Package 'SuperCurve'

January 26, 2015

Version 1.5.8

Phase Beta

Date 2015-01-25

Title SuperCurve Package

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

Maintainer ``P. Roebuck'' <proebuck@mdanderson.org>

Description A package to analyze reverse phase protein lysate arrays.

Depends R (>= 2.15), methods, cobs

Imports utils, grDevices, graphics, stats, MASS

Suggests boot, mgcv, quantreg, robustbase, splines, timeDate, SuperCurveSampleData

SystemRequirements ImageMagick

NeedsCompilation no

URL http://r-forge.r-project.org/projects/supercurve/

License Artistic-2.0

Copyright file COPYRIGHTS

LazyLoad yes

Repository R-Forge

Repository/R-Forge/Project supercurve

Repository/R-Forge/Revision 962

Repository/R-Forge/DateTimeStamp 2015-01-26 03:34:50

Date/Publication 2015-01-26 03:34:50

R topics documented:

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

Description

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

Details

Package:	SuperCurve
Type:	Package
Version:	1.5.8
Phase:	Beta
Date:	2015-01-25
License:	Artistic-2.0

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 <proebuck@mdanderson.org>

`BoundedRange-class` *Class "BoundedRange"*

Description

The BoundedRange class represents an attempt to abstract reporting of progress of a task. This class assumes that progress is reported via a progressbar and provides means to get/set values for such a widget.

Usage

```
BoundedRange(value, minimum=1, maximum=100)
is.BoundedRange(x)
## S4 method for signature 'BoundedRange'
progressMaximum(object)
## S4 replacement method for signature 'BoundedRange,numeric'
progressMaximum(object) <- value
## S4 method for signature 'BoundedRange'
progressMinimum(object)
## S4 replacement method for signature 'BoundedRange,numeric'
progressMinimum(object) <- value
## S4 method for signature 'BoundedRange'
progressValue(object)
## S4 replacement method for signature 'BoundedRange,numeric'
progressValue(object) <- value
```

Arguments

minimum	integer specifying desired minimum value of closed interval
maximum	integer specifying desired maximum value of closed interval
value	integer specifying desired current value
object	object of class BoundedRange
x	object of class BoundedRange

Value

The BoundedRange generator returns an object of class BoundedRange.

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

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 BoundedRange generator function.

Slots

minimum: integer specifying the minimum value of the closed interval. Default is 1.

maximum: integer specifying the minimum value of the closed interval. Default is 100.

value: integer specifying current value. Must be in the closed interval minimum..maximum

Methods

progressMaximum signature(object = "BoundedRange"):
Returns integer representing maximum value of interval.

progressMaximum<- signature(object = "BoundedRange", value = "numeric"):
Sets the value of the maximum slot.

progressMinimum signature(object = "BoundedRange"):
Returns integer representing minimum value of interval.

progressMinimum<- signature(object = "BoundedRange", value = "numeric"):
Sets the value of the minimum slot.

progressValue signature(object = "BoundedRange"):
Returns integer representing current value.

progressValue<- signature(object = "BoundedRange", value = "numeric"):
Sets the value of the value slot.

Author(s)

P. Roebuck <proebuck@mdanderson.org>

Examples

```
showClass("BoundedRange")
br <- BoundedRange(3, maximum=10)
progressValue(br) <- 5 # Modify current value
```

CobsFitClass-class *Class "CobsFitClass"*

Description

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

Usage

```
## S4 method for signature 'CobsFitClass'
fitSeries(object,
          diln,
          intensity,
          est.conc,
          method="nls",
          silent=TRUE,
          trace=FALSE,
          ...)
## S4 method for signature 'CobsFitClass'
fitSlide(object,
         conc,
         intensity,
         ...)
## S4 method for signature 'CobsFitClass'
fitted(object, conc, ...)
## S4 method for signature 'CobsFitClass'
trimConc(object,
         conc,
         intensity,
         design,
         trimLevel,
         ...)
```

Arguments

object	object of class CobsFitClass
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 to fit the series
silent	logical scalar. If TRUE, report of error messages will be suppressed in try(nlsmeth(...))
trace	logical scalar. Used in nls method.
conc	numeric vector containing estimates of the log concentration for each dilution 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 `fitted` method returns a numeric vector.

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.

Methods

fitSeries `signature(object = "CobsFitClass")`:
Finds the concentration for an individual dilution series given the curve fit for the slide.

fitSlide `signature(object = "CobsFitClass")`:
Uses the concentration and intensity series for an entire slide to fit a curve for the slide of intensity = f(conc).

fitted `signature(object = "CobsFitClass")`:
Extracts fitted values of the model.

trimConc `signature(object = "CobsFitClass")`:
Returns concentration and intensity cutoffs for the model.

Author(s)

P. Roebuck <proebuck@mdanderson.org>

References

Hu J, He X, Baggerly KA, Coombes KR, Hennessy BT, Mills GB.
"Non-parametric Quantification of Protein Lysate Arrays"
Bioinformatics (2007) 23(15): 1986-1994.
http://bioinformatics.oxfordjournals.org/content/23/15/1986

See Also

`FitClass`

DefaultProgressMonitor-class
Class "DefaultProgressMonitor"

Description

The DefaultProgressMonitor class represents an attempt to abstract reporting of progress of a task. This class assumes that progress is reported via a progressbar and provides means to get/set values for such a widget.

Usage

```
DefaultProgressMonitor(label, value, minimum=0, maximum=100)
## S4 method for signature 'DefaultProgressMonitor'
elapsed(object)
## S4 method for signature 'DefaultProgressMonitor'
progressDone(object)
## S4 replacement method for signature 'DefaultProgressMonitor,logical'
progressDone(object) <- value
## S4 method for signature 'DefaultProgressMonitor'
progressError(object)
## S4 replacement method for signature 'DefaultProgressMonitor,logical'
progressError(object) <- value
## S4 method for signature 'DefaultProgressMonitor'
progressLabel(object)
## S4 replacement method for signature 'DefaultProgressMonitor,character'
progressLabel(object) <- value
## S4 method for signature 'DefaultProgressMonitor'
progressMaximum(object)
## S4 replacement method for signature 'DefaultProgressMonitor,numeric'
progressMaximum(object) <- value
## S4 method for signature 'DefaultProgressMonitor'
progressMinimum(object)
## S4 replacement method for signature 'DefaultProgressMonitor,numeric'
progressMinimum(object) <- value
## S4 method for signature 'DefaultProgressMonitor'
progressValue(object)
## S4 replacement method for signature 'DefaultProgressMonitor,numeric'
progressValue(object) <- value
```

Arguments

label	string specifying label for progressbar widget
value	integer value representing current progress towards task completion
minimum	integer value representing minimum range of progress
maximum	integer value representing minimum range of progress
object	object of (sub)class DefaultProgressMonitor

Value

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

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 `DefaultProgressMonitor` generator function.

Slots

`done`: logical scalar specifying if task completed. Default is `FALSE`.

`err`: logical scalar specifying if an error has occurred. Default is `FALSE`.

`label`: string specifying label for abstract progressbar

`range`: object of class `BoundedRange`

`etime`: object of class `ElapsedTime`

`elapsed`: object of class `difftime` specifying seconds since last update

Extends

Class `ProgressMonitor`, directly.

Methods

elapsed `signature(object = "DefaultProgressMonitor")`:
Returns elapsed time since creation of abstract progressbar.

progressDone `signature(object = "DefaultProgressMonitor")`:
Returns `TRUE` if task is complete; otherwise, `FALSE`.

progressDone<- `signature(object = "DefaultProgressMonitor", value = "logical")`:
Sets value of the `done` slot.

progressError `signature(object = "DefaultProgressMonitor")`:
Returns `TRUE` if an error occurred during processing; otherwise, `FALSE`.

progressError<- `signature(object = "DefaultProgressMonitor", value = "logical")`:
Sets value of the `err` slot.

progressLabel `signature(object = "DefaultProgressMonitor")`:
Returns string representing label for abstract progressbar.

progressLabel<- `signature(object = "DefaultProgressMonitor", value = "character")`:
Sets value of the `label` slot.

progressMaximum `signature(object = "DefaultProgressMonitor")`:
Returns integer representing maximum value for abstract progressbar.

progressMaximum<- `signature(object = "DefaultProgressMonitor", value = "numeric")`:
Sets the maximum value of the `range` slot.

progressMinimum `signature(object = "DefaultProgressMonitor")`:
Returns integer representing minimum value for abstract progressbar.

progressMinimum<- `signature(object = "DefaultProgressMonitor", value = "numeric")`:
Sets the minimum value of the `range` slot.

progressValue `signature(object = "DefaultProgressMonitor")`:
Returns integer representing current value for abstract progressbar.

progressValue<- `signature(object = "DefaultProgressMonitor", value = "numeric")`:
Sets the current value of the `range` slot.

Author(s)

P. Roebuck <proebuck@mdanderson.org>

See Also

`difftime`, `BoundedRange`, `ElapsedTime`, `ProgressMonitor`

Examples

```
showClass("DefaultProgressMonitor")
niters <- 10
dpm <- DefaultProgressMonitor("my task", value=0, maximum=niters)
for (i in seq.int(niters)) {
    ## Perform portion of task
    progressValue(dpm) <- i # Modify current value
}
```

Directory-class *Class "Directory"*

Description

The Directory class represents a file system directory.

Usage

```
Directory(path)
is.Directory(x)
## S4 method for signature 'character,Directory'
coerce(from, to, strict=TRUE)
## S4 method for signature 'Directory,character'
coerce(from, to, strict=TRUE)
```

Arguments

`path`	character string specifying a directory
`x`	object of class `Directory`
`from`	object of class `Directory` or character string specifying pathname of directory
`to`	object of class `Directory` or character string specifying pathname of directory
`strict`	logical scalar. If `TRUE`, the returned object must be strictly from the target class; otherwise, any simple extension of the target class will be returned, without further change.

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

Methods

coerce `signature(from = "Directory", to = "character")`:
Coerce an object of class `Directory` to its character string pathname equivalent.

coerce `signature(from = "character", to = "Directory")`:
Coerce a character string specifying directory pathname to an equivalent object of class `Directory`.

Note

The coercion methods should not be called explicitly; instead, use an explicit call to the `as` method.

Author(s)

P. Roebuck <proebuck@mdanderson.org>

See Also

`as`

Examples

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

DS5RPPAPreFitQC-class *Class "DS5RPPAPreFitQC"*

Description

The DS5RPPAPreFitQC class represents the inputs necessary to determine the quality control rating of a reverse-phase protein array slide with 5 dilution series.

Usage

```
## S4 method for signature 'DS5RPPAPreFitQC'
qcprob(object, ...)
## S4 method for signature 'DS5RPPAPreFitQC'
summary(object, ...)
```

Arguments

object	object of class DS5RPPAPreFitQC
...	extra arguments for generic routines

Details

The prediction model used multiple training datasets from the RPPA Core Facility by fitting a logistic regression model using an expert rating of a slide's quality (good, fair, or poor) as the response variable and a host of metrics about the raw positive control data as predicting variables.

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 RPPAPreFitQC factory generator function.

Slots

slopediff: numeric scalar specifying the difference from perfect slope

cvs: numeric vector containing the coefficient of variance for each positive control dilution series

slopes: numeric vector containing the slopes for each positive control dilution series

skews: numeric vector containing the skews for each sample dilution series

drdiffs: numeric vector containing the difference in dynamic range of each positive control dilution series

percentgood: numeric scalar specifying percentage of "good" sample spots on the slide

adjusted: logical scalar specifying if adjusted measures were used

Extends

Class RPPAPreFitQC, directly.

Methods

qcprob signature(object = "DS5RPPAPreFitQC"):
Calculates the probability of good slide, returned as numeric scalar.

summary signature(object = "DS5RPPAPreFitQC"):
Prints a summary of the underlying data frame.

Author(s)

P. Roebuck <proebuck@mdanderson.org>

References

Ju Z, Liu W, Roebuck PL, Siwak DR, Zhang N, Lu Y, Davies MA, Akbani R, Weinstein JN, Mills GB, Coombes KR
Development of a Robust Classifier for Quality Control of Reverse Phase Protein Arrays.
(submitted).

`elapsed-method` *Method "elapsed"*

Description

`elapsed` is a generic function used to return the elapsed time since some operation (related to the object) began. The method invokes particular `methods` which depend on the `class` of the first argument.

Usage

```
## S4 method for signature 'ANY'
elapsed(object, ...)
```

Arguments

`object`	an object for which an elapsed time is desired
`...`	additional arguments affecting the elapsed time produced

Value

The form of the value returned by `elapsed` depends on the class of its argument. See the documentation of the particular methods for details of what is produced by that method.

Author(s)

P. Roebuck <proebuck@mdanderson.org>

`ElapsedTime-class` *Class "ElapsedTime"*

Description

The `ElapsedTime` class represents a means of reporting elapsed time.

Usage

```
ElapsedTime()
is.ElapsedTime(x)
## S4 method for signature 'ElapsedTime'
elapsed(object,
        units=c("auto", "secs", "mins", "hours", "days"))
```

Arguments

x	object of class ElapsedTime
object	object of class ElapsedTime
units	string specifying desired unit of time

Value

The ElapsedTime generator returns an object of class ElapsedTime.

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

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 ElapsedTime generator function.

Slots

start: numeric scalar specifying the elapsed time for the currently running R process (taken from proc.time when created)

Methods

elapsed signature(object = "ElapsedTime"):
Returns object of class difftime representing elapsed time difference between current time and the start slot value.

Units may be specified if desired; otherwise, the largest possible unit in which difference is greater than one will be chosen.

Author(s)

P. Roebuck <proebuck@mdanderson.org>

See Also

difftime

Examples

```
showClass("ElapsedTime")
et <- ElapsedTime()
elapsed(et)                  # 'auto' reports 'secs' as less than minute later
elapsed(et, units="secs")
elapsed(et, units="mins")
elapsed(et, units="hours")
elapsed(et, units="days")
```

FitClass-class *Class "FitClass"*

Description

The FitClass class is a virtual class representing the model that was fit in the RPPAFit routine. Functions for use with 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'
fitted(object,
       conc,
       ...)
## 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 to fit the series
silent	logical scalar. If TRUE, report of error messages will 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.

The coef and coefficients methods return NULL.

Objects from the Class

This class should not be instantiated directly; extend this class instead.

Methods

coef signature(object = "FitClass"):
Placeholder method which should be implemented by subclass if appropriate for the particular model.

coefficients signature(object = "FitClass"):
An alias for coef.

fitSeries signature(object = "FitClass"):
Placeholder method which must be implemented by subclass.

fitSlide signature(object = "FitClass"):
Placeholder method which must be implemented by subclass.

fitted signature(object = "FitClass"):
Placeholder method which must be implemented by subclass.

trimConc signature(object = "FitClass"):
Placeholder method which must be implemented by subclass.

Author(s)

P. Roebuck <proebuck@mdanderson.org>

`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,
                      progmethod=NULL)
```

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.
`progmethod`	optional function that can be used to report progress.

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 <proebuck@mdanderson.org>

See Also

`RPPAFit-class`, `RPPAFit`

Examples

```
  ## Not run:
extdata.dir <- system.file("extdata", package="SuperCurveSampleData")

txtdir <- file.path(extdata.dir, "rppaCellData")
akt <- RPPA("Akt.txt", path=txtdir)
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'
fitSeries(object,
          diln,
          intensity,
          est.conc,
          method="nls",
          silent=TRUE,
          trace=FALSE,
          ...)
## S4 method for signature 'LoessFitClass'
fitSlide(object,
         conc,
         intensity,
         ...)
## S4 method for signature 'LoessFitClass'
fitted(object,
       conc,
       ...)
## S4 method for signature 'LoessFitClass'
trimConc(object,
         conc,
         intensity,
         design,
         trimLevel,
         ...)
```

Arguments

object	object of class LoessFitClass
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 to fit the series
silent	logical scalar. If TRUE, report of error messages will be suppressed in try(nlsmeth(...))
trace	logical scalar. Used in nls method.
conc	numeric vector containing estimates of the log concentration for each dilution 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 fitted method returns a numeric vector.

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.

Methods

fitSeries signature(object = "LoessFitClass"):
Finds the concentration for an individual dilution series given the curve fit for the slide.

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

fitted signature(object = "LoessFitClass"):
Extracts fitted values of the model.

trimConc signature(object = "LoessFitClass"):
Returns concentration and intensity cutoffs for the model.

Author(s)

P. Roebuck <proebuck@mdanderson.org>

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'
fitSeries(object,
          diln,
          intensity,
          est.conc,
          method="nls",
          silent=TRUE,
          trace=FALSE,
          ...)
## S4 method for signature 'LogisticFitClass'
fitSlide(object,
         conc,
         intensity,
         ...)
## S4 method for signature 'LogisticFitClass'
fitted(object,
       conc,
       ...)
## S4 method for signature 'LogisticFitClass'
trimConc(object,
         conc,
         intensity,
         design,
         trimLevel,
         ...)
```

Arguments

object	object of class LogisticFitClass

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 to fit the series
silent	logical scalar. If TRUE, report of error messages will be suppressed in try(nlsmeth(...))
trace	logical scalar. Used in nls method.
conc	numeric vector containing estimates of the log concentration for each dilution 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 coef and coefficients methods return a named vector of length three with logistic curve coefficients.

The fitted method returns a numeric vector.

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.

Methods

coef signature(object = "LogisticFitClass"):
Extracts model coefficients from objects returned by modeling functions.

coefficients signature(object = "LogisticFitClass"):
An alias for coef

fitSeries signature(object = "LogisticFitClass"):
Finds the concentration for an individual dilution series given the curve fit for the slide.

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

fitted signature(object = "LogisticFitClass"):
Extracts fitted values of the model.

trimConc signature(object = "LogisticFitClass"):
Returns concentration and intensity cutoffs for the model.

Author(s)

P. Roebuck <proebuck@mdanderson.org>

See Also

FitClass

---

normalize *Normalization*

Description

This function performs normalization for sample loading after quantification. It is typically invoked as part of the process of creating summary information from an RPPASet object.

Usage

```
## S4 method for signature 'MatrixLike'
normalize(object,
          method=getRegisteredNormalizationMethodKeys(),
          calc.medians=TRUE,
          sweep.cols=calc.medians,
          ...)
```

Arguments

object	data frame or matrix to be normalized
method	character string specifying name of method of sample loading normalization (see section 'Details' below)
calc.medians	logical scalar. If TRUE, calculate row and column median values from the data to be normalized.
sweep.cols	logical scalar. If TRUE, subtract column medians from data values prior to invoking the normalization method.
...	extra arguments for normalization routines

Details

By default, column medians are subtracted from the input data values; these adjusted data values are then passed to the requested normalization routine for further processing.

The method argument may be augmented with user-provided normalization methods. Package-provided values are:

medpolish	Tukey's median polish normalization
median	sample median normalization
house	housekeeping normalization
vs	variable slope normalization

Specifying "median" as the method argument causes the row median to be subtracted from each sample. Specifying "house" causes the median of one or more housekeeping antibodies to be used. The names of the antibodies to be used must be supplied as a named argument to this method. Specifying "vs" causes the sample median to be used along with a multiplicative gamma (see reference below).

Value

Returns normalized concentrations as matrix appropriately annotated.

Author(s)

P. Roebuck <proebuck@mdanderson.org>, E. Shannon Neeley <sneeley@stat.byu.edu>

References

Neeley ES, Kornblau SM, Coombes KR, Baggerly KA.
Variable slope normalization of reverse phase protein arrays
Bioinformatics (2009) 25(11): 1384-1389.
http://bioinformatics.oxfordjournals.org/content/25/11/1384

See Also

RPPASet

Examples

```
antibodies <- c("FOO", "BAR", "PLUGH", "WALDO")
concs <- matrix(rnorm(1024),
                ncol=length(antibodies),
                dimnames=list(samples=NULL, antibodies=antibodies))

## Normalize using sample median
normconcs <- normalize(concs, method="median")
str(normconcs)

## Normalize using housekeeping antibodies
normconcs <- normalize(concs, method="house", antibodies=c("FOO", "PLUGH"))
str(normconcs)

## Normalize using variable slope
normconcs <- normalize(concs, method="vs")
str(normconcs)

## Normalize using Tukey's median polish (previous default method)
normconcs <- normalize(concs, method="medpolish", calc.medians=FALSE)
str(normconcs)

## Normalize using user-provided method (in this case, robust sample mean)
normalize.robustmean <- function(concs, trim=0, na.rm=FALSE) {
    stopifnot(is.matrix(concs) || is.data.frame(concs))
    stopifnot(is.numeric(trim))
```

```
    stopifnot(is.logical(na.rm))

    rowMean <- apply(concs, 1, mean, trim=trim, na.rm=na.rm)
    normconcs <- sweep(concs, 1, rowMean, FUN="-")

    ## Store method-specific info in "normalization" attribute
    attr(normconcs, "normalization") <- list(rowMean=rowMean)

    normconcs
}

registerNormalizationMethod("rmean", normalize.robustmean)
normconcs <- normalize(concs, method="rmean", trim=0.1, na.rm=TRUE)
str(normconcs)
```

---

normalize-method *Method "normalize"*

Description

normalize is a generic function used to normalize the data based on the input object. The method invokes particular methods which depend on the class of the first argument.

Usage

```
## S4 method for signature 'ANY'
normalize(object, ...)
## S4 method for signature 'NULL'
normalize(object, ...)
```

Arguments

object	an object to be normalized
...	additional arguments affecting the normalization process

Value

The form of the value returned by normalize depends on the class of its argument. See the documentation of the particular methods for details of what is produced by that method.

If the object is NULL, NA is returned.

Author(s)

P. Roebuck <proebuck@mdanderson.org>

ProgressMonitor-class *Class "ProgressMonitor"*

Description

The ProgressMonitor class is a virtual class for abstract reporting of progress of a task. All the generic methods here are placeholders that must be extended to be used.

Usage

```
is.ProgressMonitor(x)
## S4 method for signature 'ProgressMonitor'
progressDone(object, ...)
## S4 replacement method for signature 'ProgressMonitor,ANY'
progressDone(object, ...) <- value
## S4 method for signature 'ProgressMonitor'
progressError(object, ...)
## S4 replacement method for signature 'ProgressMonitor,ANY'
progressError(object, ...) <- value
## S4 method for signature 'ProgressMonitor'
progressLabel(object, ...)
## S4 replacement method for signature 'ProgressMonitor,ANY'
progressLabel(object, ...) <- value
## S4 method for signature 'ProgressMonitor'
progressMaximum(object, ...)
## S4 replacement method for signature 'ProgressMonitor,ANY'
progressMaximum(object, ...) <- value
## S4 method for signature 'ProgressMonitor'
progressMinimum(object, ...)
## S4 replacement method for signature 'ProgressMonitor,ANY'
progressMinimum(object, ...) <- value
## S4 method for signature 'ProgressMonitor'
progressValue(object, ...)
## S4 replacement method for signature 'ProgressMonitor,ANY'
progressValue(object, ...) <- value
```

Arguments

x	object of (sub)class ProgressMonitor
object	object of (sub)class ProgressMonitor
...	extra arguments for generic routines
value	value to be assigned

Value

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

Objects from the Class

This class should not be instantiated directly; extend this class instead.

Methods

progressDone `signature(object = "ProgressMonitor")`:
Placeholder method which must be implemented by subclass.

progressError `signature(object = "ProgressMonitor")`:
Placeholder method which must be implemented by subclass.

progressLabel `signature(object = "ProgressMonitor")`:
Placeholder method which must be implemented by subclass.

progressMaximum `signature(object = "ProgressMonitor")`:
Placeholder method which must be implemented by subclass.

progressMinimum `signature(object = "ProgressMonitor")`:
Placeholder method which must be implemented by subclass.

progressValue `signature(object = "ProgressMonitor")`:
Placeholder method which must be implemented by subclass.

Author(s)

P. Roebuck <proebuck@mdanderson.org>

ProgressMonitor-methods

Methods for Manipulating Progess Monitors

Description

These are generic functions used as accessors and mutators for objects of Progress-related classes.

`progressDone`**:** determines whether the task has been completed.

`progressError`**:** determines whether an error occurred attempting to complete the task.

`progressLabel`**:** determines label "applied" to the task.

`progressMinimum`**:** determines value associated with initiating the task.

`progressMaximum`**:** determines value associated with completing the task.

`progressValue`**:** determines value associated with percentage of the task completed.

The method invokes particular `methods` which depend on the `class` of the first argument.

Usage

```
## S4 method for signature 'ANY'
progressDone(object)
## S4 replacement method for signature 'ANY,ANY'
progressDone(object, ...) <- value
## S4 method for signature 'ANY'
progressError(object)
## S4 replacement method for signature 'ANY,ANY'
progressError(object, ...) <- value
## S4 method for signature 'ANY'
progressLabel(object)
## S4 replacement method for signature 'ANY,ANY'
progressLabel(object, ...) <- value
## S4 method for signature 'ANY'
progressMaximum(object)
## S4 replacement method for signature 'ANY,ANY'
progressMaximum(object, ...) <- value
## S4 method for signature 'ANY'
progressMinimum(object)
## S4 replacement method for signature 'ANY,ANY'
progressMinimum(object, ...) <- value
## S4 method for signature 'ANY'
progressValue(object)
## S4 replacement method for signature 'ANY,ANY'
progressValue(object, ...) <- value
```

Arguments

`object`	object of (sub)class `ProgressMonitor`
`value`	new value to apply
`...`	additional arguments affecting the updated values

Details

All functions are generic: you must write methods to handle specific classes of objects.

Value

The form of the value returned by these methods depends on the class of its argument. See the documentation of the particular methods for details of what is produced by that method.

Author(s)

P. Roebuck <proebuck@mdanderson.org>

See Also

`ProgressMonitor`, `DefaultProgressMonitor`

qcprob-method *Method "qcprob"*

Description

qcprob is a generic function used to produce a quality control probability based on the input object. The method invokes particular methods which depend on the class of the first argument.

Usage

```
## S4 method for signature 'ANY'
qcprob(object, ...)
## S4 method for signature 'NULL'
qcprob(object, ...)
```

Arguments

object	an object for which a QC probability is desired
...	additional arguments affecting the QC probability produced

Value

The form of the value returned by qcprob depends on the class of its argument. See the documentation of the particular methods for details of what is produced by that method.

If the object is NULL, NA is returned.

Author(s)

P. Roebuck <proebuck@mdanderson.org>

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 <proebuck@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 <proebuck@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)
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",
     alt.layout=NULL)
is.RPPA(x)
## S4 method for signature 'RPPA'
dim(x)
## S4 method for signature 'RPPA'
image(x, measure="Mean.Net", main,
      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)
alt.layout	character string specifying the name of the alternative layout to be used (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 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. Another method, read.arraypro, is also provided to read *ArrayPro* quantification files in text format.

Likewise, the logical layout of the slide can also be changed. The alt.layout argument is used in the selection of the actual method to perform this function. For example, if the argument value is "bar", the code will attempt to invoke method layout.as.bar to convert the physical layout of the data to that specified by the method itself.

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 `signature(x = "RPPA")`:
Returns the dimensions of the slide layout.

image `signature(x = "RPPA")`:
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, placed along the right edge.

summary `signature(object = "RPPA")`:
Prints a summary of the underlying data frame.

Note

The previous release provided a method, `read.singlesubgrid`, which could convert a single subgrid physical layout (in MicroVigene format) into its actual logical one.

As this package now supports more than one software package, this had to be reworked! To achieve the same thing in this release, use:
`RPPA(file, path, software="microvigene", alt.layout="superslide")`

Author(s)

Kevin R. Coombes <kcoombes@mdanderson.org>, P. Roebuck <proebuck@mdanderson.org>

See Also

RPPADesign, RPPAFit

Examples

```
extdata.dir <- system.file("extdata", package="SuperCurveSampleData")

## Converts file from single subgrid to its logical equivalent (4x12x11x11)
txtdir <- file.path(extdata.dir, "rppaSingleSubgridData")
waldo <- RPPA("Waldo.txt",
              path=txtdir,
              software="microvigene",
              alt.layout="superslide")
dim(waldo)

txtdir <- file.path(extdata.dir, "rppaTumorData")
erk2 <- RPPA("ERK2.txt", path=txtdir)
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.

See corresponding manpage of the raw data for a description of the design of the RPPA.

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.

Details

The corresponding raw datafiles are available in the '`extdata/rppaCellData`' subdirectory of the **SuperCurveSampleData** package.

Author(s)

P. Roebuck <proebuck@mdanderson.org>

See Also

rppaCell-extdata

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, main, ...)
## S4 method for signature 'RPPADesign'
names(x)
## S4 method for signature 'RPPADesignParams'
```

```
paramString(object, slots, ...)
## S4 method for signature 'RPPA,RPPADesign'
plot(x, y, measure, main, ...)
## 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	Label to use in lieu of sample name
Sample	Sample name to match

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
slots	strings specifying RPPADesignParams slotnames to display (for debuggging)
main	overall title for plot
measure	character string specifying measure to plot
...	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 *SampleType* 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 `signature(x = "RPPADesign")`:
Returns the dimensions of the slide layout.

image `signature(x = "RPPADesign")`:
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.

names `signature(x = "RPPADesign")`:
Returns the names of the samples on the slide.

paramString `signature(object = "RPPADesignParams")`:
Returns string representation of object.

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

summary `signature(object = "RPPADesign")`:
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 <proebuck@mdanderson.org>

See Also

RPPA

Examples

```
extdata.dir <- system.file("extdata", package="SuperCurveSampleData")

txtdir <- file.path(extdata.dir, "rppaTumorData")
erk2 <- RPPA("ERK2.txt", path=txtdir)
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)
```

```
txtdir <- file.path(extdata.dir, "rppaCellData")
akt <- RPPA("Akt.txt", path=txtdir)
## 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"),
      main,
      ...)
## S4 method for signature 'RPPAFit,missing'
plot(x, y,
     type=c("cloud", "series", "individual", "steps", "resid"),
```

```
     col=NULL,
     main,
     xform=NULL,
     xlab="Log Concentration",
     ylab="Intensity",
     ...)
## 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
xform	function to transform the raw data associated with the measure for the plot. If NULL, no transformation occurs.
y	not used
col	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 `signature(object = "RPPAFit")`:
Extracts model coefficients from objects returned by modeling functions.

coefficients `signature(object = "RPPAFit")`:
An alias for `coef`.

fitted `signature(object = "RPPAFit")`:
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 `signature(x = "RPPAFit")`:
Produces a histogram of the residuals. The exact form of the residuals being displayed depends on the value of the `type` argument.

image `signature(x = "RPPAFit")`:
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 `signature(x = "RPPAFit", y = "missing")`:
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 `signature(object = "RPPAFit")`:
An alias for `residuals`.

residuals `signature(object = "RPPAFit")`:
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 `signature(object = "RPPAFit")`:
Prints a summary of the `RPPAFit` object, which reports the function call used to fit the model and important fitting parameters.

Author(s)

Kevin R. Coombes <kcoombes@mdanderson.org>, P. Roebuck <proebuck@mdanderson.org>

See Also

RPPA, RPPADesign, RPPAFit, hist

Examples

```
extdata.dir <- system.file("extdata", package="SuperCurveSampleData")

txtdir <- file.path(extdata.dir, "rppaTumorData")
erk2 <- RPPA("ERK2.txt", path=txtdir)
design <- RPPADesign(erk2,
                     grouping="blockSample",
                     controls=list("neg con", "pos con"))
erk2.fit <- RPPAFit(erk2, design, "Mean.Net")
```

```
showMethods('image')
class(erk2.fit)

image(erk2.fit)
image(erk2.fit, measure="Residuals")
plot(erk2.fit, type="cloud")
coef(erk2.fit)

jnk <- RPPA("JNK.txt", path=txtdir)
jnk.fit <- RPPAFit(jnk, design, "Mean.Net")
hist(jnk.fit, type="StdRes")
plot(jnk.fit, type="series")
coef(jnk.fit)
plot(fitted(jnk.fit), resid(jnk.fit))
```

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,
                  progmethod=NULL)

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

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.
progmethod	optional function that can be used to report progress.
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, computes 90% confidence intervals on the concentration estimates.
ignoreNegative	logical scalar. If TRUE, converts 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, prints updates while fitting the data
veryVerbose	logical scalar. If TRUE, prints 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)
slots	strings specifying RPPAFitParams slotnames to display (for debuggging)
...	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 scalar; see arguments above

ignoreNegative: logical scalar; see arguments above

trace: logical scalar; see arguments above

verbose: logical scalar; see arguments above

veryVerbose: logical scalar; see arguments above

warnLevel: numeric; see arguments above

trim: numeric; see arguments above

model: character; see arguments above

Methods

paramString signature(object = "RPPAFitParams"):
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 <proebuck@mdanderson.org>, Kevin R. Coombes <kcoombes@mdanderson.org>

See Also

RPPAFit, RPPAFit-class, RPPA, RPPADesign

Examples

```
extdata.dir <- system.file("extdata", package="SuperCurveSampleData")

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

RPPANormalizationParams-class

Class "RPPANormalizationParams"

Description

The RPPANormalizationParams class is used to bundle the parameter set together that control how to perform spatial adjustment into a reuseable object.

Usage

```
RPPANormalizationParams(method,
                        arglist=NULL)
is.RPPANormalizationParams(x)
## S4 method for signature 'RPPANormalizationParams'
paramString(object, slots, ...)
```

Arguments

`method`	character string specifying normalization method to use
`arglist`	list of named key/value pairs representing argument list to be passed upon invocation of `normalize` method
`object`	object of class `RPPANormalizationParams`
`x`	object of class `RPPANormalizationParams`
`slots`	strings specifying `RPPANormalizationParams` slotnames to display (for debuggging)
`...`	extra arguments for generic routines

Details

The `method` argument is combined with the `arglist` argument prior to invocation of `normalize` method.

Value

The RPPANormalizationParams generator returns an object of class RPPANormalizationParams.

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

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 RPPANormalizationParams generator function.

Slots

name: character string; see arguments above

method: character string; see arguments above

arglist: list of named key/value pairs; see arguments above

Methods

paramString `signature(object = "RPPANormalizationParams")`:
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 <proebuck@mdanderson.org>

See Also

normalize

Examples

```
showClass("RPPANormalizationParams")
normparams <- RPPANormalizationParams(method="medpolish",
                                      arglist=list(calc.medians=FALSE))
paramString(normparams)
```

RPPAPreFitQC-class *Class "RPPAPreFitQC"*

Description

The RPPAPreFitQC class represents the inputs necessary to determine the quality control rating of a reverse-phase protein array slide.

Usage

```
RPPAPreFitQC(rppa, design, useAdjusted=TRUE)
is.RPPAPreFitQC(x)
## S4 method for signature 'RPPAPreFitQC'
qcprob(object, ...)
## S4 method for signature 'RPPAPreFitQC'
summary(object, ...)
```

Arguments

rppa	object of class RPPA containing the raw data to be assessed
design	object of class RPPADesign describing the layout of the array
useAdjusted	logical scalar. If TRUE, spatially adjusted measures are used instead of Mean.Net and Mean.Total.
object	object of (sub)class RPPAPreFitQC
x	object of (sub)class RPPAPreFitQC
...	extra arguments for generic routines

Value

The RPPAPreFitQC generator returns an object of subclass of class RPPAPreFitQC.

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

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

Objects from the Class

Objects are created by calls to the RPPAPreFitQC factory method.

Methods

qcprob signature(object = "RPPAPreFitQC"):
Placeholder method which must be implemented by subclass.

summary signature(object = "RPPAPreFitQC"):
Placeholder method which must be implemented by subclass.

Warning

The current implementation only handles designs with 5 dilution series. Anything else will fail.

Author(s)

P. Roebuck <proebuck@mdanderson.org>

Examples

```
extdata.dir <- system.file("extdata", package="SuperCurveSampleData")

txtdir <- file.path(extdata.dir, "rppaSingleSubgridData")
rppa <- RPPA("Foo.txt",
             path=txtdir,
             software="microvigene",
             alt.layout="superslide")
designfile <- file.path(txtdir, "slidedesign.tsv")
design <- RPPADesign(rppa, designfile=designfile)
rppa <- spatialAdjustment(rppa, design)
fitqc <- RPPAPreFitQC(rppa, design)
summary(fitqc)
qcprob(fitqc)
```

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,
        spatialparams=NULL,
        normparams,
        doprefitqc=FALSE,
        monitor=SCProgressMonitor(),
        antibodyfile=NULL,
        software="microvigene",
        alt.layout=NULL)
is.RPPASet(x)
## S4 method for signature 'RPPASet'
normalize(object,
          ...)
```

```
## S4 method for signature 'RPPASet'
summary(object,
        onlynormqcgood=ran.prefitqc(object),
        ...)
## S4 method for signature 'RPPASet'
write.summary(object,
              path,
              prefix="supercurve",
              graphs=TRUE,
              tiffdir=NULL,
              onlynormqcgood=ran.prefitqc(object),
              monitor=NULL,
              ...)
```

Arguments

path	character string specifying a directory. In the case of the RPPASet generator, it specifies the directory containing the quantification files to be processed. In the case of the write.summary method, it specifies the directory where output should be stored.
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
spatialparams	object of class RPPASpatialParams containing parameters used to perform spatial adjustment, or NULL
normparams	object of class RPPANormParams containing parameters used to normalize the concentrations
doprefitqc	logical scalar. If TRUE, performs pre-fit quality control.
monitor	object of (sub)class ProgressMonitor
antibodyfile	character string specifying filename containing mapping from quantification files to antibodies
software	character string specifying the software used to generate the quantification file (see section 'Details' of RPPA)
alt.layout	character string specifying the name of the alternative layout to be used (see section 'Details' of RPPA)
object	object of class RPPASet
prefix	character string used as a filename prefix on files generated by the write.summary method.
graphs	logical scalar. If TRUE, produces fit graphs.
tiffdir	character string specifying the directory containing the TIFF images corresponding to the quantification files
onlynormqcgood	logical scalar. If TRUE, filters the slides to be normalized according to their pre-fit quality control scores.
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 write.summary to write the resulting quantifications and diagnostic plots to a directory.

The write.summary method (indirectly) generates three CSV files: one for the raw concentrations, one for the R^2 statistics, and one for the normalized 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.

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 an object of class RPPASetSummary.

The write.summary method invisibly returns NULL.

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

spatialparams object of class RPPASpatialParams that was used to perform spatial adjustment, or NULL

prefitqcs: array of objects of class RPPAPreFitQCParams

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

normparams: object of class RPPANormalizationParams used to normalize the raw concentrations

fits: array of fitted objects of class RPPAFit

completed: logical matrix specifying stage completion for each slide

Methods

normalize signature(object = "RPPASet"):
Assembles matrix of concentrations from all fits in object, using the object's normalization settings.

summary signature(object = "RPPASet"):
Creates an object of class RPPASetSummary.

write.summary `signature(object = "RPPASet")`:
Writes a record of the entire RPPASet, including fitted values, residuals, and images of the processed slides.

Author(s)

Kevin R. Coombes <kcoombes@mdanderson.org>, P. Roebuck <proebuck@mdanderson.org>

See Also

RPPA, RPPADesign, RPPAFit, RPPASetSummary, SCProgressMonitor

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)
normparams <- RPPANormalizationParams(method="vs")
monitor <- SCProgressMonitor()
rppaset <- RPPASet(txtdir,
                   designparams,
                   fitparams,
                   normparams=normparams,
                   monitor=monitor)
write.summary(rppaset,
              path=outdir,
              graphs=TRUE,
              tiffdir=imgdir,
              monitor=monitor)

## End(Not run)
```

RPPASetSummary-class *Class "RPPASetSummary"*

Description

The RPPASetSummary class contains the summary information derived from an RPPASet object.

Usage

```
RPPASetSummary(rppaset,
               onlynormqcgood=ran.prefitqc(rppaset),
               monitor=NULL)
is.RPPASetSummary(x)
## S4 method for signature 'RPPASetSummary'
write.summary(object,
              path,
              prefix="supercurve",
              monitor=NULL,
              ...)
```

Arguments

rppaset	object of class RPPASet
onlynormqcgood	logical scalar. If TRUE, filters the slides to be normalized according to their pre-fit quality control scores.
monitor	object of class SCProgressMonitor
x	object of class RPPASetSummary
object	object of class RPPASetSummary
path	character string specifying the path from the current directory to the directory containing the files to be processed
prefix	character string used as a prefix on files generated by the write.summary method
...	extra arguments for generic routines

Value

The RPPASetSummary generator returns an object of class RPPASetSummary.

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

The write.summary method invisibly returns NULL.

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 RPPASetSummary generator function.

Slots

raw: numeric matrix of raw concentrations

ss: numeric matrix of R^2 statistical values

norm: numeric matrix of normalized concentrations

probs: numeric vector of goodness of fit probabilities

completed: logical matrix specifying stage completion for each slide

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

`onlynormqcgood`**:** logical scalar specifying if raw concentrations were filtered according to their pre-fit quality control scores prior to normalization

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

Methods

write.summary `signature(object = "RPPASetSummary")`:
Generates three CSV files: one for the raw concentrations, one for the R^2 statistics, and one for the normalized concentrations; a fourth file containing the goodness of fit probabilities may be present if prefit QC analysis was requested. Additionally, a TSV file detailing completion of each stage of processing for each slide is produced.

Note

The three CSV files may be reordered (to match that of the original input) when written to disk.

Author(s)

P. Roebuck <proebuck@mdanderson.org>

See Also

RPPASet

Examples

```
  ## Not run:
parentdir <- file.path("C:", "MyData")
txtdir <- file.path(parentdir, "txt")     # quantification 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)
normparams <- RPPANormalizationParams(method="vs")
rppaset <- RPPASet(txtdir,
                   designparams,
                   fitparams,
                   normparams=normparams)
## If you REALLY want to do this manually. It will be invoked
## automatically if you invoke write.summary(rppaset) instead...
write.summary(summary(rppaset),
              path=outdir,
              graphs=FALSE)
```

```
## End(Not run)
```

---

rppaSingleSubgrid-data

FOO, BAR, PLUGH, and WALDO expression

Description

This data set contains the expression levels of four imaginary proteins: FOO, BAR, PLUGH, and WALDO. Each sample on the slide is printed as a 5-step dilution series using reverse-phase protein arrays.

See corresponding manpage of the raw data for a description of the design of the RPPA.

Usage

```
data(rppaSingleSubgrid)
```

Format

The objects `foo`, `bar`, `plugh`, and `waldo` are objects of class `RPPA` converted to use logical layout. The object `sdesign` is an object of class `RPPADesign` describing that logical layout.

Details

The corresponding raw datafiles are available in the '`extdata/rppaSingleSubgridData`' subdirectory of the **SuperCurveSampleData** package.

Author(s)

P. Roebuck <proebuck@mdanderson.org>

See Also

`rppaSingleSubgrid-extdata`

RPPASpatialParams-class
Class "RPPASpatialParams"

Description

The RPPASpatialParams class is used to bundle the parameter set together that control how to perform spatial adjustment into a reuseable object.

Usage

```
RPPASpatialParams(cutoff=0.8,
                  k=100,
                  gamma=0.1,
                  plotSurface=FALSE)
is.RPPASpatialParams(x)
## S4 method for signature 'RPPASpatialParams'
paramString(object, slots, ...)
```

Arguments

cutoff	numeric scalar used to identify the background cutoff with value in closed interval [0..1]. Default is 0.8.
k	numeric scalar used as smoothing model argument. Default is 100.
gamma	numeric scalar used as model parameter with value in closed interval [0..2]. Default is 0.1.
plotSurface	logical scalar. If TRUE, plots surfaces. Default is FALSE.
object	object of class RPPASpatialParams
x	object of class RPPASpatialParams
slots	strings specifying RPPASpatialParams slotnames to display (for debuggging)
...	extra arguments for generic routines

Details

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

The RPPASpatialParams generator returns an object of class RPPASpatialParams.

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

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 RPPASpatialParams generator function.

Slots

cutoff: numeric scalar; see arguments above

k: numeric scalar; see arguments above

gamma: numeric scalar; see arguments above

plotSurface: logical scalar; 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 <proebuck@mdanderson.org>

See Also

spatialCorrection

Examples

```
showClass("RPPASpatialParams")
spatialparams <- RPPASpatialParams()
paramString(spatialparams)
```

---

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 "starved" 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.

See corresponding manpage of the raw data for a description of the design of the RPPA.

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.

Details

The corresponding raw datafiles are available in the 'extdata/rppaTripleData' subdirectory of the **SuperCurveSampleData** package.

Author(s)

P. Roebuck <proebuck@mdanderson.org>

See Also

rppaTriple-extdata

---

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.

See corresponding manpage of the raw data for a description of the design of the RPPA.

Usage

```
data(rppaTumor)
```

Format

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

Details

The corresponding raw datafiles are available in the 'extdata/rppaTumorData' subdirectory of the **SuperCurveSampleData** package.

Author(s)

P. Roebuck <proebuck@mdanderson.org>

See Also

rppaTumor-extdata

SCProgressMonitor-class

Class "SCProgressMonitor"

Description

The SCProgressMonitor class represents an attempt to abstract reporting of progress of a task. This class assumes that progress is reported via a progressbar and provides means to get/set values for such a widget.

Usage

```
SCProgressMonitor(stage="")
is.SCProgressMonitor(x)
getStages()
## S4 method for signature 'SCProgressMonitor'
progressMarquee(object)
## S4 replacement method for signature 'SCProgressMonitor,character'
progressMarquee(object) <- value
## S4 method for signature 'SCProgressMonitor'
progressStage(object)
## S4 replacement method for signature 'SCProgressMonitor,character'
progressStage(object) <- value
```

Arguments

stage	string specifying current stage of task
x	object of (sub)class SCProgressMonitor
object	object of (sub)class SCProgressMonitor
value	value to be assigned

Value

The SCProgressMonitor generator returns an object of class SCProgressMonitor.

The getStages method returns a named character vector specifying the stages processed by this package.

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

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 SCProgressMonitor generator function.

Slots

`stage`: string specifying current stage of task

`marquee`: string specifying marquee for current task

`range`: object of class `BoundedRange`

`label`: string specifying detail label for current task

`done`: logical scalar specifying if task completed. Default is `FALSE`.

`err`: logical scalar specifying if an error has occurred. Default is `FALSE`.

Extends

Class `DefaultProgressMonitor`, directly. Class `ProgressMonitor`, by class "DefaultProgressMonitor", distance 2.

Methods

progressStage `signature(object = "SCProgressMonitor")`:
Returns string representing current stage of task.

progressStage<- `signature(object = "SCProgressMonitor", value = "character")`:
Sets value of the `stage` slot.

progressMarquee `signature(object = "SCProgressMonitor")`:
Returns string representing marquee for current stage.

progressMarquee<- `signature(object = "SCProgressMonitor", value = "character")`:
Sets value of the `marquee` slot.

Author(s)

P. Roebuck <proebuck@mdanderson.org>

See Also

`BoundedRange`, `ElapsedTime`, `ProgressMonitor`, `DefaultProgressMonitor`

Examples

```
showClass("SCProgressMonitor")
niters <- 10
scpm <- SuperCurve:::SCProgressMonitor("input")
progressMarquee(scpm) <- "Read input files"
for (i in seq.int(niters)) {
    ## Perform portion of task
    progressValue(scpm) <- i # Modify current value
}
```

SCProgressMonitor-methods
Methods for Manipulating SuperCurve Progess Models

Description

These are generic functions used as accessors and mutators for SuperCurve-specific additions for objects of Progress-related subclasses.

`progressMarquee`: determines marquee "applied" to the task.

`progressStage`: determines current stage of the task.

The method invokes particular `methods` which depend on the `class` of the first argument.

Usage

```
## S4 method for signature 'ANY'
progressMarquee(object)
## S4 replacement method for signature 'ANY'
progressMarquee(object, ...) <- value
## S4 method for signature 'ANY'
progressStage(object)
## S4 replacement method for signature 'ANY'
progressStage(object, ...) <- value
```

Arguments

`object`	object of (sub)class `SCProgressMonitor`
`value`	new value to apply
`...`	additional arguments affecting the updated values

Details

All functions are generic: you must write methods to handle specific classes of objects.

Value

The form of the value returned by these methods depends on the class of its argument. See the documentation of the particular methods for details of what is produced by that method.

Author(s)

P. Roebuck <proebuck@mdanderson.org>

See Also

`ProgressMonitor`, `DefaultProgressMonitor`

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

```
spatialAdjustmentFromParams(rppa,
                            design,
                            spatialparams)
spatialAdjustment(rppa,
                  design,
                  cutoff=0.8,
                  k=100,
                  gamma=0.1,
                  plotSurface=FALSE)
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
spatialparams	object of class RPPASpatialParams containing parameters used to perform spatial adjustment
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. If TRUE, plots 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 after the `measure` with an `Adj.` prefix. For example, if the measure was `Mean.Net`, the name of the adjusted column would be `Adj.Mean.Net`.

Author(s)

P. Roebuck <proebuck@mdanderson.org>, E. Shannon Neeley <sneeley@stat.byu.edu>

References

Neeley ES, Baggerly KA, Kornblau SM.
Surface Adjustment of Reverse Phase Protein Arrays Using Positive Control Spots
Cancer Informatics (2012) 11: 77-86.
http://www.ncbi.nlm.nih.gov/pubmed/22550399

See Also

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

Examples

```
extdata.dir <- system.file("extdata", package="SuperCurveSampleData")

txtdir <- file.path(extdata.dir, "rppaSingleSubgridData")
designfile <- file.path(txtdir, "slidedesign.tsv")
rppa <- RPPA("Waldo.txt",
             path=txtdir,
             software="microvigene",
             alt.layout="superslide")
designparams <- RPPADesignParams(grouping="blockSample",
                                 designfile=designfile)
design <- RPPADesignFromParams(rppa, designparams)
rppa.adj <- spatialAdjustment(rppa, design)
colnames(rppa.adj)
```

SuperCurveSettings-class

Class "SuperCurveSettings"

Description

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

Usage

```
SuperCurveSettings(txtdir,
                   imgdir,
                   outdir,
                   designparams,
                   fitparams,
                   spatialparams=NULL,
                   normparams,
                   doprefitqc=FALSE,
                   onlynormqcgood=doprefitqc,
                   antibodyfile=NULL,
                   software=NULL,
                   alt.layout=NULL)
fitCurveAndSummarizeFromSettings(settings, monitor=NULL)
is.SuperCurveSettings(x)
## S4 method for signature 'SuperCurveSettings'
write.summary(object,
              path=as(settings@outdir, "character"),
              ...)
## S4 method for signature 'SuperCurveSettings'
paramString(object,
            designparams.slots,
            fitparams.slots,
```

```
            spatialparams.slots,
            normparams.slots,
            ...)
```

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, or NULL
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
spatialparams	object of class RPPASpatialParams, or NULL
normparams	object of class RPPANormalizationParams
doprefitqc	logical scalar. If TRUE, performs pre-fit quality control.
onlynormqcgood	logical scalar. If TRUE, filters the slides to be normalized according to their pre-fit quality control scores.
antibodyfile	character string specifying filename containing mapping from quantification files to antibodies, or NULL
software	character string specifying the software used to generate the quantification files (see section 'Details' of RPPA), or NULL to use the default value.
alt.layout	character string specifying the name of the alternative layout to be used (see section 'Details' of RPPA), or NULL to use the implicit layout.)
monitor	object of (sub)class ProgressMonitor, or NULL
object	object of class SuperCurveSettings
settings	object of class SuperCurveSettings
x	object of class SuperCurveSettings
path	character string specifying the directory where settings summary should be saved. Must be writable.
designparams.slots	strings specifying RPPADesignParams slotnames to display (for debuggging)
fitparams.slots	strings specifying RPPAFitParams slotnames to display (for debuggging)
spatialparams.slots	strings specifying RPPASpatialParams slotnames to display (for debuggging)
normparams.slots	strings specifying RPPANormalizationParams slotnames to display (for debuggging)
...	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.

The write.summary method invisibly returns 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

spatialparams: object of class RPPASpatialParams specifying the parameters that control spatial adjustment

normparams: object of class RPPANormalizationParams specifying the parameters that control normalization

doprefitqc: logical scalar specifying whether to perform pre-fit quality control

onlynormqcgood: logical scalar specifying whether to filter the slides to be normalized according to their pre-fit quality control scores

antibodyfile: character string specifying filename containing mapping from quantification files to antibodies, or NULL

software: character string specifying the software used to generate the quantification files, or NULL

alt.layout: character string specifying the name of the alternative layout to be used, or NULL

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

Methods

paramString signature(object = "SuperCurveSettings"):
Returns string representation of object.

write.summary signature(object = "SuperCurveSettings"):
Writes a text file 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 <proebuck@mdanderson.org>

See Also

Directory, RPPADesignParams, RPPASpatialParams, RPPAFitParams, RPPANormalizationParams

Examples

```
  ## Not run:
extdata.dir <- system.file("extdata", package="SuperCurveSampleData")

txtdir <- file.path(extdata.dir, "rppaTumorData")
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)
normparams <- RPPANormalizationParams(method="median")
settings <- SuperCurveSettings(txtdir=txtdir,
                               imgdir=NULL,
                               outdir=tempdir(),
                               designparams=designparams,
                               spatialparams=NULL,
                               fitparams=fitparams,
                               normparams=normparams)
fitCurveAndSummarizeFromSettings(settings)

## End(Not run)
```

---

write.summary-method *Method "write.summary"*

Description

write.summary is a generic function used like a summary method that writes to disk, saving summary information from the object in an external format. The method invokes particular methods which depend on the class of the first argument.

Usage

```
## S4 method for signature 'ANY'
write.summary(object, ...)
```

Arguments

`object`	an object for which saving summary information externally is desired
`...`	additional arguments affecting the summary information produced

Note

Exactly what is written to disk by `write.summary` depends on the class of its argument. See the documentation of the particular methods for details of what is written by that method.

Author(s)

P. Roebuck <proebuck@mdanderson.org>

Index