Usage

```
meltcurve(data, temps = NULL, fluos = NULL, window = NULL,
          norm = FALSE, span.smooth = 0.05, span.peaks = 51,
          is.deriv = FALSE, Tm.opt = NULL, Tm.border = c(1, 1),
          plot = TRUE, peaklines = TRUE, calc.Area = TRUE,
          plot.Area = TRUE, cut.Area = 0,...)
```

Arguments

`data`	a dataframe containing the temperature and fluorescence data.
`temps`	a vector of column numbers reflecting the temperature values. If `NULL`, they are assumed to be 1, 3, 5, ...
`fluos`	a vector of column numbers reflecting the fluorescence values. If `NULL`, they are assumed to be 2, 4, 6, ...
`window`	a user-defined window for the temperature region to be analyzed. See 'Details'.
`norm`	logical. If `TRUE`, the fluorescence values are scaled between [0, 1].
`span.smooth`	the window span for curve smoothing. Can be tweaked to optimize T_m identification.
`span.peaks`	the window span for peak identification. Can be tweaked to optimize T_m identification. Must be an odd number.
`is.deriv`	logical. Use `TRUE`, if `data` is already in first derivative transformed format.
`Tm.opt`	a possible vector of known T_m values to optimize `span.smooth` and `span.peaks` against. See 'Details' and 'Examples'.
`Tm.border`	for peak area calculation, a vector containing left and right border temperature values from the T_m values. Default is -1/+1 ?C.
`plot`	logical. If `TRUE`, a plot with the raw melting curve, derivative curve and identified T_m values is displayed for each sample.
`peaklines`	logical. If `TRUE`, lines that show the identified peaks are plotted.
`calc.Area`	logical. If `TRUE`, all peak areas are calculated.
`plot.Area`	logical. If `TRUE`, the baselined area identified for the peaks is plotted by filling the peaks in red.
`cut.Area`	a peak area value to identify only those peaks with a higher area.
`...`	other parameters to be passed to `plot`.

Details

The melting curve analysis is conducted with the following steps:

1a) Temperature and fluorescence values are selected in a region according to `window`.
1b) If `norm = TRUE`, the fluorescence data is scaled into [0, 1] by `qpcR:::rescale`.

Then, the function qpcR:::TmFind conducts the following steps:
2a) A cubic spline function (splinefun) is fit to the raw fluorescence melt values.
2b) The first derivative values are calculated from the spline function for each of the temperature values.
2c) Friedman's supersmoother (supsmu) is applied to the first derivative values.
2d) Melting peaks (T_m) values are identified by qpcR:::peaks.
2e) Raw melt data, first derivative data, best parameters, residual sum-of-squares and identified T_m values are returned.
Peak areas are then calculated by qpcR:::peakArea:
3a) A linear regression curve is fit from the leftmost temperature value (T_m - Tm.border[1]) to the rightmost temperature value (T_m + Tm.border[2]) by lm.
3b) A baseline curve is calculated from the regression coefficients by predict.lm.
3c) The baseline data is subtracted from the first derivative melt data (baselining).
3d) A splinefun is fit to the baselined data.
3e) The area of this spline function is integrated from the leftmost to rightmost temperature value.
4) If calculated peak areas were below cut.Area, the corresponding T_m values are removed.
Finally,
5) A matrix of xyy-plots is displayed using qpcR:::xyy.plot.

is.deriv must be set to TRUE if the exported data was already transformed to $-\frac{\partial F}{\partial T}$ by the PCR system (i.e. Stratagene MX3000P).

If values are given to Tm.opt (see 'Examples'), then meltcurve is iterated over all combinations of span.smooth = seq(0, 0.2, by = 0.01) and span.peaks = seq(11, 201, by = 10). For each iteration, T_m values are calculated and compared to those given by measuring the residual sum-of-squares between the given values Tm.opt and the Tm values obtained during the iteration:

$$RSS = \sum_{i=1}^{n} (Tm_i - Tm.opt_i)^2$$

The returned list items containing the resulting data frame each has an attribute "quality" which is set to "bad" if none of the peaks met the cut.Area criterion (or "good" otherwise).

Value

A list with as many items as melting curves, named as in data, each containing a data.frame with the temperature (*Temp*), fluorescence values (*Fluo*), first derivative (*dF.dT*) values, (optimized) parameters of *span.smooth*/*span.peaks*, residual sum-of-squares (if Tm.opt != NULL), identified melting points (*Tm*), calculated peak areas (*Area*) and peak baseline values (*baseline*).

Note

The peaks function is derived from a R-Help mailing list entry in Nov 2005 by Martin Maechler.

Author(s)

Andrej-Nikolai Spiess

Examples

```
## Default columns.
data(dyemelt)
res1 <- meltcurve(dyemelt, window = c(75, 86))
res1

## Selected columns and normalized fluo values.
res2 <- meltcurve(dyemelt, temps = c(1, 3), fluos = c(2, 4),
                  window = c(75, 86), norm = TRUE)

## Removing peaks based on peak area
## => two peaks have smaller areas and are not included.
res3 <- meltcurve(dyemelt, temps = 1, fluos = 2, window = c(75, 86),
                  cut.Area = 0.2)
attr(res3[[1]], "quality")

## If all peak areas do not meet the cutoff value, meltcurve is
## flagged as 'bad'.
res4 <- meltcurve(dyemelt, temps = 1, fluos = 2, window = c(75, 86),
                  cut.Area = 0.5)
attr(res4[[1]], "quality")

## Optimizing span and peaks values.

res5 <- meltcurve(dyemelt[, 1:6], window = c(74, 88),
                  Tm.opt = c(77.2, 80.1, 82.4, 84.8))
```

`midpoint` *Calculation of the 'midpoint' region*

Description

Calculates the exponential region midpoint using the algorithm described in Peirson *et al.* (2003).

Usage

```
midpoint(object, noise.cyc = 1:5)
```

Arguments

`object`	a fitted object of class 'pcrfit'.
`noise.cyc`	the cycles defining the background noise.

Details

The 'midpoint' region is calculated by

$$F_{noise} \cdot \sqrt{\frac{F_{max}}{F_{noise}}}$$

with F_{noise} = the standard deviation of the background cycles and F_{max} = the maximal fluorescence.

Value

A list with the following components:

`f.mp`	the 'midpoint' fluorescence.
`cyc.mp`	the 'midpoint' cycle, as predicted from `f.mp`.

Author(s)

Andrej-Nikolai Spiess

References

Experimental validation of novel and conventional approaches to quantitative real-time PCR data analysis.
Peirson SN, Butler JN & Foster RG.
Nucleic Acids Research (2003), **31**: e73.

Examples

```
m1 <- pcrfit(reps, 1, 2, l5)
mp <- midpoint(m1)
plot(m1)
abline(h = mp$f.mp, col = 2)
abline(v = mp$mp, col = 2)
```

`modlist` *Create nonlinear models from a dataframe and coerce them into a list*

Description

Essential function to create a list of nonlinear models from the columns (runs) of a qPCR dataframe. This function houses different methods for curve transformation prior to fitting, such as normalization in [0, 1], smoothing, baseline subtraction etc. Runs that failed to fit or that have been identified as kinetic outliers (by default: lack of sigmoidal structure) can be removed automatically as well as their entries in an optionally supplied label vector.

Usage

```
modlist(x, cyc = 1, fluo = NULL, model = l4, check = "uni2",
        checkPAR = parKOD(), remove = c("none", "fit", "KOD"),
        exclude = NULL, labels = NULL, norm = FALSE,
        baseline = c("none", "mean", "median", "lin", "quad", "parm"),
        basecyc = 1:8, basefac = 1, smooth = NULL, smoothPAR = NULL,
        factor = 1, opt = FALSE,
        optPAR = list(sig.level = 0.05, crit = "ftest"), verbose = TRUE, ...)
```

Arguments

x	a dataframe containing the qPCR data or a single qPCR run of class 'pcrfit'.
cyc	the column containing the cycle data. Defaults to first column.
fluo	the column(s) (runs) to be analyzed. If NULL, all runs will be considered.
model	the model to be used for all runs.
check	the method for kinetic outlier detection. Default is check for sigmoidal structure, see KOD. To turn off, use NULL.
checkPAR	parameters to be supplied to the check method, see KOD.
remove	which runs to remove. Either "none", those which failed to "fit" or from the "KOD" outlier method.
exclude	either "" for samples with missing column names or a regular expression defining columns (samples) to be excluded from modlist. See 'Details'.
labels	a vector containing labels, i.e. for defining replicate groups prior to ratiobatch.
norm	logical. Should the raw data be normalized within [0, 1] before model fitting?
baseline	type of baseline subtraction. See 'Details'.
basecyc	cycle range to be used for baseline subtraction, i.e. 1:5.
basefac	a factor for the baseline value, such as 0.95.
smooth	which curve smoothing method to use. See 'Details'.
smoothPAR	parameters to be supplied to the smoothing functions, supplied as a list. See 'Details'.
factor	a multiplication factor for the fluorescence response values (barely useful, but who knows...).
opt	logical. Should model selection be applied to each model?
optPAR	parameters to be supplied to mselect.
verbose	logical. If TRUE, fitting and tagging results will be displayed in the console.
...	other parameters to be passed to pcrfit.

Details

From version 1.4-0, the following baselining methods are available for the fluorescence values:
baseline = numeric: a numeric value such as baseline = 0.2 for subtracting from each F_i.
"mean": subtracts the mean of all basecyc cycles from each F_i.

"median": subtracts the median of all basecyc cycles from each F_i.
"lin": creates a linear model of all basecyc cycles, predicts P_i over all cycles i from this model, and subtracts $F_i - P_i$.
"quad": creates a quadratic model of all basecyc cycles, predicts P_i over all cycles i from this model, and subtracts $F_i - P_i$.
"parm": extracts the c parameter from the fitted sigmoidal model and subtracts this value from all F_i.
It is switched off by default, but in case of data with a high baseline (such as in TaqMan PCR), it should be turned on as otherwise this will give highly underestimated efficiencies and hence wrong init2 values.

From version 1.3-8, the following smoothing methods are available for the fluorescence values:
"lowess": Lowess smoothing, see lowess, parameter in smoothPAR: f.
"supsmu": Friedman's SuperSmoother, see supsmu, parameter in smoothPAR: span.
"spline": Smoothing spline, see smooth.spline, parameter in smoothPAR: spar.
"savgol": Savitzky-Golay smoother, qpcR:::savgol, parameter in smoothPAR: none.
"kalman": Kalman smoother, see arima, parameter in smoothPAR: none.
"runmean": Running mean, see qpcR:::runmean, parameter in smoothPAR: wsize.
"whit": Whittaker smoother, see qpcR:::whittaker, parameter in smoothPAR: lambda.
"ema": Exponential moving average, see qpcR:::EMA, parameter in smoothPAR: alpha.
The author of this package advocates the use of "spline", "savgol" or "whit" because these three smoothers have the least influence on overall curve structure.

In case of unsuccessful model fitting and if remove = "none" (default), the original data is included in the output, albeit with no fitting information. This is useful since using plot.pcrfit on the 'modlist' shows the non-fitted runs. If remove = "fit", the non-fitted runs are automatically removed and will thus not be displayed. If remove = "KOD", by default all runs without sigmoidal structure are removed likewise. If a labels vector lab is supplied, the labels from the failed fits are removed and a new label vector lab_mod is written to the global environment. This way, an initial labeling vector for all samples can be supplied, bad runs and their labels automatically removed and these transfered to downstream analysis (i.e. to ratiobatch) without giving errors. exclude offers an option to exclude samples from the modlist by some regular expression or by using "" for samples with empty column names. See 'Examples'.

Value

A list with each item containing the model from each column. A names item (which is tagged by *NAME*, if fitting failed) containing the column name is attached to each model as well as an item isFitted with either TRUE (fitting converged) or FALSE (a fitting error occured). This information is useful when ratiocalc is to be applied and unsuccessful fits should automatically removed from the given group definition. If kinetic outlier detection is selected, an item isOutlier is attached, defining the run as an outlier (TRUE) or not (FALSE).

Author(s)

Andrej-Nikolai Spiess

See Also

pcrbatch for batch analysis using different methods.

Examples

```
## Calculate efficiencies and ct values
## for each run in the 'reps' data,
## subtract baseline using mean of
## first 8 cycles.
ml1 <- modlist(reps, model = l5, baseline = "mean")
getPar(ml1, type = "curve")

## 'Crossing points' for the first 3 runs (normalized)
##  and using best model from Akaike weights.
ml2 <- modlist(reps, 1, 2:5, model = l5, norm = TRUE,
               opt = TRUE, optPAR = list(crit = "weights"))
sapply(ml2, function(x) efficiency(x, plot = FALSE)$cpD2)

## Convert a single run to a 'modlist'.
m <- pcrfit(reps, 1, 2, l5)
ml3 <- modlist(m)

## Using the 'testdat' set
## include failed fits.
ml4 <- modlist(testdat, 1, 2:9,  model = l5)
plot(ml4, which = "single")

## Remove failed fits and update a label vector.
GROUP <- c("g1s1", "g1s2", "g1s3", "g1s4", "g1c1", "g1c2", "g1c3", "g1c4")
ml5 <- modlist(testdat, 1, 2:9,  model = l5, labels = GROUP, remove = "KOD")
plot(ml5, which = "single")

## Smoothing by EMA and alpha = 0.8.
ml6 <- modlist(reps, model = l5, smooth = "ema",
               smoothPAR = list(alpha = 0.5))
plot(ml6)


## Use one of the mechanistic models
## get D0 values.
ml7 <- modlist(reps, model = mak3)
sapply(ml7, function(x) coef(x)[1])

## Exclude first sample in each
## replicate group of dataset 'reps'.
ml8 <- modlist(reps, exclude = ".1")
plot(ml8, which = "single")

## Using weighted fitting:
## weighted by inverse residuals.
ml9 <- modlist(reps, weights = "1/abs(resid)")
plot(ml9, which = "single")
```

```
## Use linear model of the first 10
## cycles for baselining.
ml10 <- modlist(reps, basecyc = 1:10, baseline = "lin")
plot(ml10)

## Use a single value for baselining.
ml11 <- modlist(reps, basecyc = 1:10, baseline = 0.5)
plot(ml11)
```

`mselect` *Sigmoidal model selection by different criteria*

Description

Model selection by comparison of different models using

1) the maximum log likelihood value,
2) Akaike's Information Criterion (AIC),
3) bias-corrected Akaike's Information Criterion (AICc),
4) the estimated residual variance,
5) the p-value from a nested F-test on the residual variance,
6) the p-value from the likelihood ratio,
7) the Akaike weights based on AIC,
8) the Akaike weights based on AICc, and
9) the reduced chi-square, χ^2_ν, if replicates exist.

The best model is chosen by 5), 6), 8) or 9) and returned as a new model.

Usage

```
mselect(object, fctList = NULL, sig.level = 0.05, verbose = TRUE,
        crit = c("ftest", "ratio", "weights", "chisq"), do.all = FALSE, ...)
```

Arguments

`object`	an object of class 'pcrfit' or 'replist'.
`fctList`	a list of functions to be analyzed, i.e. for a non-nested regime. Should also contain the original model.
`sig.level`	the significance level for the nested F-test.
`verbose`	logical. If `TRUE`, the result matrix is displayed in the console.
`crit`	the criterium for model selection. Either `"ftest"/"ratio"` for nested models or `"weights"/"fitprob"` for nested and non-nested models.
`do.all`	if `TRUE`, all available sigmoidal models are tested and the best one is selected based on AICc weights.
`...`	other parameters to be passed to `fitchisq`.

Details

Criteria 5) and 6) cannot be used for comparison unless the models are nested. Criterion 8), Akaike weights, can be used for nested and non-nested regimes, which also accounts for the reduced χ^2_ν. For criterion 1) the larger the better. For criteria 2), 3) and 4): the smaller the better. The best model is chosen either from the nested F-test (anova), likelihood ratio (llratio), corrected Akaike weights (akaike.weights) or reduced χ^2_ν (fitchisq) and returned as a new model. When using "ftest"/"ratio" the corresponding nested functions are analyzed automatically, i.e. b3/b4/b5/b6/b7; l3/l4/l5/l6/l7. If supplying nested models, please do this with ascending number of parameters.

Value

A model of the best fit selected by one of the criteria above. The new model has an additional list item 'retMat' with a result matrix of the criterion tests.

Author(s)

Andrej-Nikolai Spiess

See Also

llratio, akaike.weights and fitchisq.

Examples

```
## Choose best model based on F-tests
## on the corresponding nested models.
m1 <- pcrfit(reps, 1, 2, l4)
m2 <- mselect(m1)
summary(m2)  ## Converted to l7 model!

## Use Akaike weights on non-nested models
## compare to original model.
m2 <- mselect(m1, fctList = list(l4, b5, cm3), crit = "weights")
summary(m2) ## Converted to b5 model!

## Try all sigmoidal models.
m3 <- pcrfit(reps, 1, 20, l4)
mselect(m3, do.all = TRUE) ## l7 wins by far!

## On replicated data using reduced chi-square.
ml1 <- modlist(reps, fluo = 2:5, model = l4)
rl1 <- replist(ml1, group = c(1, 1, 1, 1))
mselect(rl1, crit = "chisq")  ## converted to l6!
```

parKOD *Parameters that can be changed to tweak the kinetic outlier methods*

Description

A control function with different list items that change the performance of the different (kinetic) outlier functions as defined in KOD.

Usage

```
parKOD(eff = c("sliwin", "sigfit", "expfit"), train = TRUE,
       alpha = 0.05, cp.crit = 10, cut = c(-6, 2))
```

Arguments

eff	**uni1**. The efficiency method to be used. Either sliwin, sigfit or expfit.
train	**uni1**. If TRUE, the sample's efficiency is NOT included in the calculation of the average efficiency (default), if FALSE it is.
cp.crit	**uni2**. The cycle difference between first and second derivative maxima, default is 10.
cut	**multi1**. A 2-element vector defining the lower and upper border from the first derivative maximum from where to cut the complete curve.
alpha	the p-value cutoff value for all implemented statistical tests.

Details

For more details on the function of the parameters within the different kinetic and sigmoidal outlier methods, see KOD.

Value

If called, returns a list with the parameters as items.

Author(s)

Andrej-Nikolai Spiess

Examples

```
## Multivariate outliers,
## adjusting the 'cut' parameter.
ml1 <- modlist(reps, 1, 2:5, model = l5)
res1 <- KOD(ml1, method = "multi1", par = parKOD(cut = c(-5, 2)))
```

pcrbatch *Batch calculation of qPCR efficiency and other qPCR parameters*

Description

This function batch calculates the results obtained from efficiency, sliwin, expfit, LRE or the coefficients from any of the makX/cm3 models on a dataframe containing many qPCR runs. The input can also be a list obtained from modlist, which simplifies things in many cases. The output is a dataframe with the estimated parameters and model descriptions. Very easy to use on datasheets containing many qPCR runs, i.e. as can be imported from Excel. The result is automatically copied to the clipboard.

Usage

```
pcrbatch(x, cyc = 1, fluo = NULL,
         methods = c("sigfit", "sliwin", "expfit", "LRE"),
         model = l4, check = "uni2", checkPAR = parKOD(),
         remove = c("none", "fit", "KOD"), exclude = NULL,
         type = "cpD2", labels = NULL, norm = FALSE,
         baseline = c("none", "mean", "median", "lin", "quad", "parm"),
         basecyc = 1:8, basefac = 1, smooth = NULL, smoothPAR = NULL,
        factor = 1, opt = FALSE, optPAR = list(sig.level = 0.05, crit = "ftest"),
         group = NULL, names = c("group", "first"), plot = TRUE,
         verbose = TRUE, ...)
```

Arguments

x	a dataframe containing the qPCR raw data from the different runs or a list obtained from modlist.
cyc	the column containing the cycle data. Defaults to first column.
fluo	the column(s) (runs) to be analyzed. If NULL, all runs will be considered.
methods	a character vector defining the methods to use. See 'Details'.
model	the model to be used for all runs.
check	the method for outlier detection in KOD. Default is check for sigmoidal structure.
checkPAR	parameters to be supplied to the check method.
remove	which runs to remove. Either none, those which failed to fit or from the outlier methods.
exclude	either "" for samples with missing column names or a regular expression defining columns (samples) to be excluded from pcrbatch. See 'Details' and 'Examples' in modlist.
type	the point on the amplification curve from which the efficiency is estimated. See efficiency.
labels	a vector containing labels, i.e. for defining replicate groups prior to ratiobatch.
norm	logical. Should the raw data be normalized within [0, 1] before model fitting?

baseline	type of baseline subtraction. See 'Details' in modlist.
basecyc	cycle range to be used for baseline subtraction, i.e. 1:5.
basefac	a factor when using averaged baseline cycles, such as 0.95.
smooth	which curve smoothing method to use. See modlist.
smoothPAR	parameters to be supplied to the smoothing functions, supplied as a list. See modlist.
factor	a multiplication factor for the fluorescence response values (barely useful, but who knows...).
opt	logical. Should model selection be applied to each model?
optPAR	parameters to be supplied to mselect.
group	a vector containing the grouping for possible replicates.
names	how to name the grouped fit. Either 'group_1, ...' or the first name of the replicates.
plot	logical. If TRUE, the single runs are plotted from the internal 'modlist' for diagnostics.
verbose	logical. If TRUE, fitting and tagging results will be displayed in the console.
...	other parameters to be passed to downstream methods.

Details

The methods vector is used for defining the different methods from which pcrbatch will concatenate the results. The mechanistic models (mak2, mak2i, mak3, mak3i, cm3) are omitted by default, because fitting is time-expensive. If they should be included, just add "mak3" to methods. See 'Examples'. The qPCR raw data should be arranged with the cycle numbers in the first column with the name "Cycles". All subsequent columns must be plain raw data with sensible column descriptions. If replicates are defined by group, the output will contain a numbering of groups (i.e. "group_1" for the first replicate group). The model selection process is optional, but we advocate using this for obtaining better parameter estimates. Normalization has been described to improve certain qPCR analyses, but this has still to be independently evaluated. Background subtraction is done as in modlist and efficiency. In case of unsuccessful model fitting or lack of sigmoidal structure, the names are tagged by *NAME* or **NAME**, respectively (if remove = "none"). However, if remove = "fit" or remove = "KOD", the failed runs are excluded from the output. Similar to modlist, if a labels vector lab is supplied, the labels from the failed fits are removed and a new label vector lab_mod is written to the global environment.

Value

A dataframe with the results in columns containing the calculated values, fit parameters and (tagged) model name together with the different methods used as the name prefix. A plot shows a plot matrix of all amplification curves/sigmoidal fits and failed amplifications marked with asterisks.

Note

IMPORTANT: When subsequent use of ratiocalc is desired, use pcrbatch on the single run level with group = NULL and remove = "none", so that ratiocalc can automatically delete the failed runs from its group definition. Otherwise error propagation will fail.

Author(s)

Andrej-Nikolai Spiess

See Also

The function `modlist` for creating a list of models, which is used internally by `pcrbatch`.

Examples

```
## First 4 runs and return parameters of fit
## do background subtraction using mean of the first 5 cycles.
pcrbatch(reps, fluo = 2:5, baseline = "mean", basecyc = 1:5)


##  First 8 runs, with 4 replicates each, l5 model.
pcrbatch(reps, fluo = 2:9, model = l5, group = c(1,1,1,1,2,2,2,2))

## Using model selection (Akaike weights)
## on the first 4 runs, runs 1 and 2 are replicates.
pcrbatch(reps, fluo = 2:5, group = c(1,1,2,3),
         opt = TRUE, optPAR = list(crit = "weights"))

## Fitting a sigmoidal and 'mak3' mechanistic model.
pcrbatch(reps, methods = c("sigfit", "mak3"))

## Converting a 'modlist' to 'pcrbatch'.
ml5 <- modlist(reps, 1, 2:5, b5)
res5 <- pcrbatch(ml5)

## Using Whittaker smoothing.
pcrbatch(reps, smooth = "whit")
```

---

pcrboot *Bootstrapping and jackknifing qPCR data*

Description

Confidence intervals for the estimated parameters and goodness-of-fit measures are calculated for a nonlinear qPCR data fit by either
a) boostrapping the residuals of the fit or
b) jackknifing and refitting the data.

Confidence intervals can also be calculated for all parameters obtained from the `efficiency` analysis.

Usage

```
pcrboot(object, type = c("boot", "jack"), B = 100, njack = 1,
        plot = TRUE, do.eff = TRUE, conf = 0.95, verbose = TRUE, ...)
```

Arguments

`object`	an object of class 'pcrfit'.
`type`	either `bootstrapping` or `jackknifing`.
`B`	numeric. The number of iterations.
`njack`	numeric. In case of `type = "jack"`, how many datapoints to exclude. Defaults to leave-one-out.
`plot`	should the fitting and final results be displayed as a plot?
`do.eff`	logical. If `TRUE`, `efficiency` analysis will be performed.
`conf`	the confidence level.
`verbose`	logical. If `TRUE`, the iterations will be printed on the console.
`...`	other parameters to be passed on to the plotting functions.

Details

Non-parametric bootstrapping is applied using the centered residuals.
1) Obtain the residuals from the fit:

$$\hat{\varepsilon}_t = y_t - f(x_t, \hat{\theta})$$

2) Draw bootstrap pseudodata:

$$y_t^* = f(x_t, \hat{\theta}) + \epsilon_t^*$$

where ϵ_t^* are i.i.d. from distribution $\hat{F}$, where the residuals from the original fit are centered at zero.
3) Fit $\hat{\theta}^*$ by nonlinear least-squares.
4) Repeat *B* times, yielding bootstrap replications

$$\hat{\theta}^{*1}, \hat{\theta}^{*2}, \ldots, \hat{\theta}^{*B}$$

One can then characterize the EDF and calculate confidence intervals for each parameter:

$$\theta \in [EDF^{-1}(\alpha/2), EDF^{-1}(1-\alpha/2)]$$

The jackknife alternative is to perform the bootstrap on the data-predictor vector, i.e. eliminating a certain number of datapoints.
If the residuals are correlated or have non-constant variance the latter is recommended. This may be the case in qPCR data, as the variance in the low fluorescence region (ground phase) is usually much higher than in the rest of the curve.

Value

A list containing the following items:

`ITER`	a list containing each of the results from the iterations.
`CONF`	a list containing the confidence intervals for each item in `ITER`.

Each item contains subitems for the coefficients (`coef`), root-mean-squared error (`rmse`), residual sum-of-squares (`rss`), goodness-of-fit measures (`gof`) and the efficiency analysis (`eff`). If `plot = TRUE`, all data is plotted as boxplots including confidence intervals.

Author(s)

Andrej-Nikolai Spiess

References

Nonlinear regression analysis and its applications.
Bates DM & Watts DG.
Wiley, Chichester, UK, 1988.

Nonlinear regression.
Seber GAF & Wild CJ.
Wiley, New York, 1989.

Boostrap accuracy for non-linear regression models.
Roy T.
J Chemometics (1994), **8**: 37-44.

Examples

```
## Simple bootstrapping with
## too less iterations...
m1 <- pcrfit(reps, 1, 2, l4)
pcrboot(m1, B = 20)

## Jackknifing with leaving
## 5 datapoints out.
m2 <- pcrfit(reps, 1, 2, l4)
pcrboot(m2, type = "jack", njack = 5, B = 20)
```

`pcrfit` *Workhorse function for qPCR model fitting*

Description

This is the workhorse function of the qpcR package that fits one of the available models to qPCR data using (weighted) nonlinear least-squares (Levenberg-Marquardt) fitting from `nlsLM` of the 'minpack.lm' package.

Usage

```
pcrfit(data, cyc = 1, fluo, model = l4, start = NULL,
       offset = 0, weights = NULL, verbose = TRUE, ...)
```

Arguments

data	the name of the dataframe containing the qPCR runs.
cyc	the column containing the cycle data. Defaults to 1.
fluo	the column(s) containing the raw fluorescence data of the run(s). If more than one column is given, the model will be built with the replicates. See 'Details' and 'Examples'.
model	the model to be used for the analysis. Defaults to 'l4'.
start	a vector of starting values that can be supplied externally.
offset	an offset cycle number from the second derivative cut-off cycle for all MAK methods. See 'Details' and 'Example'.
weights	a vector with same length as data containing possible weights for the nonlinear fit, or an expression to calculate weights from. See 'Details'.
verbose	logical. If TRUE, fitting and convergence results will be displayed in the console.
...	other parameters to be passed to nlsLM.

Details

This is a newer (from qpcR 1.3-7 upwards) version of pcrfit. It is a much simpler implementation containing only the LM-Algorithm for fitting, but this fitting routine has proven to be so robust that other optimization routines (such as in optim) could safely be removed. The fitting is done with the new nlsLM function of the 'minpack.lm' package, which gives a model of class 'nls' as output.

This function is to be used at the single run level or on replicates (by giving several columns). The latter will build a single model based on the replicate values. If many models should be built on a cohort of replicates, use modlist and replist.

The offset value defines the offset cycles from the second derivative maximum that is used as a cut-off criterion in the MAK methods. See 'Examples'.

Since version 1.3-7, an expression given as a character string can be supplied to the weights argument. This expression, which is transferred to qpcR:::wfct, defines how the vector of weights is calculated from the data. In principle, five different parameters can be used to define weights:
"x" relates to the cycles x_i,
"y" relates to the raw fluorescence values y_i,
"error" relates to the error $\sigma(y_{i,j})$ of replicates j,
"fitted" relates to the fitted values $\hat{y}_i$ of the fit,
"resid" relates to the residuals $y_i - \hat{y}_i$ of the fit.
For "fitted" and "resid", the model is fit unweighted by pcrfit, the fitted/residual values extracted and these subsequently used for refitting the model with weights. These parameters can be used solely or combined to create a weights vector for different regimes. The most commonly used are (see also 'Examples'):
Inverse of response (raw fluorescence) $\frac{1}{y_i}$: "1/y"
Square root of predictor (Cycles) $\sqrt{x_i}$: "sqrt(x)"
Inverse square of fitted values: $\frac{1}{\hat{y}_i^2}$: "1/fitted^2"
Inverse variance $\frac{1}{\sigma^2(y_{i,j})}$: "1/error^2"

Value

A model of class 'nls' and 'pcrfit' with the following items attached:

DATA	the initial data used for fitting.
MODEL	the model used for fitting.
call2	the call to pcrfit.
parMat	the trace of the parameter values. Can be used to track problems.
opt.method	defaults to "LM".

Author(s)

Andrej-Nikolai Spiess

References

Bioassay analysis using R.
Ritz C & Streibig JC.
J Stat Soft (2005), **12**: 1-22.

A Method for the Solution of Certain Problems in Least Squares.
K. Levenberg.
Quart Appl Math (1944), **2**: 164-168.

An Algorithm for Least-Squares Estimation of Nonlinear Parameters.
D. Marquardt.
SIAM J Appl Math (1963), **11**: 431-441.

Examples

```
## Simple l4 fit of F1.1 of the 'reps' dataset.
m1 <- pcrfit(reps, 1, 2, l4)
plot(m1)

## Supply own starting values.
pcrfit(reps, 1, 2, l4, start = c(-5, -0.05, 11, 16))

## Make a replicate model,
## use inverse variance as weights.
m2 <- pcrfit(reps, 1, 2:5, l5, weights = "1/error^2")
plot(m2)

## Fit a mechanistic 'mak2' model
## to -1 cycle from SDM.
m3 <- pcrfit(reps, 1, 2, mak2, offset = -1)
plot(m3)
```

pcrGOF *Summarize measures for the goodness-of-fit*

Description

Calculates all implemented measures for the goodness-of-fit and returns them as a list. Works for objects of class `pcrfit`, `lm`, `glm`, `nls`, `drc` and many others...

Usage

```
pcrGOF(object, PRESS = FALSE)
```

Arguments

`object`	a fitted object.
`PRESS`	logical. If `TRUE`, the more calculation intensive P^2 is also returned.

Value

A list with all implemented Information criteria (`AIC`, `AICc`, `BIC`), residual variance, root-mean-squared-error, the reduced χ^2_ν from `fitchisq` (if replicates) and the `PRESS` P^2 value (if `PRESS = TRUE`).

Author(s)

Andrej-Nikolai Spiess

Examples

```
## Single fit without replicates
## including PRESS statistic.
m1 <- pcrfit(reps, 1, 2, l5)
pcrGOF(m1, PRESS = TRUE)

## Fit containing replicates:
## calculation of reduced
## chi-square included!
m2 <- pcrfit(reps, 1, 2:5, l5)
pcrGOF(m2)
```

pcropt1 *Combinatorial elimination of plateau and ground phase cycles*

Description

The estimation of PCR efficiency and calculation of initial fluorescence F_0 is analyzed by refitting the (optimized) model on subsets of the data, thereby using all possible combinations of datapoints. The estimated parameters are then collated in a dataframe. This is intended to be the prerequisite for finding the optimal datapoints that minimize the fit or exhibit the best correlation to a calibration curve. This approch is an extension to the method described in Rutledge *et al*. (2004). The result of any collected parameter can then be displayed by a rank-colored bubbleplot. See 'Examples'.

Usage

```
pcropt1(object, fact = 3, opt = FALSE, plot = TRUE, bubble = NULL, ...)
```

Arguments

`object`	an object of class 'pcrfit'.
`fact`	numeric. The multiplier for the scan border. See 'Details'.
`opt`	logical. If `true`, model selection is applied for each combination of cycles. Beware: Slow!!
`plot`	logical. If `TRUE`, the iterative plotting is displayed, which makes the method a bit slower.
`bubble`	either `NULL` for no bubble plot or any parameter (given as a character vector) in the result matrix to be displayed as a bubble plot. See 'Examples'.
`...`	other parameters to be passed on to `efficiency`, `mselect` or `qpcR:::bubbleplot`.

Details

It has been shown by Rutledge (2004) that the estimation of PCR efficiency gives more realistic values when the number of plateau cycles are decreased. This paradigm is the basis for this function, but we also consider the cycles in the ground phase and all combinations between ground/plateau cycles. All datapoints between the lower border cpD1 - `fact` * (cpD1 - cpD2) and upper border cpD1 + `fact` * (cpD1 - cpD2) are cycled through.

Value

A matrix with the selected border values, goodness-of-fit measures as obtained from `pcrGOF` and efficiency and F_0 values from `efficiency`.

Author(s)

Andrej-Nikolai Spiess

References

Sigmoidal curve fitting redefines quantitative real-time PCR with the prospective of developing automated high-throughput applications.
Rutledge RG.
Nucleic Acids Research (2004), **32**: e178.

Examples

```
## Optimize fit and display bubbleplot of R-square.
m1 <- pcrfit(reps, 1, 2, l4)
res1 <- pcropt1(m1, plot = FALSE, bubble = "Rsq")
```

`pcrsim` *Simulation of sigmoidal qPCR data with goodness-of-fit analysis*

Description

Simulated sigmoidal qPCR curves are generated from an initial model to which some user-defined homoscedastic or heteroscedastic noise is added. One or more models can then be fit to this random data and goodness-of-fit (GOF) measures are calculated for each of the models. This is essentially a Monte-Carlo approach testing for the best model in dependence to some noise structure in sigmodal models.

Usage

```
pcrsim(object, nsim = 100, error = 0.02,
       errfun = function(y) 1, plot = TRUE,
       fitmodel = NULL, select = FALSE,
       statfun = function(y) mean(y, na.rm = TRUE),
       PRESS = FALSE, ...)
```

Arguments

`object`	an object of class 'pcrfit.
`nsim`	the number of simulated curves.
`error`	the gaussian error used for the simulation. See 'Details'.
`errfun`	an optional function for the error distribution. See 'Details'.
`plot`	should the simulated and fitted curves be displayed?
`fitmodel`	a model or model list to test against the initial model.
`select`	if `TRUE`, a matrix is returned with the best model in respect to each of the GOF measures.
`statfun`	a function to be finally applied to all collected GOF measures, default is the average.
`PRESS`	logical. If set to `TRUE`, the computationally expensive `PRESS` statistic will be calculated.
`...`	other parameters to be passed on to `plot` or `pcrfit`.

Details

The value defined under `error` is just the standard deviation added plainly to each y value from the initial model, thus generating a dataset with homoscedastic error. With aid of `errfun`, the distribution of the error along the y values can be altered and be used to generate heteroscedastic error along the curve, i.e. as a function of the magnitude.

Example:
`errfun = function(y) 1`
same variance for all y, as is.

`errfun = function(y) y`
variance as a function of the y-magnitude.

`errfun = function(y) 1/y`
variance as an inverse function of the y-magnitude.

For the effect, see 'Examples'.

Value

A list containing the following items:

`cyc`	same as in 'arguments'.
`fluoMat`	a matrix with the simulated qPCR data in columns.
`coefList`	a list with the coefficients from the fits for each model, as subitems.
`gofList`	a list with the GOF measures for each model, as subitems.
`statList`	a list with the GOF measures summarized by `statfun` for each model, as subitems.
`modelMat`	if `select = TRUE`, a matrix with the best model for each GOF measure and each simulation.

Author(s)

Andrej-Nikolai Spiess

Examples

```
## Generate initial model.
m1 <- pcrfit(reps, 1, 2, l4)

## Simulate homoscedastic error
## and test l4 and l5 on data.
res1 <- pcrsim(m1, error = 0.2, nsim = 20,
               fitmodel = list(l4, l5))


## Use heteroscedastic noise typical for
## qPCR: more noise at lower fluorescence.
res2 <- pcrsim(m1, error = 0.01, errfun = function(y) 1/y,
              nsim = 20, fitmodel = list(l4, l5, l6))
```

```
## Get 95% confidence interval for
## the models GOF in question (l4, l5, l6).
res3 <- pcrsim(m1, error = 0.2, nsim = 20, fitmodel = list(l4, l5, l6),
               statfun = function(y) quantile(y, c(0.025, 0.975)))
res3$statList

## Count the selection of the 'true' model (l4)
## for each of the GOF measures,
## use PRESS statistic => SLOW!
## BIC wins!!
res4 <- pcrsim(m1, error = 0.05, nsim = 10, fitmodel = list(l4, l5, l6),
                select = TRUE, PRESS = TRUE)
apply(res4$modelMat, 2, function(x) sum(x == 1))
```

`plot.pcrfit` *Plotting qPCR data with fitted curves*

Description

A plotting function for data of class 'pcrfit' (single curves) or 'modlist' (batch curves) displaying the data points and the fitted curve. Four different plot types are available, namely plotting all curves in a 2D graph, a 2D plot matrix, a 3D graph or a heatmap-like image plot.

Usage

```
## S3 method for class 'pcrfit'
plot(x, type = c("all", "single", "3D", "image"),
        fitted = TRUE, add = FALSE, col = NULL, par2D = list(),
        par3D = list(), ...)
```

Arguments

`x`	an object of class 'pcrfit' or 'modlist'.
`type`	plots all curves in 2D (`"all"`), a plot matrix with many curves (`"single"`), a 3D plot (`"3D"`) or a heatmap-like image plot (`image`).
`fitted`	should the fitted lines be displayed?
`add`	should the curve be added to an existing plot?
`col`	an optional color vector for the individual curves. Is recycled to the number of runs in `x`.
`par2D`	a list containing graphical parameters to change the 2D-plots: `plot`, `points` or `lines`.
`par3D`	a list containing graphical parameters to change the 3D-plot: `plot3d`, `points3d`, `lines3d`, `axis3d` or `mtext3d`.
`...`	other parameters for downstream methods.

Details

Uses the 'rgl' package for 3D plots. If the 'modlist' contains runs that failed to fit, these are displayed with RED asterisked names. In addition, sigmoidal outlier runs will be displayed in BLUE with double asterisked names. This approach makes the identification of failed runs easy and works only with `type = "single"`. See 'Examples'.
For high-throughput data, the user of this function is encouraged to use the `"image"` kind of plot, as one can see quite nicely the differences in the amplification profiles of several hundred runs. Of course, this plot type does not display the fitted curve. See 'Examples'.

Value

A 2D, multiple 2D, 3D or heatmap-like qPCR plot.

Author(s)

Andrej-Nikolai Spiess

Examples

```
## Single plot.
m1 <- pcrfit(reps, 1, 2, l5)
plot(m1)

## Add another plot in blue.
m2 <- pcrfit(reps, 1, 12, l5)
plot(m2, add = TRUE, col = 4)

## Plot a 'modlist' batch with coloring of replicates.
ml1 <- modlist(reps, 1, 2:13, model = l4)
plot(ml1, col = gl(3,4))

## Subset of cycle range.
plot(ml1, col = rep(1:3, each = 4),
     par2D = list(xlim = c(10, 30)))

## Plot single curves for diagnostics.
plot(ml1, type = "single", col = rep(1:3, each = 4))

## 3D plots of 'modlist's.
plot(ml1, type = "3D", col = rep(1:3, each = 4))
close3d()


## Example for "image" type when
## using large data.
ml2 <- modlist(vermeulen2)
plot(ml2, type = "image")

## Example for outlier identification:
## RED/*name* indicates failed fitting,
## BLUE/**name** indicates sigmoidal outlier
```

```
## using 'testdat' set.
ml3 <- modlist(testdat, model = l5)
plot(ml3, type = "single")
```

`predict.pcrfit` *Value prediction from a fitted qPCR model*

Description

After fitting the appropriate model, either the raw fluorescence values can be predicted from the cycle number or *vice versa*.

Usage

```
## S3 method for class 'pcrfit'
predict(object, newdata, which = c("y", "x"),
        interval = c("none", "confidence", "prediction"),
        level = 0.95, ...)
```

Arguments

`object`	an object of class 'pcrfit'.
`newdata`	a dataframe containing the values to estimate from, using the same variable naming as in the fitted model.
`which`	either `"y"` (default) for prediction of the raw fluorescence or `"x"` for prediction of the cycle number.
`interval`	if not `"none"`, confidence or prediction intervals are calculated.
`level`	the confidence level.
`...`	some methods for this generic require additional arguments. None are used in this method.

Details

y-values (Fluorescence) are estimated from `object$MODEL$expr`, x-values (Cycles) are estimated from `object$MODEL$inv`. Confidence intervals are calculated from the gradient of the function and the variance-covariance matrix of `object` by $\nabla f(x) \cdot cov(y) \cdot \nabla f(x)$ and are based on asymptotic normality (t-distribution).

Value

A dataframe containing the estimated values and (if chosen) standard error/upper confidence limit/lower confidence limit. The gradient is attached to the dataframe and can be accessed with `attr`.

Note

The estimation of x (cycles) from fluorescence data if `which = "x"` is problematic in the asymptotic regions of the sigmoidal curves (often gives NaN, due to logarithmation of negative values) and works fairly well in the ascending part.

Author(s)

Andrej-Nikolai Spiess

Examples

```
m1 <- pcrfit(reps, 1, 2, l5)

## Which raw fluorescence value at cycle number = 17?
predict(m1, newdata = data.frame(Cycles = 17))

## Cycle numbers 20:25, with 95% confidence?
predict(m1, newdata = data.frame(Cycles = 20:25), interval = "confidence")

## Which cycle at Fluo = 4, with 95% prediction?
predict(m1, newdata = data.frame(Fluo = 4), which = "x", interval = "prediction")
```

PRESS *Allen's PRESS (Prediction Sum-Of-Squares) statistic, aka P-square*

Description

Calculates the PRESS statistic, a leave-one-out refitting and prediction method, as described in Allen (1971). Works for any regression model with a `call` slot, an `update` and a `predict` function, hence all models of class `lm`, `glm`, `nls` and `drc` (and maybe more...). The function also returns the PRESS analog to R-square, the P-square.

Usage

```
PRESS(object, verbose = TRUE)
```

Arguments

`object`	a fitted model.
`verbose`	logical. If `TRUE`, iterations are displayed on the console.

Details

From a fitted model, each of the predictors $x_i, i = 1 \dots n$ is removed and the model is refitted to the $n - 1$ points. The predicted value $\hat{y}_{i,-i}$ is calculated at the excluded point x_i and the PRESS statistic is given by:

$$\sum_{i=1}^{n} (y_i - \hat{y}_{i,-i})^2$$

The PRESS statistic is a surrogate measure of crossvalidation of small sample sizes and a measure for internal validity. Small values indicate that the model is not overly sensitive to any single data point. The P-square value, the PRESS equivalent to R-square, is given by

$$P^2 = \frac{\sum_{i=1}^{n} \hat{\varepsilon}_{-i}^2}{\sum_{i=1}^{n} (y_i - \bar{y})^2}$$

with $\hat{\varepsilon}_{-i} = y_i - \hat{y}_{-i}$.

Value

A list with the following components:

`stat`	The PRESS statistic.
`residuals`	a vector containing the PRESS residuals for each x_i.
`P.square`	the P-square value. See 'Details'.

Note

There is also a `PRESS` function in library 'MPV' that works solely for `lm` models using the hat matrix.

Author(s)

Andrej-Nikolai Spiess

References

The relationship between variable selection and data augmentation and a method for prediction.
Allen DM.
Technometrics (1974), **16**: 25-127.

The Prediction Sum of Squares as a Criterion for Selecting Predictor Variables.
Allen DM.
Technical Report Number 23 (1971), Department of Statistics, University of Kentucky.

Classical and Modern Regression with Applications.
Myers RH.
Second Edition (1990), Duxbury Press (PWS-KENT Publishing Company), 299-304.

Examples

```
## Example for PCR analysis.
m1 <- pcrfit(reps, 1, 2, l7)
PRESS(m1)

## Compare PRESS statistic in models
## with fewer parameters.
m2 <- pcrfit(reps, 1, 2, l5)
PRESS(m2)
m3 <- pcrfit(reps, 1, 2, l4)
PRESS(m3)
```

```
## Example for linear regression.
x <- 1:10
y <- rnorm(10, x, 0.1)
mod <- lm(y ~ x)
PRESS(mod)

## Example for NLS fitting.
DNase1 <- subset(DNase, Run == 1)
fm1DNase1 <- nls(density ~ SSlogis(log(conc), Asym, xmid, scal), DNase1)
res <- PRESS(fm1DNase1)

## PRESS residuals plot.
barplot(res$residuals)
```

`propagate` *Error propagation using different methods*

Description

A general function for the calculation of error propagation by Monte Carlo simulation, permutation and first/second-order Taylor expansion including covariances. Can be used for qPCR data, but any data that should be subjected to error propagation analysis will do. The different methods can be used for any expression based on either replicate or summary data (mean & standard deviation).

Usage

```
propagate(expr, data, type = c("raw", "stat"), second.order = TRUE,
          do.sim = FALSE, dist.sim = c("norm", "tnorm"), use.cov = FALSE,
          nsim = 10000, do.perm = FALSE, perm.crit = NULL, ties = NULL,
          nperm = 2000, alpha = 0.05, plot = TRUE, logx = FALSE,
          verbose = FALSE, ...)
```

Arguments

`expr`	an expression, such as `expression(x/y)`.
`data`	a dataframe or matrix containing either a) the replicates in columns or b) the means in the first row and the standard deviations in the second row. The variable names must be defined in the column headers.
`type`	either `raw` if replicates are given, or `stat` if means and standard deviations are supplied.
`second.order`	logical. If `TRUE`, error propagation will be calculated with first AND second-order Taylor expansion. See 'Details'.
`do.sim`	logical. Should Monte Carlo simulation be applied?
`dist.sim`	`"norm"` will use a multivariate normal distribution, `"tnorm"` a multivariate truncated normal distribution. See 'Details'.

use.cov	logical or variance-covariance matrix with the same column descriptions and column order as data. If TRUE together with replicates, the covariances are calculated from these and used within Monte Carlo simulation and error propagation. If type = "stat", a square variance-covariance matrix can be supplied in the right dimensions (n x n, n = number of variables). If FALSE, Monte Carlo simulation and error propagation use only the diagonal (variances).
nsim	the number of simulations to be performed, minimum is 5000.
do.perm	logical. Should permutation error analysis be applied?
perm.crit	a character string of one or more criteria defining the null hypothesis for the permutation p-value. See 'Details'.
ties	a vector defining the columns that should be tied together for the permutations. See 'Details'.
nperm	the number of permutations to be performed.
alpha	the confidence level.
plot	logical. Should histograms with confidence intervals (in blue) be plotted for all methods?
logx	logical. Should the x-axis of the graphs have logarithmic scale?
verbose	logical. If TRUE, a longer output is given including the simulated data, derivatives, covariance matrix, etc.
...	other parameters to be supplied to hist, boxplot or abline.

Details

The implemented methods are:

1) **Monte Carlo simulation:**
For each variable in data, simulated data with nsim samples is generated from a multivariate (truncated) normal distribution using mean μ and standard deviation σ of each variable. All data is coerced into a new dataset that has the same covariance structure as the initial data. Each row of the simulated dataset is evaluated and summary statistics are calculated. In scenarios that are nonlinear in nature the distribution of the result values can be skewed, mainly due to the simulated values at the extreme end of the normal distribution. Setting dist.sim = "tnorm" will fit a multivariate normal distribution, calculate the lower/upper 2.5% quantile on each side for each input variable and use these as bounds for simulating from a multivariate truncated normal distribution. This will (in part) remove some of the skewness in the result distribution.

2) **Permutation approach:**
The original data is permutated nperm times by binding observations together according to ties. The ties bind observations that can be independent measurements from the same sample. In qPCR terms, this would be a real-time PCR for two different genes on the same sample. If ties are omitted, the observations are shuffled independently. In detail, two datasets are created for each permutation: Dataset1 samples the rows (replicates) of the data according to ties. Dataset2 is obtained by sampling the columns (samples), also binding columns as defined in ties. For both datasets, the permutations are evaluated and statistics are collected. A confidence interval is calculated from all evaluations of Dataset1. A p-value is calculated from all permutations that follow perm.crit, whereby init reflects the permutations of the initial data and perm the permutations

of the randomly reallocated samples. Thus, the p-value gives a measure against the null hypothesis that the result in the initial group is just by chance. See also 'Examples'.
The criterion for the permutation p-value (`perm.crit`) has to be defined by the user. For example, let's say we calculate some value 0.2 which is a ratio between two groups. We would hypothesize that by randomly reallocating the values between the groups, the mean values are not equal or smaller than in the initial data. We would thus define `perm.crit` as "perm < init" meaning that we want to test if the mean of the initial data (`init`) is frequently smaller than by the randomly allocated data (`perm`). The default (`NULL`) is to test all three variants "perm > init", "perm == init" and "perm < init".

3) **Error propagation:**
The propagated error is calculated by first and second-order Taylor expansion using matrix algebra. Often omitted, but important in models where the variables are correlated, is the second covariance term:

$$\sigma_y^2 = \underbrace{\sum_{i=1}^{N} \left(\frac{\partial f}{\partial x_i}\right)^2 \sigma_i^2}_{\text{variance}} + \underbrace{2 \sum_{\substack{i=1 \\ i \neq j}}^{N} \sum_{\substack{j=1 \\ j \neq i}}^{N} \left(\frac{\partial f}{\partial x_i}\right) \left(\frac{\partial f}{\partial x_j}\right) \sigma_{ij}}_{\text{covariance}}$$

`propagate` calculates the propagated error either with or without covariances using matrix algebra for first- and second-order (since version 1.3-8) Taylor expansion.
First-order:
$\mathrm{E}(y) = f(\bar{x}_i)$
$\mathbf{C}_y = \nabla_x \mathbf{C}_x \nabla_x^T$
Second-order:
$\mathrm{E}(y) = f(\bar{x}_i) + \frac{1}{2}[tr(\mathbf{H}_{xx}\mathbf{C}_x)]$
$\mathbf{C}_y = \nabla_x \mathbf{C}_x \nabla_x^T + \frac{1}{2}[tr(\mathbf{H}_{xx}\mathbf{C}_x\mathbf{H}_{xx}\mathbf{C}_x)]$

with $\mathrm{E}(y)$ = expectation of y, $\mathbf{C}_y$ = variance of y, ∇_x = the p x n gradient matrix with all partial first derivatives, $\mathbf{C}_x$ = the p x p covariance matrix, $\mathbf{H}_{xx}$ the Hessian matrix with all partial second derivatives and $tr(\cdot)$ = the trace (sum of diagonal) of the matrix. For a detailed derivation, see 'References'.
The second-order Taylor expansion (`second.order = TRUE`) corrects for bias in nonlinear expressions as the first-order Taylor expansion assumes linearity around $\bar{x}_i$.
Depending on the input formula, the error propagation may result in an error that is not normally distributed. The Monte Carlo simulation, starting with normal distributions of the variables, can clarify this. A high tendency from deviation of normality is encountered in formulas in which the error of the denominator is relatively high or in exponential models where the exponent has a high error. This is one of the problems that is inherent in real-time PCR analysis, as the classical ratio calculation with efficiencies (i.e. by the delta-ct method) is of an exponential type.

Value

A plot containing histograms of the Monte Carlo simulation, the permutation values and error propagation. Additionally inserted are a boxplot, median values in red and confidence intervals as blue borders.

A list with the following components (if `verbose = TRUE`):

`data.Sim` the Monte Carlo simulated data with evaluations in the last column.

`data.Perm`	the data of the permutated observations and samples with corresponding evaluations and the decision according to `perm.crit`.
`data.Prop`	`nsim` values generated from a normal distribution with mean and s.d. as calculated from the propagated error.
`gradient`	the evaluated gradient vector ∇_x of partial first derivatives.
`hessian`	the evaluated hessian matrix $\mathbf{H}_{xx}$ of partial second derivatives.
`covMat`	the covariance matrix $\mathbf{C}_x$ used for Monte Carlo simulation and error propagation.
`summary`	a summary of the collected statistics, given as a dataframe. These are: mean, s.d. median, mad, lower/upper confidence interval and permutation p-values.

Author(s)

Andrej-Nikolai Spiess

References

Error propagation (in general):
An Introduction to error analysis.
Taylor JR.
University Science Books (1996), New York.

Evaluation of measurement data - Guide to the expression of uncertainty in measurement.
JCGM 100:2008 (GUM 1995 with minor corrections).
https://www.bipm.org/documents/20126/2071204/JCGM_100_2008_E.pdf.

Higher-order Taylor expansion:
On higher-order corrections for propagating uncertainties.
Wang CM & Iyer HK.
Metrologia (2005), **42**: 406-410.

Accuracy of error propagation exemplified with ratios of random variables.
Winzer PJ.
Rev Sci Instrum (2000), **72**: 1447-1454.

Matrix algebra for error propagation:
An Introduction to Error Propagation: Derivation, Meaning and Examples of Equation Cy = Fx-CxFx^t.
https://www.research-collection.ethz.ch/handle/20.500.11850/82620.

Second order nonlinear uncertainty modeling in strapdown integration using MEMS IMUs.
Zhang M, Hol JD, Slot L, Luinge H.
2011 Proceedings of the 14th International Conference on Information Fusion (FUSION) (2011).

Error propagation (in qPCR):
Error propagation in relative real-time reverse transcription polymerase chain reaction quantification models: The balance between accuracy and precision.
Nordgard O, Kvaloy JT, Farmen RK, Heikkila R.
Anal Biochem (2006), **356**: 182-193.

qBase relative quantification framework and software for management and analysis of real-time quantitative PCR data.
Hellemans J, Mortier G, De Paepe A, Speleman F, Vandesompele J.
Genome Biol (2007), **8**: R19.

Multivariate normal distribution:
Stochastic Simulation.
Ripley BD.
Stochastic Simulation (1987). Wiley. Page 98.

Testing for normal distribution:
Testing for Normality.
Thode Jr. HC.
Marcel Dekker (2002), New York.

Approximating the Shapiro-Wilk W-test for non-normality.
Royston P.
Stat Comp (1992), **2**: 117-119.

See Also

Function `ratiocalc` for error analysis within qPCR ratio calculation.

Examples

```
## From summary data just calculate
## Monte-Carlo and propagated error.
EXPR <- expression(x/y)
x <- c(5, 0.01)
y <- c(1, 0.01)
DF <- cbind(x, y)
RES1 <- propagate(expr = EXPR, data = DF, type = "stat",
                  do.sim = TRUE, verbose = TRUE)

## Do Shapiro-Wilks test on Monte Carlo evaluations
## !maximum 5000 datapoints can be used!
## => p.value on border to non-normality
shapiro.test(RES1$data.Sim[1:5000, 3])
## How about a graphical analysis:
qqnorm(RES1$data.Sim[, 3])

## Using raw data
```

```
## Do permutations (swap x and y values)
## and simulations.
EXPR <- expression(x*y)
x <- c(2, 2.1, 2.2, 2, 2.3, 2.1)
y <- c(4, 4, 3.8, 4.1, 3.1, NA)
DF <- cbind(x, y)
RES2 <- propagate(EXPR, DF, type = "raw", do.perm = TRUE,
                  do.sim = TRUE, verbose = TRUE)
RES2$summary

## For replicate data, using relative
## quantification ratio from qPCR.
## How good is the estimation of the propagated error?
## Done without using covariance in the
## calculation and simulation.
## cp's and efficiencies are tied together
## because they are two observations on the
## same sample!
## As we are using an exponential type function,
## better to logarithmize the x-axis.
EXPR <- expression((E1^cp1)/(E2^cp2))
E1 <- c(1.73, 1.75, 1.77)
cp1 <- c(25.77,26.14,26.33)
E2 <-  c(1.72,1.68,1.65)
cp2 <- c(33.84,34.04,33.33)
DF <- cbind(E1, cp1, E2, cp2)
RES3 <- propagate(EXPR, DF, type = "raw", do.sim = TRUE,
                  do.perm = TRUE, verbose = TRUE, logx = TRUE)
## STRONG deviation from normality!
shapiro.test(RES3$data.Sim[1:5000, 5])
qqnorm(RES3$data.Sim[, 5])

## Same setup as above but also
## using a permutation approach
## for resampling the confidence interval.
## Cp's and efficiencies are tied together
## because they are two observations on the
## same sample!
## Similar to what REST2008 software does.
RES4 <- propagate(EXPR, DF, type = "raw", do.sim = TRUE,
                  perm.crit = NULL, do.perm = TRUE,
                  ties = c(1, 1, 2, 2), logx = TRUE, verbose = TRUE)
RES4$summary
## p-value of 0 in perm < init indicates that not a single
## exchange of group memberships resulted in a smaller ratio!

## Proof that covariance of Monte-Carlo
## simulated dataset is the same as from
## initial data.
RES4$covMat
cov(RES4$data.Sim[, 1:4])
all.equal(RES4$covMat, cov(RES4$data.Sim[, 1:4]))
```

qpcR.news *Display news and changes of qpcR package versions*

Description

Displays the latest changes (new functions, bug fixes etc.) of the different package versions in a text window.

Usage

```
qpcR.news(...)
```

Arguments

...	arguments to be passed to `file.show`. Usually needs no entry.

Value

None.

Author(s)

Andrej-Nikolai Spiess

Examples

```
qpcR.news()
```

qpcR_datasets *The (published) datasets implemented in qpcR*

Description

A compilation of published datasets for method evaluation/comparison.

Usage

```
batsch1
batsch2
batsch3
batsch4
batsch5
boggy
competimer
dil4reps94
dyemelt
```

```
guescini1
guescini2
htPCR
karlen1
karlen2
karlen3
lievens1
lievens2
lievens3
reps
reps2
reps3
reps384
rutledge
testdat
vermeulen1
vermeulen2
```

Details

batsch1-5:
Setup: Five 4-fold dilutions with 3 replicates.
Annotation: FX.Y (X = dilution number, Y = replicate number).
Hardware: Lightcycler 1.0 (Roche).
Details:
batsch1: Primers for rat SLC6A14, Taqman probes.
batsch2: Primers for human SLC22A13, Taqman probes.
batsch3: Primers for pig EMT, Taqman probes.
batsch4: Primers for chicken ETT, SybrGreen.
batsch5: Primers for human GAPDH, SybrGreen.

boggy:
Setup: Six 10-fold dilutions with 2 replicates.
Annotation: FX.Y (X = dilution number, Y = replicate number).
Hardware: Chromo4 (BioRad).
Details:
Primers for a synthetic template, consisting of a secondary structure-optimized random sequence (129 bp), Syto-13 dye.

competimer
Setup: 7 concentrations of inhibitor, six 4-fold dilutions, 3 replicates.
Annotation: X_Y_Z (X = inhibitor concentration, Y = dilution number, Z = replicate number).
X: % competimer
A 0%, B 5%, C 10%, D 20%, E 30%, F 40%, G 50%.
Y: dilution factor (-fold)
A 64, B 16, C 4, D 1, E 0.25, F 0.0625, G NTC.
Hardware: Lightcycler 480 (Roche).
Details:
Primers for human AluSx repeats, competitive primers, SybrGreen I dye. NTCs are included.

dil4reps94

Setup: Four 10-fold dilutions with 94 replicates.
Annotation: FX_Y (X = copy number, Y = replicate number).
Hardware: CFX384 (BioRad).
Details:
Primers for the human MYCN gene, synthetic MYCN oligo used as template, SybrGreen I dye. NTCs were removed.

dyemelt:
Setup: Melting curves of a 4-plex qPCR with different fluorescence dyes.
Annotation: T0 (Temperature), EvaGreen, T1 (Temperature), SybrGreen.I, T2 (Temperature), Syto13.
Hardware: Lightcycler 1.0 (Roche).
Details:
A melting curve analysis of a 4-plex real-time PCR on genomic DNA with AZF deletion-specific primers. The dyes used were EvaGreen, SybrGreen I and Syto-13.

guescini1-2:
Setup: Seven 10-fold dilutions with 12 replicates (`guescini1`). Five decreasing steps of PCR mix with 12 replicates (`guescini2`).
Annotation: FX.Y (X = dilution number, Y = replicate number).
Hardware: Lightcycler 480 (Roche).
Details:
Primers for NADH dehydrogenase 1, SybrGreen I dye, data is background subtracted.

htPCR:
Setup: High throughput experiment containing 8858 runs from a 95 x 96 PCR grid.
Annotation: PX.Y (X = plate number, Y = well number).
Hardware: Biomark HD (Fluidigm).
Details:
Proprietary primers, EvaGreen dye, data is ROX normalized.

karlen1-3:
Setup: 4 (5) dilutions (1-, 10-, 50-, 100-, (1000)-fold) with 5 (4) replicates in 4 samples.
Annotation: FX.Y.Z (X = sample number, Y = dilution number, Z = replicate number).
Hardware: ABI Prism 7700 (Applied Biosystems).
Details:
Primers for Caveolin (`karlen1`), Fibronectin (`karlen2`) and L27 (`karlen3`), SybrGreen I dye, data is background subtracted.

lievens1-3:
Setup: Five 5-fold dilutions with 18 replicates (`lievens1`). Five different concentrations of isopropanol (2.5%, 0.5%, 0.1%, 0.02% and 0.004% (v/v)) with 18 replicates (`lievens2`). Five different amounts of tannic acid per reaction (5 ng, 1 ng, 0.2 ng, 0.04 ng and 0.008 ng) and 18 replicates (`lievens3`).
Annotation: SX.Y (X = dilution number, Y = replicate number) (`lievens1`). SX.Y (X = concentration step, Y = replicate number) (`lievens2` & `lievens3`).
Hardware: ABI7300 (ABI) or Biorad IQ5 (Biorad).
Details:
Primers for the soybean lectin endogene Le1, SybrGreen I dye.

reps, reps2, reps3:
Setup: Seven 10-fold dilutions with 4 replicates (`reps`). Five 4-fold dilutions with 3 replicates, 2 different cDNAs (`reps2`). Seven 4-fold dilutions with 3 replicates (`reps3`).
Annotation: FX.Y (X = dilution number, Y = replicate number) (`reps` & `reps3`). FX.Y.Z (X =

cDNA number, Y = dilution number, Z = replicate number) (reps2).
Hardware: Lightcycler 1.0 (Roche) (reps) or MXPro3000P (Stratagene) (reps2 & reps3).
Details:
Primers for the S27a housekeeping gene, SybrGreen I dye, reps3 was ROX-normalized.

reps384
Setup: A data frame with 379 replicate runs of a 384 microtiter plate.
Annotation: A_A_X (X = replicate number).
Hardware: CFX384 (BioRad).
Details:
Primers for the human MYCN gene, synthetic MYCN oligo used as template (15000 copies), SybrGreen I dye. NTCs were removed.

rutledge:
Setup: Six 10-fold dilutions with 4 replicates in 5 individual batches.
Annotation: X.RY.Z (X = dilution number, Y = batch number, Z = replicate number).
Hardware: Opticon 2 (MJ Research).
Details:
Primers for a 102 bp amplicon, SybrGreen I dye, data is background subtracted.

testdat:
Setup: Six 10-fold dilutions with 4 replicates.
Annotation: FX.Y (X = dilution number, Y = replicate number).
Hardware: Lightcycler 1.0 (Roche).
Details:
Same as reps, but each FX.3 has noisy data which fails to fit with the l5 model, each FX.4 passes fitting but fails in sigmoidal structure detection by KOD. Used for evaluating quality checking methods.

vermeulen1-2:
Setup: A subset of the first 20 samples for each of 64 genes (vermeulen1) and the corresponding dilution data for all 64 genes with five 10-fold dilutions and 3 replicates (vermeulen2).
Annotation: X.Y (X = gene name, Y = sample name) (vermeulen1), X.STD_Y.Z (X = gene name, Y = copy number, Z = replicate number) (vermeulen2).
Hardware: Lightcycler 480 (Roche).
Details:
Primers for AHCY, AKR1C1, ALUsq(Eurogentec), ARHGEF7, BIRC5, CAMTA1, CAMTA2, CD44, CDCA5, CDH5, CDKN3, CLSTN1, CPSG3, DDC, DPYSL3, ECEL1, ELAVL4, EPB41L3, EPHA5, EPN2, FYN, GNB1, HIVEP2, HMBS, HPRT1, IGSF4, INPP1, MAP2K4, MAP7, MAPT, MCM2, MRPL3, MTSS1, MYCN(4), NHLH2, NM23A, NRCAM, NTRK1, ODC1, PAICS, PDE4DIP, PIK3R1, PLAGL1, PLAT, PMP22, PRAME, PRDM2, PRKACB, PRKCZ, PTN, PTPRF, PTPRH, PTPRN2, QPCT, SCG2, SDHA(1), SLC25A5, SLC6A8, SNAPC1, TNFRSF, TYMS, UBC(2), ULK2 and WSB1. SybrGreen I dye.
Originally, raw data was available at *http://medgen.ugent.be/jvermeulen*, but site is down. The complete (vermeulen_all) and smaller (vermeulen_sub) datasets can be downloaded from http://www.dr-spiess.de/qpcR/datasets.html.

Author(s)

Andrej-Nikolai Spiess

References

batsch1-5:
Simultaneous fitting of real-time PCR data with efficiency of amplification modeled as Gaussian function of target fluorescence.
Batsch A, Noetel A, Fork C, Urban A, Lazic D, Lucas T, Pietsch J, Lazar A, Schoemig E & Gruendemann D.
BMC Bioinformatics (2008), **9**: 95. Additional File 5 to the paper.

boggy:
A Mechanistic Model of PCR for Accurate Quantification of Quantitative PCR Data.
Boggy GJ & Woolf PJ.
PLOS One (2010), **5**: e12355. Additional File S1 to the paper.

dyemelt:
A one-step real-time multiplex PCR for screening Y-chromosomal microdeletions without downstream amplicon size analysis.
Kozina V, Cappallo-Obermann H, Gromoll J & Spiess AN.
PLOS One (2011), **6**: e23174. Figure 2 to the paper.

guescini1-2:
A new real-time PCR method to overcome significant quantitative inaccuracy due to slight amplification inhibition.
Guescini M, Sisti D, Rocchi MB, Stocchi L & Stocchi V.
BMC Bioinformatics (2008), **9**: 326. Supplemental Data 1 to the paper.

htPCR:
Kindly supplied by Roman Bruno.

karlen1-3:
Karlen Y, McNair A, Perseguers S, Mazza C & Mermod N.
Statistical significance of quantitative PCR.
BMC Bioinformatics (2007), **20**: 131. Supplemental Data 2 to the paper.

lievens1-3:
Enhanced analysis of real-time PCR data by using a variable efficiency model: FPK-PCR.
Lievens A, Van Aelst S, Van den Bulcke M & Goetghebeur E.
Nucleic Acids Res (2012), **40**: e10. Supplemental Data to the paper.

reps, reps2, reps3:
Andrej-Nikolai Spiess & Nadine Mueller, Institute for Hormone and Fertlity Research, Hamburg, Germany.

competimer, dil4reps94, reps384:
Evaluation of qPCR curve analysis methods for reliable biomarker discovery: Bias, resolution, precision, and implications.
Ruijter JM, Pfaffl MW, Zhao S, Spiess AN, Boggy G, Blom J, Rutledge RG, Sisti D, Lievens A, De Preter K, Derveaux S, Hellemans J, Vandesompele J.
Methods (2012), [Epub ahead of print] PubMed PMID: 22975077.

rutledge:
Sigmoidal curve-fitting redefines quantitative real-time PCR with the prospective of developing automated high-throughput applications.
Rutledge RG.
Nucleic Acids Research (2004), **32**: e178. Supplemental Data 1 to the paper.

vermeulen1-2:
Predicting outcomes for children with neuroblastoma using a multigene-expression signature: a retrospective SIOPEN/COG/GPOH study.
Vermeulen J, De Preter K, Naranjo A, Vercruysse L, Van Roy N, Hellemans J, Swerts K, Bravo S, Scaruffi P, et. al.
Lancet Oncol (2009), **10**:663-671.

Examples

```
## 'reps' dataset.
g1 <- gl(7, 4)
ml1 <- modlist(reps, model = l5)
plot(ml1, col = g1)


## 'rutledge' dataset.
g2 <- gl(6, 20)
ml2 <- modlist(rutledge, model = l5)
plot(ml2, col = g2)

## 'lievens1' dataset.
g3 <- gl(5, 18)
ml3 <- modlist(lievens1, model = l5)
plot(ml3, col = g3)
```

`qpcR_functions` *The nonlinear/mechanistic models implemented in qpcR*

Description

A summary of all available models implemented in this package.

Usage

```
l7
l6
l5
l4
b7
b6
b5
b4
expGrowth
expSDM
linexp
mak2
mak2i
mak3
```

```
mak3i
lin2
cm3
spl3
```

Details

The following nonlinear sigmoidal models are implemented:

l7:

$$f(x) = c + k1 \cdot x + k2 \cdot x^2 + \frac{d-c}{(1+exp(b(log(x)-log(e))))^f}$$

l6:

$$f(x) = c + k \cdot x + \frac{d-c}{(1+exp(b(log(x)-log(e))))^f}$$

l5:

$$f(x) = c + \frac{d-c}{(1+exp(b(log(x)-log(e))))^f}$$

l4:

$$f(x) = c + \frac{d-c}{1+exp(b(log(x)-log(e)))}$$

b7:

$$f(x) = c + k1 \cdot x + k2 \cdot x^2 + \frac{d-c}{(1+exp(b(x-e)))^f}$$

b6:

$$f(x) = c + k \cdot x + \frac{d-c}{(1+exp(b(x-e)))^f}$$

b5:

$$f(x) = c + \frac{d-c}{(1+exp(b(x-e)))^f}$$

b4:

$$f(x) = c + \frac{d-c}{1+exp(b(x-e))}$$

The following nonlinear models for subsets of the curve are implemented:

expGrowth:

$$f(x) = a \cdot exp(b \cdot x) + c \,|_{n_1 \le x \le n_2}$$

expSDM:

$$f(x) = a \cdot exp(b \cdot x) + c \,|_{1 \le x \le \mathrm{SDM}}$$

linexp:

$$f(x) = a \cdot exp(b \cdot x) + (k \cdot x) + c \,|_{1 \le x \le \mathrm{SDM}}$$

lin2:

$$f(x) = \eta \cdot log\left(exp\left(a1 \cdot \frac{x-\tau}{\eta}\right) + exp\left(a2 \cdot \frac{x-\tau}{\eta}\right)\right) + c \,\Big|_{1 \le x \le \mathrm{SDM}}$$

The following mechanistic models are implemented:

mak2 & mak2i:

$$F_n = F_{n-1} + k \cdot log\left(1 + \left(\frac{F_{n-1}}{k}\right)\right) + Fb \Big|_{1 \leq x \leq \text{SDM}}$$

mak3 & mak3i:

$$F_n = F_{n-1} + k \cdot log\left(1 + \left(\frac{F_{n-1}}{k}\right)\right) + (slope \cdot n + Fb) \Big|_{1 \leq x \leq \text{SDM}}$$

cm3:

$$F_n = F_{n-1} \cdot \left(1 + \left(\frac{max - F_{n-1}}{max}\right) - \left(\frac{F_{n-1}}{Kd + F_{n-1}}\right)\right) + Fb$$

Other models:

spl3:

$$S : [a, b] \rightarrow Real, a = n_0 < n_1 < \ldots < n_{k-1} < n_k = b$$

mak2 and **mak3** are two mechanistic models developed by Gregory Boggy (see references). The mechanistic models are a completely different approach in that the response value (Fluorescence) is not a function of the predictor value (Cycles), but a function of the preceeding response value, that is, $F_n = f(F_{n-1})$. These are also called 'recurrence relations' or 'iterative maps'. The implementation of these models in the 'qpcR" package is the following:
1) In case of `mak2/mak2i` or `mak3/mak3i`, all cycles up from the second derivative maximum (SDM) of a four-parameter log-logistic model (l4) are chopped off. This is because these two models do not fit to a complete sigmoidal curve. An `offset` criterion from the SDM can be defined in `pcrfit`, see there.
2) For `mak2i/mak3i`, a grid of sensible starting values is created for all parameters in the model. For `mak2/mak3` the recurrence function is fitted directly (which is much faster, but may give convergence problems), so proceed to 7).
3) For each combination of starting parameters, the model is fit.
4) The acquired parameters are collected in a parameter matrix together with the residual sum-of-squares (RSS) of the fit.
5) The parameter combination is selected that delivered the lowest RSS.
6) These parameters are transferred to `pcrfit`, and the data is refitted.
7) Parameter D_0 can be used directly to calculate expression ratios, hence making the use of threshold cycles and efficiencies expendable.
cm3 is a mechanistic model by Carr & Moore (see references). In contrast to the `mak` models, `cm3` models the complete curve, which might prove advantageous as no decision on curve subset selection has to be done. As in the `mak` models, D_0 is the essential parameter to use.
spl3 is a cubic spline function that treats each point as being exact. It is just implemented for comparison purposes.
lin2 is a bilinear model developed by P. Buchwald (see references). These are essentially two linear functions connected by a transition region.

The functions are defined as a list containing the following items:
`$expr` the function as an expression for the fitting procedure.
`$fct` the function defined as `f(x, parm)`.
`$ssfct` the self-starter function.
`$d1` the first derivative function.
`$d2` the second derivative function.
`$inv` the inverse function.
`$expr.grad` the function as an expression for gradient calculation.
`$inv.grad` the inverse functions as an expression for gradient calculation.
`$parnames` the parameter names.
`$name` the function name.
`$type` the function type as a character string.

Note

For models `l6, l7, b6, b7` there are no explicit solutions to the inverse function. The calculation of `x` from `y` (Cycles from Fluorescence) is done using `uniroot` by minimizing `model$fct(x, parm) - y` in the interval [1, 100].

Author(s)

Andrej-Nikolai Spiess

References

4-parameter logistic:
Validation of a quantitative method for real time PCR kinetics.
Liu W & Saint DA.
Biochem Biophys Res Commun (2002), **294**:347-53.

Standardized determination of real-time PCR efficiency from a single reaction set-up.
Tichopad A, Dilger M, Schwarz G & Pfaffl MW.
Nucleic Acids Res (2003), **31**:e122.

Sigmoidal curve-fitting redefines quantitative real-time PCR with the prospective of developing automated high-throughput applications.
Rutledge RG.
Nucleic Acids Res (2004), **32**:e178.

A kinetic-based sigmoidal model for the polymerase chain reaction and its application to high-capacity absolute quantitative real-time PCR.
Rutledge RG & Stewart D.
BMC Biotechnol (2008), **8**:47.

Evaluation of absolute quantitation by nonlinear regression in probe-based real-time PCR.
Goll R, Olsen T, Cui G & Florholmen J.
BMC Bioinformatics (2006), **7**:107

Comprehensive algorithm for quantitative real-time polymerase chain reaction.
Zhao S & Fernald RD.
J Comput Biol (2005), **12**:1047-64.

4-parameter log-logistic; 5-parameter logistic/log-logistic:
qpcR: an R package for sigmoidal model selection in quantitative real-time polymerase chain reaction analysis.
Ritz C & Spiess AN.
Bioinformatics (2008), **24**:1549-51.

Highly accurate sigmoidal fitting of real-time PCR data by introducing a parameter for asymmetry.
Spiess AN, Feig C & Ritz C.
BMC Bioinformatics (2008), **29**:221.

exponential model:
Standardized determination of real-time PCR efficiency from a single reaction set-up.
Tichopad A, Dilger M, Schwarz G & Pfaffl MW.
Nucleic Acids Research (2003), **31**:e122.

Comprehensive algorithm for quantitative real-time polymerase chain reaction.
Zhao S & Fernald RD.
J Comput Biol (2005), **12**:1047-64.

mak2, mak2i, mak3, mak3i:
A Mechanistic Model of PCR for Accurate Quantification of Quantitative PCR Data.
Boggy GJ & Woolf PJ.
PLoS ONE (2010), **5**:e12355.

lin2:
A general bilinear model to describe growth or decline time profiles.
Buchwald P.
Math Biosci (2007), **205**:108-36.

cm3:
Robust quantification of polymerase chain reactions using global fitting.
Carr AC & Moore SD.
PLoS One (2012), **7**:e37640.

Examples

```
m1 <- pcrfit(reps, 1, 2, b4)
m2 <- pcrfit(reps, 1, 2, b5)
m3 <- pcrfit(reps, 1, 2, l6)
m4 <- pcrfit(reps, 1, 2, l7)

## Get the second derivative
## curve of m2.
d2 <- b5$d2(m2$DATA[, 1], coef(m2))
plot(m2)
lines(d2, col = 2)
```

`ratiobatch` *Calculation of ratios in a batch format for multiple genes/samples*

Description

For multiple qPCR data from type 'pcrbatch', this function calculates ratios between samples, using normalization against one or more reference gene(s), if supplied. Multiple reference genes can be averaged according to Vandesompele *et al*. (2002). The input may be single qPCR data or (more likely) data containing replicates. This is essentially a version of `ratiocalc` that can handle multiple reference genes and genes-of-interest with multiple (replicated) samples as found in large-scale qPCR runs such as 96- or 384-Well plates. A boxplot representation for all Monte-Carlo simulations, permutations and error propagations including 95% confidence intervals is calculated for each ratio calculation.

Usage

```
ratiobatch(data, group = NULL, plot = TRUE,
           combs = c("same", "across", "all"),
           type.eff = "mean.single", which.cp = "cpD2",
           which.eff = "sli", refmean = FALSE,
           dataout = NULL, verbose = TRUE, ...)
```

Arguments

`data`	multiple qPCR data generated by `pcrbatch`.
`group`	a character vector defining the replicates (if any) of control/treatment samples and reference genes/genes-of-interest. See 'Details'
.	
`plot`	logical. If `TRUE`, plots are displayed for the diagnostics and analysis.
`combs`	type of combinations between different samples (i.e. r1s1:g1s2). See 'Details'.
`type.eff`	type of efficiency averaging used. Same as in `ratiocalc`.
`which.eff`	efficiency obtained from which method. Same as in `ratiocalc`.
`which.cp`	threshold cycle obtained from which method. Same as in `ratiocalc`.
`dataout`	an optional file path where to store the result dataframe.
`refmean`	logical. If `TRUE`, multiple reference are averaged before calculating the ratios. See 'Details'.
`verbose`	logical. If `TRUE`, the steps of analysis are shown in the console window
`...`	other parameters to be passed to `ratiocalc`.

Details

Similar to `ratiocalc`, the replicates of the 'pcrbatch' data columns are to be defined as a character vector with the following abbreviations:

"g1s1": gene-of-interest #1 in treatment sample #1
"g1c1": gene-of-interest #1 in control sample #1
"r1s1": reference gene #1 in treatment sample #1
"r1c1": reference gene #1 in control sample #1

There is no distinction between the different technical replicates so that three different runs of gene-of-interest #1 in treatment sample #2 are defined as c("g1s2", "g1s2", "g1s2").

Example:
1 control sample with 2 genes-of-interest (2 technical replicates), 2 treatment samples with 2 genes-of-interest (2 technical replicates):
"g1c1", "g1c1", "g2c1", "g2c1", "g1s1", "g1s1", "g1s2", "g1s2", "g2s1", "g2s1", "g2s2", "g2s2"

The ratios are calculated for all pairwise 'rc:gc' and 'rs:gs' combinations according to:
For all control samples $i = 1 \dots I$ and treatment samples $j = 1 \dots J$, reference genes $k = 1 \dots K$ and genes-of-interest $l = 1 \dots L$, calculate

Without reference genes:

$$\frac{E(g_l c_i)^{cp(g_l c_i)}}{E(g_l s_j)^{cp(g_l s_j)}}$$

With reference genes:

$$\frac{E(g_l c_i)^{cp(g_l c_i)}}{E(g_l s_j)^{cp(g_l s_j)}} / \frac{E(r_k c_i)^{cp(r_k c_i)}}{E(r_k s_j)^{cp(r_k s_j)}}$$

For the mechanistic models `makX/cm3` the following is calculated:

Without reference genes:

$$\frac{D_0(g_l s_j)}{D_0(g_l c_i)}$$

With reference genes:

$$\frac{D_0(g_l s_j)}{D_0(g_l c_i)} / \frac{D_0(r_k s_j)}{D_0(r_k c_i)}$$

Efficiencies can be taken from the individual curves or averaged from the replicates as described in the documentation to `ratiocalc`. It is also possible to give external efficiencies (i.e. acquired by some calibration curve) to the function. See 'Examples'. The different combinations of `type.eff`, `which.eff` and `which.cp` can yield very different results in ratio calculation. We observed a relatively stable setup which minimizes the overall variance using the combination

`type.eff = "mean.single"` # averaging efficiency across replicates
`which.eff = "sli"` # taking efficiency from the sliding window method
`which.cp = "sig"` # using the second derivative maximum of a sigmoidal fit

This is also the default setup in the function. The lowest variance can be obtained for the threshold cycles if the asymmetric 5-parameter `l5` model is used in the `pcrbatch` function.

There are three different combination setups possible when calculating the pairwise ratios:
`combs = "same"`: reference genes, genes-of-interest, control and treatment samples are the `same`, i.e. $i = k, m = o, j = n, l = p$.
`combs = "across"`: control and treatment samples are the same, while the genes are combinated, i.e. $i \neq k, m \neq o, j = n, l = p$,.
`combs = "all"`: reference genes, genes-of-interest, control and treatment samples are all combinated, i.e. $i \neq k, m \neq o, j \neq n, l \neq p$.

The last setting rarely makes sense and is very time-intensive. `combs = "same"` is the most common setting, but `combs = "across"` also makes sense if different genes-of-interest and reference gene combinations should be calculated for the same samples.

From version 1.3-6, `ratiobatch` has the option of averaging several reference genes, as described in Vandesompele *et al.* (2002). Threshold cycles and efficiency values for any i reference genes with

j replicates are averaged before calculating the ratios using the averaged value μ_r for all reference genes in a control/treatment sample. The overall error σ_r is obtained by error propagation. The whole procedure is accomplished by function `refmean`, which can be used as a stand-alone function, but is most conveniently used inside `ratiobatch` setting `refmean = TRUE`. See in 'Examples'. For details about reference gene averaging by `refmean`, see there. If none or only one per sample is found, the data is analyzed without using reference gene averaging/error propagation.

Value

A list with the following components:

`resList`	a list with the results from the combinations as list items.
`resDat`	a dataframe with the results in colums.

Both `resList` and `resDat` have as names the combinations used for the ratio calculation. If `plot = TRUE`, a boxplot matrix from the Monte-Carlo simulations, permutations and error propagations is given including 95% confidence intervals as coloured horizontal lines.

Note

This function can be used quite conveniently when the raw fluorescence data from the 96- or 384-well runs come from Excel with 'Cycles' in the first column and run descriptions as explained above in the remaining column descriptions (such as 'r1c6'). Examples for a proper format can be found under `http://www.dr-spiess.de//qpcR//datasets.html`. This data may then be imported into R by `dat <- pcrimport()`.

Author(s)

Andrej-Nikolai Spiess

References

Accurate normalization of real-time quantitative RT-PCR data by geometric averaging of multiple internal control genes.
Vandesompele J, De Preter K, Pattyn F, Poppe B, Van Roy N, De Paepe A, Speleman F.
Genome Biol (2002), **3**: research0034-research0034.11.

Examples

```
## One reference gene, one gene of interest,
## one control and one treatment sample with
## 4 replicates each => 1 x Ratio = 1.
DAT1 <- pcrbatch(reps, fluo = c(2:9, 2:9), model = l5)
GROUP1 <- c("g1c1", "g1c1", "g1c1", "g1c1",
            "g1s1", "g1s1", "g1s1", "g1s1",
            "r1c1", "r1c1", "r1c1", "r1c1",
            "r1s1", "r1s1", "r1s1", "r1s1")
ratiobatch(DAT1, GROUP1, refmean = FALSE)

## One reference gene, one gene of interest,
```

```
## two control and two treatment samples with
## 2 replicates each => 4 x Ratio = 1.
DAT2 <- pcrbatch(reps, fluo = c(2:9, 2:9), model = l5)
GROUP2 <- c("g1c1", "g1c1", "g1c2", "g1c2",
            "g1s1", "g1s1", "g1s2", "g1s2",
            "r1c1", "r1c1", "r1c2", "r1c2",
            "r1s1", "r1s1", "r1s2", "r1s2")
ratiobatch(DAT2, GROUP2, refmean = FALSE)

## Two reference genes, one gene of interest,
## one control and one treatment samples with
## 4 replicates each => 2 x Ratio = 1.
DAT3 <- pcrbatch(reps, fluo = c(2:9, 2:9, 2:9), model = l5)
GROUP3 <- c("g1c1", "g1c1", "g1c1", "g1c1",
            "g1s1", "g1s1", "g1s1", "g1s1",
            "r1c1", "r1c1", "r1c1", "r1c1",
            "r1s1", "r1s1", "r1s1", "r1s1",
            "r2c1", "r2c1", "r2c1", "r2c1",
            "r2s1", "r2s1", "r2s1", "r2s1")
ratiobatch(DAT3, GROUP3, refmean = FALSE)

## Two reference genes, one gene of interest,
## one control and one treatment samples with
## 4 replicates each.
## Reference genes are averaged => 1 x Ratio = 1.
DAT4 <- pcrbatch(reps, fluo = c(2:9, 2:9, 2:9), model = l5)
GROUP4 <- c("g1c1", "g1c1", "g1c1", "g1c1",
            "g1s1", "g1s1", "g1s1", "g1s1",
            "r1c1", "r1c1", "r1c1", "r1c1",
            "r1s1", "r1s1", "r1s1", "r1s1",
            "r2c1", "r2c1", "r2c1", "r2c1",
            "r2s1", "r2s1", "r2s1", "r2s1")
ratiobatch(DAT4, GROUP4, refmean = TRUE)

## Same as above, but use same efficiency E = 2.
ratiobatch(DAT4, GROUP4, which.eff = 2)

## No reference genes, two genes-of-interest,
## two control and two treatment samples with
## 2 replicates each, efficiency from sigmoidal model.
DAT6 <- pcrbatch(reps, fluo = 2:17, model = l5)
GROUP6 <- c("g1s1", "g1s1", "g1s2", "g1s2",
            "g2s1", "g2s1", "g2s2", "g2s2",
            "g1c1", "g1c1", "g1c2", "g1c2",
            "g2c1", "g2c1", "g2c2", "g2c2")
ratiobatch(DAT6, GROUP6, which.eff = "sig")

## Same as above, but using a mechanistic model (mak3).
## BEWARE: type.eff must be "individual"!
DAT7 <- pcrbatch(reps, fluo = 2:17, model = l5,
                 methods = c("sigfit", "mak3"))
GROUP7 <- c("g1s1", "g1s1", "g1s2", "g1s2",
            "g2s1", "g2s1", "g2s2", "g2s2",
```

```
             "g1c1", "g1c1", "g1c2", "g1c2",
             "g2c1", "g2c1", "g2c2", "g2c2")
ratiobatch(DAT7, GROUP7, which.eff = "mak",
           type.eff = "individual")
```

`ratiocalc` *Calculation of ratios from qPCR runs with/without reference genes*

Description

For multiple qPCR data from type 'pcrbatch', this function calculates ratios between two samples (control/treatment) of a gene-of-interest, using normalization against a reference gene, if supplied. The input can be single qPCR data or (more likely) data containing replicates. Errors and confidence intervals for the obtained ratios can be calculated by Monte-Carlo simulation, a permutation approach similar to the popular REST software and by error propagation. Statistical significance for the ratios is calculated by a permutation approach of randomly reallocated vs. non-reallocated data. See 'Details'.

Usage

```
ratiocalc(data, group = NULL,
          which.eff = c("sig", "sli", "exp", "mak", "cm3", "ext"),
          type.eff = c("individual", "mean.single", "median.single",
                       "mean.pair", "median.pair"),
          which.cp = c("cpD2", "cpD1", "cpE", "cpR", "cpT", "Cy0", "ext"),
          ...)
```

Arguments

`data`	multiple qPCR data generated by `pcrbatch`.
`group`	a character vector defining the replicates (if any) of control/treatment samples and reference genes/genes-of-interest. See 'Details'.
`which.eff`	efficiency calculated by which method. Defaults to sigmoidal fit. See output of `pcrbatch`. Alternatively, a numeric value for all runs, a vector of external efficiencies with one element per run or directly transferred from `ratioPar`.
`type.eff`	type of efficiency pre-processing prior to error analysis. See 'Details'.
`which.cp`	type of threshold cycles to be used for the analysis. See output of `efficiency`. Alternatively, a vector of external threshold cycles with one element per run or directly transferred from `ratioPar`.
`...`	other parameters to be passed to `propagate`.

Details

The replicates for the data columns are to be defined as a character vector with the following abbreviations:

"gs": gene-of-interest in treatment sample
"gc": gene-of-interest in control sample
"rs": reference gene in treatment sample
"rc": reference gene in control sample

There is no distinction between the different runs of the same sample, so that three different runs of a gene-of-interest in a treatment sample are defined as c("gs", "gs", "gs"). The error analysis calculates statistics from ALL replicates, so that a further sub-categorization of runs is superfluous. NOTE: If the setup consists of different sample or gene combinations, use `ratiobatch`!

Examples:
No replicates: NULL.
2 runs with 2 replicates each, no reference gene: c("gs", "gs", "gs", "gs", "gc", "gc", "gc", "gc").
1 run with two replicates each and reference gene: c("gs", "gs", "gc", "gc", "rs", "rs", "rc", "rc").

`type.eff` defines the pre-processing of the efficiencies before being transferred to `propagate`. The qPCR community sometimes uses single efficiencies, or averaged over replicates etc., so that different settings were implemented. In detail, these are the following:

"individual": The individual efficiencies from each run are used.
"mean.single": Efficiencies are averaged over all replicates.
"median.single": Same as above but median instead of mean.
"mean.pair": Efficiencies are averaged from all replicates of treatment sample AND control.
"median.pair": Same as above but median instead of mean.

Efficiencies can be taken from the individual curves or averaged from the replicates as described in the documentation to `ratiocalc`. The different combinations of `type.eff`, `which.eff` and `which.cp` can yield very different results in ratio calculation. We observed a relatively stable setup which minimizes the overall variance using the combination

```
type.eff = "mean.single" # averaging efficiency across replicates
which.eff = "sli" # taking efficiency from the sliding window method
which.cp = "sig" # using the second derivative maximum of a sigmoidal fit
```

The ratios are calculated according to the following formulas:
Without reference gene:

$$\frac{E(gc)^{cp(gc)}}{E(gs)^{cp(gs)}}$$

With reference gene:

$$\frac{E(gc)^{cp(gc)}}{E(gs)^{cp(gs)}} / \frac{E(rc)^{cp(rc)}}{E(rs)^{cp(rs)}}$$

The permutation approach permutates threshold cycles and efficiency replicates within treatment and control samples. The treatment/control samples (and their respective efficiencies) are tied together, which is similar to the popular REST software approach ("pairwise-reallocation test"). Ratios are calculated for each permutation and compared to ratios obtained if samples were randomly reallocated from the treatment to the control group. Three p-values are calculated from all permutations that gave a higher/equal/lower ratio than the original data. The resulting p-values are thus an indication for the significance in any direction AGAINST the null hypothesis that ratios calculated by permutation are just by chance.

If the mechanistic `mak2/mak2i/mak3/mak3i/cm3` models are used in `pcrbatch`, set `which.eff = "mak"` for ratio calculations from the $D0$ values of the model:
Without reference gene:

$$\frac{D_0(gs)}{D_0(gc)}$$

With reference gene:

$$\frac{D_0(gs)}{D_0(gc)} / \frac{D_0(rs)}{D_0(rc)}$$

Confidence values are returned for all three methods (Monte Carlo, permutation, error propagation) as follows:
Monte-Carlo: From the evaluations of the Monte-Carlo simulated data.
Permutation: From the evaluations of the within-group permutated data.
Propagation: From the propagated error, assuming normality.

Value

A list with the following components:
`data`: the data that was transferred to `propagate` for the error analysis.
`data.Sim`, `data.Perm`, `data.Prop`, `derivs`, `covMat`: The complete output from `propagate`.
`summary`: a summary of the results obtained from the Monte Carlo simulation, permutation and error propagation.

Note

The error calculated from qPCR data by `propagate` often seems quite high. This largely depends on the error of the base (i.e. efficiency) of the exponential function. The error usually decreases when setting `use.cov = TRUE` in the `...` part of the function. It can be debated anyhow, if the variables 'efficiency' and 'threshold cycles' have a covariance structure. As the efficiency is deduced at the second derivative maximum of the sigmoidal curve, variance in the second should show an effect on the first, such that the use of a var-cov matrix might be feasible. It is also commonly encountered that the propagated error is much higher when using reference genes, as the number of partial derivative functions increases.

Author(s)

Andrej-Nikolai Spiess

References

Analysis of relative gene expression data using real-time quantitative PCR and the 2(-Delta Delta C(T)) method.
Livak KJ & Schmittgen TD.
Methods (2001), **25**: 402-428.

Standardized determination of real-time PCR efficiency from a single reaction set-up.
Tichopad A, Dilger M, Schwarz G, Pfaffl MW.
Nucleic Acids Res (2003), **31**: e122.

Validation of a quantitative method for real time PCR kinetics.
Liu W & Saint DA.
Biochem Biophys Res Commun (2002), **294**: 347-53.

Relative expression software tool (REST) for group-wise comparison and statistical analysis of relative expression results in real-time PCR.
Pfaffl MW, Horgan GW, Dempfle L.
Nucl Acids Res (2002), **30**: e36.

See Also

The function ratioPar, which is a much better and simpler way if only external threshold cycles/efficiencies should be used.

Examples

```
## Only treatment sample and control,
## no reference gene, 4 replicates each.
## Individual efficiencies for error calculation.
DAT1 <- pcrbatch(reps, fluo = 2:9, model = l4)
GROUP1 <- c("gs", "gs", "gs", "gs", "gc", "gc", "gc", "gc")
RES1 <- ratiocalc(DAT1, group = GROUP1, which.eff = "sli",
                  type.eff = "individual", which.cp = "cpD2")
RES1$summary


## Gets even better using averaged efficiencies
## over all replicates.
## p-value indicates significant upregulation in
## comparison to randomly reallocated
## samples (similar to REST software)
RES2 <- ratiocalc(DAT1, GROUP1, which.eff = "sli",
                   type.eff = "mean.single", which.cp = "cpD2")
RES2$summary

## Using reference data.
## Toy example is same data as above
## but replicated as reference such
## that the ratio should be 1.
```

```
DAT3 <- pcrbatch(reps, fluo = c(2:9, 2:9), model = l4)
GROUP3 <- c("gs", "gs", "gs", "gs",
            "gc", "gc", "gc", "gc",
            "rs", "rs", "rs", "rs",
            "rc", "rc", "rc", "rc")
RES3 <- ratiocalc(DAT3, GROUP3, which.eff = "sli",
                  type.eff = "mean.single", which.cp = "cpD2")
RES3$summary

## Using one of the mechanistic models
## => ratios are calculated from the replicate
## D0 values, without reference genes.
DAT4 <- pcrbatch(reps, fluo = 2:9,
                 methods = c("sigfit", "sliwin", "mak3"))
GROUP4 <- c("gs", "gs", "gs", "gs", "gc", "gc", "gc", "gc")
RES4 <- ratiocalc(DAT4, GROUP4, which.eff = "mak")
RES4$summary

## Example without replicates
## => no Monte-Carlo simulations
## and hence no plots.
DAT5 <- pcrbatch(reps, fluo = 2:5, model = l4)
GROUP5 <- c("gs", "gc", "rs", "rc")
RES5 <- ratiocalc(DAT5, GROUP5, which.eff = "sli",
                  type.eff = "individual", which.cp = "cpD2")
RES5$summary

## Using external efficiencies.
DAT6 <- pcrbatch(reps, fluo = 2:9, model = l5)
GROUP6 <- c("gs", "gs", "gs", "gs", "gc", "gc", "gc", "gc")
EFF6 <- rep(c(1.72, 1.76), c(4, 4))
RES6 <- ratiocalc(DAT6, GROUP6, which.eff = EFF6,
                  type.eff = "individual", which.cp = "cpD2")
RES6$summary

## Using external efficiencies AND
## external threshold cycles.
DAT7 <- pcrbatch(reps, fluo = 2:9, model = l5)
GROUP7 <- c("gs", "gs", "gs", "gs", "gc", "gc", "gc", "gc")
EFF7 <- rep(c(1.72, 1.76), c(4, 4))
CP7 <- c(15.44, 15.33, 14.84, 15.34, 18.89, 18.71, 18.13, 17.22)
RES7 <- ratiocalc(DAT7, GROUP7, which.eff = EFF7,
                  type.eff = "individual", which.cp = CP7)
RES7$summary

## Compare 'ratiocalc' to REST software
## using the data from the REST 2008
## manual (http://rest.gene-quantification.info/).
## We supply the threshold cycles/efficiencies from the
## manual as external data to 'dummy' pcrbatch data.
## BETTER: use 'ratioPar' function!
cp.rc <- c(26.74, 26.85, 26.83, 26.68, 27.39, 27.03, 26.78, 27.32)
cp.rs <- c(26.77, 26.47, 27.03, 26.92, 26.97, 26.97, 26.07, 26.3, 26.14, 26.81)
```

```
cp.gc <- c(27.57, 27.61, 27.82, 27.12, 27.76, 27.74, 26.91, 27.49)
cp.gs <- c(24.54, 24.95, 24.57, 24.63, 24.66, 24.89, 24.71, 24.9, 24.26, 24.44)
eff.rc <- rep(1.97, 8)
eff.rs <- rep(1.97, 10)
eff.gc <- rep(2.01, 8)
eff.gs <- rep(2.01, 10)
CP8 <- c(cp.rc, cp.rs, cp.gc, cp.gs)
EFF8 <- c(eff.rc, eff.rs, eff.gc, eff.gs)
DAT8 <- pcrbatch(rutledge, 1, 2:37, model = l4)
GROUP8 <- rep(c("rc", "rs", "gc", "gs"), c(8, 10, 8, 10))
RES8 <- ratiocalc(DAT8, GROUP8, which.eff = EFF8, which.cp = CP8)
RES8$summary
## => Confidence interval: 2.983/9.996
## REST 2008 manual, page 10: 2.983/9.996
```

`ratioPar` *Calculation of ratios in a batch format from external PCR parameters*

Description

Starting from external PCR parameters such as threshold cycles/efficiency values commonly obtained from other programs, this function calculates ratios between samples, using normalization against one or more reference gene(s), if supplied. By default, multiple reference genes are averaged according to Vandesompele *et al*. (2002). The input can be single qPCR data or (more likely) data containing replicates. It is similar to `ratiobatch` and can handle multiple reference genes and genes-of-interest with multiple (replicated) samples as found in large-scale qPCR runs such as 96- or 384-Well plates. The results are automatically stored as a file or copied into the clipboard. A boxplot representation for all Monte-Carlo simulations, permutations and error propagations including 95% confidence intervals is also given.

Usage

```
ratioPar(group = NULL, effVec = NULL, cpVec = NULL,
         type.eff = "individual", plot = TRUE,
         combs = c("same", "across", "all"),
         refmean = FALSE, verbose = TRUE, ...)
```

Arguments

`group`	a character vector defining the replicates (if any) of control/treatment samples and reference genes/genes-of-interest. See 'Details'.
`effVec`	a vector of efficiency values with the same length of `group`.
`cpVec`	a vector of threshold cycle values with the same length of `group`.
`type.eff`	type of efficiency averaging used. Same as in `ratiocalc`.
`plot`	logical. If `TRUE`, plots are displayed for the diagnostics and analysis.
`combs`	type of combinations between different samples (i.e. r1s1:g1s2). See 'Details'.

refmean	logical. If TRUE, multiple reference are averaged before calculating the ratios. See 'Details'.
verbose	logical. If TRUE, the steps of analysis are shown in the console window
...	other parameters to be passed to ratiocalc.

Details

As in ratiobatch, the replicates are to be defined as a character vector with the following abbreviations:

"g1s1": gene-of-interest #1 in treatment sample #1
"g1c1": gene-of-interest #1 in control sample #1
"r1s1": reference gene #1 in treatment sample #1
"r1c1": reference gene #1 in control sample #1

There is no distinction between the different technical replicates so that three different runs of gene-of-interest #1 in treatment sample #2 are defined as c("g1s2", "g1s2", "g1s2").

Example:
1 control sample with 2 genes-of-interest (2 technical replicates), 2 treatment samples with 2 genes-of-interest (2 technical replicates):
"g1c1", "g1c1", "g2c1", "g2c1", "g1s1", "g1s1", "g1s2", "g1s2", "g2s1", "g2s1", "g2s2", "g2s2"

The ratios are calculated for all pairwise 'rc:gc' and 'rs:gs' combinations according to:
For all control samples $i = 1 \dots I$ and treatment samples $j = 1 \dots J$, reference genes $k = 1 \dots K$ and genes-of-interest $l = 1 \dots L$, calculate

Without reference genes:

$$\frac{E(g_l c_i)^{cp(g_l c_i)}}{E(g_l s_j)^{cp(g_l s_j)}}$$

With reference genes:

$$\frac{E(g_l c_i)^{cp(g_l c_i)}}{E(g_l s_j)^{cp(g_l s_j)}} \Big/ \frac{E(r_k c_i)^{cp(r_k c_i)}}{E(r_k s_j)^{cp(r_k s_j)}}$$

For the mechanistic models makX/cm3 the following is calculated:

Without reference genes:

$$\frac{D_0(g_l s_j)}{D_0(g_l c_i)}$$

With reference genes:

$$\frac{D_0(g_l s_j)}{D_0(g_l c_i)} \Big/ \frac{D_0(r_k s_j)}{D_0(r_k c_i)}$$

Efficiencies can be taken from the individual samples or averaged from the replicates as described in the documentation to ratiocalc. Different settings in type.eff can yield very different results in ratio calculation. We observed a relatively stable setup which minimizes the overall variance using the type.eff = "mean.single".

There are three different combination setups possible when calculating the pairwise ratios:
combs = "same": reference genes, genes-of-interest, control and treatment samples are the same, i.e. $i = k, m = o, j = n, l = p$.

`combs = "across"`: control and treatment samples are the same, while the genes are combinated, i.e. $i \neq k, m \neq o, j = n, l = p$,.
`combs = "all"`: reference genes, genes-of-interest, control and treatment samples are all combinated, i.e. $i \neq k, m \neq o, j \neq n, l \neq p$.

The last setting rarely makes sense and is very time-intensive. `combs = "same"` is the most common setting, but `combs = "across"` also makes sense if different genes-of-interest and reference gene combinations should be calculated for the same samples.

`ratioPar` has an option of averaging several reference genes, as described in Vandesompele *et al.* (2002). Threshold cycles and efficiency values for any i reference genes with j replicates are averaged before calculating the ratios using the averaged value μ_r for all reference genes in a control/treatment sample. The overall error σ_r is obtained by error propagation. The whole procedure is accomplished by function `refmean`, which can be used as a stand-alone function, but is most conveniently used inside `ratioPar` setting `refmean = TRUE`. For details about reference gene averaging by `refmean`, see there.

Value

A list with the following components:

`resList`	a list with the results from the combinations as list items.
`resDat`	a dataframe with the results in colums.

Both `resList` and `resDat` have as names the combinations used for the ratio calculation. If `plot = TRUE`, a boxplot matrix from the Monte-Carlo simulations, permutations and error propagations is given including 95% confidence intervals as coloured horizontal lines.

Note

This function can be used quite conveniently when the PCR parameters are from 96- or 384-well runs plates and exported to a tab-delimited file.

Author(s)

Andrej-Nikolai Spiess

References

Accurate normalization of real-time quantitative RT-PCR data by geometric averaging of multiple internal control genes.
Vandesompele J, De Preter K, Pattyn F, Poppe B, Van Roy N, De Paepe A, Speleman F.
Genome Biol (2002), **3**: research0034-research0034.11.

Examples

```
## One control sample, two treatment samples,
## one gene-of-interest, two reference genes,
## two replicates each. Replicates are averaged,
## but reference genes not, so that we have 4 ratios.
GROUP1 <- c("r1c1", "r1c1", "r2c1", "r2c1", "g1c1", "g1c1",
```

```
           "r1s1", "r1s1", "r1s2", "r1s2", "r2s1", "r2s1",
           "r2s2", "r2s2", "g1s1", "g1s1", "g1s2", "g1s2")
EFF1 <- c(1.96, 2.03, 1.60, 1.67, 1.91, 1.97, 1.53, 1.61, 1.87,
          1.92, 1.52, 1.58, 1.84, 1.90, 1.49, 1.56, 1.83, 1.87)
CP1 <- c(15.44, 15.33, 14.84, 15.34, 18.89, 18.71, 18.13, 17.22, 22.06,
        21.85, 21.03, 20.92, 25.34, 25.12, 25.00, 24.62, 28.39, 28.28)
RES1 <- ratioPar(group = GROUP1, effVec = EFF1, cpVec= CP1, refmean = FALSE)



## Same as above, but now we average the two
## reference genes, so that we have 2 ratios.
RES2 <- ratioPar(group = GROUP1, effVec = EFF1, cpVec= CP1, refmean = TRUE)

## Two control samples, one treatment sample,
## one gene-of-interest, one reference gene,
## no replicates. Reference gene has efficiency = 1.8,
## gene-of-interest has efficiency = 1.9.
GROUP3 <- c("r1c1", "r1c2", "g1c1", "g1c2",
            "r1s1", "g1s1")
EFF3 <- c(1.8, 1.8, 1.9, 1.9, 1.8, 1.9)
CP3 <- c(17.25, 17.38, 22.52, 23.18, 21.42, 19.83)
RES3 <- ratioPar(group = GROUP3, effVec = EFF3, cpVec= CP3, refmean = TRUE)

## One control sample, one treatment sample,
## three genes-of-interest, no reference gene,
## three replicates. Using efficiency from sigmoidal model.
GROUP4 <- c("g1c1", "g1c1", "g1c1", "g2c1", "g2c1", "g2c1", "g3c1", "g3c1", "g3c1",
            "g1s1", "g1s1", "g1s1", "g2s1", "g2s1", "g2s1", "g3s1", "g3s1", "g3s1")
EFF4 <- c(1.79, 1.71, 1.83, 1.98, 1.85, 1.76, 1.76, 1.91, 1.84, 1.80, 1.79, 1.91,
          1.88, 1.79, 1.78, 1.89, 1.86, 1.81)
CP4 <- c(15.68, 15.84, 14.47, 14.96, 18.97, 19.04, 17.65, 16.76, 22.11, 22.03, 20.43,
         20.36, 25.29, 25.29, 24.27, 23.99, 28.34, 28.38)
RES4 <- ratioPar(group = GROUP4, effVec = EFF4, cpVec= CP4, refmean = TRUE)

## Compare to REST software using the data from the
## REST 2008 manual (http://rest.gene-quantification.info/)
cp.rc <- c(26.74, 26.85, 26.83, 26.68, 27.39, 27.03, 26.78, 27.32)
cp.rs <- c(26.77, 26.47, 27.03, 26.92, 26.97, 26.97, 26.07, 26.3, 26.14, 26.81)
cp.gc <- c(27.57, 27.61, 27.82, 27.12, 27.76, 27.74, 26.91, 27.49)
cp.gs <- c(24.54, 24.95, 24.57, 24.63, 24.66, 24.89, 24.71, 24.9, 24.26, 24.44)
eff.rc <- rep(1.97, 8)
eff.rs <- rep(1.97, 10)
eff.gc <- rep(2.01, 8)
eff.gs <- rep(2.01, 10)
CP5 <- c(cp.rc, cp.rs, cp.gc, cp.gs)
EFF5 <- c(eff.rc, eff.rs, eff.gc, eff.gs)
GROUP5 <- rep(c("r1c1", "r1s1", "g1c1", "g1s1"), c(8, 10, 8, 10))
RES5 <- ratioPar(group = GROUP5, effVec = EFF5, cpVec = CP5)
RES5$resDat
```

refmean *Averaging of multiple reference genes*

Description

This function averages the expression of several reference genes before calculation of gene expression ratios by ratiocalc or ratiobatch. The method is similar to that described in Vandesompele *et al.* (2002), but uses **arithmetic** averaging of threshold cycles/efficiencies and **not geometric** averaging of relative expression values. This is equivalent, as discussed in 'Details' and as shown in 'Examples'. An essential extension of this method is, that if replicates for the reference genes are supplied, the threshold cycles and efficiencies are subjected to error propagation prior to ratio calculation. The propagated error is then included in the calculation of the gene expression ratios, as advocated in Nordgard *et al.* (2006).

Usage

```
refmean(data, group, which.eff = c("sig", "sli", "exp", "mak", "ext"),
        type.eff = c("individual", "mean.single"),
        which.cp = c("cpD2", "cpD1", "cpE", "cpR", "cpT", "Cy0", "ext"),
        verbose = TRUE, ...)
```

Arguments

data	multiple qPCR data generated by pcrbatch.
group	a character vector defining the replicates (if any) of control/treatment samples and reference genes/genes-of-interest. See 'Details'
which.eff	efficiency calculated by which method. Defaults to sigmoidal fit. Can also be a value such as 1.8, as shown in 'Examples'. See ratiocalc.
type.eff	using individual or averaged efficiencies for the replicates of a reference gene. See 'Details'.
which.cp	type of threshold cycles to be used for the analysis. Defaults to cpD2. See ratiocalc.
verbose	logical. If TRUE, the steps of analysis are shown in the console window.
...	parameters to be supplied to propagate.

Details

As in ratiobatch, the samples are to be defined as a character vector in the style of "g1s1", "g1c1", "r1s1" and "r1c1" etc. If refmean is used as a standalone function or switched on in ratiobatch using refmean = TRUE, different reference genes per control/treatment samples are averaged when supplied either as single runs or as replicates.

Examples (omitting genes-of-interest in control/treatment samples):
2 reference genes, 2 replicates each:
c("r1s1", "r1s1, "r2s1", "r2s1", "r1c1", "r1c1, "r2c1", "r2c1", ...).

3 reference genes, no replicates:
c("r1s1", "r2s1, "r3s1", "r1c1", "r2c1, "r3c1", ...)

Averaging of multiple reference genes is accomplished the following way:
Given i reference genes with j replicates in a sample, all replicates r_{ij} are used for calculating mean μ_{r_i} and standard deviation σ_{r_i} of the threshold cycles and efficiencies. The overall (grand) mean μ_r and propagated error σ_r is calculated using `propagate` with first-order Taylor expansion including covariance: $\sigma_r = F_{r_i} C_{r_i} F_{r_i}^T$. Finally, a vector of length $L = n(r_{ij})$ containing equidistant numbers $X = (x_1, x_2, x_3, \dots x_L)$ with mean μ_r and standard deviation σ_r is generated for a new overall reference gene r_1. This is done using the internal function `qpcR:::makeStat` which calculates a shifted (μ_r) and scaled (σ_r) Z-transformation on a vector $x_1 \dots x_L$:

$$Z_i = \mu_r + \frac{(x_i - \bar{X})}{\sigma_X} \cdot \sigma_r$$

The new Z_i threshold cycle and efficiency values replace all values of r_{ij} in `data`. When using `ratiobatch`, this modified data is then used for the ratio calculation, again using `propagate` to calculate errors for ratios using the Z_i values as mentioned above.
By using logarithmic identities (https://en.wikipedia.org/wiki/Logarithmic_identities), it can be shown that the **geometric** mean can be transformed to the **arithmetic** mean by logarithmation (assuming constant E):

$$\left(\prod_{i=1}^{n} E^{x_i}\right)^{\frac{1}{n}} = \frac{1}{n} \cdot \log_E \left(\prod_{i=1}^{n} E^{x_i}\right) = \frac{1}{n} \sum_{i=1}^{n} x_i$$

Hence, **arithmetic** averaging of the threshold cycles **BEFORE** ratio calculation is the same as doing **geometric** averaging on relative quantities **AFTER** ratio calculation. This is demonstrated in 'Examples' and also mentioned in the geNorm manual (https://www.gene-quantification.com/geNorm_manual.pdf) in Q8 (page 12).

When setting `type.eff = "individual"` (default), all efficiencies from replicates of a reference gene in a control/treatment sample $E(r_{ij})$ are used for calculating mean $\mu_{E(r_i)}$ and standard deviation $\sigma_{E(r_i)}$, the latter being used for calculating a propagated error for all reference gene efficiencies $\sigma_{E(r)}$. If `type.eff = "mean.single"`, all $E(r_{ij})$ values from the replicates are set to the same value $\mu_{E(r_i)}$, that is, there is no variation assumed between the different $E(r_{ij})$. In this case, $\sigma_{E(r_i)} = 0$, so that no error of the replicates is propagated to $\sigma_{E(r)}$. This results in smaller overall errors of the output, but it can be debated if this is a realistic approach, hence both settings were implemented.

`which.eff` can be supplied with an efficiency value such as `1.8`, which is then used as the efficiency for all reference runs $E(r_{ij})$.

Value

The same dataset `data` which was supplied to the function, but with modified threshold cycle/efficiency values in which the values are created per sample in a way, that they have the mean of all averaged reference genes and the same standard deviation as obtained by error propagation. See 'Details' for a more thorough explanation. Furthermore, a modified label vector `"NAME_mod"` is written to the global environment (if `"NAME"` was supplied for `group`) in which the different reference gene labels are aggregated, i.e. c("r1c1", "r2c1", "r3c1") will become c("r1c1", "r1c1", "r1c1"). This new label vector is also attached as an attribute to the output data and can be obtained by `attr(RES1, "group")`.

Note

The function checks stringently if the same number of different reference genes are used for control and treatment samples, although the number of replicates may differ.
Example:
GROUP <- c("r1c1", "r1c1", "r2c1", "r2c1", "r1s1", "r2s1") will work (2 reference genes in control/treatment samples), but GROUP <- c("r1c1", "r2c1", "r3c1", "r1s1", "r1s1", "r1s2", "r1s2", "r2s1", "r2s1") will not work (3 reference genes in controls, only 2 in treatment samples). Also, when no or only one reference genes are detected, the original data is not averaged and returned unchanged.

Author(s)

Andrej-Nikolai Spiess

References

Accurate normalization of real-time quantitative RT-PCR data by geometric averaging of multiple internal control genes.
Vandesompele J, De Preter K, Pattyn F, Poppe B, Van Roy N, De Paepe A, Speleman F.
Genome Biol (2002), **3**: research0034-research0034.11.

Error propagation in relative real-time reverse transcription polymerase chain reaction quantification models: the balance between accuracy and precision.
Nordgard O, Kvaloy JT, Farmen RK, Heikkil? R.
Anal Biochem (2006), **356**: 182-193.

See Also

In ratiobatch, reference gene averaging can be done automatically by setting `refmean = TRUE`. See there.

Examples

```
## Replacing the reference gene values by
## averaged ones in the original data.
## => RES1 is new dataset.
## => GROUP1_mod in global environment is
## new labeling vector.
DAT1 <- pcrbatch(reps, fluo = 2:19, model = l5)
GROUP1 <- c("r1c1", "r1c1", "r2c1", "r2c1", "g1c1", "g1c1",
            "r1s1", "r1s1", "r1s2", "r1s2", "r2s1", "r2s1",
            "r2s2", "r2s2", "g1s1", "g1s1", "g1s2", "g1s2")
RES1 <- refmean(DAT1, GROUP1, which.eff = "sig", which.cp = "cpD2")

## Using three reference genes without replicates
## and then 'ratiobatch'.
## This can also be called in 'ratiobatch' directly
## with parameter 'refmean = TRUE'. See there.
## In this example, already averaged dataset and
```

```
## new labeling vector are supplied to 'ratiobatch',
## so one has to set 'refmean = FALSE'.
DAT2 <- pcrbatch(reps, fluo = 2:9, model = l5)
GROUP2 <- c("r1c1", "r2c1", "r3c1", "g1c1", "r1s1", "r2s1", "r3s1", "g1s1" )
RES2 <- refmean(DAT2, GROUP2, which.eff = "sig", which.cp = "cpD2")
ratiobatch(RES2, GROUP2, refmean = FALSE)

## Comparison between 'refmean' ct-value arithmetic averaging
## and 'geNorm' relative quantities geometric averaging
## using data from the geNorm manual (2008), page 6.
## We will use HK1-HK3 as in the manual (no replicates).
## First we create a 'pcrbatch' dataset and then
## override the ct values with those of the manual and all
## efficiencies with E = 2. Sample A is considered as control sample.
DAT3 <- pcrbatch(reps, fluo = 2:17, model = l5)
DAT3[8, -1] <- c(32.10, 27.00, 34.90, 23.00,
                 33.30, 28.40, 36.10, 24.20,
                 31.00, 27.50, 34.00, 26.35,
                 30.50, 28.20, 33.00, 25.45)
DAT3[1, -1] <- 2
GROUP3 <- c("r1c1", "r2c1", "r3c1", "g1c1",
            "r1s1", "r2s1", "r3s1", "g1s1",
            "r1s2", "r2s2", "r3s2", "g1s2",
            "r1s3", "r2s3", "r3s3", "g1s3")
RES3 <- refmean(DAT3, GROUP3, which.eff = "sig", which.cp = "cpD2")
ratiobatch(RES3, GROUP3, which.cp = "cpD2",
           which.eff = "sig", refmean = FALSE)
```

`replist` *Amalgamation of single data models into a model containing replicates*

Description

Starting from a 'modlist' containing qPCR models from single data, `replist` amalgamates the models according to the grouping structure as defined in `group`. The result is a 'replist' with models obtained from fitting the replicates by `pcrfit`. A kinetic outlier detection and removal option is included.

Usage

```
replist(object, group = NULL, check = "none",
        checkPAR = parKOD(), remove = c("none", "KOD"),
        names = c("group", "first"), doFit = TRUE, opt = FALSE,
        optPAR = list(sig.level = 0.05, crit = "ftest"),
        verbose = TRUE, ...)
```

Arguments

object	an object of class 'modlist'.
group	a vector defining the replicates for each group.
check	which method to use for kinetic outlier detection. Either none or any of the methods in KOD.
checkPAR	parameters to be supplied to the check method, see KOD.
remove	which runs to remove. Either none or those that failed from the method defined in check.
names	how to name the grouped fit. Either 'group_1, ...' or the first name of the replicates.
doFit	logical. If set to FALSE, the replicate data is only aggregated without doing a refitting. See 'Details'.
opt	logical. Should model selection be applied to the final model?
optPAR	parameters to be supplied to mselect.
verbose	if TRUE, the analysis is printed to the console.
...	other parameters to be supplied to mselect.

Details

As being defined by group, the 'modlist' is split into groups of runs and these amalgamated into a nonlinear model. Runs which have failed to be fitted by modlist are automatically removed and group is updated (that is, the correpsonding entries also removed) prior to fitting the replicate model by pcrfit. Model selection can be applied to the final replicate model by setting opt = TRUE and changing the parameters in optPAR. If check is set to any of the methods in "KOD", kinetic outliers are identified and optionally removed, if remove is set to "KOD".
If doFit = FALSE, the replicate data is only aggregated and no refitting is done. This is useful when plotting replicate data by some grouping vector. See 'Examples'.

Value

An object of class 'replist' containing the replicate models of class 'nls'/'pcrfit'.

Author(s)

Andrej-Nikolai Spiess

See Also

modlist, pcrfit.

Examples

```
## Convert 'modlist' into 'replist'.
ml1 <- modlist(reps, model = l4)
rl1 <- replist(ml1, group = gl(7, 4))
plot(rl1)
```

```
summary(rl1[[1]])

## Optimize model based on Akaike weights.
rl2 <- replist(ml1, group = gl(7, 4), opt = TRUE,
               optPARS = list(crit = "weights"))
plot(rl2)


## Remove kinetic outliers,
## use first replicate name for output.
ml3 <- modlist(reps, model = l4)
rl3 <- replist(ml3, group = gl(7, 4), check = "uni1",
               remove = "KOD", names = "first")
plot(rl3, which = "single")

## Just aggregation and no refitting.
ml4 <- modlist(reps, model = l4)
rl4 <- replist(ml4, group = gl(7, 4), doFit = FALSE)
plot(rl4, which = "single")

## Scenario 1:
## automatic removal of runs that failed to
## fit during 'modlist' by using 'testdat' set.
ml5 <- modlist(testdat, model = l5)
rl5 <- replist(ml5, gl(6, 4))
plot(rl5, which = "single")

## Scenario 2:
## automatic removal of runs that failed to
## fit during 'replist':
## samples F3.1-F3.4 is set to 1.
dat1 <- reps
ml6 <- modlist(dat1)
ml6[[9]]$DATA[, 2] <- 1
ml6[[10]]$DATA[, 2] <- 1
ml6[[11]]$DATA[, 2] <- 1
ml6[[12]]$DATA[, 2] <- 1
rl6 <- replist(ml6, gl(7, 4))
plot(rl6, which = "single")
```

`resplot` *An (overlayed) residuals barplot*

Description

A plotting function which displays a barplot of the (standardized) residuals. The bars are colour-coded with heat colours proportional to the residual value. As default, the residuals are displayed together with the points of the fitted PCR curve.

Usage

```
resplot(object, overlay = TRUE, ylim = NULL, ...)
```

Arguments

object	an object of class 'pcrfit.
overlay	logical. If TRUE, the residuals are plotted on top of the fitted curve, else alone.
ylim	graphical ylim values for tweaking the scale and position of the barplot overlay.
...	any other parameters to be passed to barplot.

Details

If replicate data is present in the fitted curve, the residuals from all replicates i, j are summed up from the absolute values: $Y_i = \sum |\hat{\varepsilon}_{i,j}|$.

Value

A plot as described above.

Author(s)

Andrej-Nikolai Spiess

Examples

```
## Create l5 model and plot
## standardized residuals.
m1 <- pcrfit(reps, 1, 2, l5)
resplot(m1)

## Using replicates.
m2 <- pcrfit(reps, 1, 2:5, l5)
resplot(m2)
```

---

resVar *Residual variance of a fitted model*

Description

Calculates the residual variance for objects of class nls, lm, glm, drc or any other models from which coef and residuals can be extracted.

Usage

```
resVar(object)
```

Arguments

object	a fitted model.

Details

$$resVar = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n - p}$$

where n is the number of response values and p the number of parameters in the model.

Value

The residual variance of the fit.

Author(s)

Andrej-Nikolai Spiess

Examples

```
m1 <- pcrfit(reps, 1, 2, l5)
resVar(m1)
```

RMSE *Root-mean-squared-error of a fitted model*

Description

Calculates the root-mean-squared-error (RMSE) for objects of class `nls`, `lm`, `glm`, `drc` or any other models from which `residuals` can be extacted.

Usage

```
RMSE(object, which = NULL)
```

Arguments

object	a fitted model.
which	a subset of the curve to be used for RMSE calculation. If not defined, the complete curve is used.

Details

$$RMSE = \sqrt{(y_i - \hat{y}_i)^2}$$

Value

The root-mean-squared-error from the fit or a part thereof.

Author(s)

Andrej-Nikolai Spiess

Examples

```
## For a curve subset.
m1 <- pcrfit(reps, 1, 2, l5)
RMSE(m1, 10:15)
```

`Rsq` *R-square value of a fitted model*

Description

Calculates the R^2 value for objects of class `nls`, `lm`, `glm`, `drc` or any other models from which `fitted` and `residuals` can be extracted. Since version 1.2-9 it calculates a weighted R^2 if the object has an item `object$weights` containing weighting values.

Usage

```
Rsq(object)
```

Arguments

`object`	a fitted model.

Details

Uses the most general definition of R^2:

$$R^2 \equiv 1 - \frac{RSS}{TSS}$$

where

$$RSS = \sum_{i=1}^{n} w_i \cdot (y_i - \hat{y}_i)^2$$

and

$$TSS = \sum_{i=1}^{n} w_i \cdot (y_i - \bar{y})^2$$

using the weighted mean

$$\bar{y} = \frac{\sum_{i=1}^{n} w_i x_i}{\sum_{i=1}^{n} w_i}$$

Value

The R^2 value of the fit.

Author(s)

Andrej-Nikolai Spiess

Examples

```
m1 <- pcrfit(reps, 1, 2, l5)
Rsq(m1)
```

`Rsq.ad` *Adjusted R-square value of a fitted model*

Description

Calculates the adjusted R^2_{adj} value for objects of class `nls`, `lm`, `glm`, `drc` or any other models from which `fitted`, `residuals` and `coef` can be extracted.

Usage

```
Rsq.ad(object)
```

Arguments

`object`	a fitted model.

Details

$$R^2_{adj} = 1 - \frac{n-1}{n-p} \cdot (1 - R^2)$$

with n = sample size, p = number of parameters.

Value

The adjusted R^2_{adj} value of the fit.

Author(s)

Andrej-Nikolai Spiess

Examples

```
## Single model.
m1 <- pcrfit(reps, 1, 2, l7)
Rsq.ad(m1)

## Compare different models with increasing
## number of parameters.
ml1 <- lapply(list(l4, l5, l6), function(x) pcrfit(reps, 1, 2, x))
sapply(ml1, function(x) Rsq.ad(x))
```

RSS *Residual sum-of-squares of a fitted model*

Description

Calculates the residual sum-of-squares for objects of class `nls`, `lm`, `glm`, `drc` or any other models from which `residuals` can be extacted. From version 1.3-6, this function uses weights, if `object` has an item `$weights`.

Usage

```
RSS(object)
```

Arguments

`object`	a fitted model.

Details

$$RSS = \sum_{i=1}^{n} w_i \cdot (y_i - \hat{y}_i)^2$$

Value

The (weighted) residual sum-of-squares from the fit.

Author(s)

Andrej-Nikolai Spiess

Examples

```
m1 <- pcrfit(reps, 1, 2, l5)
RSS(m1)
```

`sliwin` *Calculation of qPCR efficiency by the 'window-of-linearity' method*

Description

A linear model of Cycles versus log(Fluorescence) is fit within a sliding window of defined size(s) and within a defined border. Regression coefficients are calculated for each window, and at the point of maximum regression (log-linear range) or least variation in slope, parameters such as PCR efficiency and initial template fluorescence are calculated. From version 1.3-5, an approach "not unlike" to Ruijter et al. (2009) has been implemented, in which baseline values can be iteratively subtracted from the data prior to fitting the sliding window. See 'Details' for more information.

Usage

```
sliwin(object, wsize = 6, basecyc = 1:6, base = 0, border = NULL,
       type = c("rsq", "slope"), plot = TRUE, verbose = TRUE, ...)
```

Arguments

`object`	an object of class 'pcrfit'.
`wsize`	the size(s) of the sliding window(s), default is `6`. A sequence such as `4:6` can be used to optimize the window size.
`basecyc`	if `base != 0`, which cycles to use for an initial baseline estimation based on the averaged fluorescence values.
`base`	either `0` for no baseline optimization, or a scalar defining multiples of the standard deviation of all baseline points obtained from `basecyc`. These are iteratively subtracted from the raw data. See 'Details' and 'Examples'.
`border`	either `NULL` (default) or a two-element vector which defines the border from the take-off point to points nearby the upper asymptote (saturation phase). See 'Details'.
`type`	selection of the window with best baseline + maximum R^2 (`"rsq"`) or best baseline + minimal variance in slope + maximum R^2 (`"slope"`).
`plot`	if `TRUE`, the result is plotted with the logarithmized curve, sliding window, regression line and baseline.
`verbose`	logical. If `TRUE`, more information is displayed in the console window.
`...`	only used internally for passing the parameter matrix.

Details

To avoid fits with a high R^2 in the baseline region, some border in the data must be defined. In `sliwin`, this is by default (`base = NULL`) the region in the curve starting at the take-off cycle (top) as calculated from `takeoff` and ending at the transition region to the upper asymptote (saturation region). The latter is calculated from the first and second derivative maxima: $asympt = cpD1 + (cpD1 - cpD2)$. If the border is to be set by the user, `border` values such as `c(-2, 4)` extend these values by $top + border[1]$ and $asympt + border[2]$. The log_{10} transformed raw fluorescence values

are regressed against the cycle number $log_{10}(F) = n\beta + \epsilon$ and the efficiency is then calculated by $E = 10^{slope}$. For the baseline optimization, 100 baseline values Fb_i are interpolated in the range of the data:

$$F_{min} \leq Fb_i \leq base \cdot \sigma(F_{basecyc[1]} ... F_{basecyc[2]})$$

and subtracted from F_n. If `type = "rsq"`, the best window in terms of R^2 is selected from all iterations, as defined by `wsize` and `border`. If `type = "slope"`, the baseline value delivering the smallest variance in the slope of the upper/lower part of the sliding window and highest R^2 is selected. This approach is quite similar to the one in Ruijter et al. (2009) but has to be tweaked in order to obtain the same values as in the 'LinRegPCR' software. Especially the `border` value has significant influence on the calculation of the best window's efficiency value.

Value

A list with the following components:

`eff`	the optimized PCR efficiency found within the sliding window.
`rsq`	the maximum R-squared.
`init`	the initial template fluorescence F0.
`base`	the optimized baseline value.
`window`	the best window found within the `borders`.
`parMat`	a matrix containing the parameters as above for each iteration.

Author(s)

Andrej-Nikolai Spiess

References

Assumption-free analysis of quantitative real-time polymerase chain reaction (PCR) data.
Ramakers C, Ruijter JM, Deprez RH, Moorman AF.
Neurosci Lett (2003), **339**: 62-65.

Amplification efficiency: linking baseline and bias in the analysis of quantitative PCR data.
Ruijter JM, Ramakers C, Hoogaars WM, Karlen Y, Bakker O, van den Hoff MJ, Moorman AF.
Nucleic Acids Res (2009), **37**: e45

Examples

```
## Sliding window of size 5 between
## take-off point and upper asymptote,
## no baseline optimization.
m1 <- pcrfit(reps, 1, 2, l4)
sliwin(m1, wsize = 5)


## Optimizing with window sizes of 4 to 6,
## between 0/+2 from lower/upper border,
## and baseline up to 2 standard deviations.
```

```
sliwin(m1, wsize = 4:6, border = c(0, 2), base = 2)
```

`takeoff` *Calculation of the qPCR takeoff point*

Description

Calculates the first significant cycle of the exponential region (takeoff point) using externally studentized residuals as described in Tichopad *et al.* (2003).

Usage

```
takeoff(object, pval = 0.05, nsig = 3)
```

Arguments

`object`	an object of class 'pcrfit'.
`pval`	the p-value for the takeoff test.
`nsig`	the number of successive takeoff tests. See 'Details'.

Details

Takeoff points are calculated essentially as described in the reference below. The steps are:

1) Fitting a linear model to background cycles $1:n$, starting with $n = 5$.
2) Calculation of the external studentized residuals using `rstudent`, which uses the hat matrix of the linear model and leave-one-out:

$$\langle\hat{\varepsilon}_i\rangle = \frac{\hat{\varepsilon}_i}{\hat{\sigma}_{(i)}\sqrt{1-h_{ii}}}, \hat{\sigma}_{(i)} = \sqrt{\frac{1}{n-p-1}\sum_{\substack{j=1\\ j\neq i}}^{n}\hat{\varepsilon}_j^2}$$

with h_{ii} being the ith diagonal entry in the hat matrix $H = X(X^TX)^{-1}X^T$.
3) Test if the last studentized residual $\langle\hat{\varepsilon}_n\rangle$ is an outlier in terms of t-distribution:

$$1 - pt(\langle\hat{\varepsilon}_n\rangle, n-p) < 0.05$$

with n = number of residuals and p = number of parameters.
4) Test if the next `nsig` - 1 cycles are also outlier cycles.
5) If so, take cycle number from 3), otherwise $n = n + 1$ and start at 1).

Value

A list with the following components:

`top`	the takeoff point.
`f.top`	the fluorescence at `top`.

Author(s)

Andrej-Nikolai Spiess

References

Standardized determination of real-time PCR efficiency from a single reaction set-up.
Tichopad A, Dilger M, Schwarz G & Pfaffl MW.
Nucleic Acids Research (2003), **e122**.

Examples

```
m1 <- pcrfit(reps, 1, 2, l5)
res1 <- takeoff(m1)
plot(m1)
abline(v = res1$top, col = 2)
abline(h = res1$f.top, col = 2)
```

---

update.pcrfit *Updating and refitting a qPCR model*

Description

Updates and re-fits a model of class 'pcrfit'.

Usage

```
## S3 method for class 'pcrfit'
update(object, ..., evaluate = TRUE)
```

Arguments

`object`	a fitted model of class 'pcrfit'.
`...`	arguments to alter in object.
`evaluate`	logical. If `TRUE`, model is re-fit; otherwise an unevaluated call is returned.

Value

An updated model of class 'pcrfit' and 'nls'.

Author(s)

Andrej-Nikolai Spiess

See Also

The function `pcrfit` in this package.

Examples

```
m1 <- pcrfit(reps, 1, 2, l4)

## Update model.
update(m1, model = l5)

## Update qPCR run.
update(m1, fluo = 20)

## Update data.
update(m1, data = guescini1)
```

Index

∗ **distribution**
propagate, 56
∗ **documentation**
qpcR.news, 62
∗ **htest**
propagate, 56
∗ **models**
AICc, 3
akaike.weights, 4
calib, 5
Cy0, 7
eff, 8
efficiency, 10
evidence, 13
expcomp, 14
expfit, 15
fitchisq, 17
getPar, 18
is.outlier, 20
KOD, 21
llratio, 23
LOF.test, 24
LRE, 26
maxRatio, 28
meltcurve, 29
midpoint, 32
modlist, 33
mselect, 37
parKOD, 39
pcrbatch, 40
pcrboot, 42
pcrfit, 44
pcrGOF, 47
pcropt1, 48
pcrsim, 49
plot.pcrfit, 51
predict.pcrfit, 53
PRESS, 54
qpcR_datasets, 62
qpcR_functions, 67
replist, 88
resplot, 90
resVar, 91
RMSE, 92
Rsq, 93
Rsq.ad, 94
RSS, 95
sliwin, 96
takeoff, 98
update.pcrfit, 99
∗ **nonlinear**
AICc, 3
akaike.weights, 4
calib, 5
Cy0, 7
eff, 8
efficiency, 10
evidence, 13
expcomp, 14
expfit, 15
fitchisq, 17
getPar, 18
is.outlier, 20
KOD, 21
llratio, 23
LOF.test, 24
LRE, 26
maxRatio, 28
meltcurve, 29
midpoint, 32
modlist, 33
mselect, 37
parKOD, 39
pcrbatch, 40
pcrboot, 42
pcrfit, 44
pcrGOF, 47
pcropt1, 48

pcrsim, 49
plot.pcrfit, 51
predict.pcrfit, 53
PRESS, 54
qpcR_datasets, 62
qpcR_functions, 67
ratiobatch, 71
ratiocalc, 76
ratioPar, 81
refmean, 85
replist, 88
resplot, 90
resVar, 91
RMSE, 92
Rsq, 93
Rsq.ad, 94
RSS, 95
sliwin, 96
takeoff, 98
update.pcrfit, 99
∗ **utilities**
qpcR.news, 62

AIC, *3*, *5*, *24*
AICc, 3
akaike.weights, 4, *38*
anova, *38*
arima, *35*

b4 *(*qpcR_functions*)*, 67
b5 *(*qpcR_functions*)*, 67
b6 *(*qpcR_functions*)*, 67
b7 *(*qpcR_functions*)*, 67
barplot, *91*
batsch1 *(*qpcR_datasets*)*, 62
batsch2 *(*qpcR_datasets*)*, 62
batsch3 *(*qpcR_datasets*)*, 62
batsch4 *(*qpcR_datasets*)*, 62
batsch5 *(*qpcR_datasets*)*, 62
boggy *(*qpcR_datasets*)*, 62

calib, 5
cm3 *(*qpcR_functions*)*, 67
coef, *91*, *94*
coefficients, *3*
competimer *(*qpcR_datasets*)*, 62
Cy0, 7

dil4reps94 *(*qpcR_datasets*)*, 62
dyemelt *(*qpcR_datasets*)*, 62

eff, 8, *10*, *12*, *28*
efficiency, 10, *16*, *19*, *21*, *40–43*, *48*, *76*
evidence, 13
expcomp, 14
expfit, *14*, 15, *19*, *21*, *40*
expGrowth, *15*
expGrowth *(*qpcR_functions*)*, 67
expSDM *(*qpcR_functions*)*, 67

file.show, *62*
filter, *9*, *28*
fitchisq, 17, *37*, *38*, *47*
fitted, *93*, *94*

getPar, 18
guescini1 *(*qpcR_datasets*)*, 62
guescini2 *(*qpcR_datasets*)*, 62

htPCR *(*qpcR_datasets*)*, 62

integrate, *31*
is.outlier, 20, *22*, *23*

karlen1 *(*qpcR_datasets*)*, 62
karlen2 *(*qpcR_datasets*)*, 62
karlen3 *(*qpcR_datasets*)*, 62
KOD, *20*, 21, *34*, *39*, *40*, *65*, *89*

l4 *(*qpcR_functions*)*, 67
l5, *65*
l5 *(*qpcR_functions*)*, 67
l6 *(*qpcR_functions*)*, 67
l7 *(*qpcR_functions*)*, 67
lievens1 *(*qpcR_datasets*)*, 62
lievens2 *(*qpcR_datasets*)*, 62
lievens3 *(*qpcR_datasets*)*, 62
lin2 *(*qpcR_functions*)*, 67
lines, *51*
linexp, *15*
linexp *(*qpcR_functions*)*, 67
llratio, 23, *38*
lm, *31*
LOF.test, 24
logLik, *3*, *5*, *23*, *24*
lowess, *35*
LRE, 26, *40*

mahalanobis, *21*, *22*

mak2 (qpcR_functions), 67
mak2i (qpcR_functions), 67
mak3 (qpcR_functions), 67
mak3i (qpcR_functions), 67
maxRatio, *9*, *11*, 28
meltcurve, 29
midpoint, 32
modlist, *22*, 33, *40–42*, *45*, *89*
mselect, *34*, 37, *41*, *48*, *89*

optim, *45*

parKOD, *21, 22*, 39
pcrbatch, *18*, *36*, 40, *72, 73*, *76*, *78*, *85*
pcrboot, 42
pcrfit, *34*, 44, *45*, *49*, *69*, *88, 89*, *99*
pcrGOF, 47, *48*
pcropt1, 48
pcrsim, 49
plot, *28*, *30*, *49*, *51*
plot.pcrfit, *7*, *10*, 51
points, *7*, *51*
predict, *54*
predict.lm, *31*
predict.pcrfit, 53
PRESS, *47*, *49*, 54
princomp, *22*
propagate, 56, *76–78*, *85, 86*

qpcR.news, 62
qpcR_datasets, 62
qpcR_functions, 67

ratiobatch, *34, 35*, *40*, 71, *77*, *81, 82*, *85–87*
ratiocalc, *35*, *41*, *60*, *72, 73*, 76, *77*, *81, 82*, *85*
ratioPar, *76*, *79*, 81
refmean, *74*, *83*, 85
replist, *45*, 88
reps (qpcR_datasets), 62
reps2 (qpcR_datasets), 62
reps3 (qpcR_datasets), 62
reps384 (qpcR_datasets), 62
residuals, *3*, *91–95*
resplot, 90
resVar, 91
RMSE, 92
Rsq, 93
Rsq.ad, 94
RSS, 95
rstudent, *98*
rutledge (qpcR_datasets), 62

sliwin, *19*, *21*, *26*, *40*, 96
smooth.spline, *35*
spl3 (qpcR_functions), 67
splinefun, *31*
supsmu, *31*, *35*

takeoff, *16*, *26*, *96*, 98
testdat (qpcR_datasets), 62
tryCatch, *18*

uniroot, *70*
update, *54*
update.pcrfit, 99

vermeulen1 (qpcR_datasets), 62
vermeulen2 (qpcR_datasets), 62