Package 'SuperCurve'

July 24, 2009

Version 1.3.3

Date 2009/01/19

Title SuperCurve Package

Author Kevin R. Coombes, Shannon Neeley, Corwin Joy, Jianhua Hu, Keith Baggerly, and P. Roebuck.

Maintainer P. Roebuck <plroebuck@mdanderson.org>

Description A package to analyze reverse phase protein lysate arrays.

Depends R (>= 2.7), methods, stats, utils, graphics, grDevices, MASS, cobs

Suggests boot, mgcv, quantreg, robustbase, splines

SystemRequirements ImageMagick

License file LICENSE

LazyLoad yes

R topics documented:

`SuperCurve-package` *Reverse phase protein lysate array analysis*

Description

A package for analyzing reverse phase protein lysate arrays (RPPA).

Details

For a complete list of functions, use `library(help="SuperCurve")`. For a high-level summary of the changes for each revision, use `file.show(system.file("NEWS", package="SuperCurve"))`

Author(s)

Kevin R. Coombes ⟨kcoombes@mdanderson.org⟩, P. Roebuck ⟨plroebuck@mdanderson.org⟩

`CobsFitClass-class` *Class "CobsFitClass"*

Description

The CobsFitClass class represents models that were fit with the nonparametric model.

Usage

```
## S4 method for signature 'CobsFitClass':
fitted(object, conc, ...)
```

Arguments

`object`	object of class `CobsFitClass`
`conc`	numeric vector containing estimates of the log concentration for each dilution series
`...`	extra arguments for generic routines

Value

The `coef` and `coefficients` methods return `NULL`.

The `fitted` method returns a numeric vector.

Methods

fitted(object, conc, ...) Extracts fitted values of the model

Objects from the Class

Objects are created internally by calls to the methods `fitSlide` or `RPPAFit`.

Slots

`model`: object of class `cobs` summarizing nonparametric fit

`lambda`: numeric

Extends

Class `FitClass`, directly.

Author(s)

P. Roebuck ⟨plroebuck@mdanderson.org⟩

References

KRC

See Also

`FitClass`

`Directory-class` *Class "Directory"*

Description

The Directory class represents a file system directory.

Usage

```
Directory(path)
is.Directory(x)
```

Arguments

`path`	character string specifying a directory
`x`	object of class `Directory`

Value

The `Directory` generator returns an object of class `Directory`.

The `is.Directory` method returns `TRUE` if its argument is an object of class `Directory`.

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 `Directory` generator function.

Slots

`path`: character string specifying a directory

Author(s)

P. Roebuck ⟨plroebuck@mdanderson.org⟩

Examples

```
txtdir <- Directory(system.file("rppaTumorData", package="SuperCurve"))
txtdir.path <- as(txtdir, "character")
```

FitClass-class *Class "FitClass"*

Description

The FitClass class represents the model that was fit in the `RPPAFit` routine. Functions for use with the FitClass are only to be used internally.

Usage

```
is.FitClass(x)
## S4 method for signature 'FitClass':
coef(object, ...)
## S4 method for signature 'FitClass':
coefficients(object, ...)
## S4 method for signature 'FitClass':
fitSeries(object, diln, intensity, est.conc, method="nls",
                               silent=TRUE, trace=FALSE, ...)
## S4 method for signature 'FitClass':
fitSlide(object, conc, intensity, ...)
## S4 method for signature 'FitClass':
trimConc(object, conc, intensity, design, trimLevel, ...)
```

Arguments

`x`	object of (sub)class `FitClass`
`object`	object of (sub)class `FitClass`
`diln`	numeric vector of dilutions for series to be fit
`intensity`	numeric vector of observed intensities for series to be fit
`est.conc`	numeric estimated concentration for dilution = 0
`method`	character string specifying regression method to use fit the series
`silent`	logical scalar. If `TRUE`, report of error messages will not be suppressed in try(nlsmeth(...))
`trace`	logical scalar. Used in `nls` method.
`conc`	numeric vector containing current estimates of concentration for each series
`design`	object of class `RPPADesign` describing the layout of the array
`trimLevel`	numeric scalar multiplied to MAD
`...`	extra arguments for generic routines

Value

The `is.FitClass` method returns `TRUE` if its argument is an object of subclass of class `FitClass`.

Methods

coef(object, ...) Extracts model coefficients from objects returned by modeling functions. Returns `NULL` if subclass does not override.

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

fitSeries(object, diln, intensity, est.conc, method="nls", silent=TRUE, trace=FALSE, ...) Finds the concentration for an individual dilution series given the curve fit for the slide.

fitSlide(object, conc, intensity, ...) Uses the concentration and intensity series for an entire slide to fit a curve for the slide of intensity = f(conc).

trimConc(object, conc, intensity, design, trimLevel, ...) Returns concentration and intensity cutoffs for the model.

Author(s)

P. Roebuck ⟨plroebuck@mdanderson.org⟩

References

KRC

`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`	object of class `RPPAFit` representing the result of fitting a four-parameter logistic model
`alpha`	numeric scalar specifying desired significance of the confidence interval; the width of the resulting interval is 1 - alpha.
`nSim`	numeric scalar specifying 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 class `RPPAFit`, containing updated values for the slots `lower`, `upper`, and `conf.width` that describe the confidence interval.

Author(s)

Kevin R. Coombes ⟨kcoombes@mdanderson.org⟩, P. Roebuck ⟨plroebuck@mdanderson.org⟩

References

KRC

See Also

`RPPAFit-class`, `RPPAFit`

Examples

```
  ## Not run:
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")
## N.B.: this takes a while!
fit.nls <- getConfidenceInterval(fit.nls, alpha=0.10, nSim=50)

## End(Not run)
```

`LoessFitClass-class` *Class "LoessFitClass"*

Description

The LoessFitClass class represents models that were fit with the nonparametric model.

Usage

```
## S4 method for signature 'LoessFitClass':
fitted(object, conc, ...)
```

Arguments

`object`	object of class `LoessFitClass`
`conc`	numeric vector containing estimates of the log concentration for each dilution series
`...`	extra arguments for generic routines

Value

The `coef` and `coefficients` methods return `NULL`.

The `fitted` method returns a numeric vector.

Methods

fitted(object, conc, ...) Extracts fitted values of the model

Objects from the Class

Objects are created internally by calls to the methods `fitSlide` or `RPPAFit`.

Slots

`model`: object of class `loess` summarizing loess fit

Extends

Class `FitClass`, directly.

Author(s)

P. Roebuck ⟨plroebuck@mdanderson.org⟩

References

KRC

See Also

`FitClass`

`LogisticFitClass-class`
Class "LogisticFitClass"

Description

The LogisticFitClass class represents models that were fit with the logistic model.

Usage

```
## S4 method for signature 'LogisticFitClass':
coef(object, ...)
## S4 method for signature 'LogisticFitClass':
coefficients(object, ...)
## S4 method for signature 'LogisticFitClass':
fitted(object, conc, ...)
```

Arguments

`object`	object of class `LogisticFitClass`
`conc`	numeric vector containing estimates of the log concentration for each dilution series
`...`	extra arguments for generic routines

Value

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

The `fitted` method returns a numeric vector.

Methods

coef(object, ...) Extracts model coefficients

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

fitted(object, conc, ...) Extracts fitted values of the model

Objects from the Class

Objects are created internally by calls to the methods `fitSlide` or `RPPAFit`.

Slots

`coefficients`: numeric vector of length 3, representing alpha, beta, and gamma respectively.

Extends

Class `FitClass`, directly.

Author(s)

P. Roebuck ⟨plroebuck@mdanderson.org⟩

References

KRC

See Also

`FitClass`

`registerModel` *Model Registration Methods*

Description

These routines represent the high-level access for model registration, which enables data-driven access by other routines. This represents the initial implementation and may change in the future.

Usage

```
getRegisteredModel(key)
getRegisteredModelLabel(key)
getRegisteredModelKeys()
registerModel(key, classname, ui.label=names(key))
```

Arguments

`key`	character string representing a registered model
`classname`	character string specifying Model class name to register
`ui.label`	character string specifying label to display by UI

Value

`getRegisteredModel` returns the `classname` associated with `key`.

`getRegisteredModelLabel` returns the `ui.label` associated with `key`.

`getRegisteredModelKeys` returns vector of `keys` for all registered models.

`registerModel` is invoked for its side effect, which is registering `classname` and `ui.label` by association to `key`.

Author(s)

P. Roebuck ⟨plroebuck@mdanderson.org⟩

See Also

`getRegisteredObject`, `getRegisteredObjectKeys`, `registerClassname`

Examples

```
## Create new (but nonfunctional) fit model
setClass("FooFitClass",
         representation("FitClass",
                        foo="character"),
         prototype(foo="fighter"))

## Register fit model to enable its use by package
registerModel("foo", "FooFitClass", "Foo R You")

## Show all registered fit models
sapply(getRegisteredModelKeys(),
       function(key) {
           c(model=getRegisteredModel(key),
             label=getRegisteredModelLabel(key))
       })
```

---

registerNormalizationMethod

Normalization Method Registration Methods

Description

These routines represent the high-level access for normalization method registration, which enables data-driven access by other routines. This represents the initial implementation and may change in the future.

Usage

```
getRegisteredNormalizationMethod(key)
getRegisteredNormalizationMethodLabel(key)
getRegisteredNormalizationMethodKeys()
registerNormalizationMethod(key, method, ui.label=names(key))
```

Arguments

`key`	character string representing a registered normalization method
`method`	function to invoke for normalization
`ui.label`	character string specifying label to display by UI

Value

`getRegisteredNormalizationMethod` returns the `method` associated with `key`.

`getRegisteredNormalizationMethodLabel` returns the `ui.label` associated with `key`.

`getRegisteredNormalizationMethodKeys` returns vector of `keys` for all registered normalization methods.

`registerNormalizationMethod` is invoked for its side effect, which is registering `method` and `ui.label` by association to `key`.

Author(s)

P. Roebuck ⟨plroebuck@mdanderson.org⟩

See Also

`getRegisteredObject`, `getRegisteredObjectKeys`, `registerMethod`

Examples

```
  ## Not run:
## Create new normalization method
normalize.foo <- function(concs, bar) {
    return(normconcs <- concs - bar)
}

## Register normalization method to enable its use by package
registerNormalizationMethod("foo", normalize.foo, "Foo is as foo does")

## Use it...
concs <- matrix(runif(500), nrow=10)
        # :TBD: Remove package prefix when released to public...
SuperCurve:::normalize(concs, method="foo", bar=0.005)
  ## End(Not run)
```

`RPPA-class` *Class "RPPA"*

Description

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

Usage

```
RPPA(file, path=".", antibody=NULL, software="microvigene")
is.RPPA(x)
## S4 method for signature 'RPPA':
dim(x)
## S4 method for signature 'RPPA':
image(x, measure="Mean.Net", main=measure,
    colorbar=FALSE, col=terrain.colors(256), ...)
## S4 method for signature 'RPPA':
summary(object, ...)
```

Arguments

`file`	character string or connection specifying text file containing quantifications of a reverse-phase protein array experiment
`path`	character string specifying the path from the current directory to the file. The default value assumes the file is contained in the current directory. If `file` is a connection, this argument is ignored.
`antibody`	character string specifying antibody name. If missing, default value is filename (referenced by `file` argument) without extension.
`software`	character string specifying the software used to generate the quantification file (see section 'Details' below)
`object`	object of class `RPPA`
`x`	object of class `RPPA`

`measure`	character string containing the name of the measurement column in `data` that should be displayed by the `image` method
`main`	character string used to title the image plot
`colorbar`	logical scalar that determines whether to include a color bar in the plot. If `TRUE`, the image cannot be used as one panel in a window with multiple plots. Default is `FALSE`.
`col`	graphics parameter used by image. It is included here to change the default color scheme to use terrain.colors.
`...`	extra arguments for generic or plotting routines

Details

The data frame slot (`data`) in a valid RPPA object constructed from a quantification file using the `RPPA` generator function is guaranteed to contain at least 6 columns of information:

`Main.Row`	logical location of spot on the array
`Main.Col`	logical location of spot on the array
`Sub.Row`	logical location of spot on the array
`Sub.Col`	logical location of spot on the array
`Sample`	unique identifier of sample spotted at location
`Mean.Net`	measurement representing background-corrected mean intensity of the spot

The first four components (taken together) give the logical location of a spot on an array. Additional columns may be included or may be added later.

Other methods can be specified to read the quantification files. The `software` argument is used in the selection of the actual method to perform this function. For example, if the argument value is "foo", the code will attempt to invoke method `read.foo` to read the file. The method will be passed a connection object to the file and should return a data frame containing the file's data. The method will be searched for in the global namespace, then within the package itself. The default value selects method `read.microvigene`, which this package provides to read *MicroVigene* quantification files in text format.

Value

The `RPPA` generator returns an object of class `RPPA`.

The `is.RPPA` method returns `TRUE` if its argument is an object of class `RPPA`.

The `dim` method returns a numeric vector of length 4.

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

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

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` generator function.

Slots

`data`: data.frame containing the contents of a quantification file

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

`antibody`: character string specifying name of antibody

Methods

dim(x) Returns the dimensions of the slide layout.

image(x, measure="Mean.Net", main=measure, colorbar=FALSE, col=terrain.colors(256), . . .)
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.

summary(object, . . .) Prints a summary of the underlying data frame.

Author(s)

Kevin R. Coombes ⟨kcoombes@mdanderson.org⟩, P. Roebuck ⟨plroebuck@mdanderson.org⟩

References

KRC

See Also

`RPPAFit`, `RPPADesign`

Examples

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

`rppaCell-data` *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 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 class `RPPA`. The object `design40` is an object of class `RPPADesign`.

Source

Bryan Hennessey and Gordon Mills

References

KRC

`RPPADesign-class` *Class "RPPADesign" and Class "RPPADesignParams"*

Description

The `RPPADesign` class represents the information that describes how a particular set of RPPA slides was designed. The `RPPADesignParams` class is used to bundle the parameter set together for easier re-use.

Usage

```
RPPADesign(raw,
           steps=rep(0, 1),
           series=factor(rep(0, 1)),
           grouping=c("byRow","byCol", "bySample", "blockSample"),
           ordering=c("decreasing","increasing"),
           alias=NULL,
           center=FALSE,
           controls=NULL,
           aliasfile=NULL,
           designfile=NULL,
           path=".")

RPPADesignParams(steps=rep(0,1),
                 series=factor(rep(0,1)),
                 grouping=c("byRow", "byCol", "bySample", "blockSample"),
                 ordering=c("decreasing", "increasing"),
                 alias=NULL,
                 center=FALSE,
                 controls=NULL,
                 aliasfile=NULL,
                 designfile=NULL,
                 path=".")

RPPADesignFromParams(raw, designparams)

getSteps(design)
is.RPPADesign(x)
is.RPPADesignParams(x)
```

```
seriesNames(design)
## S4 method for signature 'RPPADesign':
dim(x)
## S4 method for signature 'RPPADesign':
image(x, ...)
## S4 method for signature 'RPPADesign':
names(x)
## S4 method for signature 'RPPADesignParams':
paramString(object, ...)
## S4 method for signature 'RPPA, RPPADesign':
plot(x, y, ...)
## S4 method for signature 'RPPADesign':
summary(object, ...)
```

Arguments

`raw`	data frame, matrix, or object of class `RPPA`.
`designparams`	object of class `RPPADesignParams`.
`steps`	numeric vector listing the dilution step associated with each spot, on a logarithmic scale.
`series`	character vector or factor identifying the dilution series to which each spot corresponds.
`grouping`	character string specifying the orientation of the dilution series on the array. Valid values are:

`"byRow"`	each row of a subgrid is its own dilution series
`"byColumn"`	each column of a subgrid is its own dilution series
`"bySample"`	each unique sample id is its own dilution series
`"blockSample"`	all occurrences of sample id in subgrid refer to same series

`ordering`	character string specifying arrangement of dilution series. Valid values are:

`"decreasing"`	arranged in order of decreasing concentrations
`"increasing"`	arranged in order of increasing concentrations

`alias`	optional list or data frame for attaching sample labels or biologically relevant descriptors to the dilution series with the following required named components:

`Alias`	TBD description
`Sample`	TBD description

`aliasfile`	optional character string specifying filename. Data would be read by `read.delim` and expected format is as described above for `alias` argument.
`designfile`	optional character string specifying filename. Data would be read by `read.delim` and expected format is that produced as output by the **SlideDesignerGUI** package.
`path`	optional character string specifying directory path to prepend when either `aliasfile` or `designfile` argument refer to relative filename; ignored when filename is absolute.
`center`	logical scalar. If `TRUE`, then dilution steps are centered around 0.

`controls`	optional list containing the character strings that identify control spots on the array. `RPPADesignParams` will also coerce a character vector appropriately.
`x`	object of class `RPPADesign` (or `RPPA` in plot method)
`y`	object of class `RPPADesign`
`object`	object of class `RPPADesign` (or `RPPADesignParams` in `paramString` method)
`design`	object of class `RPPADesign`
`...`	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 allows 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.

The arguments `alias` and `aliasfile` are mutually exclusive; they specify the exact same thing. The arguments `controls` and `designfile` are also mutually exclusive. The *Sample-Type* column of the slide design datafile is used to automatically populate the `controls` slot of `RPPADesign` class.

Value

The `RPPADesign` generator returns an object of class `RPPADesign`.

The `RPPADesignParams` generator returns an object of class `RPPADesignParams`.

The `is.RPPADesign` method returns `TRUE` if its argument is an object of class `RPPADesign`.

The `is.RPPADesignParams` method returns `TRUE` if its argument is an object of class `RPPADesignParams`.

The `dim` method returns a numeric vector of length 4.

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

The `names` method returns a character vector.

The `paramString` method returns a character vector, possibly empty but never `NULL`.

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

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

Although objects of these classes can be created by a direct call to new, the preferred method is to start with the `RPPADesignParams` generator, followed by the `RPPADesignFromParams` function to construct the final object (the `RPPADesign` generator is directly implemented in this way).

Slots

For `RPPADesign` class:

`call`: object of class `call` specifying the function call that was used during construction

`layout`: data frame

`alias`: list

`sampleMap`: character vector

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

For `RPPADesignParams` class:

`steps`: see corresponding argument above

`series`: see corresponding argument above

`grouping`: see corresponding argument above

`ordering`: see corresponding argument above

`center`: see corresponding argument above

`controls`: list or `NULL`. see corresponding argument above

`alias`: list or `NULL`. see corresponding argument above

`aliasfile`: character specifying absolute pathname of file containing alias information, or `NULL`

`designfile`: character specifying absolute pathname of file containing slide design information, or `NULL`

Methods

dim(x) Returns the dimensions of the slide layout.

image(x, ...) 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.

paramString(object) Returns string representation of object.

plot(x, y, ...) Plots an object of class `RPPA` by showing its dilution series with respect to the corresponding object of class `RPPADesign`.

summary(object, ...) Lists the names of the control spots on the array and prints a summary of the data frame describing the layout.

Warning

The `paramString` method should not be called by user except for informational purposes. The content and format of the returned string may vary between different versions of this package.

Author(s)

Kevin R. Coombes ⟨kcoombes@mdanderson.org⟩, P. Roebuck ⟨plroebuck@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)
dim(design)
image(design)
summary(design)

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

plot(erk2, 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 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)
dim(design40)
image(design40)
summary(design40)
```

`RPPAFit-class` *Class "RPPAFit"*

Description

Objects of the `RPPAFit` class represent the results of fitting a statistical model of response to the dilution series in a reverse-phase protein array experiment.

Usage

```
## 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':
hist(x, type=c("Residuals", "StdRes", "ResidualsR2"),
  xlab=NULL, main=NULL, ...)
## S4 method for signature 'RPPAFit':
image(x, measure=c("Residuals", "ResidualsR2", "StdRes", "X", "Y"), ...)
## S4 method for signature 'RPPAFit, missing':
plot(x, y, type=c("cloud", "series", "individual"),
  xlab="Log Concentration", ylab="Intensity", colors=NULL, ...)
## S4 method for signature 'RPPAFit':
resid(object, type=c("raw", "standardized", "r2"),
  ...)
## S4 method for signature 'RPPAFit':
residuals(object, type=c("raw", "standardized", "r2"),
  ...)
## S4 method for signature 'RPPAFit':
summary(object, ...)
```

Arguments

`object`	object of class `RPPAFit`
`x`	object of class `RPPAFit`
`type`	character string describing the type of fitted values, residuals, images, histograms, or plots
`measure`	character string specifying measure to compute from fit
`xlab`	graphics parameter specifying how the x-axis should be labeled
`ylab`	graphics parameter specifying how the y-axis should be labeled
`main`	character string specifying title for the plot
`y`	not used
`colors`	graphics parameter, used only if `type='series'`, to color the lines connecting different dilution series. Eight default colors are used if the argument is `NULL`.
`...`	extra arguments for generic or plotting routines

Details

The `RPPAFit` class holds the results of fitting a response model to all the dilution series on a reverse-phase protein array. For details on how the model is fit, see the `RPPAFit` function. By fitting a joint model, we assume that the response curve is 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) concentration of the protein present in different dilution series.

Value

The `coef` and `coefficients` methods return the numeric model coefficients from objects returned by modeling functions.

The `fitted` method returns a numeric vector.

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

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

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

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

The `summary` method invisibly returns `NULL`.

Objects from the Class

Objects should be constructed using the `RPPAFit` function.

Slots

`call`: object of class `call` specifying the function call that was used to generate this model fit

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

`design`: object of class `RPPADesign` describing the layout of the array

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

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

`trimset`: numeric vector of length 5 containing the low and high intensities, the low and high concentrations that mark the trimming boundaries, and the trim level used

`model`: object of class `FitClass` unique to the model that was fit

`concentrations`: numeric vector of estimates of the relative log concentration of protein present in each sample

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

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

`conf.width`: numeric scalar specifying width of the confidence interval

`intensities`: numeric vector containing the predicted observed intensity at the estimated concentrations for each dilution series

`ss.ratio`: numeric vector containing statistic measuring the R^2 for each individual dilution series

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

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

Methods

coef(object, . . .) Extracts model coefficients from objects returned by modeling functions.

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

fitted(object, type=c("Y", "y", "X", "x"), . . .) Extracts the fitted values of the model. This process is more complicated 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 $X_i = X - \delta_i$.

hist(x, type=c("Residuals", "StdRes", "ResidualsR2"), xlab=NULL, main=NULL, . . .) Produces a histogram of the residuals. The exact form of the residuals being displayed depends on the value of the `type` argument.

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

plot(x, y, type=c("cloud", "series", "individual", "steps", "resid"), xlab="Log Concentration", ylab="Intensity" 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.

resid(object, type=c("raw", "standardized", "r2"), . . .) An alias for `residuals`.

residuals(object, type=c("raw", "standardized", "r2"), . . .) Reports 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.

summary(object, . . .) Prints a summary of the `RPPAFit` object. At present, this reports the function call used to fit the model and important fitting parameters.

Author(s)

Kevin R. Coombes ⟨kcoombes@mdanderson.org⟩, P. Roebuck ⟨plroebuck@mdanderson.org⟩

References

KRC

See Also

`RPPA`, `RPPADesign`, `RPPAFit`, `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")
showMethods('image')
class(fit.nls)
```

```
image(fit.nls)
image(fit.nls, measure="Residuals")
plot(fit.nls, type="cloud")

## :TBD: Why recreate exact same fit as above?
fit.q <- RPPAFit(erk2, design, "Mean.Net")
hist(fit.q, type="StdRes")
plot(fit.q, type="series")

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

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

`RPPAFitParams-class` *Fitting Dilution Curves to Protein Lysate Arrays with Class "RPPAFitParams"*

Description

The `RPPAFit` function fits an intensity response 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. The `RPPAFitParams` class is a convenient place to wrap the parameters that control the model fit into a reusable object.

Usage

```
RPPAFit(rppa,
        design,
        measure,
        model="logistic",
        xform=NULL,
        method=c("nls", "nlrob", "nlrq"),
        trim=2,
        ci=FALSE,
        ignoreNegative=TRUE,
        trace=FALSE,
        verbose=FALSE,
        veryVerbose=FALSE,
        warnLevel=0)

RPPAFitParams(measure,
              model="logistic",
              xform=NULL,
              method=c("nls", "nlrob", "nlrq"),
              trim=2,
              ci=FALSE,
              ignoreNegative=TRUE,
              trace=FALSE,
              verbose=FALSE,
                 veryVerbose=FALSE,
                 warnLevel=0)

RPPAFitFromParams(rppa,
                  design,
                  fitparams)

is.RPPAFit(x)
is.RPPAFitParams(x)
## S4 method for signature 'RPPAFitParams':
paramString(object, ...)
```

Arguments

`rppa`	object of class `RPPA` containing the raw data to be fit
`design`	object of class `RPPADesign` describing the layout of the array
`fitparams`	object of the class `RPPAFitParams`, bundling together the following arguments.
`measure`	character string identifying the column of the raw RPPA data that should be used to fit to the model.
`model`	character string specifying the model for the response curve fitted for the slide. Valid values are:

`"logistic"`	assumes a logistic shape for the curve
`"loess"`	fits a loess curve to the response
`"cobs"`	fits a b-spline curve to the slide with the constraint that curve be strictly increasing

`xform`	optional 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.
`method`	character string specifying the method for matching the individual dilution series to the response curve fitted for the slide. Valid values are:

`"nls"`	uses the optimal fit based on nonlinear least squares
`"nlrob"`	uses `nlrob` which is robust nls from **robustbase** package
`"nlrq"`	uses `nlrq` which is robust median regression from **quantreg** package

`trim`	numeric or logical scalar specifying trim level for concentrations. If positive, concentrations will be trimmed to reflect min and max concentrations we can estimate given the background noise. If `TRUE`, the trim level defaults to `2`, which was originally the hardcoded value; otherwise, raw concentrations are returned without trimming.
`ci`	logical scalar. If `TRUE`, compute 90% confidence intervals on the concentration estimates.
`ignoreNegative`	logical scalar. If `TRUE`, convert negative values to `NA` before fitting the model.
`trace`	logical scalar passed to nls in the `method` portion of the routine
`verbose`	logical scalar. If `TRUE`, print updates while fitting the data
`veryVerbose`	logical scalar. If `TRUE`, print voluminous updates as each individual dilution series is fitted

`warnLevel`	integer scalar used to set the `warn` option before calling `method`. 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.
`object`	object of class `RPPAFitParams`
`x`	object of class `RPPAFit` (or `RPPAFitParams`)
`...`	extra arguments for generic routines.

Details

The basic mathematical model is given by

$$Y = f(X - \delta_i),$$

where Y is the observed intensity, X is the designed dilution step and f is the model for the protein response curve. By fitting a joint model, we assume that the response curve is 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) concentration of the protein present in different dilution series.

As the first step in fitting the model, we compute crude estimates of the individual δ_i assuming a rough logistic shape for the protein response curve.

Next, we fit an overall response curve for the slide f using the estimated concentrations and observed intensities $Y = f(\delta_i)$. The model for f is specified in the *model* parameter.

Next, we update the estimates of the individual δ_i using our improved fitted model f for the overall slide response curve. These individual series are matched to the overall slide response curve using the algorithm specified in `method`. The default method is `nls`, a least squares matchup, but we also offer robust alternatives which can do better.

Finally, we re-estimate f using the improved estimates for δ_i. We continue to iterate between f and δ_i. We do this twice since that seems to give reasonable convergence.

If the `ci` argument is `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.

Value

The `RPPAFit` generator and `RPPAFitFromParams` function return an object of class `RPPAFit`.

The `RPPAFitParams` generator returns an object of class `RPPAFitParams`.

The `is.RPPAFit` method returns `TRUE` if its argument is an object of class `RPPAFit`.

The `is.RPPAFitParams` method returns `TRUE` if its argument is an object of class `RPPAFitParams`.

The `paramString` method returns a character vector, possibly empty but never `NULL`.

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 `RPPAFitParams` function.

Slots

`measure`: character; see arguments above

`xform`: function or NULL; see arguments above

`method`: character; see arguments above

`ci`: logical; see arguments above

`ignoreNegative`: logical; see arguments above

`trace`: logical; see arguments above

`verbose`: logical; see arguments above

`veryVerbose`: logical; see arguments above

`warnLevel`: numeric; see arguments above

`trim`: numeric; see arguments above

`model`: character; see arguments above

Methods

paramString(object) Returns string representation of object.

Warning

The `paramString` method should not be called by user except for informational purposes. The content and format of the returned string may vary between different versions of this package.

Author(s)

P. Roebuck ⟨plroebuck@mdanderson.org⟩, Kevin R. Coombes ⟨kcoombes@mdanderson.org⟩

References

KRC

See Also

`RPPAFit` `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)
```

`RPPASet-class` *Class "RPPASet"*

Description

The RPPASet class fits supercurves to an entire directory of reverse-phase protein array experiments.

Usage

```
RPPASet(path,
        designparams,
        fitparams,
        antibodyfile=NULL,
        software="microvigene")
is.RPPASet(x)
## S4 method for signature 'RPPASet':
summary(object, path, prefix="supercurve", graphs=TRUE, tiffdir=TRUE, ...)
```

Arguments

`path`	character string specifying the path from the current directory to the directory containing the files to be processed
`designparams`	object of class `RPPADesignParams` describing features common to all quantification files
`fitparams`	object of class `RPPAFitParams` containing parameters used to fit the supercurve model
`antibodyfile`	character string specifying filename cantaining mapping from quantification files to antibodies
`software`	character string specifying the software used to generate the quantification file (see section 'Details' of `RPPA`)
`object`	object of class `RPPASet`
`prefix`	character string used as a prefix on files generated by the summary method.
`graphs`	logical scalar specifying if standard graphs should be produced
`tiffdir`	character string specifying path to the directory containing the original TIFF files
`x`	object of class `RPPASet`
`...`	extra arguments for generic or plotting routines

Details

Quantify all the slides in a directory using `RPPASet` generator. This returns an object containing slide data and fits for each slide. Typically this is followed by a call to `summary` or `write.summary` to write the resulting quantifications and diagnostic plots to a directory.

Value

The `RPPASet` generator returns an object of class `RPPASet`.

The `is.RPPASet` method returns `TRUE` if its argument is an object of class `RPPASet`.

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

Objects from the Class

Although objects of the class can (in theory) be created by a direct call to new, the only realistic method is to use the `RPPASet` generator function.

Slots

`call`: object of class `call` specifying the function call that was used during construction

`version`: character string containing the version of this package used to construct the object

`design`: object of class `RPPADesign`, common to all the slides

`rppas`: array of objects of class `RPPA`

`fitparams`: object of class `RPPAFitParams` that was used to construct the model fits

`fits`: array of fitted objects of class `RPPAFit`

Methods

summary(object, path, prefix, graphs, tiffdir, . . .) Creates a record of the entire RPPASet, including fitted values, residuals, and images of the processed slides. See the documentation of `write.summary` for details.

Author(s)

Kevin R. Coombes ⟨kcoombes@mdanderson.org⟩, P. Roebuck ⟨plroebuck@mdanderson.org⟩

References

KRC

See Also

`RPPA`, `RPPADesign`, `RPPAFit`

Examples

```
  ## Not run:
parentdir <- file.path("C:", "MyData")
txtdir <- file.path(parentdir, "txt")     # quantification files
imgdir <- file.path(parentdir, "tif")     # and corresponding image files
outdir <- file.path(parentdir, "results") # output files

designparams <- RPPADesignParams(grouping="blockSample",
                                 center=FALSE,
                                 aliasfile="layoutInfo.tsv",
                                 designfile="slidedesign.tsv")
fitparams <- RPPAFitParams(measure="Mean.Net",
                           method="nlrob",
                           model="cobs",
                           ignoreNegative=FALSE,
                           warnLevel=-1,
                           verbose=FALSE)

fitset <- RPPASet(txtdir,
                  designparams,
                  fitparams)
```

```
write.summary(fitset,
              path=outdir,
              graphs=TRUE,
              tiffdir=imgdir)
  ## End(Not run)
```

`rppaTriple-data` *ACTB, CAS3, FAK, and ODC1 expression in 14 fed/starved cell lines*

Description

This data set contains the expression levels of four proteins: beta-Actin (ACTB), Caspase 3 (CAS3), Focal adhesion kinase (FAK), and Ornithine decarboxylase (ODC1) from a study that was done to compare protein levels in 14 cell lines from both a “fed” and a dQuotestarved state. There are two files included for beta-Actin, one that was scanned in color (actb) and the other in 16-bit grayscale (actb.gray); all other proteins were scanned in color.

This data set also contains a description of the design used for the reverse-phase protein arrays from this study. Cell lysates were spotted on the array in six-step dilution series. The layout of the array consists of a grid of 6x6 subgrids. The first three rows of a subgrid contain 3 replicates of a cell line. The last three rows contain 3 replicates of another cell line. Each subgrid is replicated on the array 3 times, so that there are a total of 9 replicates per cell line per state. The top part of the array contains the fed cell lines and the bottom part of the array contains the starved cell lines. There is one subgrid on the array that contains only buffer material and another subgrid that did not have anything printed (blank). There are a total of 18 spots each of buffer and blank material.

Usage

```
data(rppaTriple)
```

Format

The objects `actb`, `actb.gray`, `cas3`, `fak`, and `odc1` are objects of class `RPPA`. The object `tripledesign` is an object of class `RPPADesign`.

Source

Victor Levin

References

KRC

`rppaTumor-data` *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. Cell lysates were spotted on the array in seven-step dilution series with eith a positive or negative control at the end of the series. The layout of the array consisted of a grid of 4x4 subgrids. The first two rows of a subgrid contained a single dilution series and a negative control spot. The last two rows of the subgrid contained another dilution series and a positive control spot.

Usage

```
data(rppaTumor)
```

Format

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

Source

Doris Swank and Gordon Mills

References

KRC

`spatialCorrection` *Spatial Correction*

Description

This function estimates a smoothed surface from positive control spots on an RPPA slide. The surface is used to perform spatial corrections (i.e., because of uneven hybridization) on the array. It is used before `RPPAFit`, one slide at a time.

Usage

```
spatialCorrection(rppa,
                  design,
                  measure=c("Mean.Net", "Mean.Total"),
                  cutoff=0.8,
                  k=100,
                  gamma=0.1,
                  plotSurface=FALSE)
```

Arguments

`rppa`	object of class `RPPA`
`design`	object of class `RPPADesign`
`measure`	character string specifying fit measure to smooth
`cutoff`	numeric scalar used to identify the background cutoff with value in range [0..1]
`k`	numeric scalar used as smoothing model argument.
`gamma`	numeric scalar used as model parameter with value in range [0..2]
`plotSurface`	logical scalar specifying whether to plot surfaces

Details

The observed spot intensities are assumed to be a combination of true signal, background noise, and hybridization effects according to the following model:

$$Y_rc = Y * H_rc + B_rc$$

where Y_rc is the observed intensity, Y is the true signal, H_rc is the effect of hybridization, and B_rc is the background noise. The subscripts "r" and "c" refer to the physical row and column of the spot on the array. Background noise is estimated locally by the array software. The hybridization effect is estimated fitting a generalized additive model (GAM) to positive control spots printed uniformly across the array.

The estimated surface is used to scale the intensities on the array. Each intensity is adjusted by the amount that is needed to make the positive control surface flat at the value of the median of the surface. This is done by dividing each spot by the estimated surface value and then multiplying by the median of the surface.

Positive control spots that are expressed below the cutoff for the noise region are excluded from the computation of the surface.

Sometimes, positive control spots are printed in a dilution series to avoid saturation problems with these spots. When this happens, the observed intensities are adjusted by the positive control surface that has the most similar expression level.

The `design` argument must have already been augmented with slide design information.

The `cutoff` argument passed to `quantile` is percentile of the background estimates used to define the noise region of slide.

The `k` argument passed to `s` sets upper limit on degrees of freedom associated with smoothing.

The `gamma` argument passed to `gam` provides a constant multiplier used to inflate model degrees of freedom in the GCV or UBRE/AIC score.

Value

Returns modified `rppa` with an additional measurement column named `Spatial.Adj`.

Note

This code may not yet work with slides containing more than one positive control series. Currently untested.

Author(s)

P. Roebuck ⟨plroebuck@mdanderson.org⟩, E. Shannon Neeley ⟨sneeley@stat.byu.edu⟩

References

"Spatial Corrections for Reverse Phase Protein Arrays" (in progress).

See Also

`quantile`, `gam`, `s`, `choose.k`

Examples

```
  ## Not run:
## :TODO: Need to be able to run this with included data...
path <- system.file("rppaTumorData", package="SuperCurve")
erk2 <- RPPA("ERK2.txt", path=path)
designparams <- RPPADesignParams(grouping="blockSample",
                                 controls=list("neg con", "pos con"))
design <- RPPADesignFromParams(erk2, designparams)

erk2.sc <- spatialCorrection(rppa,
                             design,
                             measure="Mean.Net")
  ## End(Not run)
```

`SuperCurveSettings-class`
Class "SuperCurveSettings"

Description

The SuperCurveSettings class represents the arguments needed to perform curve fitting.

Usage

```
SuperCurveSettings(txtdir,
                   imgdir,
                   outdir,
                   designparams,
                   fitparams)

fitCurveAndSummarizeFromSettings(settings)
is.SuperCurveSettings(x)
## S4 method for signature 'SuperCurveSettings':
paramString(object, ...)
```

Arguments

`txtdir`	character string specifying the directory containing quantification files in text format
`imgdir`	character string specifying the directory containing TIFF image files associated with each of the aforementioned quantification files
`outdir`	character string specifying the directory where output from analysis should be stored. Must be writable.

`designparams`	object of class `RPPADesignParams`
`fitparams`	object of class `RPPAFitParams`
`object`	object of class `SuperCurveSettings`
`settings`	object of class `SuperCurveSettings`
`x`	object of class `SuperCurveSettings`
`...`	extra arguments for generic routines.

Value

The `SuperCurveSettings` generator returns an object of class `SuperCurveSettings`.

The `is.SuperCurveSettings` method returns `TRUE` if its argument is an object of class `SuperCurveSettings`.

The `paramString` method returns a character vector, possibly empty but never `NULL`.

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 `SuperCurveSettings` generator function.

Slots

`txtdir`: object of class `Directory` specifying the directory containing quantification files in text format

`imgdir`: object of class `Directory` specifying the directory containing TIFF image files

`outdir`: object of class `Directory` specifying the directory where analysis results should be stored

`designparams`: object of class `RPPADesignParams` specifying the parameters that describe how a particular set of RPPA slides was designed

`fitparams`: object of class `RPPAFitParams` specifying the parameters that control model fit

`version`: character string containing the version of this package used to construct the object

Methods

paramString(object) Returns string representation of object.

Warning

The `paramString` method should not be called by user except for informational purposes. The content and format of the returned string may vary between different versions of this package.

Author(s)

P. Roebuck ⟨plroebuck@mdanderson.org⟩

See Also

`Directory`, `RPPAFitParams`, `RPPADesignParams`

Examples

```
  ## Not run:
designparams <- RPPADesignParams(center=FALSE,
                                 controls=list("neg con", "pos con"),
                                 grouping="blockSample")
fitparams <- RPPAFitParams(ignoreNegative=FALSE,
                           measure="Mean.Total",
                           method="nlrob",
                           model="loess",
                           warnLevel=-1)
fitparams <- RPPAFitParams(grouping="blockSample")
settings <- SuperCurveSettings(txtdir=system.file("rppaTumorData",
                                                  package="SuperCurve"),
                               imgdir=file.path("", "path", "to", "images"),
                               outdir=tempdir(),
                               designparams=designparams,
                               fitparams=fitparams)
fitCurveAndSummarizeFromSettings(settings)

## End(Not run)
```

`write.summary` *TBD*

Description

This function produces a graphical summary for each array in an `RPPASet` and summarizes the quantification results into various files.

Usage

```
write.summary(rppaset,
              path,
              prefix="supercurve",
              graphs=TRUE,
              tiffdir=NULL)
```

Arguments

`rppaset`	object of class `RPPASet`
`path`	character string specifying directory where output should be stored
`prefix`	character string providing a filename prefix to be applied when creating result files
`graphs`	logical scalar specifying whether to save fit graphs
`tiffdir`	character string specifying directory containing TIFF images corresponding to the TEXT quantification files

Details

Generates three CSV files: one for the raw concentrations, one for the R^2 statistics, and one for the polished concentrations. If `tiffdir` is `NULL`, the directory is assumed to be a sibling directory to `path` named "tif". If `graphs` is `TRUE`, two PNG files containing output graphs are created per antibody. The ImageMagick 'convert' binary is then used to merge these output graphs with the source TIFF files, generating an additional JPEG file per antibody.

Author(s)

P. Roebuck ⟨plroebuck@mdanderson.org⟩

References

KRC

See Also

`RPPASet`

Examples

```
  ## Not run:
parentdir <- file.path("lysate")
txtdir <- file.path(parentdir, "txt")     # quantification files
imgdir <- file.path(parentdir, "tif")     # and corresponding image files
outdir <- file.path(parentdir, "results") # output files

designparams <- RPPADesignParams(grouping="blockSample",
                                 center=FALSE,
                                 controls=list("control",
                                               "pos con",
                                               "neg con"))
fitparams <- RPPAFitParams(measure="Mean.Net",
                           method="nlrob",
                           model="logistic",
                           ignoreNegative=FALSE,
                           warnLevel=-1,
                           verbose=FALSE)
fitset <- RPPASet(txtdir,
                  designparams,
                  fitparams,
                  antibodyfile="proteinAssay.tsv")
write.summary(fitset,
              outdir,
              graphs=TRUE,
              tiffdir=imgdir)
  ## End(Not run)
```

Index