`getConfidenceInterval` *Compute Confidence Intervals for a Model Fit to Dilution Series*

Description

This function computes confidence intervals for the estimated concentrations in a four-parameter logistic model fit to a set of dilution series in a reverse-phase protein array experiment.

Usage

```
getConfidenceInterval(result, alpha = 0.1, nSim = 50)
```

Arguments

`result`	A `RPPAFit` object representing the result of fitting a four-parameter logistic model
`alpha`	The desired significance of the confidence interval; the width of the resulting interval is 1 - alpha.
`nSim`	The number of times to resample the data in order to estimate the confidence intervals.

Details

In order to compute the confidence intervals, the function assumes that the errors in the observed Y intensities are independent normal values, with mean centered on the estimated curve and standard deviation that varies smoothly as a function of the (log) concentration. The smooth function is estimated using `loess`. The residuals are resampled from this estimate and the model is refit; the confidence intervals are computed empirically as symmetrically defined quantiles of the refit parameter sets.

Value

An object of the `RPPAFit` class, containing updated values for the slots `lower`, `upper`, and `conf.width` that describe the confidence interval.

Author(s)

Kevin R. Coombes <kcoombes@mdanderson.org>

References

KRC

See Also

`RPPAFit-class`, `RPPAFit`

Examples

```
path <- system.file("rppaCellData", package="SuperCurve")
akt <- RPPA("Akt.txt", path=path)
design <- RPPADesign(akt, grouping="blockSample",
                     controls=list("neg con", "pos con"))
fit.nls <- RPPAFit(akt, design, "Mean.Net")
# Warning: this takes a while!
fit.nls <- getConfidenceInterval(fit.nls, alpha=0.10, nSim=50)
```

RPPA-class *The RPPA Class*

Description

The RPPA class represents the raw quantification data from a reverse-phase protein array experiment.

Usage

```
RPPA(filename, path = ".")
## S4 method for signature 'RPPA':
summary(object, ...)
## S4 method for signature 'RPPA':
image(x, measure = "Mean.Net", main = measure, colorbar
= FALSE, col = terrain.colors(256), ...)
```

Arguments

`filename`	The name of a file containing MicroVigene quantifications of a reverse-phase protein array experiment.
`path`	An optional argument giving the path from the current directory to the file. The default value assumes the file is contained in the current direectory.
`object`	An RPPA object.
`x`	An RPPA object.
`measure`	A character string containing the name of the measurement column in `data` that should be displayed by the `image` method.
`main`	A character string used to title the image plot
`colorbar`	A logical value that determines whether to include a color bar in the plot. Default is FALSE.
`col`	The usual graphics parameter used by image. It is included here to change the default color scheme to use terrain.colors.
`...`	The usual extra arguments for generic or plotting routines.

Details

The data frame slot (`data`) in a valid RPPA object constructed from a MicroVigene input file using the `RPPA` function is guaranteed to contain at least 6 columns of information: `Main.Row`, `Main.Col`, `Sub.Row`, `Sub.Col`, `Sample`, and `Mean.Net`. The first four pieces of information give the logical location of a spot on an array, after which we get a unique identifier of the sample spotted at that location and a measurement that represents the background-corrected mean intensity of the spot. Additional columns may be included or may be added later.

Value

The `RPPA` constructor returns an object of the `RPPA` class.

The `summary` method returns a summary of the underlying data frame.

The `image` method invisibly returns the `RPPA` object on which it was invoked.

Objects from the Class

Although objects of the class can be created by a direct call to new, the preferred method is to use the `RPPA` function.

Slots

`data`: A data.frame containing the contents of a MicroVigene or other quantification file

`file`: A character string: the name of the file that the data was loaded from

Methods

summary(object, ...) The summary method prints a summary of the underlying data frame.

image(x, measure="Mean.Net", main=measure, colorbar=FALSE, col=terrain.colors(256), ...) The image method produces a "geographic" image of the measurement column named by the `measure` argument. The colors in the image represent the intensity of the measurement at each spot on the array, and the display locations match the row and column locations of the spot. Any measurement column can be displayed using this function. An optional color bar can be added; this will be placed at the right edge.

Author(s)

Kevin R. Coombes <kcoombes@mdanderson.org>

References

KRC

See Also

`RPPAFit`, `RPPADesign`

Examples

```
path <- system.file("rppaTumorData", package="SuperCurve")
erk2 <- RPPA("ERK2.txt", path=path)
summary(erk2)
image(erk2)
image(erk2, colorbar=TRUE)
image(erk2, "Vol.Bkg", main="Background Estimates", colorbar=TRUE)
```

`rppaCellData` *AKT, ERK2, and CTNNB1 expression in cell lines*

Description

This data set contains the expression levels of three proteins: AKT, ERK2, and beta catenin (CTNNB1) in 40 cell lines, measured in duplicate dilution series using reverse-phase protein arrays.

The data set also contains a description of the design of the reverse-phase protein array used in a set of experiments to measure protein levels in 40 different cell lines. Cell lysates were spotted on the array in duplicate in eight-step dilution series. The layout of the array consisted of a 4x10 grid of 4x4 subgrids. Each subgrid contained the duplicate dilution series for a single sample. Each of the identical top two rows of a subgrid contained the four most concentrated dilution steps (in decreasing concentrations from left to right), and the identical bottom two rows contained the four least concentrated dilution steps.

Usage

```
data(rppaCell)
```

Format

The objects `akt`, `c.erk2`, and `ctnnb1` are objects of the `RPPA` class. The object `design40` is an object of the `RPPADesign` class.

Source

Bryan Hennessey and Gordon Mills

References

KRC

RPPADesign-class *The RPPADesign Class*

Description

This class represents the information that describes how a particular set of RPPA slides was designed.

Usage

```
RPPADesign(raw, steps = NULL, series = NULL,
  grouping = c("byRow","byCol", "bySample", "blockSample"),
  ordering = c("decreasing","increasing"),
  alias = NULL, center = FALSE, controls = list())
seriesNames(design)
getSteps(design)
## S4 method for signature 'RPPADesign':
image(x, ...)
## S4 method for signature 'RPPADesign':
summary(object, ...)
## S4 method for signature 'RPPADesign':
names(x)
```

Arguments

raw	A data frame or an RPPA object.
steps	An optional numeric vector listing the dilution step associated with each spot, on a logarithmic scale.
series	An optional character vector or factor identifying the dilution series to which each spot corresponds.
grouping	Describes the way dilution series are oriented on the array.
ordering	Are dilution series arranges in order of `increasing` or `decreasing` concentrations. Default is `decreasing`.
alias	A data frame containing two columns: Alias and Sample
center	A logical value: if TRUE, then dilution steps are centered around 0.
controls	A list containing the character strings that identify control spots on the array.
x	A RPPADesign object
object	A RPPADesign object
design	A RPPADesign object
...	The usual extra arguments for generic or plotting routines.

Details

From their inception, reverse-phase protein array experiments have spotted samples on the array in dilution series. Thus, a critical aspect of the design and analysis is to understand how the dilution series are placed on the array.

The optional `grouping` and `ordering` arguments allow the user to specify several standard layouts without having to go into great detail. The most common layout is `byRow`, which indicates that each row of a subgrid on the array should be considered as a separate dilution series. Although considerably less common (for reasons related to the robotics of how arrays are printed), the `byCol` layout indicates that each column of a subgrid is its own dilution series. The `bySample` layout means that each unique sample name indicates its own dilution series. Finally, the `blockSample` layout indicates that all occurrences of a sample name within a subgrid (or block) refer to the same dilution series. The `blockSample` layout can be used, for example, when a dilution series is long enough to extend over more than one row of a subgrid. One layout we have seen used seven dilution steps followed by a control spot, contained in two successive rows of a design with 4x4 subgrids, leading to the pattern:

$$7654$$

$$321C$$

If the design of an RPPA experiment does not follow one of the built-in patterns, you can create an object by supplying vectors of dilution series names (in the `series` argument) and corresponding dilution steps (in the `steps` argument) that explicitly provide the mapping for each spot.

Value

The `image` method invisibly returns the displayed matrix of dilution steps.

The `summary` method returns the summary object of the `layout` data frame.

The `names` method returns a character vector.

The `getSteps` function returns a numeric vector containing, for each non-control spot, the step represented by that spot in its dilution series.

The `seriesNames` function returns a character vector containing the names of the unique (non-control) dilution series on the array.

Objects from the Class

Objects of the RPPADesign class should be constructed using the `RPPADesign` function.

Slots

`layout:` A data frame

`alias:` A data frame

`sampleMap:` A character vector

`controls:` A list containing the character strings that identify control spots on the array. Controls are not included as part of any dilution series.

Methods

image(x, ...) The `image` method produces a two-dimensional graphical display of the layout design. Colors are used to represent different dilution steps, and laid out in the same pattern as the rows and columns of the array. This provides a visual check that the design has been specified correctly.

summary(object, ...) The `summary` method lists the names of the control spots on the array and then prints a summary of the data frame describing the layout.

names(x) The `names` method returns a character vector containing, for each non-control spot, the name of the dilution series to which that spot belongs.

Author(s)

Kevin R. Coombes <kcoombes@mdanderson.org>

References

KRC

See Also

RPPA

Examples

```
path <- system.file("rppaTumorData", package="SuperCurve")
erk2 <- RPPA("ERK2.txt", path=path)
design <- RPPADesign(erk2, grouping="blockSample", center=TRUE)
image(design)
summary(design)

design <- RPPADesign(erk2, grouping="blockSample",
                     controls=list("neg con", "pos con"))
image(design)
summary(design)

path <- system.file("rppaCellData", package="SuperCurve")
akt <- RPPA("Akt.txt", path=path)
# Uses duplicate 8-step dilution series within 4x4 subgrids.
# They are interleaved, with the top two identical rows
# containing the first 4 steps and the bottom two identical
# rows containing the last 4 steps.
steps <- rep(c(rep(8:5, 2), rep(4:1, 2)), 40) - 4.5
rep.temp <- factor(paste('Rep', rep(rep(1:2, each=4), 80), sep=''))
series <- factor(paste(as.character(akt@data$Sample),
                               as.character(rep.temp),
                               sep='.'))
design40 <- RPPADesign(akt, steps=steps, series=series)
image(design40)
summary(design40)
```

RPPAFit-class *The RPPAFit Class*

Description

Objects of the RPPAFit class represent the results of fitting a four-parameter logistic model to the dilution series in a reverse-phase protein array experiment.

Usage

```
## S4 method for signature 'RPPAFit':
summary(object, ...)
## S4 method for signature 'RPPAFit':
coef(object, ...)
## S4 method for signature 'RPPAFit':
coefficients(object, ...)
## S4 method for signature 'RPPAFit':
fitted(object, type=c("Y", "y", "X", "x"), ...)
## S4 method for signature 'RPPAFit':
residuals(object, type=c("raw", "standardized", "linear"), ...)
## S4 method for signature 'RPPAFit':
resid(object, ...)
## S4 method for signature 'RPPAFit':
image(x, type=c("Residuals", "StdRes", "LinRes","X", "Y"), ...)
## S4 method for signature 'RPPAFit':
hist(x, type=c("Residuals", "StdRes", "LinRes"),
  xlab = NULL, main = NULL, ...)
## S4 method for signature 'RPPAFit':
plot(x, y, type=c("cloud", "series", "individual"),
xlab="Log Concentration", ylab="Intensity", colors=NULL, ...)
```

Arguments

object	A RPPAFit object.
x	A RPPAFit object.
type	A list of options describing the type of fitted values, residuals, images, histograms, or plots.
xlab	Graphics parameter; how the x-axis should be labeled.
ylab	Graphics parameter; how the y-axis should be labeled.
main	Character string used as a title for the plot.
y	not used.
colors	A graphical parameter, used only if type='series', to color the lines connecting different dilution series. Eight default colors are used if the argument is NULL.
...	The usual extra arguments for generic or plotting routines.

Details

The `RPPAFit` class holds the results of fitting a four-parameter joint logistic model to all the dilution series on a reverse-phase protein array. For details on how the model is fit, see the `RPPAFit` function. The basic mathematical model is given by

$$Y = \alpha + \beta * g(\gamma * (X + \delta_i)),$$

where Y is the observed intensity and X is the designed dilution step. The heart of the model is the function $g(x) = \frac{e^x}{1+e^x}$, which is the inverse of the logistic function

$$f(x) = \log(p/(1-p)).$$

By fitting a joint model, we assume that the parameters α, β, and γ are the same for all dilution series on the array. The real point of the model, however, is to be able to draw inferences on the δ_i, which represent the (log) concentrations of the protein present in different dilution series.

Value

The `summary` method has no return value.

The `coef` and `coefficients` methods return a named vector of length three.

The `fitted` method returns a numeric vector.

The `resid` and `residuals` methods return a numeric vector.

The `image` method invisibly returns the object `x` on which it was invoked.

The `hist` method returns an object of the `histogram` class.

The `plot` method invisibly returns the object `x` on which it was invoked.

Objects from the Class

Objects should be constructed using the `RPPAFit` function.

Slots

`call`: A call object: the function call that was used to generate this model fit.

`rppa`: The RPPA object containing the raw data that was fit

`design`: The RPPADesign object describing the layout of the array.

`measure`: A character string containing the name of the measurement column in the raw data that was fit by the model.

`method`: A character string containing the name of the method that was used to estimate the upper and lower limit parameters in the model.

`coefficients`: A named list containing the estimates of model parameters alpha, beta, and gamma.

`concentration`: A vector of estimates of the relative log concentration of protein present in each sample.

`lower`: A vector containing the lower bounds on the confidence interval of the log concentration estimates.

`upper`: A vector containing the upper bounds on the confidence interval of the log concentration estimates.

`conf.width`: The width of the confidence interval.

`intensities`: The predicted observed intensity at the estimated concentrations for each dilution series.

`linear.p`: A vector of p-values for how well a linear model fits each dilution series

`ss.ratio`: A statistic measuring the goodness-of-fit for each dilution series, computed as a ratio of two different sums-of-squares.

`warn`: A character vector containing any warnings that arose when trying to fit the model to individual dilution series.

`version`: A character string containing the version of SuperCurve that produced the fit

Methods

summary(object, ...) Print a summary of the `RPPAFit` object. At present, this only reports the function call used to fit the model.

coefficients(object, ...) Extract a named vector (NOT a list) containing the coefficients *alpha*, β, and γ of the model.

coef(object, ...) An alias for `coefficients`.

fitted(object, type=c("Y", "y", "X", "x"), ...) Extract the fitted values of the model. This process is more complictaed than it may seem at first, since we are estimating values on both the X and Y axes. By default, the fitted values are assumed to be the intensities, Y, which are obtained using either an uppercase or lowercase 'y' as the `type` argument. The fitted log concentrations are returned when `type` is set to either uppercase or lowercase 'x'. In the notation used above to describe the model, these fitted values are given by $X_i = X + \delta_i$.

residuals(object, type=c("raw", "standardized", "linear"), ...) Report the residual errors. The 'raw' residuals are defined as the difference between the observed intensities and the fitted intensities, as computed by the `fitted` function. The 'standardized' residuals are obtained by standardizing the raw residuals. The 'linear' residuals, by contrast, arise from a reinterpretation of the model. Converting from the observed intensity scale by a logistic transformation yields

$$f((Y - \alpha)/\beta) = \gamma * (X + \delta_i) = \gamma * X_i.$$

To compute the linear residuals, we use the observed Y values on the left side and the fitted X values on the right side and take the difference.

resid(object, ...) An alias for `residuals`.

image(x, type=c("Residuals", "StdRes", "LinRes","X", "Y"), ...) The `image` method produces a 'geographic' plot of the residuals or of the fitted values, depending on the value of the `type` argument. The implementation reuses code from the `image` method for an `RPPA` object.

hist(x, type=c("Residuals", "StdRes", "LinRes"), xlab = NULL, main = NULL, ...) The `hist` method produces a histogram of the residuals. The exact form of the residuals being displayed depends on the value of the `type` argument.

plot(x, y, type=c("cloud", "series", "individual"), xlab="Log Concentration", ylab="Intensity", ...)
The `plot` method produces a diagnostic plot of the model fit. The default `type`, 'cloud', simply plots the fitted X values against the observed Y values as a cloud of points around the jointly estimated sigmoid curve. The 'series' plot uses different colored lines to join points belonging to the same dilution series. The 'individual' plot produces separate graphs for each dilution series, laying each one alongside the jointly fitted sigmoid curve.

Author(s)

Kevin R. Coombes <kcoombes@mdanderson.org>

References

See Also

RPPAFit, RPPA, RPPADesign, hist

Examples

```
path <- system.file("rppaTumorData", package="SuperCurve")
erk2 <- RPPA("ERK2.txt", path=path)
design <- RPPADesign(erk2, grouping="blockSample",
                     controls=list("neg con", "pos con"))
fit.nls <- RPPAFit(erk2, design, "Mean.Net")
image(fit.nls, measure="Residuals")
image(fit.nls, measure="LinRes")
plot(fit.nls, type="cloud")

fit.q <- RPPAFit(erk2, design, "Mean.Net", method="quantiles")
hist(fit.q, type="StdRes")
plot(fit.q, type="series")

coef(fit.nls)
coef(fit.q)

plot(fitted(fit.q), resid(fit.q))
```

`RPPAFit` *Fit Dilution Curves to Protein Lysate Series*

Description

`RPPAFit` fits a four-parameter logistic model to the dilution series in a reverse-phase protein array experiment. Individual sample concentrations are estimated by matching individual sample dilution series to the overall logistic response for the slide.

Usage

```
RPPAFit(rppa, design, measure, xform=function(x) x,
    method = c("pure", "mixed", "quantiles", "rlm"),
    ci = FALSE, ignoreNegative = TRUE, bayesian = FALSE,
    trace = FALSE, verbose = FALSE, veryVerbose = FALSE,
    warnLevel = 0)
```

Arguments

`rppa`	An RPPA object containing the raw data to be fit.
`design`	A RPPADesign object describing the layout of the array.
`measure`	A character string identifying the column of the raw RPPA data that should be used to fit to the model.
`xform`	(Experimental) A function that takes a single input vector and returns a single output vector of the same length. The `measure` column is transformed using this function before fitting the model. NOT YET IMPLEMENTED.
`method`	optional parameter specifying the method for fitting the parameters `alpha` and `beta`. Default method is `pure`, which simply uses the optimal fit based on nonlinear least squares. Setting `method` to `mixed` uses `nls` to fit the three general model parametrs, but uses `rlm` to fit the sample-specific parameters. Setting `method` to `quantiles` uses the 5th and 95th quantiles from the raw data. Setting `method` to `rlm` tries to refit the values (afer an appropriate transformation) with a robust linear model.
`ci`	A logical value: if TRUE, then compute 90% confidence intervals on the concentration estimates.
`ignoreNegative`	A logical value: if TRUE, then negative values are converted to NA before fitting the model.
`bayesian`	A logical value: if TRUE, we use bayesian methods to estimate per sample values of the lower bound alpha.
`trace`	this is passed to nls in the bayesian portion of the routine.
`verbose`	a logical value; if TRUE, the function prints updates while it is fitting the data.
`veryVerbose`	a logical value; if TRUE, then the function prints voluminous updates as it is fitting each individual dilution series.
`warnLevel`	used to set the `warn` option before calling `rlm`. Since this is wrapped in a `try` function, it won't cause failure but will give us a chance to figure out which dilution series are failing. Setting `warnLevel` to two or greater may change the values returned by the function.

Details

The basic mathematical model is given by

$$Y = \alpha + \beta * g(\gamma * (X + \delta_i)),$$

where Y is the observed intensity and X is the designed dilution step. The heart of the model is the function $g(x) = \frac{e^x}{1+e^x}$, which is the inverse of the logistic function

$$f(x) = \log(p/(1-p)).$$

By fitting a joint model, we assume that the parameters α, β, and γ are the same for all dilution series on the array. The real point of the model, however, is to be able to draw inferences on the δ_i, which represent the (log) concentrations of the protein present in different dilution series.

As the first step in fitting the model, we get crude estimates of the parameters α and β by computing the min and max of the observed intensities Y. We then perform a logistic transformation, working with the values $W = f((Y-\alpha)/\beta)$. We then compute an initial estimate of γ as the median (over all dilution series) of the slope of a robust linear fit to W as a function of the dilution steps X. Initial estimates of the individual δ_i are also computed robustly, conditional on the previously estimated parameters.

The next step depends on which `method` has been specified for model fitting. If `method="pure"` or `method="mixed"`, then we use the non-linear least squares function `nls`. Conditional on the current estimates of the δ_i, we use `nls` to update the estimates of the other three parameters. Then, conditional on the updated values of α, β, and γ, we update the estimates of the δ_i one dilution series at a time. The update uses `nls` when `method="pure"` and uses `rlm` when `method="mixed"`.

If `method='quantiles'`, then we retain quantile estimates of α and β. In this case, we first use `nls` to update the value of γ and then, conditional on that estimate, update the δ_i.

If `method="rlm"`, we first follow the procedure described for `method='nls'`. We follow this by trying to refit the estimates of α and β using a robust linear model with the `rlm` function from the `MASS` package. This computation is peformed conditionally on the estimates of `gamma` and
`delta_i`, in which case the observed intensities Y are linear in the sigmoid-transformed dilution steps X.

The `bayesian` option alters the model by assuming that the baseline level α can be different for each dilution series. The globally estimated α is used as a strong prior, and the individual estimates of *alpha* are shrunk toward the global value. This idea is motivated by the possibility that background levels might be different on different parts of the reverse phase protein array.

If the `ci` argument is set to TRUE, then the function also computes confidence intervals around the estimates of the log concentration. Since this step can be time-consuming, it is not performed by default. Moreover, confidence intervals can be computed after the main model is fit and evaluated, using the `getConfidenceInterval` function; see its documentation for details on how the intervals are estimated.

Value

This function constructs and returns an object of the `RPPAFit` class.

Author(s)

Kevin R. Coombes <kcoombes@mdanderson.org>

References

KRC

See Also

RPPAFit-class, RPPA, RPPADesign

Examples

```
path <- system.file("rppaTumorData", package="SuperCurve")
erk2 <- RPPA("ERK2.txt", path=path)
design <- RPPADesign(erk2, grouping="blockSample",
                     controls=list("neg con", "pos con"))
fit.nls <- RPPAFit(erk2, design, "Mean.Net")
summary(fit.nls)
coef(fit.nls)
```

---

rppaTumorData *ERK2, GSK3, and JNK expression in tumor samples*

Description

This data set contains the expression levels of three proteins: ERK2, GSK3, and JNK in 96 breast tumor samples and controls, measured in dilution series using reverse-phase protein arrays.

This data set also contains a description of the design of the reverse-phase protein array used in a set of experiments to measure protein levels in 101 different tumors. Cell lysates were spotted on the array in six-step dilution series. The layout of the array consisted of a grid of 4x6 subgrids. Each row of a subgrid contained the dilution series for a single sample, in decreasing concentration from left to right.

Usage

```
data(rppaTumorData)
```

Format

The objects `erk2`, `gsk3`, and `jnk` are objects of the RPPA class. The object `tDesign` is an object of the RPPADesign class.

Source

Bryan Hennessey and Gordon Mills

References

KRC