Package 'TIMP' documentation

of

August 31, 2007

Type Package

Title a problem solving environment for fitting superposition models

Version 1.3

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

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

Depends R (>= 2.5.0), methods, tcltk, vcd, fields, gplots, splines

Suggests gclus

Description Measurements often represent a superposition of the contributions of distinct sub-systems resolved with respect to many experimental variables (time, temperature, wavelength, pH, polarization, etc). TIMP allows parametric models for such superpositions to be fit and validated. The package has been extensively applied to modeling data arising in spectroscopy experiments.

License GPL version 2 or newer

R topics documented:

`baseIRF` *Baseline subtraction from a vector, usually representing an IRF.*

Description

Baseline subtraction from a vector, usually representing an IRF.

Usage

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

Arguments

`irfvec`	Vector to subtract a baseline from
`indexlow`	Lowest index to base the baseline estimation on
`indexhigh`	Highest index to base the baseline estimation on
`removeNeg`	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. These slots 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

`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

`nl`: 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

`mod_type`: Object of class `"character"` character string defining the model type, e.g., `"kin"` or `"spec"`

`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

`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, `weight_par = 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.

weight: Object of class "logical" TRUE when the specification in weightpar is to be applied and FALSE otherwise

weightM: Object of class "matrix" weights

weightsmooth: Object of class "list" type of smoothing to apply with weighting; not currently used

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 appling 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.

makeps: Object of class "character" specifyies the prefix of files written to postscript

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 specfied 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 constrainted, 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))`

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

`clpequ`: Object of class `"vector"` describes the parameters governing the clp equality constraints specified in `clpequspec`

`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

`prel`: Object of class `"vector"` vector of starting values for the relations described in prelspec

`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

`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.

`positivepar`: Object of class `"vector"` containing character strings of those parameter vectors to constrain to positivity, e.g., `positivepar=c("kinpar")`

`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

 # 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))
```

`examineFit` *Examines the results of a call to fitModel*

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

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

`fit-class` *Class "fit" to store the results of model fitting associated with all datasets analyzed.*

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

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

`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

list with element `toPlotter`.

`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

normal-bracket124bracket-normal

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`

`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 `dat`; function switches on this argument.
`multimodel`	Object of class `multimodel` 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 parmeters; 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`

getResid — *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.*

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

`data`	As in the `fitModel` function
`modspec`	As in the `fitModel` function
`datasetind`	As in the `fitModel` function
`modeldiffs`	As in the `fitModel` function
`opt`	As in the `fitModel` 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`

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`

`Internals` *TIMP function used internally*

Description

TIMP function used internally

Details

The functions linked to below are for direct use.

Author(s)

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

See Also

`baseIRF`,`readData`, `preProcess`,`initModel`, `fitModel`,`examineFit`, `sumKinSpecEst`

`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

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"` 1-6 determining the numerical convolution algorithm used in the case of modeling a measured IRF; if `6` then supply a reference lifetime in the slot `reftau`.

`reftau`: Object of class `"numeric"` containing a reference lifetime to be used when `convalg=6`

`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` if `type` is `"irf"`, the coherent artifact/scatter has the time profile of the IRF. if `type` is `"freeirfdisp"` the coherent artifact/scatter has a Gaussian time profile whose location and width are parameterized in the vector `coh`. if `type` is `"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. if `type` is `"seq"` a sequential exponential decay model is applied, whose parameters are contained in `coh`. if `type` is `"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.

`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 `x2`-dependence

Extends

Class `dat-class`, directly.

Author(s)

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

See Also

`dat-class`, `spec-class`

`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` that contains options applicable to all model types

Details

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

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 `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 `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 in 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.

plotcohcolspec 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

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.

`lzero fule`: 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., `1 218.800000000000011 220.099999999999994`

Extends

Class `kin-class`, directly.

Author(s)

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

See Also

`kin-class`, `spec-class`

`modifyModel` *Allows the starting values for parameters associated with a model to be updated with the values found in fitting the model.*

Description

Allows the starting values for parameters associated with a model to be updated with the values found in fitting the model. A call `model_w_new_starting_vals <- modifyModel(old_model)` will plug in the optimized parameter values the last model fit so that are the starting values in the model specification `model_w_new_starting_vals`.

Usage

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

Arguments

`model`	an object of class `dat` returned by `initModel`; 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 `.currModel@model`)
`newest`	an object of class `theta` 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 `.currTheta[[1]]`)
`exceptslots`	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 `.currModel` 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.

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 ith 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` and `specopt-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

`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`

`writeclpwerr`: 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.

`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 preffix 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

`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 respresenting 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

Author(s)

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

See Also

`kinopt-class`, `specopt-class`

`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

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

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)
```

Arguments

`data`	Object of class `dat`
`sample`	integer describing sampling interval to take in both time and `x2`; e.g., `sample=2` will sample every 2nd time and every 2nd point in `x2`.
`sample_time`	integer describing sampling interval in time; e.g., `sample_time=2` will sample every 2nd element of the time vector.
`sample_lambda`	integer describing sampling interval in `x2`; e.g., `sample_lambda=2` will sample every 2nd element in the `x2` vector.
`sel_time`	vector of length 2 describing the first and last time index of data to select; e.g., `sel_time=c(5,120)` will select data at times indexed 5-120.
`sel_lambda`	vector of length 2 describing the first and last `x2` index of data to select; e.g., `sel_lambda=c(5,120)` will select data at `x2` indexed 5-120.
`baselinetime`	a vector of form `c(timeIndexmin, timeIndexmax, lambdaIndexmin, lambdaIndexmax)`. The average of data between `x2` indexes `lambdaIndexmin` and `lambdaIndexmax` is subtracted from data with time index between `timeIndexmin` and `timeIndexmax`.
`baselinelambda`	a vector of form `c(timeIndexmin, timeIndexmax, lambdaIndexmin, lambdaIndexmax)`. The average of data between time indexes `timeIndexmin` and `timeIndexmax` is subtracted from data with `x2` index between `lambdaIndexmin` and `lambdaIndexmax`.
`scalx`	numeric by which to linearly scale the `x` axis (which often represents time), so that newx = oldx * scalx
`scalx2`	vector of length 2 by which to linearly scale the `x2` axis, so that newx2 = oldx2 * scalx2[1] + scalx2[2]
`sel_lambda_ab`	vector of length 2 describing the absolute values (e.g., wavelengths, wavenumbers, etc.) between which data should be selected. e.g., `sel_lambda_ab = c(400, 600)` will select data associated with `x2` values between 400 and 600.
`sel_time_ab`	vector of length 2 describing the absolute times between which data should be selected. e.g., `sel_time_ab = c(50, 5000)` will select data associated with time values between 50 and 5000 picoseconds.
`rm_x2`	vector of `x2` indices to remove from the data
`rm_x`	vector of `x` indices to remove from the data
`svdResid`	list returned from the `getResid` function, containing residuals to be used in data correction.
`numV`	numeric specifying how many singular vectors to use in data correction. Maximum is five.

Value

object of class `dat`.

Author(s)

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

See Also

`readData`, `getResid`

`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 may be read from file into an R object for analysis.

Usage

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

Arguments

`filenm`	This is the path to the file to read in, as a quoted string.
`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 `x2`

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

`model`	Object of class `dat`; switches on this argument.
`group`	list of vector pairs (x2 index, dataset index) for which the part of the residual vector is to be determined
`multimodel`	Object of class `multimodel`
`thetalist`	Object of class `multitheta`
`clpindepX`	Object of class `matrix` 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 `x2` index
`finished`	logical determining whether fitting is finished that triggers the storage of results
`returnX`	logical determining whether to just return the matrix `X` directly dependent on nonlinear parameters; this is used in the finite difference derivative of `X` used to get standard error estimates on the conditionally linear parameters.
`rawtheta`	numeric vector of nonlinear parameters to be optimized by `nls`; this is used in the finite difference derivative of `X` 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`

`specopt-class` *Class "specopt" stores options for fitting and plotting spectral models*

Description

Class "specopt" stores options for fitting and plotting spectral models in particular; this is a subclass of class `opt` that contains options applicable to all model types.

Details

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

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` *Makes a summary plot of spectra associated with kinetic components alongside a plot showing parameter estimates*

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 arguement `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

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

Details

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

Value

Author(s)

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

See Also

`fitModel`, `examineFit`

theta-class — *Class "theta" for storage of nonlinear parameter estimates*

Description

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

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

`parmu`: 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`

TIMP-package *a problem solving environment for fitting superposition models*

Description

Measurements often represent a superposition of the contributions of distinct sub-systems resolved with respect to many experimental variables (time, temperature, wavelength, pH, polarization, etc). A parametric model for each component may be desirable to apply to the data, but only to the evolution of components with respect to a subset of the independent variables. For instance, given time-resolved spectroscopy data, a parametric model for the time-evolution of components may be available, while a physically-inspired parametric model for the spectra of components may be difficult to formulate and interpret. Such situations give rise to a separable nonlinear parameter estimation problem, namely that of estimating the (nonlinear) parameters associated with the parametric model, while estimating parameters representing the evolution of components with respect to the independent variables to which a parametric model does not apply as conditionally linear. The partitioned variable projection algorithm is well-suited to solving such problems under the criteria of efficiency, quality of standard error estimates, and precision of parameter estimates. TIMP implements the partitioned variable projection algorithm and allows its application to fitting a wide range of models, including those for the simultaneous analysis of multiple datasets collected under different experimental conditions. The package has been extensively applied to modeling data arising in spectroscopy experiments.

Details

Package: TIMP Type: Package Title: a problem solving environment for fitting superposition models Version: 1.2 Author: Katharine M. Mullen, Ivo H. M. van Stokkum Maintainer: Katharine M. Mullen <kate@nat.vu.nl> Depends: R (>= 2.5.0), methods, tcltk, vcd, fields, gplots, splines Suggests: gclus License: GPL version 2 or newer

Author(s)

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

References

See `http://www.nat.vu.nl/~kate/TIMP/` for further documentation.

Index