

# Package ‘TIMP’

April 17, 2009

**Type** Package

**Title** a problem solving environment for fitting separable nonlinear models in physics and chemistry applications

**Version** 1.8

**Author** Katharine M. Mullen, Sergey Laptanok, David Nicolaides, Ivo H. M. van Stokkum

**Maintainer** Katharine M. Mullen <kate@nat.vu.nl>

**Depends** R (>= 2.7.0), methods, tcltk, vcd, fields (>= 4.1), gplots, splines, gclus, nls (>= 1.1), odesolve, minpack.lm (>= 1.1-1)

**Description** TIMP is a problem solving environment for fitting separable nonlinear models to measurements arising in physics and chemistry experiments, and has been extensively applied to time-resolved spectroscopy and FLIM-FRET data.

**License** GPL (>= 2)

**URL** <http://timp.r-forge.r-project.org/>, <http://www.nat.vu.nl/~kate/TIMP/>

**Repository** CRAN

**Date/Publication** 2008-10-13 09:57:51

## R topics documented:

TIMP-package	2
amp-class	3
baseIRF	4
dat-class	5
denS4	9
divergeZimage	9
donorAcceptorTagged	11
donorTagged	11
eFit2file	12
examineFit	13
fit-class	13

fitModel . . . . .	14
FLIMplots . . . . .	17
getClpindepX-methods . . . . .	21
getResid . . . . .	22
getResults . . . . .	22
initModel . . . . .	27
kin-class . . . . .	30
kinopt-class . . . . .	35
mass-class . . . . .	36
massopt-class . . . . .	37
mea_IRF . . . . .	38
modifyModel . . . . .	39
multimodel-class . . . . .	40
multitheta-class . . . . .	41
opt-class . . . . .	41
outlierCorr . . . . .	44
plotter-methods . . . . .	45
preProcess . . . . .	46
readclp0 . . . . .	50
readData . . . . .	51
res-class . . . . .	52
residPart-methods . . . . .	53
spec-class . . . . .	54
specopt-class . . . . .	55
sumKinSpecEst . . . . .	56
target . . . . .	57
theta-class . . . . .	57
writeAverage . . . . .	58
<b>Index</b>	<b>60</b>

---

TIMP-package	<i>a problem solving environment for fitting separable nonlinear models in physics and chemistry applications</i>
--------------	---

---

## Description

TIMP is a problem solving environment for fitting separable nonlinear models to measurements arising in physics and chemistry experiments, and has been extensively applied to time-resolved spectroscopy and FLIM-FRET data.

## Author(s)

Katharine M. Mullen, Ivo H. M. van Stokkum Maintainer: Katharine M. Mullen (kate@nat.vu.nl)

## References

Mullen KM, van Stokkum IHM (2007). "TIMP: An R Package for Modeling Multi-way Spectroscopic Measurements." Journal of Statistical Software, 18(3). URL <http://www.jstatsoft.org/v18/i03/>.

Laptenok S, Mullen KM, Borst JW, van Stokkum IHM, Apanasovich VV, Visser AJWG (2007). "Fluorescence Lifetime Imaging Microscopy (FLIM) Data Analysis with TIMP." Journal of Statistical Software, 18(8). URL <http://www.jstatsoft.org/v18/i08/>.

See <http://timp.r-forge.r-project.org/> or <http://www.nat.vu.nl/~kate/TIMP/> for further documentation.

---

amp-class

*Class "amp" for diagonal matrix model specification.*

---

## Description

amp is the class for diagonal matrix model specification; such models are internally initialized when a tri-linear-type model is fit to the data via passing the argument `opt` to `fitModel` as an object of class `opt` in which the slot `trilinear` has the value `TRUE`. All objects of class `amp` are sub-classes of class `dat`; see documentation for `dat` for a description of these slots.

## Details

See [kin-class](#) for an example of the initialization of a `kin` object via the `initModel` function.

## Objects from the Class

Objects can be created by calls of the form `new("amp", ...)` or `amp(...)`.

## Slots

**amp** list of vectors of starting values for the parameters of the amplitudes for each dataset; one vector of values is used to parameterize the values corresponding to each dataset.

## Extends

Class [kin-class](#), directly.

## Author(s)

Katharine M. Mullen, Ivo H. M. van Stokkum

## See Also

[kin-class](#), [spec-class](#), [opt-class](#)

---

baseIRF	<i>Baseline subtraction from a vector, usually representing an IRF.</i>
---------	---

---

**Description**

Baseline subtraction from a vector, usually representing an IRF.

**Usage**

```
baseIRF(irfvec, indexlow, indexhigh, removeNeg = FALSE)
```

**Arguments**

<code>irfvec</code>	Vector to subtract a baseline from
<code>indexlow</code>	Lowest index to base the baseline estimation on
<code>indexhigh</code>	Highest index to base the baseline estimation on
<code>removeNeg</code>	Whether negative values should be replaced with 0.

**Details**

Currently estimates the baseline as the mean of data between `indexlow` and `indexhigh`, and subtracts the result from the entire vector.

**Value**

vector

**Author(s)**

Katharine M. Mullen, Ivo H. M. van Stokkum

**Examples**

```
irfvec <- rnorm(128, mean=1)
plot(irfvec, type="l")
irfvec_corrected <- baseIRF(irfvec, 1, 10)
lines(irfvec_corrected, col=2)
```

dat-class

Class "dat" for model and data storage

## Description

dat is the super-class of other classes representing models and data, so that other model/data classes (e.g., kin and spec for kinetic and spectral models respectively) also have the slots defined here. Slots whose description are marked with **\*\*\*** may be specified in the ... argument of the `initModel` function.

## Objects from the Class

Objects from the class can be created by calls of the form `new("dat", ...)` or `dat(...)`, but most are most often made by invoking another function such as `readData` or `initModel`.

## Slots

**weightpar:** **\*\*\*** Object of class "list" list of vectors `c(first_x, last_x, first_x2, last_x2, weight)`, where each vector is of length 5 and specifies an interval in which to weight the data.

`first_x` first(absolute, not an index) x to weight

`last_x` last (absolute, not an index) x to weight

`first_x2` first (absolute, not an index) x2 to weight

`last_x2` last (absolute, not an index) x2 to weight

`weight` numeric by which to weight data

Note that if vector elements 1-4 are NA (not a number), the firstmost point of the data is taken for elements 1 and 3, and the lastmost points are taken for 2 and 4. For example, `weightpar = list(c(40, 1500, 400, 600, .9), c(NA, NA, 700, 800, .1))` will weight data between times 40 and 1500 picoseconds and 700 and 800 wavelengths by .9, and will weight data at all times between wavelength 700 and 800 by .1. Note also that for single photon counting data `weightpar = list(poisson = TRUE)` will apply poisson weighting to all non-zero elements of the data.

**mod\_type:** **\*\*\*** Object of class "character" character string defining the model type, e.g., "kin" or "spec"

**fixed:** **\*\*\*** Object of class "list" list of lists or vectors giving the parameter values to fix (at their starting values) during optimization.

**free:** **\*\*\*** Object of class "list" list of lists or vectors giving the parameter values to free during optimization; if this list is present then all parameters not specified in it are fixed, e.g., `free = list(irfpar = 2)` will fix every parameter at its starting value except for the 2nd `irfpar`. If `fix = list(none=TRUE)` (or if the element `none` has length greater than 0) then all parameters in the model are fixed. Note that this option only should be applied to multiexperiment models in which at least one parameter applying to some other dataset is optimized (`nls` always must have at least one parameter to optimize).

- constrained:** \*\*\* Object of class "list" list whose elements are lists containing a character vector *what*, a vector *ind*, and either (but not both) a character vector *low* and *high*. *what* should specify the parameter type to constrain. *ind* should give the index of the parameter to be constrained, e.g., 1 if indexing into a vector, and *c*(1,2) if indexing into a list. *low* gives a number that the parameter should always remain lower than and *high* gives a number that the parameter should always remain higher than (so that *low* bounds the parameter value from above and *high* bounds the parameter value from below). It is not now possible to specify both *low* and *high* for a single parameter value. An example of a complete constrained specification is `constrained = list(list(what = "kinpar", ind = 2, low = .3), list(what = "parmu", ind = c(1,1), high = .002))`
- clp0:** \*\*\* Object of class "list" list of lists with elements *low*, *high* and *comp*, specifying the least value in *x2* to constrain to zero, the greatest value in *x2* to constrain to zero, and the component to which to apply the zero constraint, respectively. e.g., `clp0 = list(list(low=400, high = 600, comp=2), list(low = 600, high = 650, comp=4))` applies zero constraints to the spectra associated with components 2 and 4.
- autoclpo:** \*\*\* Object of class "list" that has two elements; *oldRes*, the output of `fitModel` and an index *ind* representing the index of the dataset to use in *oldRes*; *ind* defaults to one. The *clp* that are negative in *oldRes* are constrained to zero in the new model; this is primarily useful when fitting a model, finding some negative *clp*, and constraining them to zero by fitting again with this option. See also the help page for `opt` for other ways to constrain the *clp* to non-negativity.
- clpequspec:** \*\*\* Object of class "list" list of lists each of which has elements *to*, *from*, *low*, *high*, and optional element *dataset* to specify the dataset from which to get the reference *clp* (that is, a spectrum for kinetic models). *to* is the component to be fixed in relation to some other component; *from* is the reference component. *low* and *high* are the least and greatest absolute values of the *clp* vector to constrain. e.g., `clpequspec = list(list(low = 400, high = 600, to = 1, from = 2))` will constrain the first component to equality to the second component between wavelengths 400 and 600. Note that equality constraints are actually constraints to a linear relationship. For each of the equality constraints specified as a list in the `clpequspec` list, specify a starting value parameterizing this linear relation in the vector *clpequ*; if true equality is desired then fix the corresponding parameter in *clpequ* to 1. Note that if multiple components are constrained, the *from* in the sublists should be increasing order, (i.e., `(list(to=2, from=1, low=100, high=10000), list(to=3, from=1, low=10000, high=100))`, not `list(to=3, from=1, low=10000, high=100), list(to=2, from=1, low=10000, high=100)`)
- clpequ:** \*\*\* Object of class "vector" describes the parameters governing the *clp* equality constraints specified in `clpequspec`
- prel:** \*\*\* Object of class "vector" vector of starting values for the relations described in `prel-spec`
- prelspec:** \*\*\* Object of class "list" list of lists to specify the functional relationship between parameters, each of which has elements
- what1* character string describing the parameter type to relate, e.g., "kinpar"
  - what2* the parameter type on which the relation is based; usually the same as *what1*
  - ind1* index into *what1*
  - ind2* index into *what2*
  - rel* character string, optional argument to specify functional relation type, by default linear

e.g., `prelspec = list(list(what1 = "kinpar", what2 = "kinpar", ind1 = 1, ind2 = 5))` relates the 1st element of `kinpar` to the 5th element of `kinpar`. The starting values parameterizing the relationship are given in the `prel` vector

- positivepar:** `***` Object of class "vector" containing character strings of those parameter vectors to constrain to positivity, e.g., `positivepar=c("kinpar")`
- weight:** Object of class "logical" TRUE when the specification in `weightpar` is to be applied and FALSE otherwise
- psi.df:** Object of class "matrix" dataset from 1 experiment
- psi.weight:** Object of class "matrix" weighted dataset from 1 experiment
- x:** Object of class "vector" time or other independent variable.
- nt:** Object of class "integer" length `x`
- x2:** Object of class "vector" vector of points in 2nd independent dimension, such as wavelengths of wavenumbers
- n1:** Object of class "integer" length `x2`
- C2:** Object of class "matrix" concentration matrix for simulated data
- E2:** Object of class "matrix" matrix of spectra for simulated data
- sigma:** Object of class "numeric" noise level in simulated data
- parnames:** Object of class "vector" vector of parameter names, used internally
- finished:** Object of class "logical" describes whether optimization is complete
- simdata:** Object of class "logical" logical that is TRUE if the data is simulated, FALSE otherwise; will determine whether values in `C2` and `E2` are plotted with results
- weightM:** Object of class "matrix" weights
- weightsmooth:** Object of class "list" type of smoothing to apply with weighting; not currently used
- makeps:** Object of class "character" specifies the prefix of files written to postscript
- lclp0:** Object of class "logical" TRUE if specification in `clp0` is to be applied and FALSE otherwise
- lclpequ:** Object of class "logical" TRUE if specification in `clpequspec` is to be applied and FALSE otherwise
- title:** Object of class "character" displayed on output plots
- mhist:** Object of class "list" list describing fitting history
- datCall:** Object of class "list" list of calls to functions
- drel** vector of starting parameters for dataset scaling relations
- dscalspec:** Object of class "list"
- drel:** Object of class "vector" vector of starting parameters for dataset scaling relations
- scalx:** Object of class "numeric" numeric by which to scale the `x` axis in plotting
- prel** vector of starting values for the relations described in `prelspec`
- fvecind:** Object of class "vector" vector containing indices of fixed parameters
- pvecind:** Object of class "vector" used internally to store indices of related parameters.

**groups:** Object of class "list" list containing lists of pairs c(x2 index, dataset index). the x2 values (which are solved for as conditionally linear parameters) are equated for all pairs in a list.

**iter:** Object of class "numeric" describing the number of iterations that is run; this is sometimes stored after fitting, but has not effect as an argument to `initModel`

**clpCon:** Object of class "list" used internally to enforce constraints on the clp

**ncomp:** Object of class "numeric" describing the number of components in a model

**clpdep:** Object of class "logical" describing whether a model is dependent on the index of x2

**inten:** Object of class "matrix" for use with FLIM data; represents the number of photons per pixel measured over the course of all times  $t$  represented by the dataset. See the help for the `readData` function for more information.

**datafile:** Object of class "character" containing the name of a datafile associated with the `psi.df`

**clpType:** Object of class "character" that is "nt" if the model has clp in the "x" dimension and "nl" otherwise (so that, e.g., if `mod_type = "kin"`, then `clpType = "nl"`).

### Author(s)

Katharine M. Mullen, Ivo H. M. van Stokkum

### See Also

[kin-class](#), [spec-class](#)

### Examples

```
# simulate data

C <- matrix(nrow = 51, ncol = 2)
k <- c(.5, 1)
t <- seq(0, 2, by = 2/50)
C[, 1] <- exp(- k[1] * t)
C[, 2] <- exp(- k[2] * t)
E <- matrix(nrow = 51, ncol = 2)
wavenum <- seq(18000, 28000, by=200)
location <- c(25000, 20000)
delta <- c(5000, 7000)
amp <- c(1, 2)
E[, 1] <- amp[1] * exp(- log(2) * (2 * (wavenum - location[1])/delta[1]))^2)
E[, 2] <- amp[2] * exp(- log(2) * (2 * (wavenum - location[2])/delta[2]))^2)
sigma <- .001
Psi_q <- C %**% t(E) + sigma * rnorm(nrow(C) * nrow(E))

# initialize an object of class dat
Psi_q_data <- dat(psi.df = Psi_q, x = t, nt = length(t),
x2 = wavenum, nl = length(wavenum))

# initialize an object of class dat via initModel
```

```
# this dat object is also a kin object
kinetic_model <- initModel(mod_type = "kin", seqmod = FALSE,
kinpar = c(.1, 2))
```

---

denS4	<i>Time-resolved absorption data</i>
-------	--------------------------------------

---

### Description

Time-resolved absorption data measured at two different laser intensities

### Usage

```
data("denS4")
data("denS5")
```

### Format

denS4 is an object of class `dat` representing absorption data. denS5 is an object of class `dat` representing absorption data measured at half the laser intensity as compared to the intensity used to measure denS4.

### References

This data was described in Mullen KM, van Stokkum IHM (2007). TIMP: An R Package for Modeling Multi-way Spectroscopic Measurements. *Journal of Statistical Software*, **18**(3), <http://www.jstatsoft.org/v18/i03/>.

### Examples

```
data("denS4")
image.plot(denS4@x, denS4@x2, denS4@psi.df)
```

---

divergeZimage	<i>Plots a matrix with a diverging palette, with the center value of the palette possible to set</i>
---------------	--

---

### Description

An image plot of a matrix is a way of visualizing data; when the data represents a quantity like transient absorption, where negative values represent a different phenomena than positive values, it can be useful to set values at zero in the image plot to grey, whereas positive values are assigned to red, and negative values are assigned to blue. Alternately, when comparing image plots of several matrices, it may be useful to set the value assigned to grey uniformly, with values above this threshold assigned to red, and below this threshold assigned to blue.

**Usage**

```
divergeZimage(ob, out=FALSE, file="divergeZimage.pdf",
             lin = 1, title = "", center = 0,
             x2 = vector(), x= vector(),
             plainmat = FALSE, ylab="wavelength (nm)",
             xlab = "time (ns)")
```

**Arguments**

<code>ob</code>	either an object of class <code>dat</code> or a numeric matrix; if a numeric matrix is given then set <code>plainmat=TRUE</code> and specify labels for the columns of matrix in <code>x2</code> and for the rows of the matrix in <code>x</code>
<code>out</code>	a logical indicating whether to write to the screen in the case that this is possible or to a file; if <code>TRUE</code> , writes to a pdf file
<code>file</code>	a character vector giving a filename to write to in the case that <code>out=TRUE</code>
<code>lin</code>	range of <code>x</code> to plot linearly; values not between <code>-lin</code> and <code>lin</code> are plotted on a log scale
<code>title</code>	character vector giving a title for the plot
<code>center</code>	point assigned to grey in the diverging palette.
<code>x2</code>	vector of labels for the columns of the matrix; used only if <code>plainmat=TRUE</code>
<code>x</code>	vector of labels for the rows of the matrix; used only if <code>plainmat=TRUE</code>
<code>plainmat</code>	logical indicating whether <code>ob</code> is a matrix, as opposed to an object of class <code>dat</code>
<code>ylab</code>	character vector giving a label to put on the y-axis
<code>xlab</code>	character vector giving a label to put on the x-axis

**Author(s)**

Katharine M. Mullen, Ivo H. M. van Stokkum

**See Also**

[dat](#)

**Examples**

```
exd <- dat(psi.df=matrix(rnorm(1000), ncol=100, nrow=100),
          x=1:100, x2=1:100, nl=as.integer(100), nt=as.integer(100))

## by default linear range until 1 is used, logarithmic thereafter
divergeZimage(exd)

## can change this as desired
divergeZimage(exd, lin=10, title="plot linearly to 10")
```

---

donorAcceptorTagged  
*Fluorescent lifetime imaging microscopy (FLIM) data*

---

**Description**

Donor-and-acceptor tagged fluorescent lifetime imaging microscopy (FLIM) data.

**Usage**

```
data("donorAcceptorTagged")
```

**Format**

cy005c and cy006 are objects of class `dat` representing donor-and-acceptor tagged data.

**Details**

See [FLIMplots](#) for examples using this data.

**References**

This data was described in

Mullen KM, van Stokkum IHM (2008). The variable projection algorithm in time-resolved spectroscopy, microscopy and mass-spectroscopy applications, *Numerical Algorithms*, **in press**, <http://dx.doi.org/10.1007/s11075-008-9235-2>.

---

donorTagged  
*Fluorescent lifetime imaging microscopy (FLIM) data*

---

**Description**

Donor-only tagged fluorescent lifetime imaging microscopy (FLIM) data.

**Usage**

```
data("donorTagged")
```

**Format**

c001 and c003 are objects of class `dat` representing donor-only data.

**Details**

See [FLIMplots](#) for examples using this data.

## References

This data was described in

Mullen KM, van Stokkum IHM (2008). The variable projection algorithm in time-resolved spectroscopy, microscopy and mass-spectroscopy applications, *Numerical Algorithms*, **in press**, <http://dx.doi.org/10.1007/s11075-008-9235-2>.

---

efit2file

*convert 'tim' FORTRAN efit files to plain matrices in ASCII files*

---

## Description

'tim' efit files sometimes represent spectra associated with multiple datasets; for each matrix of spectra stored in such a file, this function writes a plain text file.

## Usage

```
efit2file(filename, skip = 2, numcol, nrows=vector())
```

## Arguments

filename	This is the path to the file to read in, as a quoted string.
skip	number of lines at the top of the file before the data begins
numcol	number of columns the data
nrows	a vector saying how many rows are in each of the matrices of spectra in the file; for instance <code>nrows = c(256, 256, 256)</code> would indicate that the file stores spectra associated with 3 datasets, each of which contains 256 wavelengths. If <code>nrows</code> is not given, then a single file containing all data is written.

## Author(s)

Katharine M. Mullen, Ivo H. M. van Stokkum

## See Also

[readData](#)

---

examineFit	<i>Examines the results of a call to fitModel</i>
------------	---

---

**Description**

Examine the results of a call to `fitModel` by a call to plotting functions; call this function with argument an object returned from `fitModel`. Possibly also supply a new specification of plots to be generated.

**Usage**

```
examineFit(resultfitModel, opt=vector())
```

**Arguments**

<code>resultfitModel</code>	list returned by a call to <code>fitModel</code>
<code>opt</code>	possibly an object of class <code>opt</code> giving options for plotting; if <code>opt</code> has length zero (the default) then the plotting options given in the <code>opt</code> list of <code>resultFitModel</code> are applied

**Details**

The `fitModel` function returns a list of results, and initiates plotting functions. Given the `resultfitModel` list `fitModel` returns, `examineFit` initiates the plotting functions, and thus may be used to examine results.

**Author(s)**

Katharine M. Mullen, Ivo H. M. van Stokkum

**See Also**

`fitModel`, `opt`

---

<code>fit-class</code>	<i>Class "fit" to store the results of model fitting associated with all datasets analyzed.</i>
------------------------	---

---

**Description**

Class to store results of model fitting associated with all datasets in a single call to the `fitModel` function. An object of class `fit` is stored in the slot `fit` of objects of class `multimodel`.

### Objects from the Class

Objects can be created by calls of the form `new("fit", ...)`.

### Slots

**resultlist:** Object of class "list" that contains an object of class `res` for each dataset modeled, in the order that they were specified.

**nlsres:** Object of class "list" containing named elements

`onls` output of the call to `nls` used in model optimization.

`sumonls` result of call `summary(onls)`

### Author(s)

Katharine M. Mullen, Ivo H. M. van Stokkum

### See Also

[res-class](#), [multimodel-class](#)

---

fitModel

*Performs optimization of (possibly multidataset) models.*

---

### Description

Performs optimization of (possibly multidataset) models and outputs plots and files representing the fit of the model to the data.

### Usage

```
fitModel(data, modspec=list(), datasetind = vector(),
  modeldiffs = list(), opt = opt() )
```

### Arguments

<code>data</code>	list of objects of class <code>dat</code> containing the data to be modeled
<code>modspec</code>	list whose elements are models of class <code>dat</code> describing the models as results from a call to the function <code>initModel</code>
<code>datasetind</code>	vector that has the same length as <code>data</code> ; for each dataset in <code>data</code> specify the model it should have as an index into <code>modspec</code> ; default mapping is that all datasets use the first model given in <code>modspec</code>
<code>modeldiffs</code>	list whose elements specify any dataset-specific model differences.

- linkclp** list of vectors containing the indices of datasets. If the two dataset indices are in the same vector, their conditionally linear parameters will be equated if they represent the same condition (e.g., a wavelength) within `thresh`. For example, `linkclp = list(1:10, 11:15)` will let datasets 1-10 and 11-15 have the same `clp`. Note that if `linkclp` is not given, it will default to `list(1:length(data))`, so that the `clp` from all datasets are equated when they represent conditions within `thresh` of each other. Consider the situation where the `clp` from many different datasets are equated. **Then it is important to note that the specification of constraints applicable to the `clp` will also be equated, and will be read from the model assigned to the first dataset in the group.**
- dscal** list of lists specifying linear scaling relations between datasets; each list has elements `to`, `from`, `value`. The index of the dataset to be scaled is given in `to`; the index of the dataset on which the scaling is to be based is given in `from`. The starting value parameterizing the relationship is given as `value`. For example, `dscal = list(list(to=2, from=1, value=.457))`.
- thresh** numeric describing the tolerance with which `clp` from different datasets are to be considered as equal. For instance, for two datasets containing data at 750 and 751 nm, respectively, `thresh=1.5` will equate the `clp` at 750 and 751 between datasets. Specify a negative value of `thresh` to estimate `clp` per-dataset. See Section 2.2 of the paper in the references for the model equations.
- free** list of lists specifying individual parameters to free for a given dataset. each sublist has named elements  
**what** character string naming parameter type, e.g., "kinpar"  
**ind** vector of indices into parameter vector or list, e.g., `c(2, 3)` or 4  
**dataset** dataset index in which parameter is to be freed  
**start** starting value for freed parameter  
 For example, `free = list(list(what = "irfpar", ind = 1, dataset = 2, start=-.1932), list(what = "kinpar", ind = 5, dataset = 2, start=.0004), list(what = "kinpar", ind = 4, dataset = 2, start= .0159))`.
- remove** list of lists specifying individual parameters to remove from parameter groups for a given dataset. each sublist has named elements  
**what** character string naming parameter type, e.g., "kinpar"  
**dataset** dataset index in which parameter group is to be removed  
**ind** vector of indices into parameter vector or list, e.g., `c(2, 3)` or 4 where parameter should be removed
- add** list of lists specifying individual parameters to add to parameter groups for a given dataset. each sublist has named elements  
**what** character string naming parameter type, e.g., "kinpar"  
**dataset** dataset index in which parameter group is to change  
**start** starting value for added parameter  
**ind** vector of indices into parameter vector or list, e.g., `c(2, 3)` or 4 where parameter should be added.

change list of lists specifying entire parameter groups to change for a given dataset. each sublist has named elements

what character string naming parameter type, e.g., "kinpar"

dataset dataset index in which parameter group is to change

spec new specification that in `initModel` would follow "what", e.g., for `c(.1, .3)` if `what="kinpar"`

rel list of lists specifying parameters to relate between datasets each sublist has named elements

what1 character string naming parameter type to be determined in relation to some other parameter type, e.g., "kinpar"

what2 character string naming parameter type on which another parameter type is to depend, e.g., "kinpar"

ind1 vector of indices into parameter vector or list, e.g., `c(2, 3)` or 4 of the dependent parameter.

ind2 vector or numeric of indices into parameter vector or list, e.g., `c(2, 3)` or 4 of the parameter on which another parameter will depend

dataset1 dataset index of the dependent parameter

dataset2 dataset index of the parameter on which another parameter will depend

rel optional character string describing functional relationship between parameters; defaults to "lin" for linear relationship

start starting value or vector of values parameterizing relationship between parameters

weightlist List of new weights for the datasets returned by the function `outlierCorrs` (as the element `weightList` of the list that is the return value of this function).

opt Object of class `kinopt` or `specopt` specifying fitting and plotting options.

### Details

This function applies the `nls` function internally to optimize nonlinear parameters and to solve for conditionally linear parameters (clp) via the partitioned variable projection algorithm.

### Value

A list is returned containing the following elements:

`currTheta` is a list of objects of class `theta` whose elements contain the parameter estimates associated with each dataset modeled.

`currModel` is an object of class `multimodel` containing the results of fitting as well as the model specification

`toPlotter` is a list containing all arguments used by the plotting function; it is used to regenerate plots and other output by the `examineFit` function

### Author(s)

Katharine M. Mullen, Ivo H. M. van Stokkum

## References

Mullen KM, van Stokkum IHM (2007). “TIMP: an R package for modeling multi-way spectroscopic measurements.” *Journal of Statistical Software*, 18(3). <http://www.jstatsoft.org/v18/i03/>.

## See Also

[readData](#), [initModel](#), [examineFit](#)

## Examples

```
## 2 simulated concentration profiles in time
C <- matrix(nrow = 51, ncol = 2)
k <- c(.5, 1)
t <- seq(0, 2, by = 2/50)
C[, 1] <- exp(- k[1] * t)
C[, 2] <- exp(- k[2] * t)

## 2 simulated spectra in wavelength
E <- matrix(nrow = 51, ncol = 2)
wavenum <- seq(18000, 28000, by=200)
location <- c(25000, 20000)
delta <- c(5000, 7000)
amp <- c(1, 2)
E[, 1] <- amp[1] * exp(- log(2) * (2 * (wavenum - location[1])/delta[1])^2)
E[, 2] <- amp[2] * exp(- log(2) * (2 * (wavenum - location[2])/delta[2])^2)

## simulated time-resolved spectra
sigma <- .001
Psi_q <- C %*% t(E) + sigma * rnorm(nrow(C) * nrow(E))

## as an object of class dat
Psi_q_data <- dat(psi.df = Psi_q, x = t, nt = length(t), x2 = wavenum, nl =
length(wavenum))

## model for the data in the time-domain
kinetic_model <- initModel(mod_type = "kin", seqmod = FALSE,
kinpar = c(.1, 2))

## fit the model
kinetic_fit <- fitModel(data = list(Psi_q_data),
modspec = list(kinetic_model), opt = kinopt(iter=4, plot=FALSE))
```

## Description

Functions to plot FLIM results.

**Usage**

```

plotHistAmp(multimodel, t, i=1)
plotHistNormComp(multimodel, t, i=1)
plotIntenImage(multimodel, t, i=1, tit=c("Intensity Image"))
plotSelIntenImage(multimodel, t, i=1, tit=c("Region of Interest"),
  cex=1)
plotTau(multimodel, t, i=1, tit=" < tau > ", plotoptions=kinopt(),
  lifetimes=TRUE)
plotNormComp(multimodel, t, i=1)

```

**Arguments**

<code>multimodel</code>	the <code>currModel</code> element of the list returned by <code>fitModel</code>
<code>t</code>	the <code>currTheta</code> element of the list returned by <code>fitModel</code>
<code>i</code>	dataset index to make plot for
<code>tit</code>	Character vector giving the title
<code>plotoptions</code>	object of class <code>kinopt</code> giving the plotting options
<code>cex</code>	A numerical value giving the amount by which plotting text and symbols should be magnified relative to the default
<code>lifetimes</code>	A logical value indicating whether the averages per-pixel should be for lifetimes or their inverse, decay rates.

**Author(s)**

Katharine M. Mullen, Sergey Laptinok, Ivo H. M. van Stokkum

**See Also**

[fitModel](#)

**Examples**

```

#####
## READ IN DATA, PREPROCESS DATA
#####

## data representing only donor tagged

data("donorTagged")

D1 <- preProcess(c001, sel_time=c(25,230))
D2 <- preProcess(c003, sel_time=c(25,230))

## data representing donor-acceptor tagged

data("donorAcceptorTagged")

DA1 <- preProcess(cy005c, sel_time=c(25,230))

```

```

DA2 <- preProcess(cy006, sel_time=c(25,230))

#####
## READ IN MEASURED IRF, PREPROCESS IRF
#####

data("mea_IRF")
mea_IRF <- baseIRF(mea_IRF, 100, 150)[25:230]

#####
## SPECIFY INITIAL MODEL
#####

modelC <- initModel(mod_type = "kin",
  ## starting values for decays
  kinpar=c(1.42, 0.36),
  ## numerical convolution algorithm to use
  convalg = 2,
  ## measured IRF
  measured_irf = mea_IRF,
  ## shift of the irf is fixed
  parmu = list(0), fixed = list(parmu=1),
  ## one component represents a pulse-following with the IRF shape
  cohspec = list(type = "irf"),
  ## parallel kinetics
  seqmod=FALSE,
  ## decay parameters are non-negative
  positivepar=c("kinpar"),
  title="Global CFP bi-exp model with pulse-follower")

#####
## FIT MODEL FOR DONOR ONLY DATA
#####

fitD <- fitModel(list(D1,D2),
  list(modelC),
  ## estimate the linear coefficients per-dataset
  modeldiffs = list(linkclp=list(1,2)),
  opt=kinopt(iter=20, linrange = 10,
    addfilename = TRUE,
    output = "pdf",
    makeps = "globalD",
    notraces = TRUE,
    selectedtraces = seq(1, length(c001@x2), by=11),
    summaryplotcol = 4, summaryplotrow = 4,
    ylimspec = c(1, 2.5),
    xlab = "time (ns)", ylab = "pixel number",
    FLIM=TRUE))

#####
## FIT MODEL FOR DONOR-ACCEPTOR DATA
#####

```

```

fitDA <- fitModel(list(DA1,DA2),
                  list(modelC),
                  ## estimate the linear coefficients per-dataset
                  modeldiffs = list(linkclp=list(1,2)),
                  opt=kinopt(iter=20, linrange = 10,
                             addfilename = TRUE,
                             output = "pdf",
                             makeps = "globalDA",
                             notraces = TRUE,
                             selectedtraces = seq(1, length(c001@x2), by=11),
                             summaryplotcol = 4, summaryplotrow = 4,
                             ylimspec = c(1, 2.5),
                             xlab = "time (ns)", ylab = "pixel number",
                             FLIM=TRUE))

#####
## COMPARE THE DECAY RATES
#####

parEst(fitD)

parEst(fitDA)

#####
## ADDITIONAL FIGURES
#####

par(mfrow=c(2,2), mar=c(1,3,1,12))

par(cex=1.5)
plotIntenImage(fitD$currModel, fitD$currTheta, 1, tit="")

par(cex=1.5)
plotIntenImage(fitDA$currModel, fitD$currTheta, 1, tit="")

par(cex=1.5)
plotIntenImage(fitD$currModel, fitD$currTheta, 2, tit="")

par(cex=1.5)
plotIntenImage(fitDA$currModel, fitD$currTheta, 2, tit="")

#####

plo <- kinopt(ylimspec = c(.25,1.1), imagepal=grey(seq(1,0,length=100)))

par(mfrow=c(2,2), mar=c(1,3,1,12))

par(cex=1.5)
plotTau(fitD$currModel, fitD$currTheta, 1, tit="",plotoptions=plo,
        lifetimes=FALSE)

par(cex=1.5)
plotTau(fitDA$currModel, fitD$currTheta, 1, tit="",plotoptions=plo,

```

```
        lifetimes=FALSE)

par(cex=1.5)
plotTau(fitD$currModel, fitD$currTheta, 2, tit="", plotoptions=plo,
        lifetimes=FALSE)

par(cex=1.5)
plotTau(fitDA$currModel, fitD$currTheta, 2, tit="", plotoptions=plo,
        lifetimes=FALSE)
```

---

getClpindepX-methods

*Generic function getClpindepX in Package ‘TIMP’*

---

### Description

Gets the matrix associated with nonlinear parameter estimates for the case that this matrix is not re-calculated per conditionally linear parameter.

### Usage

```
getClpindepX(model, multimodel, theta, returnX, rawtheta, dind)
```

### Arguments

model	Object of class <code>dat</code> ; function switches on this argument.
multimodel	Object of class <code>multimodel</code> used in standard error determination
theta	Vector of nonlinear parameter estimates.
returnX	logical indicating whether to return a vectorized version of the X matrix
rawtheta	vector of nonlinear parameters; used in standard error determination
dind	numeric indicating the dataset index; used in standard error determination

### Author(s)

Katharine M. Mullen, Ivo H. M. van Stokkum

### See Also

[dat-class](#)

---

<code>getResid</code>	<i>For data correction, fits a model (but ignores plotting commands) in order to obtain the SVD of the residuals, which then can be used in data-correction.</i>
-----------------------	--

---

### Description

For data correction, fits a model exactly as does `fitModel` (but ignores plotting commands) in order to obtain the SVD of the residuals. These residuals can then be subtracted away from the original data to some extent with the `preProcess` function.

### Usage

```
getResid(data, modspec=list(), datasetind = vector(),
         modeldiffs = list(), opt = opt() )
```

### Arguments

<code>data</code>	As in the <code>fitModel</code> function
<code>modspec</code>	As in the <code>fitModel</code> function
<code>datasetind</code>	As in the <code>fitModel</code> function
<code>modeldiffs</code>	As in the <code>fitModel</code> function
<code>opt</code>	As in the <code>fitModel</code> function

### Value

list containing the first five left and right singular vectors of the residuals, as well as the first five singular values. A weight matrix (if used) is also included in this list.

### See Also

[fitModel](#), [preProcess](#)

---

<code>getResults</code>	<i>Functions to print and return parts of the object returned by the fitting routines.</i>
-------------------------	--

---

### Description

Functions to print and return parts of the object returned by `fitModel`. `onls` returns the output of the `nls` function. `sumonls` returns the result of calling `summary` on `onls` function. `parEst` returns a summary of model parameter estimates. The remaining functions return lists representing various aspects of the results returned by the function `fitModel`.

**Usage**

```

onls(result)
sumnls(result)
parEst(result, param = "", dataset = NA, verbose = TRUE, file="",
  stderr=TRUE)
getXList(result, group = vector(), file="")
getCLPList(result, getclperr = FALSE, file="")
getX(result, group = vector(), dataset=1, file="")
getCLP(result, getclperr = FALSE, dataset=1, file="")
getData(result, dataset = 1, weighted = FALSE)
getResiduals(result, dataset = 1)
getSVDResiduals(result, numsing = 2, dataset = 1)
getTraces(result, dataset = 1, file="")
getdim1(result, dataset = 1)
getdim2(result, dataset = 1)

```

**Arguments**

result	return value of fitModel
param	character vector of the particular parameters to return; if param="" then all parameters are given.
dataset	index of the dataset from which to return results; by default dataset=NA in which case results from all datasets are returned
verbose	logical that defaults to TRUE that determines whether parEst just returns a list invisibly or prints as well.
getclperr	logical that defaults to FALSE that determines whether a list containing the standard error estimates associated with the conditionally linear parameters, as opposed to the conditionally linear parameters themselves
numsing	integer that defaults to 2; determines the number of singular vectors to return
weighted	logical indicating whether to return weighted or unweighted data
file	character vector; if not "" writes the results to a file with name file.
group	The value at which to determine the X matrix (maybe a wavelenth index, for example)
stderr	Whether to return standard error estimates on parameters, if they were calculated in fitting.

**Value**

sumnls returns an object of class "summary.nls".

onls returns an object of class "nls".

parEst returns an object of class "list" representing the parameter estimates and the standard errors if stderr=TRUE and they have been calculated.

getXList returns a "list" of length equal to the number of datasets modeled, where each element represents the matrix determined by the nonlinear parameters (under a kinetic model, the concentrations).

getCLPList returns a "list" of length equal to the number of datasets modeled, where each element represents the matrix determined as conditionally linear parameters (under a kinetic model, the spectra).

getX returns a numeric "matrix" that represents the matrix determined by the nonlinear parameters (under a kinetic model, the concentrations).

getCLPList returns a numeric "matrix" that represents the matrix determined as conditionally linear parameters (under a kinetic model, the spectra).

getSVDDData returns a "list" of length 3 with named elements values, left and right, where values contains the singular values, left contains numsing left singular vectors, and right contains numsing right singular vectors, all of the unweighted data. The number of singular vectors returned is determined by numsing.

getData returns the dataset specified by the argument dataset (weighted data in the case that weighted=TRUE) as a "matrix"

getResiduals returns a "matrix" of residuals for the dataset with index given by the argument dataset; the matrix returned has the dimension of the dataset itself.

getSVDResiduals returns a "list" of length 3 with named elements values, left and right, where values contains the singular values, left contains numsing left singular vectors, and right contains numsing right singular vectors, all of the residuals. The number of singular vectors returned is determined by numsing.

getTraces returns a "matrix" of model estimates for the dataset with index given by the argument dataset; the matrix returned has the dimension of the dataset itself.

getdim1 returns a "vector" of x values in the dataset (times for kinetic models).

getdim2 returns a "vector" of x2 values (wavelengths for kinetic models).

### Author(s)

Katharine M. Mullen, Ivo H. M. van Stokkum

### See Also

[fitModel](#)

### Examples

```
## Example showing the addition of non-negativity constraints to
## conditionally linear parameters (here the spectra associated with
## a kinetic model)

## For the 1st simulated dataset, the constraints offer a modest improvement
## in the estimated spectra, whereas for the 2nd simulated dataset, they
## prevent a catastrophe in which the estimated components are hugely
## compensating.

#####
## load TIMP
#####
```

```

require(TIMP)

#####
## set random seed for reproducibility of noise
#####

set.seed(80)

#####
## SIMULATE DATA, noise realization 1
#####

dt4 <- simndecay_gen(kinpar = c(.4, .8, 2), seqmod = FALSE, tmax
  = 2, deltat = .04, specpar = list(c(25000, 3000, .01), c(22000,
  3000, .01), c(18000, 3000, .01)), lmin=350, lmax=550, deltal = 2,
  sigma=.01)

#####
## SPECIFY INITIAL MODEL
#####

mod1 <- initModel(mod_type = "kin", kinpar = c(.4, .8, 2),
  seqmod=FALSE)

#####
## FIT INITIAL MODEL
#####

sT <- fitModel(list(dt4), list(mod1), opt=kinopt(iter=50, plot=FALSE))

#####
## EXTRACT ESTIMATED SPECTRA
## these spectra have some negative values
#####

sTcp <- getCLP(sT)

## plot the estimated spectra with the values used in
## simulation (before adding noise) for comparison
matplot(dt4@x2, sTcp, xlab = "wavelength (nm)", col = 2:4, type="l",
  ylab="", lty=1, main =
  paste("Estimated spectra, adding no constraints\n"))
matplot(dt4@x2, dt4@E2, add=TRUE, type="l", col=1, lty=2)
abline(0,0)

#####
## FIT INITIAL MODEL
## adding constraints to non-negativity of the
## spectra via the opt option nnls=TRUE
#####

sV <- fitModel(list(dt4), list(mod1), opt=kinopt(iter=50, nnls=TRUE,
  plot=FALSE))

```

```
#####
## EXTRACT ESTIMATED SPECTRA
## these spectra have no negative values
#####

sVcp <- getCLP(sV)

## plot the estimated spectra with the values used in
## simulation (before adding noise) for comparison
matplot(dt4@x2, sVcp, xlab = "wavelength (nm)", col = 2:4, type="l",
        ylab="", lty=1,
        main = paste("Estimated spectra, with non-negativity constraints\n"))
matplot(dt4@x2, dt4@E2, add=TRUE, type="l", col=1, lty=2)
abline(0,0)

#####
## SIMULATE DATA, noise realization 2
#####

dt4_2 <- simndecay_gen(kinpar = c(.4, .8, 2), seqmod = FALSE, tmax
                    = 2, deltat = .04, specpar = list(c(25000, 3000, .01), c(22000,
                    3000, .01), c(18000, 3000, .01)), lmin=350, lmax=550, deltal = 2,
                    sigma=.01)

#####
## SPECIFY INITIAL MODEL
#####

mod1 <- initModel(mod_type = "kin", kinpar = c(.4, .8, 2),
                 seqmod=FALSE)

#####
## FIT INITIAL MODEL
#####

sT <- fitModel(list(dt4_2), list(mod1), opt=kinopt(iter=50, plot=FALSE))

#####
## EXTRACT ESTIMATED SPECTRA
## these spectra have some negative values
#####

sTcp <- getCLP(sT)

## plot the estimated spectra with the values used in
## simulation (before adding noise) for comparison
matplot(dt4@x2, sTcp, xlab = "wavelength (nm)", col = 2:4, type="l",
        ylab="", lty=1, main =
        paste("Estimated spectra, adding no constraints\n"))
matplot(dt4@x2, dt4@E2, add=TRUE, type="l", col=1, lty=2)
abline(0,0)
```

```
#####
## FIT INITIAL MODEL
## adding constraints to non-negativity of the
## spectra via the opt option nnls=TRUE
#####

sV <- fitModel(list(dt4_2), list(mod1), opt=kinopt(iter=50, nnls=TRUE,
                                                plot=FALSE))

#####
## EXTRACT ESTIMATED SPECTRA
## these spectra have no negative values
#####

sVcp <- getCLP(sV)

## plot the estimated spectra with the values used in
## simulation (before adding noise) for comparison
matplot(dt4@x2, sVcp, xlab = "wavelength (nm)", col = 2:4, type="l",
        ylab="", lty=1,
        main = paste("Estimated spectra, with non-negativity constraints\n"))
matplot(dt4@x2, dt4@E2, add=TRUE, type="l", col=1, lty=2)
abline(0,0)
```

---

initModel

*Defines the model to be used in analysis.*


---

## Description

Allows definition of a model of class "dat" to be used in analysis. The arguments specify the model.

## Usage

```
initModel(...)
```

## Arguments

... specify the model class via the character string e.g., [kin-class](#) or [spec](#) and any of the slots associated with that model type (which is a subclass of class `dat`, so that all slots in `dat` may also be specified), e.g., `mod_type = "kin"` will initialize a model with class `kin`, for a kinetic model.

## Details

For examples, see the help files for [dat-class](#) and [fitModel](#)

## Value

an object of class `dat` with the sub-class given by the value of the `mod_type` input.

**Author(s)**

Katharine M. Mullen, Ivo H. M. van Stokkum

**See Also**

[dat-class](#), [kin-class](#), [spec-class](#), [fitModel](#)

**Examples**

```
#####
## READ IN PSI 1
#####

data(denS4)

#####
## PREPROCESS PSI 1
#####

denS4<-preProcess(data = denS4, scalx2 = c(3.78, 643.5))

#####
## READ IN PSI 2
#####

data(denS5)

#####
## PREPROCESS PSI 2
#####

denS5<-preProcess(data = denS5, scalx2 = c(3.78, 643.5))

#####
## DEFINE INITIAL MODEL
#####

modell1<- initModel(mod_type = "kin",
kinpar= c(7.9, 1.08, 0.129, .0225, .00156) ,
irfpar=c( -.1018, 0.0434),
disptau=FALSE, dispmu=TRUE, parmu = list(c(.230)),
lambdac = 650,
seqmod=TRUE,
positivepar=c("kinpar"),
title="S4",
cohspec = list( type = "irf"))

#####
## FIT INITIAL MODEL
#####
```

```

denRes1 <- fitModel(data=list(denS4, denS5), list(model1),
opt=kinopt(iter=5, divdrel = TRUE, linrange = .2,
makeeps = "den1", selectedtraces = c(1,5,10), plotkinspec =TRUE,
output="pdf", xlab = "time (ps)", ylab = "wavelength"))

#####
## REFINE INITIAL MODEL, RE-FIT
## adding some per-dataset parameters
#####

denRes2 <- fitModel(data = list(denS4, denS5), modspec = list(model1),
modeldiffs = list(dscal = list(list(to=2,from=1,value=.457)),
free = list(
list(what = "irfpar", ind = 1, dataset = 2, start=-.1932),
list(what = "kinpar", ind = 5, dataset = 2, start=.0004),
list(what = "kinpar", ind = 4, dataset = 2, start= .0159)
)),
opt=kinopt(iter=5, divdrel = TRUE, linrange = .2,
xlab = "time (ps)", ylab = "wavelength", output="pdf",
makeeps = "den2", selectedtraces = c(1,5,10)))

#####
## REFINE MODEL FURTHER AS NEW MODEL OBJECT
#####

model2 <- initModel(mod_type = "kin",
kinpar= c(7.9, 1.08, 0.129, .0225, .00156),
irfpar=c( -.1018, 0.0434),
parmu = list(c(.230)),
lambdac = 650,
positivepar=c("kinpar", "coh"),
cohspec = list( type = "seq", start = c(8000, 1800)))

#####
## FIT NEW MODEL OBJECT
#####

denRes3 <- fitModel(data = list(denS4, denS5), list(model2),
modeldiffs = list(dscal = list(list(to=2,from=1,value=.457)),
free = list(
list(what = "irfpar", ind = 1, dataset = 2, start=-.1932),
list(what = "kinpar", ind = 5, dataset = 2, start=.0004),
list(what = "kinpar", ind = 4, dataset = 2, start= .0159)
)),
opt=kinopt(iter=5, divdrel = TRUE, linrange = .2,
makeeps = "den3", selectedtraces = c(1,5,10), plotkinspec =TRUE,
stderrclp = TRUE, kinspecerr=TRUE, output="pdf",
xlab = "time (ps)", ylab = "wavelength",
breakdown = list(plot=c(643.50, 658.62, 677.5)))

```

---

 kin-class

 Class "kin" for kinetic model storage.
 

---

## Description

`kin` is the class for kinetic models; an object of class "kin" is initialized if `mod_type = "kin"` is an argument of `initModel`. All objects of class `kin` are sub-classes of class `dat`; see documentation for `dat` for a description of these slots.

## Details

See [dat-class](#) for an example of the initialization of a `kin` object via the `initModel` function.

## Objects from the Class

Objects can be created by calls of the form `new("kin", ...)` or `kin(...)`. Slots whose description are marked with `***` may be specified in the `...` argument of the `initModel` function.

## Slots

**kinpar** `***` vector of rate constants to be used as starting values for the exponential decay of components; the length of this vector determines the number of components of the kinetic model.

**specpar**: `***` Object of class "list" parameters for spectral constraints

**seqmod**: `***` Object of class "logical" that is TRUE if a sequential model is to be applied and FALSE otherwise

**irf**: Object of class "logical" that is TRUE is an IRF is modeled and FALSE otherwise

**mirf**: Object of class "logical" that is TRUE if a measured IRF is modeled and FALSE otherwise

**measured\_irf**: `***` Object of class "vector" containing a measured IRF

**convalg**: `***` Object of class "numeric" 1-3 determining the numerical convolution algorithm used in the case of modeling a measured IRF; if 3 then supply a reference lifetime in the slot `reftau`.

**reftau**: `***` Object of class "numeric" containing a reference lifetime to be used when `convalg=3`

**irffun**: `***` Object of class "character" describing the function to use to describe the IRF, by default "gaus"

**irfpar**: `***` Object of class "vector" of IRF parameters; for the common Gaussian IRF this vector is ordered `c(location, width)`

**dispmu**: Object of class "logical" that is TRUE if dispersion of the parameter for IRF location is to be modeled and FALSE otherwise

**dispmufun**: `***`Object of class "character" describing the functional form of the dispersion of the IRF location parameter; if equal to "discrete" then the IRF location is shifted per element of `x2` and `parmu` should have the same length as `x2`. defaults to a polynomial description

**parmu:** \*\*\* Object of class "list" of starting values for the dispersion model for the IRF location

**disptau:** Object of class "logical" that is TRUE if dispersion of the parameter for IRF width is to be modeled and FALSE otherwise

**disptaufun:** \*\*\* Object of class "character" describing the functional form of the dispersion of the IRF width parameter; if equal to "discrete" then the IRF width is parameterized per element of `x2` and `partau` should have the same length as `x2`. defaults to a polynomial description

**partau:** \*\*\* Object of class "vector" of starting values for the dispersion model for the IRF FWHM

**fullk:** Object of class "logical" that is TRUE if the data are to be modeled using a compartmental model defined in a K matrix and FALSE otherwise

**kmat:** \*\*\* Object of class "array" containing the K matrix descriptive of a compartmental model

**jvec:** \*\*\* Object of class "vector" containing the J vector descriptive of the inputs to a compartmental model

**ncolc:** Object of class "vector" describing the number of columns of the C matrix for each `clp` in `x2`

**kinscal:** \*\*\* Object of class "vector" of starting values for branching parameters in a compartmental model

**kmatfit:** Object of class "array" of fitted values for a compartmental model

**cohspec:** \*\*\* Object of class "list" describing the model for coherent artifact/scatter component(s) containing the element `type` and optionally the element `numdatasets`. The element `type` can be set as follows:

- "**irf**": if `type="irf"`, the coherent artifact/scatter has the time profile of the IRF.
- "**freeirfdisp**": if `type="freeirfdisp"`, the coherent artifact/scatter has a Gaussian time profile whose location and width are parameterized in the vector `coh`.
- "**irfmulti**": if `type="irfmulti"` the time profile of the IRF is used for the coherent artifact/scatter model, but the IRF parameters are taken per dataset (for the multidataset case), and the integer argument `numdatasets` must be equal to the number of datasets modeled.
- "**seq**": if `type="seq"` a sequential exponential decay model is applied, whose starting value are contained in an additional list element `start`. This often models oscillating behavior well, where the number of oscillations is the number of parameter starting values given in `start`. The starting values after optimization will be found in the slot `coh` of the object of class `theta` corresponding to each dataset modeled.
- "**mix**": if `type="mix"` if `type="mix"` a sequential exponential decay model is applied along with a model that follows the time profile of the IRF; the coherent artifact/scatter is then a linear superposition of these two models; see the above description of `seq` for how to supply the starting values.

**coh:** \*\*\* Object of class "vector" of starting values for the parameterization of a coherent artifact

**wavedep:** Object of class "logical" describing whether the kinetic model is dependent on `x2` index (i.e., whether there is `clp`-dependence)

- lambdac:** \*\*\* Object of class "numeric" for the center wavelength to be used in a polynomial description of  $\lambda^2$ -dependence
- amplitudes:** \*\*\* Object of class "vector" that may be used to multiply the concentrations by a square diagonal matrix with the number of columns that the concentration matrix has; the diagonal is given in `amplitudes` and these values will be treated as parameters to be optimized.
- streak:** \*\*\* Object of class "logical" that defaults to FALSE; if `streak=TRUE` then the period of the laser is expected via `streakT`.
- streakT:** \*\*\* Object of class "numeric" the period of the laser; this will be used to add a backsweep term to the concentration matrix and should be set in conjunction `streak=TRUE`.
- doublegaus:** \*\*\* Object of class "logical" that defaults to FALSE and determines whether a double Gaussian should be used to model the IRF. If `doublegaus=TRUE` then `irfpar` should contain four numeric values corresponding to the location (mean) of the IRF, the FWHM of the first Gaussian, the FWHM of the second Gaussian, and the relative amplitude of the second Gaussian, respectively.
- numericalintegration:** \*\*\* Object of class "logical" that defaults to FALSE and determines whether a kinetic theory model of a reaction mechanism should be numerically integrated (using `odesolve`) to find the concentrations. If `numericalintegration=TRUE` then `initialvals` should specify the initial concentrations and `reactantstoichiometrymatrix` and `stoichiometrymatrix` should specify the reaction mechanism, as per Puxty et. al. (2006).
- initialvals:** \*\*\* Object of class "vector" giving the concentrations at the initial time step.
- reactantstoichiometrymatrix:** \*\*\* Object of class "vector" giving the (integer) *stoichiometric coefficients* for the reactants; this is the matrix  $\mathbf{X}_r$  of Puxty et. al. (2006) with `dim=NULL`.
- stoichiometrymatrix:** \*\*\* Object of class "vector" giving the (integer) *stoichiometric coefficients* for the reactions; this is the matrix  $\mathbf{X}$  of Puxty et. al. (2006) with `dim=NULL`.

## Extends

Class `dat-class`, directly.

## Author(s)

Katharine M. Mullen, David Nicolaides, Ivo H. M. van Stokkum

## References

Puxty, G., Maeder, M., and Hungerbuhler, K. (2006) Tutorial on the fitting of kinetics models to multivariate spectroscopic measurements with non-linear least-squares regression, *Chemometrics and Intelligent Laboratory Systems* **81**, 149-164.

## See Also

`dat-class`, `spec-class`

**Examples**

```

## Example in modeling second order kinetics, by
## David Nicolaides.

## On simulated data.

#####
## load TIMP
#####

library("TIMP")

#####
## SIMULATE DATA
#####

## set up the Example problem, a la in-situ UV-Vis spectroscopy of a simple
## reaction.
## A + 2B -> C + D, 2C -> E

cstart <- c(A = 1.0, B = 0.8, C = 0.0, D = 0.0, E = 0.0)
times <- c(seq(0,2, length=21), seq(3,10, length=8))
k <- c(kA = 0.5, k2C = 1)

## stoichiometry matrices

rsmatrix <- c(1,2,0,0,0,0,2,0,0)
smatrix <- c(-1,-2,1,1,0,0,0,-2,0,1)
concentrations <- calcD(k, times, cstart, rsmatrix, smatrix)

wavelengths <- seq(500, 700, by=2)
spectra <- matrix(nrow = length(wavelengths), ncol = length(cstart))
location <- c(550, 575, 625, 650, 675)
delta <- c(10, 10, 10, 10, 10)
spectra[, 1] <- exp( - log(2) *
(2 * (wavelengths - location[1])/delta[1])^2)
spectra[, 2] <- exp( - log(2) *
(2 * (wavelengths - location[2])/delta[2])^2)
spectra[, 3] <- exp( - log(2) *
(2 * (wavelengths - location[3])/delta[3])^2)
spectra[, 4] <- exp( - log(2) *
(2 * (wavelengths - location[4])/delta[4])^2)
spectra[, 5] <- exp( - log(2) *
(2 * (wavelengths - location[5])/delta[5])^2)

sigma <- .001
Psi_q <- concentrations %*% t(spectra) + sigma *
  rnorm(dim(concentrations)[1] * dim(spectra)[1])

## store the simulated data in an object of class "dat"
kinetic_data <- dat(psi.df=Psi_q , x = times, nt = length(times),
  x2 = wavelengths, nl = length(wavelengths))

```

```
#####
## DEFINE MODEL
#####

## starting values
kstart <- c(kA = 1, k2C = 0.5)

## model definition for 2nd order kinetics
kinetic_model <- initModel(mod_type = "kin", seqmod = FALSE,
                           kinpar = kstart,
                           numericalintegration = TRUE,
                           initialvals = cstart,
                           reactantstoichiometrymatrix = rsmatrix,
                           stoichiometrymatrix = smatrix )

#####
## FIT INITIAL MODEL
## adding constraints to non-negativity of the
## spectra via the opt option nnls=TRUE
#####

kinetic_fit <- fitModel(data=list(kinetic_data),
                        modspec = list(kinetic_model),
                        opt = kinopt(nnls = TRUE, iter=80,
                                     selectedtraces = seq(1,kinetic_data@n1,by=2)))

## look at estimated parameters

parEst(kinetic_fit)

## various results

## concentrations

conRes <- getX(kinetic_fit)

matplot(times, conRes, type="b", col=1,pch=21, bg=1:5, xlab="time (sec)",
        ylab="concentrations", main="Concentrations (2nd order kinetics)")

## spectra

specRes <- getCLP(kinetic_fit)

matplot(wavelengths, specRes, type="b", col=1,pch=21, bg=1:5,
        xlab="wavelength (nm)",
        ylab="amplitude", main="Spectra")

## see help(getResults) for how to get more results information from
## kinetic_fit
```

---

kinopt-class                      *Class "kinopt" stores options for fitting and plotting kinetic models*

---

## Description

Class "kinopt" stores options for fitting and plotting kinetic models in particular; this is a subclass of class `opt`

## Details

See `opt-class` for the specification of fitting/plotting options that are not specific to the class type.

## Objects from the Class

Objects can be created by calls of the form `new("kinopt", ...)` or `kinopt(...)`

## Slots

**notraces:** Object of class "logical" that defaults to FALSE; if TRUE, do not plot traces

**selectedtraces:** Object of class "vector" containing x indices for which plots of traces are desired under a kinetic model

**breakdown:** Object of class "list" with the following elements:

`plot` vector of x2 values to plot the breakdown for. These values be specified in a fuzzy way: an x2 value within  $\text{abs}(x2[1] - x2[2])/100$  a value given in `plot` means that a plot for that x2 value will be generated, where the reference `x2[1]` and `x2[2]` are from the first dataset modelled.

`tol` numeric giving a tolerance by which the values in `plot` are compared to x2 values for near-equality. The default is defined as  $\text{abs}(x2[1] - x2[2])/100$ .

`superimpose` vector of dataset indices for which results should be superimposed if the dataset has an x2 value at a value in `plot`.

**FLIM:** Object of class "logical" that defaults to FALSE; if TRUE, the data represent a FLIM experiment and special plots are generated.

**FLIMresidimag:** Object of class "logical" that defaults to TRUE; if FALSE and a FLIM image is analyzed, the residuals are not plotted as an image.

**noFLIMsummary:** Object of class "logical" that defaults to FALSE; if TRUE and a FLIM image is analyzed, only other plots requested by the user (such as traces or residuals) are generated, and no summary plot is made.

**kinspecest** Object of class "logical" that defaults to FALSE; if TRUE, make a plot of the spectra associated with the kinetic components as well as the lifetime estimates.

**writeplaincon** Object of class "list"; if length is greater than 0, then the concentration model will be evaluated at the vector of x values supplied as the element "x" of `writeplaincon` and the result will be written to file for each dataset.

**writerawcon** Object of class "logical" that defaults to FALSE; if TRUE, then the representation of the concentration profiles before the application of constraints (to account for the equality of spectra, etc.) is written to file for each dataset.

**plotcoholspec** Object of class "logical" that defaults to TRUE; if FALSE then the spectra associated with the coherent artifact (pulse-follower) are not included in the summary plots

**plotpulsefol**: Object of class "logical" defaults to FALSE; if TRUE adding imageplots of pulsefollower amplitudes in summary plot (only with FLIM plots).

**ylimcomp** Object of class "vector" that defaults to `vector()`; Works In the case of plotting the results of FLIM image analysis, `ylimspec` can be used to determine the range used in the image plot of normalized amplitudes.

#### Author(s)

Katharine M. Mullen, Ivo H. M. van Stokkum

#### See Also

[examineFit](#), [fitModel](#), [opt-class](#), [specopt-class](#)

---

mass-class

*Class "mass" for mass spectrometry model storage.*

---

#### Description

`mass` is the class for mass spectrometry models; an object of class "mass" is initialized if `mod_type = "mass"` is an argument of `initModel`. All objects of class `mass` are sub-classes of class `kin`; see documentation for `kin` for a description of these slots.

#### Details

See [kin-class](#) for an example of the initialization of a `kin` object via the `initModel` function.

#### Objects from the Class

Objects can be created by calls of the form `new("mass", ...)` or `kin(...)`.

#### Slots

**peakpar** list of vectors of starting values for the parameters of components; one vector of values is used to parameterize each component.

**peakfunct**: Object of class "character" that specifies the function by which components are parameterized in time; this is by default "expmodgaus" for the exponentially modified Gaussian function.

**lzerofile**: Object of class "character" that specifies the filename of the `lzero` specification to read in from file. This file has the format: 1st line not read; lines thereafter are the space-delimited index of the component to constrain, the lower bound of the constraint, and the upper bound of the constraint, e.g., `1 218.80 220.09`

**extracomp:** Object of class "logical" that defaults to TRUE and determines whether a component with constant concentration in time is added to the model to represent a baseline.

**shift:** Object of class "vector" that represents a shift of the location of each elution profile peak; this can be specified per-component, in which case `length(shift)` is the number of components (not including a baseline component) or for all components, in which case `length(shift == 1)`.

### Extends

Class `kin-class`, directly.

### Author(s)

Katharine M. Mullen, Ivo H. M. van Stokkum

### See Also

`kin-class`, `spec-class`

---

<code>massopt-class</code>	<i>Class "massopt" stores options for fitting and plotting models for mass spectrometry data</i>
----------------------------	--

---

### Description

Class "massopt" stores options for fitting and plotting models for mass spectrometry data in particular; this is a subclass of class `opt` that contains options applicable to all model types

### Details

See `opt-class` and for the specification of fitting/plotting options that are not specific to the `mass` class type.

### Objects from the Class

Objects can be created by calls of the form `new("massopt", ...)` or `massopt(...)`

### Slots

**axis.by:** Object of class "numeric" that allows labels on the bars representing the mass spectra to be skipped, e.g., `axis.by=2` will add a label to every second bar

**scale.concen:** Object of class "logical" that scales the concentration matrix using the algorithm found in the function `scaleConList`.

**nummaxtraces:** Object of class "nummaxtraces" that defaults to zero; if greater than zero then this number of the traces with the maximum amplitude are plotted

**Author(s)**

Katharine M. Mullen, Ivo H. M. van Stokkum

**See Also**

[examineFit](#), [fitModel](#), [opt-class](#), [specopt-class](#)

---

mea\_IRF

*Instrument response for fluorescent lifetime imaging microscopy (FLIM) data*

---

**Description**

A measured instrument response over 256 time channels for a set of fluorescent lifetime imaging microscopy (FLIM) datasets.

**Usage**

```
data("mea_IRF")
```

**Format**

mea\_IRF represents a measured instrument response as a numeric vector over 256 time channels.

**Details**

See [FLIMplots](#) for examples using this data.

**References**

This data was described in

Mullen KM, van Stokkum IHM (2008). The variable projection algorithm in time-resolved spectroscopy, microscopy and mass-spectroscopy applications, *Numerical Algorithms*, **in press**, <http://dx.doi.org/10.1007/s11075-008-9235-2>.

---

modifyModel	<i>Allows the starting values for parameters associated with a model to be updated with the values found in fitting the model.</i>
-------------	--

---

### Description

Allows the starting values for parameters associated with a model to be updated with the values found in fitting the model. That is, a model is specified with `initModel`. Then `fitModel` is used to optimize the starting values for parameters. `modifyModel` allows modification of the starting values in the model specification with the optimized values found via `fitModel`.

### Usage

```
modifyModel(model = list(), newest = list(), exceptslots = vector() )
```

### Arguments

<code>model</code>	an object of class <code>dat</code> returned by <code>initModel</code> ; if this argument is of length(0), which is the default, then the last model fit is used (which is found in the global variable <code>.currModel@model</code> )
<code>newest</code>	an object of class <code>theta</code> containing new parameter estimates; if this argument is of length(0), which is the default, then the parameter estimates associated with dataset 1 in the last model fit are used (which are found in the global variable <code>.currTheta[[1]]</code> )
<code>exceptslots</code>	a vector of character vector of slot names whose corresponding slots are to be left out of the update

### Value

an object of class `dat` that returns the results of calling `initModel` with the new starting values.

### Author(s)

Katharine M. Mullen, Ivo H. M. van Stokkum

### See Also

[initModel](#), [fitModel](#)

---

`multimodel-class`    *Class "multimodel" for storage of multidataset models, data and the results of fitting.*

---

### Description

`multimodel` is the class to store data, a generally applicable model, a list of per-data models, a specification of per-dataset model differences, and results for the analysis of possibly many datasets. After a call to `fitModel` an object is initialized of the `multimodel` class.

### Details

after a call to `fitModel`, an object of class `multimodel` exists in the global environment as the variable `currModel`

### Objects from the Class

Objects can be created by calls of the form `new("multimodel", ...)` or `multimodel(...)`.

### Slots

**data:** Object of class "list" of objects of class `dat` containing data

**model:** Object of class "dat" of class `dat` containing a model specification to be applied to all datasets

**modellist:** Object of class "list" of length `n` where `n` is the number of datasets given in `data`, and each element `i` is an object of class `dat` giving the dataset-specific model applicable to `data[[i]]`

**modeldiffs:** Object of class "list" of per-dataset model differences input as an argument to the `fitModel` function

**fit:** Object of class "fit" containing a list of results per-dataset as well as the output of optimization returned by the `nls` function.

**groups:** Object of class "list" containing a list of lists of the groups of `clp` to link across datasets. Each component list contains vectors of form (clp condition index, dataset index), and such vectors in the same component list are linked between datasets. See `fitModel` for more details on the linking possibilities.

**stderrclp:** Object of class "logical" describing whether standard error estimates on conditionally linear parameters should be calculated; this is determined by the `opt` argument of `fitModel` and defaults to `FALSE`

### Author(s)

Katharine M. Mullen, Ivo H. M. van Stokkum

### See Also

[fitModel](#)

---

`multitheta-class`    *Class "multitheta" that stores a list with one element of class "theta" for each dataset modeled.*

---

### Description

Class `multitheta` stores a list with one element of class `theta` for each dataset modeled, corresponding to the parameter estimates associated with that dataset.

### Objects from the Class

Objects can be created by calls of the form `new("multitheta", ...)` or `multitheta(...)`.

### Slots

**th:** Object of class `"list"` with element `i` corresponding to the `theta` object for the `i`th dataset modeled.

### Author(s)

Katharine M. Mullen, Ivo H. M. van Stokkum

### See Also

[theta-class](#), [dat-class](#)

---

`opt-class`    *Class "opt" stores options for fitting and plotting*

---

### Description

Class `"opt"` stores options for fitting and plotting applicable to all model types

### Details

See [kinopt-class](#), [specopt-class](#) and [massopt-class](#) for the specification of fitting/plotting options that are specific to the class type.

### Objects from the Class

Objects can be created by calls of the form `new("opt", ...)` or `opt(...)`.

**Slots**

**algorithm:** Object of class "character" that defaults to `algorithm="nls"`, so that the function `nls` is used to optimize nonlinear parameters under least squares criteria. Other options are

**nls.lm:** optimize nonlinear parameters under least squares criteria using `nls.lm`

**optim:** optimize nonlinear parameters under poisson regression criteria with the Nelder-Mead algorithm in `optim`; if this option is used then it **MUST** be used in conjunction with `nnls=TRUE`. Currently, it must also be used with `stderrclp=FALSE`.

**nnls:** Object of class "logical" that defaults to `FALSE`. If `nnls=TRUE`, constrain the conditionally linear parameters to nonnegativity via a nonnegative least squares algorithm as implemented via the function `nnls` from the package by the same name.

**writecon:** Object of class "logical" that defaults to `FALSE`; if true then concentrations are written to a txt file; row labels are `x`

**writespec:** Object of class "logical" that defaults to `FALSE`; if `TRUE` then spectra are written to a txt file; row labels are `x2`

**writenormspec:** Object of class "logical" that defaults to `FALSE`; if `TRUE` then normalized spectra are written to a txt file; row labels are `x2`

**writefit:** Object of class "logical" that defaults to `FALSE`; if `TRUE` then fit is written to a txt file; row and column labels are `x` and `x2`

**writeclperr:** Object of class "logical" that defaults to `FALSE`; if true then the error bars for `clp` are written to a txt file. This option is only sensible with `stderrclp=TRUE`.

**output:** Object of class "character" that defaults to `"ps"`, which means that plots written to file are postscript. Alternatively, specify `output = "pdf"`, and plots are written as pdf files

**addfilename:** Object of class "logical" that, for each data file, tries to add the filename to plots associated with `output` for that data.

**residplot:** Object of class "logical" defaults to `FALSE`; if `TRUE` generate a plot of residuals in a separate window.

**adddataimage:** Object of class "logical" defaults to `FALSE`; if `TRUE` adding imageplot of data in summary plot.

**plot:** Object of class "logical" that defaults to `TRUE`; if `FALSE` then do not write output in the form of plots and other windows to the screen.

**divdrel:** Object of class "logical" that defaults to `FALSE`; if `TRUE`, plot traces and concentration profiles divided by the dataset scaling parameters where they apply; this allows for the fit of datasets having different intensities on the same scale.

**plotkinspec:** Object of class "logical" that defaults to `FALSE`; if `TRUE`, generates a separate plot of the spectra associated with the components that are not a part of a coherent artifact/scatter model.

**superimpose:** Object of class "vector" containing dataset indices whose results should be superimposed in plots

**xlab:** Object of class "character" containing label for x-axis, e.g., `"nanoseconds"` or `"picoseconds"`

**ylab:** Object of class "character" containing label for y-axis, e.g., `"wavelength"`

- title:** Object of class "character" containing title to write at the top of plots.
- makeps:** Object of class "character" containing prefix to plot files written to postscript; if present postscript will be written. Note that this string is also used as the prefix of txt output files
- linrange:** Object of class "numeric" giving linear range of time axis for plotting; time will be plotted linearly from -linrange to linrange and plotted on a logarithmic (base 10) axis elsewhere
- summaryplotrow:** Object of class "numeric" giving number of rows in summary plot; defaults to 4
- summaryplotcol:** Object of class "numeric" giving number of columns in summary plot; defaults to 4
- iter:** Object of class "numeric" giving number of iterations to optimize model parameters; if `nls=FALSE` so that the Levenberg-Marquardt algorithm is applied, then `iter` is interpreted as the maximum number of residual function evaluations (see the help page of the function `nls.lm` for details)
- paropt:** Object of class "list" of graphical parameters in format `par(...)` to apply to plots.
- stderrclp:** Object of class "logical" that defaults to `FALSE`; if `TRUE`, estimates of the standard error of conditionally linear parameters are made
- addest:** Object of class "vector" containing character strings of which parameter estimates should be added to the summary plot, e.g., `addest = c("kinpar", "irfpar")`
- kinspecerr** Object of class "logical" that defaults to `FALSE`; if `TRUE`, add standard error estimates to the `clp` a plot generated with `kinspecest=TRUE` or `plotkinspec=TRUE`. This option can only be used if the estimates were generated during fitting via the option `stderrclp=TRUE`
- xlimspec** Object of class "vector" that defaults to `vector()`; if changed, it should specify the desired x-limits of the plot of `clp`
- ylimspec** Object of class "vector" that defaults to `vector()`; if changed, it should specify the desired y-limits of the plot of `clp`. In the case of plotting the results of FLIM image analysis, `ylimspec` can be used to determine the range used in the image plot of lifetimes.
- ylimspecplus** Object of class "vector" that defaults to `vector()`; if changed, the first value should specify a vector to add to the y-limits of the plot of `clp`
- samespecline** Object of class "logical" that defaults to `FALSE`; if `TRUE`, then the line-type for `clp` is the same for all datasets
- specinterpol** Object of class "logical" that defaults to `FALSE`; if `TRUE`, use spline instead of lines between the points representing estimated `clp`
- specinterpolpoints** Object of class "logical" that defaults to `TRUE`; if `TRUE`, add points representing the actual estimates for `clp` to plots of the curves representing smoothed `clp`
- specinterpolseg** Object of class "numeric" that defaults to 50; represents the number of segments used in a spline-based representation of `clp`
- specinterpolbspline** Object of class "logical" that defaults to `FALSE`; determines whether a B-spline based representation of `clp` is used (when `specinterpol=TRUE`) or a piecewise polynomial representation
- normspec** Object of class "logical" that determines whether `clp` are normalized in plots

**writespecinterpol** Object of class "logical" that defaults to FALSE; if TRUE, a spline-based representation of `clp` is written to ASCII files

**nlsalgorithm** Object of class "character" that defaults to "default" and determines the algorithm used by `nls`, if `nls` is used in optimization. See `help(nls)` for other possibilities, such as "port", which is more stable with respect to starting values but requires more time.

**ltyfit** Object of class "numeric" if given, sets the line type of the fit in plots of the fit/data; see `lty` in `help(par)` for options.

**ltydata** Object of class "numeric" if given, sets the line type of the data in plots of the fit/data; see `lty` in `help(par)` for options.

**colfit** Object of class "vector" if given, sets the color of the fit corresponding to each dataset in plots of the fit/data; see `col` in `help(par)` for options. If given `length(colfit)` must be equal to the number of datasets in the analysis

**coldata** Object of class "vector" if given, sets the color of the data for each dataset in plots of the fit/data; see `col` in `help(par)` for options. If given, `length(coldata)` must be equal to the number of datasets in the analysis

#### Author(s)

Katharine M. Mullen, Ivo H. M. van Stokkum

#### See Also

[kinopt-class](#), [specopt-class](#)

---

outlierCorr

*Finds and removes outliers from a datasets*

---

#### Description

Finds and removes outliers from datasets given the results of fitting as returned by `fitModel`. Uses the residuals in the fitted results to return a list of corrected datasets to be used in place of the datasets used in the call to `fitModel` as well as a list of weights. The data returned contains the fitted values at pointed that are outliers and will be assigned zero weight in subsequent fits.

#### Usage

```
outlierCorr(oldRes, fence=3, saturCorr=FALSE, saturThresh=.05,
            saturMin=NA, saturDivMax=3, outlierCorr=TRUE,
            newM = TRUE)
```

#### Arguments

<code>oldRes</code>	Object returned by <code>fitModel</code> function
<code>fence</code>	Object of class "numeric" determining what points to consider outliers.
<code>saturCorr</code>	whether to correct for saturation

saturThresh	See code.
saturMin	See code.
saturDivMax	See code.
outlierCorr	whether to perform outlier correction
newM	whether to add to the outliers and saturation points detected previously

### Details

We calculate the fourth spread at a given value of  $x_2$  in a dataset. Those points that are less than the first quartile minus the fourth spread times `fence` are outliers, as are those points that are more than the third quartile plus the fourth spread times `fence`. Outliers are assigned a weight of zero and are assigned the values found in fitting for the purpose of generating smooth-looking plots.

### Value

list containing the elements `dt`, a list of corrected datasets, and `weightList`, a list of new weight matrices.

### See Also

[fitModel](#), [preProcess](#)

---

plotter-methods      *Generic function plotter in Package ‘TIMP’*

---

### Description

Methods for function `plotter` in Package ‘TIMP’ that call plotting and output functions.

### Usage

```
plotter(model, multimodel, multitheta, plotoptions)
```

### Arguments

<code>model</code>	Object of class <code>dat</code> ; function switches on this argument.
<code>multimodel</code>	Object of class <code>multimodel</code>
<code>multitheta</code>	Object of class <code>multitheta</code>
<code>plotoptions</code>	list of output options input to <code>fitModel</code> as the argument <code>opt</code>

### Author(s)

Katharine M. Mullen, Ivo H. M. van Stokkum

### See Also

[dat-class](#)

preProcess

*Performs preprocessing on data stored as an objects of class dat.***Description**

Performs data sampling, selection, baseline correction, scaling, and data correction on an object of class `dat`.

**Usage**

```
preProcess(data, sample = 1, sample_time = 1, sample_lambda = 1,
           sel_time = vector(), sel_lambda = vector(), baselinetime = vector(),
           baselinelambda = vector(), scalx = NULL, scalx2 = NULL,
           sel_lambda_ab = vector(), sel_time_ab = vector(), rm_x2=vector(),
           rm_x = vector(), svdResid = list(), numV = 0, sel_special = list(),
           doubleDiff = FALSE, doubleDiffFile = "doubleDiff.txt")
```

**Arguments**

<code>data</code>	Object of class <code>dat</code>
<code>sample</code>	integer describing sampling interval to take in both time and <code>x2</code> ; e.g., <code>sample=2</code> will sample every 2nd time and every 2nd point in <code>x2</code> .
<code>sample_time</code>	integer describing sampling interval in time; e.g., <code>sample_time=2</code> will sample every 2nd element of the time vector.
<code>sample_lambda</code>	integer describing sampling interval in <code>x2</code> ; e.g., <code>sample_lambda=2</code> will sample every 2nd element in the <code>x2</code> vector.
<code>sel_time</code>	vector of length 2 describing the first and last time index of data to select; e.g., <code>sel_time=c(5, 120)</code> will select data at times indexed 5-120.
<code>sel_lambda</code>	vector of length 2 describing the first and last <code>x2</code> index of data to select; e.g., <code>sel_lambda=c(5, 120)</code> will select data at <code>x2</code> indexed 5-120.
<code>baselinetime</code>	a vector of form <code>c(timeIndexmin, timeIndexmax, lambdaIndexmin, lambdaIndexmax)</code> . The average of data between <code>x2</code> indexes <code>lambdaIndexmin</code> and <code>lambdaIndexmax</code> is subtracted from data with time index between <code>timeIndexmin</code> and <code>timeIndexmax</code> .
<code>baselinelambda</code>	a vector of form <code>c(timeIndexmin, timeIndexmax, lambdaIndexmin, lambdaIndexmax)</code> . The average of data between time indexes <code>timeIndexmin</code> and <code>timeIndexmax</code> is subtracted from data with <code>x2</code> index between <code>lambdaIndexmin</code> and <code>lambdaIndexmax</code> .
<code>scalx</code>	numeric by which to linearly scale the <code>x</code> axis (which often represents time), so that <code>newx = oldx * scalx</code>
<code>scalx2</code>	vector of length 2 by which to linearly scale the <code>x2</code> axis, so that <code>newx2 = oldx2 * scalx2[1] + scalx2[2]</code>

<code>sel_lambda_ab</code>	vector of length 2 describing the absolute values (e.g., wavelengths, wavenumbers, etc.) between which data should be selected. e.g., <code>sel_lambda_ab = c(400, 600)</code> will select data associated with <code>x2</code> values between 400 and 600.
<code>sel_time_ab</code>	vector of length 2 describing the absolute times between which data should be selected. e.g., <code>sel_time_ab = c(50, 5000)</code> will select data associated with time values between 50 and 5000 picoseconds.
<code>rm_x2</code>	vector of <code>x2</code> indices to remove from the data
<code>rm_x</code>	vector of <code>x</code> indices to remove from the data
<code>svdResid</code>	list returned from the <code>getResid</code> function, containing residuals to be used in data correction.
<code>numV</code>	numeric specifying how many singular vectors to use in data correction. Maximum is five.
<code>sel_special</code>	list of lists specifying <code>x</code> indices to remove from individual wavelength ranges, e.g., <code>sel_special = list(c(400, 600, 10, 12), c(600, 800, 11, 13))</code> indicates that between wavelength 400 and 600, time indices between 10 and 12 should be removed from the data, and between wavelengths 600 and 800, time indices between 11 and 13 should be removed from the data. Note that the number of time indices to remove should be the same in each wavelength interval specified. Also note that the time vector associated with the data after the first set of indices is removed will be associated with the resulting dataset.
<code>doubleDiff</code>	logical indicating whether the data should be converted to represent differences between times.
<code>doubleDiffFile</code>	character string indicating the file name of time difference data to create in the case that <code>doubleDiff=TRUE</code> .

**Value**

object of class `dat`.

**Author(s)**

Katharine M. Mullen, Ivo H. M. van Stokkum

**See Also**

[readData](#), [getResid](#)

**Examples**

```
#####
## READ DATA
#####

data("target")
```

```
#####
## PREPROCESS DATA
#####

# select certain wavelengths for modeling

C1_1 <- preProcess(data = C1, baselinelambda = c(1,12,1,32) )
C1_1 <- preProcess(data = C1_1, sel_lambda = c(8, 27))
C1_1 <- preProcess(data = C1_1, rm_x = c(40, 41, 101, 116))
C1_1 <- preProcess(data = C1_1, sel_time_ab = c(-10, 100000))

C2_1 <- preProcess(data = C2, sel_lambda = c(2, 32))
C2_1 <- preProcess(data = C2_1, baselinelambda = c(1,12,1,32) )
C2_1 <- preProcess(data = C2_1, sel_time_ab = c(-10, 100000))

C3_1 <- preProcess(data = C3, sel_lambda = c(1, 25))
C3_1 <- preProcess(data = C3_1, baselinelambda = c(1,12,1,32) )

#####
## SPECIFY K Matrix and J vector
#####

## initialize 2 7x7 arrays to 0

delK <- array(0, dim=c(7,7,2))

## the matrix is indexed:
## delK[ ROW K MATRIX, COL K MATRIX, matrix number]

## in the first matrix, put the index of compartments
## that are non-zero
## the transfer rate of the compartment is governed by
## kinpar[index]

delK[1,1,1] <- 4
delK[5,1,1] <- 1
delK[2,2,1] <- 4
delK[5,2,1] <- 2
delK[3,3,1] <- 4
delK[5,3,1] <- 3
delK[4,4,1] <- 4
delK[6,5,1] <- 5
delK[7,6,1] <- 6
delK[7,7,1] <- 7

## print out the resulting array to make sure it's right

delK

jvector <- c(.48443195136500550341, .28740782363398824522,
.13749071230100625137, 0.9066953510E-01, 0, 0, 0)

datalist <- list(C1, C2, C3)
```

```

## for plotting selected traces, get a vector of all the wavenumbers
allx2 <- vector()
for(i in 1:length(datalist))
  allx2 <- append(allx2,datalist[[i]]@x2)
allx2 <- sort(unique(allx2))

#####
## SPECIFY INITIAL MODEL
## note that low is the larger wavenumber in the clpequ spec!
#####

modell1 <- initModel(mod_type = "kin",
kinpar=c( 0.13698630, 0.3448275849E-01, 0.1020408142E-01, 0.2941176528E-02,
0.17000, 0.015, 0.1074082902E-03),
fixed = list(prel = 1:6, clpequ=1:3, kinpar=1:7, irfpar=1, parmu=1),
irfpar=c(0.4211619198, 0.6299000233E-01),
prelspec = list(
list(what1="kinpar", ind1=1, what2 = "kinpar", ind2=4,
start=c(-1,0.1369863003)),
list(what1="kinpar", ind1=2, what2 = "kinpar", ind2=4,
start=c(-1,0.3448275849E-01)),
list(what1="kinpar", ind1=3, what2 = "kinpar", ind2=4,
start=c(-1,0.1020408142E-01))
),
parmu = list(c(-0.1411073953)),
lambdac = 1290,
kmat = delK,
jvec = jvector,
positivepar="kinpar",
weightpar=list( c(-20,1.4,1,2000,.2)),
clpequspec =list(
list(to=2, from=1, low=100, high=10000),
list(to=3, from=1, low=100, high=10000),
list(to=4, from=1, low=100, high=10000)),
clpequ = c(1,1,1),
cohspec = list( type = "irf"))

#####
## GET RESID
## same format as call to fitModel, but does not plot
#####

serResid <- getResid(list(C1_1, C2_1, C3_1), list(modell1),
modeldiffs = list(thresh = 0.00005,
dscal = list(list(to=2,from=1,value=4),
list(to=3,from=1,value=0.8000000119)),
free = list(
list(what="irfpar", ind=1, start= c(0.1231127158), dataset=2),
list(what="parmu", ind=c(1,1), start= c(0.1219962388), dataset=2),
list(what="irfpar", ind=1, start= c(0.3724052608), dataset=3),
list(what="parmu", ind=c(1,1), start= c(0.8844097704E-01), dataset=3)),
change = list(
list(what="fixed", spec=list(clpequ=1:3, kinpar=1:7, irfpar=1:2,

```

```

parmu=1, drel = 1, prel=1:6), dataset=2:3))),
opt=kinopt(iter=0, title="Cosimo Spectra, Not Normalized, with Error",
stderrclp=TRUE, kinspecerr=TRUE, writespec = TRUE,
plotkinspec = TRUE, plotcohcolspec=FALSE,
selectedtraces = seq(1, length(allx2), by=2),
specinterpol = TRUE, specinterpolpoints=FALSE,
divdrel=TRUE, xlab="wavenumber", writeclperr = TRUE,
makeps = "err", linrange = 1, superimpose=1:3))

#####
## MAKE CORRECTED DATASETS USING RESID INFO
#####

C1_3 <- preProcess(data = C1_1, svdResid = serResid[[1]], numV = 2)
C2_3 <- preProcess(data = C2_1, svdResid = serResid[[2]], numV = 2)
C3_3 <- preProcess(data = C3_1, svdResid = serResid[[3]], numV = 2)

#####
## FIT MODEL
#####

serRes<-fitModel(list(C1_3, C2_3, C3_3), list(model1),
modeldiffs = list(thresh = 0.00005,
dscal = list(list(to=2, from=1, value=4),
list(to=3, from=1, value=0.8000000119)),
free = list(
list(what="irfpar", ind=1, start= c(0.1231127158), dataset=2),
list(what="parmu", ind=c(1,1), start= c(0.1219962388), dataset=2),
list(what="irfpar", ind=1, start= c(0.3724052608), dataset=3),
list(what="parmu", ind=c(1,1), start= c(0.8844097704E-01), dataset=3)),
change = list(
list(what="fixed", spec=list(clpequ=1:3, kinpar=1:7, irfpar=1:2,
parmu=1, drel = 1, prel=1:6), dataset=2:3))),
opt=kinopt(iter=0, title="Cosimo Spectra, Not Normalized, with Error",
stderrclp=TRUE, kinspecerr=TRUE, writespec = TRUE,
plotkinspec = TRUE, plotcohcolspec=FALSE, writerawcon = TRUE,
selectedtraces = seq(1, length(allx2), by=2),
specinterpol = TRUE, specinterpolpoints=FALSE,
divdrel=TRUE, xlab="wavenumber", writeclperr = TRUE,
makeps = "h20", linrange = 1, superimpose=1:3))

```

---

readclp0

---

*This function reads in a specification of constraints to zero on the clp.*


---

## Description

This function is useful for the case that there are many constraints to zero in the model, as is the case for some mass spectrometry models.

**Usage**

```
readclp0(filenm)
```

**Arguments**

`filenm` Object of class "character" that gives is the path to the file to read in.

**Details**

The file to be read in should have the following format: 1st line is not read. Lines thereafter are the space-delimited index of the component to constrain, the lower bound of the constraint, and the upper bound of the constraint, e.g., 1 218.800000000000011 220.099999999999994.

**Value**

The constraints to zero in the format documented in the help file for the "dat" class. Therefore a call to "readclp0" may be used inside a call to "initModel", as in `clp0 = readclp0("filename")`.

**Author(s)**

Katharine M. Mullen, Ivo H. M. van Stokkum

**See Also**

[initModel](#)

---

readData

*This function reads in data the ivo file format*

---

**Description**

Data in the formats described at <http://www.nat.vu.nl/~kate/TIM/tim/node74.html> and [http://www.nat.vu.nl/~kate/FLIM\\_format](http://www.nat.vu.nl/~kate/FLIM_format) may be read from file into an R object for analysis.

**Usage**

```
readData(filenm, typ="", sep = "")
```

**Arguments**

`filenm` This is the path to the file to read in, as a quoted string.

`typ` if `typ="plain"` the the file being read in stores data as a plain matrix, with `x` values as the first element of each row except the first and `x2` values as the first row.

`sep` This is an optional argument describing how the data is delimited; defaults to ""

**Value**

an object of class `dat`

**Author(s)**

Katharine M. Mullen, Ivo H. M. van Stokkum

**See Also**

[preProcess](#)

---

res-class

*Class "res" to store the results of model fitting associated with a single dataset.*

---

**Description**

Class to store results of model fitting associated with a single dataset. A list containing objects of class `res` is a slot in class `fit`. An object of class `fit` is stored in the slot `fit` of objects of class `multimodel`.

**Objects from the Class**

Objects can be created by calls of the form `new("res", ...)`. A `res` object is created after model fitting via the residual function `residPart`.

**Slots**

**cp:** Object of class `"list"` that contains the estimates for conditionally linear parameters.

**resid:** Object of class `"list"` of residuals, with one element for each dataset modeled.

**fitted:** Object of class `"list"` of fits, with one element for each dataset modeled.

**irfvec:** Object of class `"list"` with a vector of elements for each element of the `clp`  $\times$  2

**Author(s)**

Katharine M. Mullen, Ivo H. M. van Stokkum

**See Also**

[fit-class](#), [multimodel-class](#)

---

residPart-methods    *Generic function residPart in Package ‘TIMP’*

---

## Description

Methods for function `residPart` in Package ‘TIMP’ determine the part of the residual vector associated with a single ‘part’ of the dataset(s).

## Usage

```
residPart(model, group, multimodel, thetalist, clpindepX, finished,
returnX, rawtheta)
```

## Arguments

<code>model</code>	Object of class <code>dat</code> ; switches on this argument.
<code>group</code>	list of vector pairs ( <code>x2</code> index, dataset index) for which the part of the residual vector is to be determined
<code>multimodel</code>	Object of class <code>multimodel</code>
<code>thetalist</code>	Object of class <code>multitheta</code>
<code>clpindepX</code>	Object of class <code>matrix</code> containing the matrix determined directly by the non-linear parameters (e.g., a concentration matrix in the case of a kinetic model) in the case that this matrix does not depend on the <code>x2</code> index
<code>finished</code>	logical determining whether fitting is finished that triggers the storage of results
<code>returnX</code>	logical determining whether to just return the matrix <code>X</code> directly dependent on nonlinear parameters; this is used in the finite difference derivative of <code>X</code> used to get standard error estimates on the conditionally linear parameters.
<code>rawtheta</code>	numeric vector of nonlinear parameters to be optimized by <code>nls</code> ; this is used in the finite difference derivative of <code>X</code> used to get standard error estimates on the conditionally linear parameters.

## See Also

[dat-class](#), [spec-class](#), [kin-class](#)

spec-class

Class "spec" for the storage of spectral models.

## Description

spec is the class for spectral models; an object of class "mass" is initialized if `mod_type = "spec"` is an argument of `initModel`. All objects of class `spec` are also of class `dat`; see documentation for `dat` for a description of these slots. Note that here `x2` will refer to the independent variable in which traces are resolved, e.g., wavelength or wavenumber.

## Objects from the Class

Objects can be created by calls of the form `new("spec", ...)` or `spec(...)`.

## Slots

- clpequ:** Object of class "vector" of starting values for linear relationships between `clp`
- specpar:** Object of class "list" of vectors of starting values for spectral parameters; the number of vectors gives the number of components in the resulting spectral model; each vector contains the parameters associated with a component. e.g., `specpar = list(c(20000, 3000, .3, 21000, 2000, .4), c(18000, 1000, .2))`; the parameters in each vector are grouped `c(location_spectra, width_spectra, skew_spectra)`. the location and width parameters are given in wavenumbers.
- specfun:** Object of class "character", "gaus" for a spectral model of a superposition of skewed Gaussians; "bspline" for a bspline-based model.
- specref:** Object of class "numeric" index defining the center value of the `x2` variable.
- specCon:** Object of class "list" used internally to store constraints.
- specdisp:** Object of class "logical" TRUE if time-dependence of the spectral parameters is to be taken into account and FALSE otherwise
- specdisppar:** Object of class "list"
- specdispindex:** Object of class "list" of vectors defining those indexes of `specpar` whose time-dependence is to be modeled. e.g., `specdispindex = list(c(1,1), c(1,2), c(1,3))` says that parameters 1-3 of spectra 1 are to be modeled as time-dependent.
- nupow:** Object of class "numeric" describing the power to which wavenumbers are raised in the model equation; see Equation 30 of the paper in the references section for a complete description
- timedep:** Object of class "logical" describing whether the model for spectra E is dependent on x-index (i.e., whether it is `clp`-dependent).
- parmufunc:** Object of class "character" describing the function form of the time-dependence of spectral parameters; options are "exp" for exponential time dependence, "multiexp" for multiexponential time dependence, and "poly" for polynomial time dependence. defaults to polynomial time dependence.
- ncole** vector describing the number of columns of the E matrix for each value in the `x` vector

**Extends**

Class `dat-class`, directly.

**Author(s)**

Katharine M. Mullen, Ivo H. M. van Stokkum

**References**

Ivo H. M. van Stokkum, "Global and target analysis of time-resolved spectra, Lecture notes for the Troisieme Cycle de la Physique en Suisse Romande", Department of Physics and Astronomy, Faculty of Sciences, Vrije Universiteit, Amsterdam, The Netherlands, 2005, <http://www.nat.vu.nl/~ivo/lecturenotes.pdf>

**See Also**

`kin-class`, `dat-class`

---

<code>specopt-class</code>	<i>Class "specopt" stores options for fitting and plotting spectral models</i>
----------------------------	--

---

**Description**

Class "specopt" stores options for fitting and plotting spectral models in particular; this is a subclass of class `opt`.

**Details**

See `opt-class` for the specification of fitting/plotting options that are not specific to the class type.

**Objects from the Class**

Objects can be created by calls of the form `new ("specopt", ...)` or `specopt (...)`

**Slots**

**nospectra:** Object of class "logical" that defaults to FALSE; if TRUE, do not plot time-resolved spectra

**selectedspectra:** Object of class "vector" containing x indices for which plots of time-resolved spectra are desired under a spectral model

**Author(s)**

Katharine M. Mullen, Ivo H. M. van Stokkum

**See Also**

`opt-class`, `kinopt-class`

---

sumKinSpecEst	<i>Makes a summary plot of spectra associated with kinetic components alongside a plot showing parameter estimates</i>
---------------	--

---

### Description

Makes a summary plot of spectra associated with kinetic components alongside a plot showing parameter estimates for, by default, kinetic parameters. If the analysis had more parameters in the `addEst` slot of the argument `opt`, then more parameters are displayed. Note that this summary leaves out the spectra associated with coherent artifact or scatter.

### Usage

```
sumKinSpecEst(listFits, addtitle = TRUE, customtitle = "", preps = "",
ylimlist=list(), kinspecerr=TRUE)
```

### Arguments

<code>listFits</code>	list of objects returned by the <code>fitModel</code> function
<code>addtitle</code>	logical regarding whether to add a title; if <code>TRUE</code> and <code>customtitle</code> is not given then the title is "Summary of EADS for: " plus the analysis titles
<code>customtitle</code>	character vector containing a title
<code>preps</code>	character vector describing the prefix of the postscript filename given as output
<code>ylimlist</code>	list with elements <code>list(ind, ylim)</code> . <code>ind</code> is an index into <code>listFits</code> ; <code>ylim</code> is the desired <code>ylim</code> for the plot for that analysis
<code>kinspecerr</code>	logical regarding whether to add error bars for to the estimated spectra.

### Details

This looks best with less than five objects in `listFits`.

### Author(s)

Katharine M. Mullen, Ivo H. M. van Stokkum

### See Also

[fitModel](#), [examineFit](#)

---

target	<i>Ultrafast time-resolved fluorescence data</i>
--------	--

---

### Description

Ultrafast time-resolved absorption data

### Usage

```
data("target")
```

### Format

C1, C2 and C3 are objects of class `dat`.

### Details

See [preProcess](#) for examples using this data.

### References

This data was described in Bonetti C, Mathes T, van Stokkum IHM, Mullen KM, Groot ML and van Grondelle R, Hegemann P and Kennis, JTM (2008), The variable projection algorithm in time-resolved spectroscopy, microscopy and mass-spectroscopy applications, Hydrogen bond switching among flavin and amino acid side chains in the BLUF photoreceptor observed by ultrafast infrared spectroscopy, *Biophysical Journal*, **in press**, <http://www.biophysj.org/cgi/content/abstract/biophysj.108.139246v1>.

---

theta-class	<i>Class "theta" for storage of nonlinear parameter estimates</i>
-------------	---

---

### Description

`theta` is the class to store parameter estimates associated with possibly many datasets; after a call to `fitModel` a list containing `theta` objects for each of the `n` datasets analyzed in the call to `fitModel` is created. To see the parameter estimates associated with the datasets, examine the object `currTheta` in the list returned by `fitModel`

### Details

after a call to `fitModel`, an object of class `theta` exists in the global environment as the variable `currTheta`

### Objects from the Class

Objects can be created by calls of the form `new("theta", ...)` or `theta(...)`.

**Slots**

- kinpar:** Object of class "vector" of rate constant estimates
- specpar:** Object of class "list" of spectral shape parameter estimates
- irfpar:** Object of class "vector" of IRF parameter estimates
- paramu:** Object of class "list" of parameter estimates describing dispersion of the location of other parameters (in time, temp., etc.)
- partau:** Object of class "vector" of parameter estimates describing dispersion of the width of other parameters (in time)
- clpequ:** Object of class "vector" of parameter estimates describing conditionally linear parameters (spectra, in a kinetic model) relations
- specdisppar:** Object of class "list" of parameter estimates describing dispersion of spectra
- kinscal:** Object of class "vector" of parameters describing kinetic relations in the context of a compartmental scheme
- prel:** Object of class "vector" of parameters describing relations between parameters (which may be linear, exponential, etc.)
- coh:** Object of class "vector" of parameters describing a coherent artifact or pulse follower.
- drel:** Object of class "vector" of parameters describing relations between datasets (linear, and possibly per-wavelength or, in general, per-clp)

**Author(s)**

Katharine M. Mullen, Ivo H. M. van Stokkum

**See Also**

[fitModel](#) , [multitheta-class](#)

---

<code>writeAverage</code>	<i>Writes the average of scans stored in a file to a new file in the 'ivo' format</i>
---------------------------	---

---

**Description**

Some measurement set-ups dump a set of matrices stacked on top of each other to a file; each matrix represents a scan. This function writes the average of the scans to a file in the '.ivo' format.

**Usage**

```
writeAverage(filename, ntimes, nwave, scans,
             fileout = paste(filename, "Average.ivo", sep=""),
             calibration = 1:nwave, wexplicit=FALSE)
```

**Arguments**

<code>filename</code>	This is the path to the file to read in, as a quoted string.
<code>ntimes</code>	number of times in each scan
<code>nwave</code>	number of wavelengths in each scan
<code>scans</code>	number of full scans to read
<code>fileout</code>	a character vector specifying the filename to write the averaged data to; the default is to write a file named "filenameAverage.ivo"
<code>calibration</code>	a numeric vector representing the wavelength labels; by default the labels "1, 2, ..., nwave" are used
<code>wexplicit</code>	logical whether the file is written in the 'wavelength explicit' format, with each column of the matrix written representing a wavelength, as opposed to the 'time explicit' format, where each column represents a timepoint.

**Author(s)**

Katharine M. Mullen, Ivo H. M. van Stokkum

**See Also**

[readData](#)

# Index

## \*Topic **classes**

- amp-class, 2
- dat-class, 4
- fit-class, 13
- kin-class, 29
- kinopt-class, 34
- mass-class, 35
- massopt-class, 36
- multimodel-class, 39
- multitheta-class, 40
- opt-class, 40
- res-class, 51
- spec-class, 53
- specopt-class, 54
- theta-class, 56

## \*Topic **datasets**

- denS4, 8
- donorAcceptorTagged, 10
- donorTagged, 11
- mea\_IRF, 37
- target, 56

## \*Topic **file**

- baseIRF, 3
- efit2file, 11
- examineFit, 12
- fitModel, 13
- getResid, 21
- getResults, 22
- initModel, 26
- modifyModel, 38
- outlierCorr, 43
- preProcess, 45
- readclp0, 49
- readData, 50
- sumKinSpecEst, 55
- writeAverage, 57

## \*Topic **hplot**

- divergeZimage, 9
- FLIMplots, 17

## \*Topic **methods**

- getClpindepX-methods, 20
- plotter-methods, 44
- residPart-methods, 52

## \*Topic **package**

- TIMP-package, 2

- amp (*amp-class*), 2

- amp-class, 2

- baseIRF, 3

- c001 (*donorTagged*), 11

- c003 (*donorTagged*), 11

- C1 (*target*), 56

- C2 (*target*), 56

- C3 (*target*), 56

- cy005c (*donorAcceptorTagged*), 10

- cy006 (*donorAcceptorTagged*), 10

- dat, 10

- dat (*dat-class*), 4

- dat-class, 21, 27, 29, 31, 32, 40, 44, 52, 54

- dat-class, 4

- denS4, 8

- denS5 (*denS4*), 8

- divergeZimage, 9

- donorAcceptorTagged, 10

- donorTagged, 11

- efit2file, 11

- examineFit, 12, 16, 35, 37, 55

- fit (*fit-class*), 13

- fit-class, 51

- fit-class, 13

- fitModel, 12, 13, 17, 21, 23, 27, 35, 37–39, 44, 55–57

- FLIMplots, 10, 11, 17, 37

- getCLP (*getResults*), 22

- getClpindepX
  - (*getClpindepX-methods*), 20
- getClpindepX, amp-method
  - (*getClpindepX-methods*), 20
- getClpindepX, kin-method
  - (*getClpindepX-methods*), 20
- getClpindepX, mass-method
  - (*getClpindepX-methods*), 20
- getClpindepX, spec-method
  - (*getClpindepX-methods*), 20
- getClpindepX-methods, 20
- getCLPList (*getResults*), 22
- getData (*getResults*), 22
- getdim1 (*getResults*), 22
- getdim2 (*getResults*), 22
- getResid, 21, 46
- getResiduals (*getResults*), 22
- getResults, 22
- getSVDDData (*getResults*), 22
- getSVDResiduals (*getResults*), 22
- getTraces (*getResults*), 22
- getX (*getResults*), 22
- getXList (*getResults*), 22
  
- initModel, 4, 7, 16, 26, 29, 38, 50
  
- kin (*kin-class*), 29
- kin-class, 2, 3, 7, 27, 35, 36, 52, 54
- kin-class, 29
- kinopt (*kinopt-class*), 34
- kinopt-class, 40, 43, 54
- kinopt-class, 34
  
- mass (*mass-class*), 35
- mass-class, 35
- massopt (*massopt-class*), 36
- massopt-class, 40
- massopt-class, 36
- mea\_IRF, 37
- modifyModel, 38
- multimodel (*multimodel-class*), 39
- multimodel-class, 13, 51
- multimodel-class, 39
- multitheta (*multitheta-class*), 40
- multitheta-class, 57
- multitheta-class, 40
  
- nls, 15
  
- onls (*getResults*), 22
  
- opt, 12
- opt (*opt-class*), 40
- opt-class, 3, 34–37, 54
- opt-class, 40
- outlierCorr, 43
  
- parEst (*getResults*), 22
- plotHistAmp (*FLIMplots*), 17
- plotHistNormComp (*FLIMplots*), 17
- plotIntenImage (*FLIMplots*), 17
- plotNormComp (*FLIMplots*), 17
- plotSelIntenImage (*FLIMplots*), 17
- plotTau (*FLIMplots*), 17
- plotter (*plotter-methods*), 44
- plotter, kin-method
  - (*plotter-methods*), 44
- plotter, mass-method
  - (*plotter-methods*), 44
- plotter, spec-method
  - (*plotter-methods*), 44
- plotter-methods, 44
- preProcess, 21, 44, 45, 51, 56
  
- readclp0, 49
- readData, 12, 16, 46, 50, 58
- res (*res-class*), 51
- res-class, 13
- res-class, 51
- residPart (*residPart-methods*), 52
- residPart, amp-method
  - (*residPart-methods*), 52
- residPart, kin-method
  - (*residPart-methods*), 52
- residPart, mass-method
  - (*residPart-methods*), 52
- residPart, spec-method
  - (*residPart-methods*), 52
- residPart-methods, 52
  
- spec, 27
- spec (*spec-class*), 53
- spec-class, 3, 7, 27, 32, 36, 52
- spec-class, 53
- specopt (*specopt-class*), 54
- specopt-class, 35, 37, 40, 43
- specopt-class, 54
- sumKinSpecEst, 55
- sumnls (*getResults*), 22
  
- target, 56

`theta` (*theta-class*), [56](#)  
`theta-class`, [40](#)  
`theta-class`, [56](#)  
`TIMP` (*TIMP-package*), [2](#)  
`TIMP-package`, [2](#)  
`writeAverage`, [57](#)