Package 'Data2LD'

August 6, 2020

Type Package

Title Functional Data Analysis with Linear Differential Equations

Version 3.2.1

Date 2020-7-5

Maintainer James Ramsay <James.Ramsay@mcgill.ca>

Depends R (>= 3.5), fda, splines, graphics, deSolve, knitr, rmarkdown

Description Provides methods for using differential equations as modelling objects as described in J. Ramsay G. and Hooker (2017,ISBN 978-1-4939-7188-6) Dynamic Data Analysis, New York: Springer. Data sets and script files for analyzing many of the examples in this book are included. This version corrects bugs and adds two new demos, CruiseControl and monotoneSmoothDemo. This version incorporates a new system for indexing coefficient list objects. The step of setting up a set of coefficient list objects has been eliminated and the functions make.Coef() and coefCheck have been dropped. 'Matlab' versions of the code and sample analyses are available by ftp from the website. There you find a set of .zip files containing the functions and sample analyses, as well as two .txt files giving instructions for installation and some additional information.

License GPL (>= 2)

URL http://www.functionaldata.org

LazyData true

VignetteBuilder knitr, rmarkdown

NeedsCompilation yes

Author James Ramsay [aut, cre]

Repository CRAN

Date/Publication 2020-08-05 23:22:10 UTC

R topics documented:

`Data2LD2-package` *A short title line describing what the package does*

Description

A more detailed description of what the package does. A length of about one to five lines is recommended.

Details

This section should provide a more detailed overview of how to use the package, including the most important functions.

Author(s)

James Ramsay, email optional.

Maintainer: James Ramsay <your@email.com>

References

This optional section can contain literature or other references for background information.

See Also

Optional links to other man pages

Examples

```
## Not run:
   ## Optional simple examples of the most important functions
   ## These can be in \dontrun{} and \donttest{} blocks.

## End(Not run)
```

`Atensorfn` *Compute the four-way tensors corresponding to pairs of forcing terms in systems of linear differential equations.*

Description

Forcing terms consist of the product of a coefficient function that must be estimated and a known forcing function that does not. When both of these functions in a forcing term are defined by B-splines, the product involves an inner product of two B-spline basis systems. When a product of two forcing terms are required, as is usual in the the use of the Data2LD package, a great improvement in efficiency of computation can be acheived by an initial computation of the four-way array or tensor resulting by taking the inner products of all possible quadruples of the B-spline basis functions. Memoization is the process of storing these tensors in memory so that they do not need to be re-computed each time the Data2LD.R function is called. Memoization is taken care of automatically in the code using the R.cache package, and is activated the first time a new `modelList` object is encountered. Normally the user does not have to worry about the memorization procedure. It is possible, however, to manually re-activate the memoization. However, users may also want to construct these four-way tensors manually for debugging and other purposes, and this function is made available for this reason.

Usage

```
Atensorfn(modelList)
```

Arguments

`modelList`	A list object containing the specification of a Data2LD model. Each member of this list contains a list object that defines a single linear differential equation.

Details

Variable specifications can be set by an invocation of function `make.Variable` for each linear differential equation in the system. Functions `Atensorfn`, `BAtensorfn` and `Btensorfn` will not normally be required by users since they are invoked automatically with the function `checkModel`, which is required to be invoked before the analysis of the data.

Value

A list object of length equal to the number of variables in the system. Each of the members of this list is a two-dimensional list object, and the members of this list are the four-way tensors set up as vectors for each of the possible pairs of forcing terms. All levels of the this list structure are designed to be accessed numerically by a call like `myAtensor[[ivar]][[ntermj]][[ntermk]]`.

References

J. O. Ramsay and G. Hooker (2017) *Dynamic Data Analysis*. Springer.

`BAtensorfn` *Compute the four-way tensors corresponding to pairs of terms in the homogeneous portions of systems of linear differential equations.*

Description

A linear differential equation involves a set of terms consisting of the product of a coefficient function that must be estimated and a derivative (including a derivative order 0) of one of the variables in the system. We call this portion of the equation the homogeneous part of the equation, as opposed to the part consisting of forcing terms involving known forcing functions. When both of the functions in either a homogeneous term or a forcing term are defined by B-splines, the product involves an inner product of two B-spline basis systems. When a product of a homogeneous term and a forcing term are required, as is usual in the the use of the Data2LD package, a great improvement in efficiency of computation can be acheived by an initial computation of the four-way array or tensor resulting by taking the inner products of all possible quadruples of of the B-spline basis functions involved. Memoization is the process of storing these tensors in memory so that they do not need to be re-computed each time the Data2LD.R function is called. Memoization is taken care of automatically in the code using the R.cache package, and is activated the first time a new `modelList` object is encountered. Normally the user does not have to worry about the memorization procedure. It is possible, however, to manually re-activate the memoization. However, users may also want to construct these four-way tensors manually for debugging and other purposes, and this function is made available for this reason.

Usage

```
BAtensorfn(XbasisList, modelList)
```

Arguments

`XbasisList`	A list object of length equal to the number of equations in the system. Each member of this list is a functional basis object used to approximate the values of the corresponding variable.
`modelList`	A list object containing the specification of a Data2LD model. Each member of this list contains a list object that defines a single linear differential equation.

Details

Variable specifications can be set by an invocation of function `make.Variable` for each linear differential equation in the system. Functions `Atensorfn`, `BAtensorfn` and `Btensorfn` will not normally be required by users since they are invoked automatically with the function `checkModel`, which is required to be invoked before the analysis of the data.

Value

A list object of length equal to the number of variables in the system. Each of the members of this list is a two-dimensional list object, and the members of this list are the four-way tensors set up as vectors for each of the possible pairs of forcing terms. All levels of the this list structure are designed to be accessed numerically by a call like `myBAtensor[[ivar]][[ntermj]][[ntermk]]`.

References

J. O. Ramsay and G. Hooker (2017) *Dynamic Data Analysis*. Springer.

`Btensorfn` *Compute the four-way tensors corresponding to pairs of terms in the homogeneous portions of systems of linear differential equations.*

Description

A linear differential equation involves a set of terms consisting of the product of a coefficient function that must be estimated and a derivative (including a derivative order 0) of one of the variables in the system. We call this portion of the equation the homogeneous part of the equation, as opposed to the part consisting of forcing terms involving known forcing functions. When both of the functions in a homogeneous term are defined by B-splines, the product involves an inner product of two B-spline basis systems. When a product of two homogeneous terms are required, as is usual in the the use of the Data2LD package, a great improvement in efficiency of computation can be acheived by an initial computation of the four-way array or tensor resulting by taking the inner products of all possible quadruples of the B-spline basis functions. Memoization is the process of storing these tensors in memory so that they do not need to be re-computed each time the Data2LD.R function is called. Memoization is taken care of automatically in the code using the R.cache package, and is activated the first time a new `modelList` object is encountered. Normally the user does not have to worry about the memorization procedure. It is possible, however, to manually re-activate the memoization. However, users may also want to construct these four-way tensors manually for debugging and other purposes, and this function is made available for this reason.

Usage

```
Btensorfn(XbasisList, modelList)
```

Arguments

`XbasisList`	A list object of length equal to the number of equations in the system. Each member of this list is a functional basis object used to approximate the values of the corresponding variable.
`modelList`	A list object containing the specification of a Data2LD model. Each member of this list contains a list object that defines a single linear differential equation.

Details

Variable specifications can be set by an invocation of function `make.Variable` for each linear differential equation in the system. Functions `Atensorfn`, `BAtensorfn` and `Btensorfn` will not normally be required by users since they are invoked automatically with the function `checkModel`, which is required to be invoked before the analysis of the data.

Value

A list object of length equal to the number of variables in the system. Each of the members of this list is a two-dimensional list object, and the members of this list are the four-way tensors set up as vectors for each of the possible pairs of forcing terms. All levels of the this list structure are designed to be accessed numerically by a call like `myBtensor[[ivar]][[ntermj]][[ntermk]]`.

References

J. O. Ramsay and G. Hooker (2017) *Dynamic Data Analysis*. Springer.

`CanadianWeather` *Canadian average annual weather cycle*

Description

Daily temperature and precipitation at 35 different locations in Canada averaged over 1960 to 1994.

Usage

```
CanadianWeather
```

Format

'CanadianWeather' and 'daily' are lists containing essentially the same data. 'CanadianWeather' may be preferred for most purposes; 'daily' is included primarily for compatability with scripts written before the other format became available and for compatability with the Matlab 'fda' code. 'CanadianWeather' contains the following objects:

- dailyAv: a three dimensional array c(365, 35, 3) summarizing data collected at 35 different weather stations in Canada on the following: [,,1] = [,, 'Temperature.C']: average daily temperature for each day of the year [,,2] = [,, 'Precipitation.mm']: average daily rainfall for each day of the year rounded to 0.1 mm. [,,3] = [,, 'log10precip']: base 10 logarithm of Precipitation.mm after first replacing 27 zeros by 0.05 mm (Ramsay and Silverman 2005, p. 248).
- place: Names of the 35 different weather stations in Canada whose data are summarized in 'dailyAv'. These names vary between 6 and 11 characters in length. By contrast, daily[["place"]] which are all 11 characters, with names having fewer characters being extended with trailing blanks.
- province: names of the Canadian province containing each place
- coordinates: a numeric matrix giving 'N.latitude' and 'W.longitude' for each place.

- region: Which of 4 climate zones contain each place: Atlantic, Pacific, Continental, Arctic.
- monthlyTemp: A matrix of dimensions (12, 35) giving the average temperature in degrees Celcius for each month of the year.
- monthlyPrecip: A matrix of dimensions (12, 35) giving the average daily precipitation in milimeters for each month of the year.
- geogindex: Order the weather stations from East to West to North

Source

Ramsay, James O., and Silverman, Bernard W. (2005), *Functional Data Analysis, 2nd ed.*, Springer, New York. Ramsay, James O., and Silverman, Bernard W. (2002), *Applied Functional Data Analysis*, Springer, New York

Examples

```
##
## 1.  Plot (latitude & longitude) of stations by region
##
with(CanadianWeather, plot(-coordinates[, 2], coordinates[, 1], type='n',
                           xlab="West Longitude", ylab="North Latitude",
                           axes=FALSE) )
Wlon <- pretty(CanadianWeather$coordinates[, 2])
axis(1, -Wlon, Wlon)
axis(2)

rgns <- 1:4
names(rgns) <- c('Arctic', 'Atlantic', 'Continental', 'Pacific')
Rgns <- rgns[CanadianWeather$region]
with(CanadianWeather, points(-coordinates[, 2], coordinates[, 1],
                             col=Rgns, pch=Rgns) )
legend('topright', legend=names(rgns), col=rgns, pch=rgns)

##
## 2.  Plot dailyAv[, 'Temperature.C'] for 4 stations
##
data(CanadianWeather)
# Expand the left margin to allow space for place names
op <- par(mar=c(5, 4, 4, 5)+.1)
# Plot
stations <- c("Pr. Rupert", "Montreal", "Edmonton", "Resolute")
matplot(day.5, CanadianWeather$dailyAv[, stations, "Temperature.C"],
        type="l", axes=FALSE, xlab="", ylab="Mean Temperature (deg C)")
axis(2, las=1)
# Label the horizontal axis with the month names
axis(1, monthBegin.5, labels=FALSE)
axis(1, monthEnd.5, labels=FALSE)
axis(1, monthMid, monthLetters, tick=FALSE)
# Add the monthly averages
matpoints(monthMid, CanadianWeather$monthlyTemp[, stations])
# Add the names of the weather stations
mtext(stations, side=4,
```

```
      at=CanadianWeather$dailyAv[365, stations, "Temperature.C"],
     las=1)
# clean up
par(op)
```

---

checkModel *Check linear differential equation specifications.*

Description

A system of linear differential equations is defined by a model list of length equal to the number of variables in the system. Each variable is defined an invocation of function `make.Variable`, and then assigned to the corresponding position in the model list. so that each member of this list defines a single linear differential equation.

Within each equation there are typically one or more terms involving a coefficient function multiplying either a derivative of one of the variables, or multiplying a forcing function.

This function checks that all linear differential equation specifications conform to what is required. A list of errors is displayed if there are problems. Default values are supplied where needed. New named members are also supplied.

In addition, function `checkModel` also assigns position(s) in the master parameter vector corresponding to coefficient functions in the system. This master parameter vector is optimized by function `Data2LD.opt`.

Finally, `checkModel` also displays a detailed summary of the linear differential system if the argument `summarywrd` is TRUE. It is easy to make mistakes in setting up lists of variable specifications, and it is ESSENTIAL to invoke this function prior to invoking any function using this list.

Usage

```
checkModel(XbasisList, modelList, summarywrd=TRUE)
```

Arguments

`XbasisList`	A list object containing the functional basis objects for representing the variables as functional data objects.
`modelList`	A list object containing the specification of a Data2LD model. Each member of this list contains a list object that defines a single linear differential equation.
`summarywrd`	If TRUE, display a report on the structure of the dynamic system.

Details

A list object containing:

- model list object: A list object with length equal to the number of variables in the system. It contains variable specifications that can either be made manually, or can set up by an invocation of function `make.Variable`.

- nparam: The total number of parameters in the master parameter vector. This master vector contains all of the parameters in the system, including any that are fixed as well as those that are to be optimized.

If certain values are not supplied, default values are assigned.

Two new named members are also supplied for each variable. Member `nallXterm` is the number of terms in the right side homogeneous portion of the equation, which involve a derivative of a variable in the system multiplied by a coefficient function. Member `nallFterm` is the number of forcing terms in the equation. These two new members are essential for function `Data2LD`, and, although `checkModel` is also invoked inside `Data2LD`, an inovking of `checkModel` is essential prior to beginning the data analysis.

Value

A named list object with the same structure as the argument along with added default values and members `nallXterm` and `nallFterm`.

References

J. O. Ramsay and G. Hooker (2017) *Dynamic Data Analysis*. Springer.

See Also

`make.Fterm`, `make.Xterm`, `make.Variable`

Examples

```
#  For examples of the use of this function,
#  see these examples in the description of the function \code{make.variable}.
#
#  Example 1:  The refinery data
#
#
#  Example 2:  The X coordinate of the fda script data
#
#
#  Example 3:  The X coordinate of the fda script data
#
#
#  Example 4  Average temperature for Montreal
#
```

`Data2LD` *Evaluate the mean of squared errors data-fitting criterion and its gradient for a system of linear differential equations.*

Description

Data2LD ... stands for "Data to Linear Dynamics." The function approximates the data in argument YLIST by one or smooth functions $latex$. This approximation is defined by a set of *d* linear differential or algebraic equations defined by a set of parameters some of which require estimation from the data. The approximation minimizes the mean of squared residuals, expressed as follows:

$$latex$$

where:

- $latex$ indexes equations in a system differential and algebraic equations.
- $latex$ indexes times of observation of a variable.
- $latex$ indexes replications of observations.
- $latex$ is a positive fixed real number weighting the contribution of a variable to the fitting criterion.

Only a subset of the variables may be observed, so that not all values of index i are actually used. The number and location of times of observation $latex$ can vary from one observed variable to another. The fitting functions $latex$ in turn, here assumed to be defined over the interval *[0,T]*, are defined by basis function expansions:

$$x_i(t_j) = \sum_{k}^{K_i} c_{ik}(\theta|\rho)\phi_{ik}(t_j)$$

where $latex$ is the *k*th function in a system of $latex$ basis functions used to approximate the ith variable, and $latex$ is the coefficientin the expansion of $latex$. The number of $latex$ basis functions and the type of basis function system can vary from one variable to another. This information is contained in argument `XbasisList` described below. The coefficients $latex$ defining the smoothing functions are functions of the unknown parameters in vector $latex$ that define the differential and algebraic equations and that require estimation from the data. The smoothing parameter $latex$ is a value in the interval $[0, 1)$. The coefficient functions $latex$ minimize the inner least squares criterion, expressed here for simplicity for a single variable or equation:

$$latex$$

Linear differential operator $latex$ is a linear differential or algebraic equation rewritten as an operator by subtracting the right side of the equation from the left, so that a solution x of the equation satisfies $latex$. Each L in a system of equation depends on one or more parameters that need to be estimated and are contained in parameter vector $latex$. The linear differential equation is of the form

$$latex$$

where each coefficient is expanded in terms of its own number of B-spline basis functions:

$$latex$$

$$latex$$

As smoothing parameter $latex$ increases toward its upper limit of 1, the second roughness penalty term in J is more and more emphasized and the fit to the data less and less emphasized, so that

$latex$ is required to approach a solution to its respective equation. The highest order of derivative can vary from one equation to another, and in particular can be larger than 1. Each right side may contain contributions from all variables in the equation. Morever, these contributions may be from all permissible derivatives of each equation, wher "permissible" means up to one less than the highest order in the variable. In addition, each equation may be forced by a number of forcing terms that can vary from equation to equation. This version approximates the integrals in the penalty terms by using inprod_basis to compute the cross-product matrices for the $latex$-coefficient basis functions and the corresponding derivative of the x-basis functions,and the cross-product matrices for the $latex$-coefficients and the corresponding U functions. These are computed upon the first call to D2LD, and then retained for subsequent calls by using the persistent command. This technique for speeding up computaton is called memoization. The structure of the model is defined in list `modelList`, which is described below. This version disassociates coefficient functions from equation definitions to allow some coefficients to be used repeatedly and for both homogeneous and forcing terms. It requires an extra argument COEFLIST that contains the coefficients and the position of their coefficient vectors in vector $latex$.

Usage

```
Data2LD(yList, XbasisList, modelList, rhoVec = 0.5*rep(1,nvar),
        summary=TRUE)
```

Arguments

`yList`	A list of length NVAR. Each list contains a list object with these fields: • argvals: a vector of length $latex$ of observation times • y: a matrix with $latex$ rows and *N* columns. The number of columns must be the same for all variables, except that, if a list is empty, that variable is taken to be not observed.
`XbasisList`	A list array of length d. Each member contains in turn a functional data object or a functional basis object.
`modelList`	A list each member of which specifies one of linear differential equations in the system. These can be constructed using function `make.Variable`. See the help file for that function for further details.
`rhoVec`	A real number in the half-open interval $[0, T)]$. Its value determines the relative emphasis on fitting the data versus being a solution of the differential equation. The smaller the value, the more the emphasis on data fitting.
`summary`	A logical constant. If TRUE, a variety of summeary measures are computed, including degrees of freedom (`df`), the generalized cross-validation index (`gcv`) and an estimate of the sampling variance-covariance matrix. However, these can in some circumstances be computationally demanding, and `summary=FALSE` limits the output to the fitting criterion, gradient and hessian matrix, which can greatly speed up computation. Function `Data2LD` is called in this way within function `Data2LD.opt`, for example.

Value

A named list object containing these results of the analysis:

MSE	The weighted mean squared errors computed over the variables with data.
DpMSE	The gradient of the objective function MSE with respect to the estimated parameters.
D2ppMSE	A square symmetric matrx of that contains the second partial derivatives of the objective function with respect to the parameter being estimated.
XfdParList	A list of length *d* containing functional parameter objects for the estimated functions $latex$.
df	An equivalent degrees of freedom value $latex$ where *YM* is the matrix y2cMap described below.
gcv	The generalized cross-validation measure GCV. The value of $latex$ corresponding to the minimim of GCV across values of smoothing parameter $latex$ is often chose for an automatic data-driven level of smoothing.
ISE	The sum across variables of the integrated squared value of the differential operator. This value multiplied by $latex$ and divided by *T*, the width of the domain, is the second term the objective function.
Var.theta	A square symmetric matrix containing estimates of the sampling variances and covariances for the estimated parameter values.
Rmat	A square symmetric matrix associated with the homogeneous portions of the equations that has many useful applications, including constructions of more compact and meaningful basis function expansions for the $latex$'s.
Smat	A matrix containing information on the covariation of the basis functions and the forcing functions.

References

J. O. Ramsay and G. Hooker (2017) *Dynamic Data Analysis*. Springer.

See Also

checkModel, make.Variable, make.Xterm, make.Fterm checkModel

Examples

```
#
#  Example 1:  The refinery data
#
N <- 194
TimeData <- RefineryData[,1]
TrayData <- RefineryData[,2]
ValvData <- RefineryData[,3]
#  Define the List array containing the tray data
TrayList <- list(argvals=RefineryData[,1], y=RefineryData[,2])
TrayDataList  <- vector("list",1)
TrayDataList[[1]] <- TrayList
# construct the basis object for tray variable
#  Order 5 spline basis with four knots at the 67
#   to allow discontinuity in the first derivative
#  and 15 knots between 67 and 193
```

```
Traynorder <- 5
Traybreaks <- c(0, rep(67,3), seq(67, 193, len=15))
Traynbasis <- 22
TrayBasis  <- create.bspline.basis(c(0,193), Traynbasis, Traynorder,
                                   Traybreaks)
#  Set up the basis list for the tray variable
TrayBasisList    <- vector("list",1)
TrayBasisList[[1]] <- TrayBasis
#  Both alpha and beta coefficient functions will be constant.
#  Define the constant basis
conbasis <- create.constant.basis(c(0,193))
betafdPar  <- fdPar(conbasis)
alphafdPar <- fdPar(conbasis)
#  Construct a step basis (order 1) and valve level
Valvbreaks <- c(0,67,193)
Valvnbasis <- 2
Valvnorder <- 1
Valvbasis  <- create.bspline.basis(c(0,193), Valvnbasis, Valvnorder, Valvbreaks)
#  smooth the valve data
Valvfd <- smooth.basis(TimeData, ValvData, Valvbasis)$fd
#  Set up the model list for the tray variable
#  Define single homogeneous term
#  Xterm Fields:    funobj     parvec  estimate  variable deriv. factor
XTerm <- make.Xterm(betafdPar, 0.04,  TRUE,     1,       0,     -1)
XList <- vector("list", 1)
XList[[1]] <- XTerm
#  Define the single forcing term
#  Fterm Fields:    funobj      parvec  estimate  Ufd      factor
FTerm <- make.Fterm(alphafdPar, 1.0,    TRUE,     Valvfd,  1)
FList <- vector("list", 1)
FList[[1]] <- FTerm
#  Define the single differential equation in the model
TrayVariable <- make.Variable(XList=XList, FList=FList,
                              name="Tray level", order=1)
#  Set up the model List array
TrayModelList <- vector("list",1)
TrayModelList[[1]] <- TrayVariable
#  Run a check on TrayVariableList and make the modelList object
checkModelList <- checkModel(TrayBasisList, TrayModelList, summarywrd=TRUE)
TrayModelList  <- checkModelList$model
nparam         <- checkModelList$nparam
#  Evaluate the fit to the data given the initial parameter estimates (0 and 0)
#  This also initializes the four-way tensors so that they are not re-computed
#  for subsquent analyses.
rhoVec <- 0.5
Data2LDList <- Data2LD(TrayDataList, TrayBasisList, TrayModelList, rhoVec)
MSE  <- Data2LDList$MSE    # Mean squared error for fit to data
DMSE <- Data2LDList$DpMSE  #  gradient with respect to parameter values
#
#  Example 2:  The head impact data
#
ImpactTime <- HeadImpactData[,2]  #  time in milliseconds
ImpactData <- HeadImpactData[,3]*100  #  acceleration in millimeters/millisecond^2
```

```
ImpactRng  <- c(0,60) # Define range time for estimated model
#  Define the List array containing the data
ImpactDataList1 <- list(argvals=ImpactTime, y=ImpactData)
ImpactDataList <- vector("list",1)
ImpactDataList[[1]] <- ImpactDataList1
# Define a unit pulse function located at times 14-15
Pulsebasis <- create.bspline.basis(ImpactRng, 3, 1, c(0,14,15,60))
Pulsefd    <- fd(matrix(c(0,1,0),3,1),Pulsebasis)
#  Construct basis for output x(t),
#  multiple knots at times 14 and 15.
#  Order 6 spline basis, with three knots at the impact point and
#  three knots at impact + delta to permit discontinuous first
#  derivatives at these points
knots     <- c(0,14,14,14,15,15,seq(15,60,len=11))
norder    <- 6
nbasis    <- 21
ImpactBasis <- create.bspline.basis(ImpactRng,nbasis,norder,knots)
#  plot the basis
ImpactTimefine <- seq(0,60,len=201)
ImpactBasisfine <- eval.basis(ImpactTimefine, ImpactBasis)
#  set up basis list
ImpactBasisList <- vector("list",1)
ImpactBasisList[[1]] <- ImpactBasis
# Set up the constant basis, make three coefficients and check them
conbasis <- create.constant.basis(ImpactRng)
# Define the terms in the second order linear equation
# Define the two terms in the homogeneous part of the equation
#  Xterm Fields:     funobj     parvec  estimate  variable deriv. factor
Xterm1 <- make.Xterm(conbasis,  1,      TRUE,     1,       0,     -1)
Xterm2 <- make.Xterm(conbasis,  1,      TRUE,     1,       1,     -1)
# Set up the XList vector of length two
XList <- vector("list",2)
XList[[1]] <- Xterm1
XList[[2]] <- Xterm2
# Define the forcing term
#  Fterm Fields:    funobj    parvec  estimate  Ufd      factor
Fterm <- make.Fterm(conbasis, 1.0,   TRUE,     Pulsefd, 1)
# Set up the forcing term list of length one
FList      <- vector("list",1)
FList[[1]] <- Fterm
#  Define the single differential equation in the model
ImpactVariable <- make.Variable(name="Impact", order=2, XList=XList, FList=FList)
#  Define list of length one containing the equation definition
ImpactModelList=vector("list",1)
ImpactModelList[[1]] <- ImpactVariable
#  Check the object for internal consistency and
#  set up the model object
#  Run a check on TrayVariableList and make the modelList object
checkModelList  <- checkModel(ImpactBasisList, ImpactModelList, summarywrd=TRUE)
ImpactModelList <- checkModelList$model
nparam          <- checkModelList$nparam
# An evaluation of the criterion at the initial values
rhoVec <- 0.5  #  light smoothing
```

```
Data2LDResult <- Data2LD(ImpactDataList, ImpactBasisList, ImpactModelList, rhoVec)
MSE        <- Data2LDResult$MSE        # Mean squared error for fit to data
DpMSE      <- Data2LDResult$DpMSE      #  gradient with respect to parameter values
D2ppMSE    <- Data2LDResult$D2ppMSE    #  Hessian matrix
```

`Data2LD.opt` *Optimize the mean of squared errors data-fitting criterion for a system of linear differential equations.*

Description

This function calls function `Data2LD` in each iteration of a quasi-Newton type optimization algorithm, allowing for a set of linear restrictions on the parameter vector.

Usage

```
Data2LD.opt(yList, XbasisList, modelList, rhoMat,
         convcrit = 1e-4, iterlim = 20, dbglev = 1, parMap = diag(rep(1,nparam)))
```

Arguments

`yList`	A list of length NVAR. Each list contains in turn a struct object with fields: **argvals** is a vector of length n_i of observation times **y** is a matrix with n_i rows and *N* columns. The number of columns must be the same for all variables, except that, if a list is empty, that variable is taken to be not observed.
`XbasisList`	A list array of length *d*. Each member contains in turn a functional data object or a functional basis object.
`modelList`	A list each member of which specifies one of linear differential equations in the system. These can be constructed using function `make.Variable`. See the help file for that function for further details.
`rhoMat`	A matrix with the number of rows equal to the number of values of the smoothing parameter per variable to be used in the optimization, and number of columns equal to the number of variables. Each entry in the matrix is a value rho in the interval [0,1) so that 0 <= rho < 1. For each variable ivar in the system of equations and each optimization iopt, the data are weighted by rho(iopt,ivar) and the roughness penalty by 1-rho(iopt,ivar). It is expected that the values of rho within each column will be in ascending order, and the estimated parameters for each row are passed along as initial values to be used for the optimization defined by the values of rho in the next row.
`convcrit`	A one- or two-part convergence criterion. The first part is for testing the convergence of the criterion values, and the second part, if present in a vector of length 2, is for testing the norm of the gradient.
`iterlim`	Maximum number of iterations allowed.

dbglev
: Control the output on each iteration. If 0, no output is displayed. If 1, the criterion and gradient norm at each iteration. If 2, the stepsize, slope and criterion at each step in the line search. if 3, displays, in addition to 2, a plot of the criterion values and slope after each line search followed by a pause.

parMap
: A rectangular matrix with number of rows equal to the number of parameters to be estimated, and number of columns equal to the number of parameters less the number of linear constraints on the estimated parameters. The columns of `parMap` must be orthnormal so that the crossproduct is an identity matrix. The crossproduct of `t(parMap)` and `THETA` maps unconstrained parameters and the corresponding gradient into constrained parameter space. `parMap` will usually be set up using the full QR decomposition of a linear constraint coefficient matrix `t(A)` where the constraints are of the form A P = B, A and B being known matrices. An example of such a constraint that arises often is one where two estimated coefficients are constrained to be equal. For example, if a variable X involved in an equation the form a(x - x.0), where x.0 is a fixed set point or defined target level for variable X, then this would be set up as a.1 x + a.2 x.0, where coefficients a.1 and a.2 are constrained to be equal in magnitude but opposite in sign, or a.1 + a.2 = 0.

Value

A named list object containing these results of the analysis:\

- theta.opt: A vector containing the optized parameter values.
- modelList.opt: A list object defining the optimized linear equation system. This may be displayed using function `printModel`.

References

J. O. Ramsay and G. Hooker (2017) *Dynamic Data Analysis*. Springer.

See Also

Data2LD

Examples

```
#
#  Example 1:  The refinery data
#
N <- 194
TimeData <- RefineryData[,1]
TrayData <- RefineryData[,2]
ValvData <- RefineryData[,3]
#  Define the List array containing the tray data
TrayList <- list(argvals=RefineryData[,1], y=RefineryData[,2])
TrayDataList  <- vector("list",1)
TrayDataList[[1]] <- TrayList
# construct the basis object for tray variable
#  Order 5 spline basis with four knots at the 67
```

```
#   to allow discontinuity in the first derivative
#  and 15 knots between 67 and 193
Traynorder <- 5
Traybreaks <- c(0, rep(67,3), seq(67, 193, len=15))
Traynbasis <- 22
TrayBasis  <- create.bspline.basis(c(0,193), Traynbasis, Traynorder,
                                   Traybreaks)
#  Set up the basis list for the tray variable
TrayBasisList    <- vector("list",1)
TrayBasisList[[1]] <- TrayBasis
#  Both alpha and beta coefficient functions will be constant.
#  Define the constant basis
conbasis <- create.constant.basis(c(0,193))
betafdPar  <- fdPar(conbasis)
alphafdPar <- fdPar(conbasis)
#  Construct a step basis (order 1) and valve level
Valvbreaks <- c(0,67,193)
Valvnbasis <- 2
Valvnorder <- 1
Valvbasis  <- create.bspline.basis(c(0,193), Valvnbasis, Valvnorder, Valvbreaks)
#  smooth the valve data
Valvfd <- smooth.basis(TimeData, ValvData, Valvbasis)$fd
#  Set up the model list for the tray variable
#  Define single homogeneous term
#  Xterm Fields:    funobj     parvec  estimate  variable deriv. factor
XTerm <- make.Xterm(betafdPar, 0.04,  TRUE,     1,       0,     -1)
XList <- vector("list", 1)
XList[[1]] <- XTerm
#  Define the single forcing term
#  Fterm Fields:    funobj      parvec  estimate  Ufd      factor
FTerm <- make.Fterm(alphafdPar, 1.0,    TRUE,     Valvfd,  1)
FList <- vector("list", 1)
FList[[1]] <- FTerm
#  Define the single differential equation in the model
TrayVariable <- make.Variable(XList=XList, FList=FList,
                              name="Tray level", order=1)
#  Set up the model List array
TrayModelList <- vector("list",1)
TrayModelList[[1]] <- TrayVariable
#  Run a check on TrayVariableList and make the modelList object
checkModelList <- checkModel(TrayBasisList, TrayModelList, summarywrd=TRUE)
TrayModelList  <- checkModelList$model
nparam         <- checkModelList$nparam
#  Evaluate the fit to the data given the initial parameter estimates (0 and 0)
#  This also initializes the four-way tensors so that they are not re-computed
#  for subsquent analyses.
rhoVec <- 0.5
Data2LDList <- Data2LD(TrayDataList, TrayBasisList, TrayModelList, rhoVec)
MSE  <- Data2LDList$MSE    # Mean squared error for fit to data
DMSE <- Data2LDList$DpMSE  #  gradient with respect to parameter values
#  These parameters control the optimization.
dbglev   <-  1    #  debugging level
iterlim  <- 50    #  maximum number of iterations
```

```
convrg   <- c(1e-8, 1e-4)  #  convergence criterion
#  This sets up an increasing sequence of rho values
gammavec <- 0:7
rhoMat   <- as.matrix(exp(gammavec)/(1+exp(gammavec)),length(gammavec),1)
nrho     <- length(rhoMat)
dfesave  <- matrix(0,nrho,1)
gcvsave  <- matrix(0,nrho,1)
MSEsave  <- matrix(0,nrho,1)
thesave  <- matrix(0,nrho,nparam)
#  This initializes the list object containing coefficient estimates
TrayModelList.opt <- TrayModelList
# Loop through values of rho.
# for test purposes, only the first value rho = 0.5 is used here
for (irho in 1:1) {
  rhoi <- rhoMat[irho,]
  print(paste("rho <- ",round(rhoi,5)))
  Data2LDResult <-
    Data2LD.opt(TrayDataList, TrayBasisList, TrayModelList.opt,
                rhoi, convrg, iterlim, dbglev)
  theta.opti <- Data2LDResult$thetastore
  TrayModelList.opt <- modelVec2List(TrayModelList.opt, theta.opti)
  Data2LDList <- Data2LD(TrayDataList, TrayBasisList, TrayModelList.opt, rhoi)
  MSE       <- Data2LDList$MSE
  df        <- Data2LDList$df
  gcv       <- Data2LDList$gcv
  ISE       <- Data2LDList$ISE
  Var.theta <- Data2LDList$Var.theta
  thesave[irho,] <- theta.opti
  dfesave[irho]  <- df
  gcvsave[irho]  <- gcv
  MSEsave[irho]  <- MSE
}
```

`fdascript` *Positions of the tip of a pen in centimeters while writing in cursive script the letters "fda" twenty times.*

Description

The writing was on a horizontal surface with the position of the pen recorded 400 times per second with a background noise level of about 0.05 centimeters.

Usage

```
fdascript
```

Format

An array object `fdascript` with dimensions 1401, 20, and 2. The first dimension is the number of observations in each record, the second is the records, and the third dimension is for the X and Y coordinate.

Details

The times are in seconds over an interval of 2.3 seconds, but in the analysis seconds are converted to "centiseconds" by multiplication by 100 in order to ensure numerical accuracy in the calculations.

Source

The data were published and analyzed in the 1997 and 2005 editions of the book by J. O. Ramsay and B. W. Silverman, Functional Data Analysis, and made available in the R package fda.

Examples

```
#  load the first record
fdascriptX <- fdascript[,1,1]
fdascriptY <- fdascript[,1,2]
#  Define the observation times in 100ths of a second
centisec <- seq(0,2.3,len=1401)*100
#  plot the script
plot(fdascriptX, fdascriptY, type="l")
```

`fun.Dexplinear` *A function to evaluate the derivatives with respect to the coefficients of the exponential of a functional data object.*

Description

Coefficient functions in linear differential systems are often required to be strictly positive. This can be achieved by defining the coefficient function to be the exponential of a functional data object. This function evaluates the partial derivatives of the coefficient function with respect to the coefficients in the functional data object. It is used as the member `fun.Dfd` of a user-defined coefficient function.

Usage

```
fun.Dexplinear(tvec, bvec, Bbasisobj)
```

Arguments

`tvec`	A vector of times at which the function is to be evaluated.
`bvec`	A vector or a single column of coefficient values for the functional data object.
`Bbasisobj`	A functional basis object.

Details

The length of argument `bvec` must match the number of basis functions in argument `Bbasisobj`.

Value

A matrix with the number of rows equal to the length of argument `tvec` and number of columns equal to the number of basis functions. This matrix contains the partial derivatives with respect to the coefficients defining the functional data object evaluated at the time values in argument `tvec`.

References

J. O. Ramsay and G. Hooker (2017) *Dynamic Data Analysis*. Springer.

See Also

`fun.explinear`

`fun.explinear` *A function to evaluate the exponential of a functional data object.*

Description

Coefficient functions in linear differential systems are often required to be strictly positive. This can be achieved by defining the coefficient function to be the exponential of a functional data object, and using this as the user-defined coefficient function `fun$fd` that evaluates the coefficient.

Usage

```
fun.explinear(tvec, bvec, Bbasisobj)
```

Arguments

`tvec`	A vector of times at which the function is to be evaluated.
`bvec`	A vector or a single column of coefficient values for the functional data object.
`Bbasisobj`	A functional basis object.

Details

The length of argument `bvec` must match the number of basis functions in argument `Bbasisobj`.

Value

A numeric vector of the same length as that of argument `tvec` containing the exponentials of the values of the functional basis object.

References

J. O. Ramsay and G. Hooker (2017) *Dynamic Data Analysis*. Springer.

See Also

`fun.explinear`

growth *Berkeley Growth Study data*

Description

A list containing the heights of 39 boys and 54 girls from age 1 to 18 and the ages at which they were collected.

Format

This list contains the following components:

hgtm a 31 by 39 numeric matrix giving the heights in centimeters of 39 boys at 31 ages.

hgtf a 31 by 54 numeric matrix giving the heights in centimeters of 54 girls at 31 ages.

age a numeric vector of length 31 giving the ages at which the heights were measured.

Details

The ages are not equally spaced.

Source

Ramsay, James O., and Silverman, Bernard W. (2006), *Functional Data Analysis, 2nd ed.*, Springer, New York.

Ramsay, James O., and Silverman, Bernard W. (2002), *Applied Functional Data Analysis*, Springer, New York, ch. 6.

Tuddenham, R. D., and Snyder, M. M. (1954) "Physical growth of California boys and girls from birth to age 18", *University of California Publications in Child Development*, 1, 183-364.

Examples

```
with(growth, matplot(age, hgtf[, 1:10], type="b"))
```

HeadImpactData *Acceleration of brain tissue before and after a blow to the head.*

Description

The data are the acceleration of brain tissue in mm/msec in a cadaver before and after five blows to the cranium measured over 60 milliseconds. The times of the blows within this interval were at 14 msec.

Usage

```
HeadImpactData
```

Format

A 133 by 3 matrix of real numbers. The first column contains row numbers, the second the times of observations, and third the tissue acceleration in millimeters per second per second.

Source

The data are widely reported in various books on data smoothing. The first report was in Wolfgang Hartle's 1990 book Appplied Nonparamemtric Regression. The original experiment was conducted at the University of Heidleberg, and as far as we know, seems to have been unpublished.

Examples

```
HeadImpactTime <- HeadImpactData[,2]  #  time in milliseconds
HeadImpact     <- HeadImpactData[,3]  #  acceleration in millimeters/millisecond^2
HeadImpactRng  <- c(0,60) # Define range time for estimated model
# plot the data along with a unit pulse
plot(HeadImpactTime, HeadImpact, type="p", xlim=c(0,60),  ylim=c(-1.0,1.5),
     xlab="Time (milliseconds)", ylab="Acceleration (mm/msec^2)")
lines(c( 0,60), c(0,0), lty=3)
lines(c(14,14), c(0,1), lty=2)
lines(c(15,15), c(0,1), lty=2)
lines(c(14,15), c(1,1), lty=2)
```

---

make.Fterm *Check each specification in list of forcing term specifications for a linear differential equation.*

Description

A system of linear differential equations is defined by a list of length equal to the number of variables in the system. Each member of this list defines a single linear differential equation. Within each equation there may be one or more terms involving a coefficient function multiplying either a known fixed function, called a forcing function. Forcing terms often involve a fixed constant multiplier, which is frequently either 1 or -1. This function sets up a list object that specifies the structure of a single term.

Usage

```
make.Fterm(funobj, parvec, estimate, Ufd, factor=1)
```

Arguments

funobj	A specification of the coefficient function. If funobj is of class basisobj, fd or fdPar, the coefficient function is a functional data object for a single function. If the class is list, its members define a function, its derivative, and other information that may be needed in the definition.
parvec	The vector of parameters defining coefficient function.

estimate	A vector of the same length as parvec containing logical values defining which parameteres are to be estimated (TRUE) or kept fixed (FALSE). If all values are the same, a single TRUE or FALSE is sufficient.
Ufd	A known forcing function. This may be specified as a functional data or functional parameter object, but may also be specified as a user-defined function. See function make.coef for more details.
factor	A real number that is treated as fixed. For example, it is frequently the case that a variable will appear in two or more places in a system of equations, and sometimes multipled by -1.

Details

This function checks that all supplied terms conform to what is required.

Value

A named list object defining a forcing term in a linear differential equation.

References

J. O. Ramsay and G. Hooker (2017) *Dynamic Data Analysis*. Springer.

See Also

checkModel, make.Variable, make.Xterm, printModel

Examples

```
#  Here is the code that sets up the single forcing term for the
#  refinery data examnple:
TimeData <- RefineryData[,1]
TrayData <- RefineryData[,2]
ValvData <- RefineryData[,3]
conbasis <- create.constant.basis(c(0,193))
alphafdPar <- fdPar(conbasis)
Valvbreaks <- c(0,67,193)
Valvnbasis <- 2
Valvnorder <- 1
Valvbasis  <- create.bspline.basis(c(0,193), Valvnbasis, Valvnorder, Valvbreaks)
Valvfd     <- smooth.basis(TimeData, ValvData, Valvbasis)$fd
#  Fterm Fields:    funobj      parvec  estimate  Ufd      factor
FTerm <- make.Fterm(alphafdPar, 1.0,    TRUE,     Valvfd,  1)
FList <- vector("list", 1)
FList[[1]] <- FTerm
#
#  For other examples of the use of this function,
#  see these examples in the description of the function \code{make.Variable}.
#
#  The single forcing term of the head impact data
#
#  The cosine and constant forcing terms for the average temperature in
```

```
#  Montreal.
#
#  The 20-step forcing function and its coefficient for the "fda" script data.
#
#  The set point forcing term for control variable of the cruise control data.
#
```

`make.Variable` *Check a list of linear differential equation specifications.*

Description

A system of linear differential equations is defined by a list of length equal to the number of variables in the system. Each member of this list defines a single linear differential equation. Within each equation there are typically one or more terms involving a coefficient function multiplying either a derivative of one of the variables, or multiplying a forcing function. This function sets up a list object that specifies the structure of a single coefficient function.

Usage

```
make.Variable(name = "", order = 1, XList = NULL, FList = NULL, weight=1)
```

Arguments

`name`	A string to be used as a name for the equation or variable. By default, it is "xn" where "n" is the position in the list.
`order`	A nonnegative integer specifying the order of derivative on the left side of the equation.
`XList`	A list each member of which specifies one of the terms in the homogeneous portion of the right side of the equation. These can be constructed using function `make.Xterm`.
`FList`	A list each member of which specifies one of the forcing terms in the forcing portion of the right side of the equation. These can be constructed using function `make.Fterm`.
`weight`	A positive weight that can be applied when variables are differently weighted.

Details

This function checks that all supplied terms conform to what is required. Use of the functions `make.Xterm` and `make.Fterm` is recommended.

Value

A named list object defining a linear differential equation.

References

J. O. Ramsay and G. Hooker (2017) *Dynamic Data Analysis*. Springer.

See Also

make.Xterm, make.Fterm, printModel

Examples

```
#
#  Example 1:  The refinery data
#
N <- 194
TimeData <- RefineryData[,1]
TrayData <- RefineryData[,2]
ValvData <- RefineryData[,3]
#  Define the List array containing the tray data
TrayList <- list(argvals=RefineryData[,1], y=RefineryData[,2])
TrayDataList  <- vector("list",1)
TrayDataList[[1]] <- TrayList
# construct the basis object for tray variable
#  Order 5 spline basis with four knots at the 67
#   to allow discontinuity in the first derivative
#  and 15 knots between 67 and 193
Traynorder <- 5
Traybreaks <- c(0, rep(67,3), seq(67, 193, len=15))
Traynbasis <- 22
TrayBasis  <- create.bspline.basis(c(0,193), Traynbasis, Traynorder,
                                   Traybreaks)
#  Set up the basis list for the tray variable
TrayBasisList    <- vector("list",1)
TrayBasisList[[1]] <- TrayBasis
#  Both alpha and beta coefficient functions will be constant.
#  Define the constant basis
conbasis <- create.constant.basis(c(0,193))
betafdPar  <- fdPar(conbasis)
alphafdPar <- fdPar(conbasis)
#  Construct a step basis (order 1) and valve level
Valvbreaks <- c(0,67,193)
Valvnbasis <- 2
Valvnorder <- 1
Valvbasis  <- create.bspline.basis(c(0,193), Valvnbasis, Valvnorder, Valvbreaks)
#  smooth the valve data
Valvfd <- smooth.basis(TimeData, ValvData, Valvbasis)$fd
#  Set up the model list for the tray variable
#  Define single homogeneous term
#  Xterm Fields:    funobj     parvec  estimate  variable deriv. factor
XTerm <- make.Xterm(betafdPar, 0.04,  TRUE,     1,       0,     -1)
XList <- vector("list", 1)
XList[[1]] <- XTerm
#  Define the single forcing term
#  Fterm Fields:    funobj      parvec  estimate  Ufd      factor
FTerm <- make.Fterm(alphafdPar, 1.0,    TRUE,     Valvfd,  1)
FList <- vector("list", 1)
FList[[1]] <- FTerm
#  Define the single differential equation in the model
```

```
TrayVariable <- make.Variable(XList=XList, FList=FList,
                              name="Tray level", order=1)
#  Set up the model List array
TrayModelList <- vector("list",1)
TrayModelList[[1]] <- TrayVariable
#  Run a check on TrayVariableList and make the modelList object
checkModelList <- checkModel(TrayBasisList, TrayModelList, summarywrd=TRUE)
TrayModelList  <- checkModelList$model
nparam         <- checkModelList$nparam
#
#  Example 2:  The head impact data
#
ImpactTime <- HeadImpactData[,2]  #  time in milliseconds
ImpactData <- HeadImpactData[,3]*100  #  acceleration in millimeters/millisecond^2
ImpactRng  <- c(0,60) # Define range time for estimated model
#  Define the List array containing the data
ImpactDataList1 <- list(argvals=ImpactTime, y=ImpactData)
ImpactDataList <- vector("list",1)
ImpactDataList[[1]] <- ImpactDataList1
# Define a unit pulse function located at times 14-15
Pulsebasis <- create.bspline.basis(ImpactRng, 3, 1, c(0,14,15,60))
Pulsefd    <- fd(matrix(c(0,1,0),3,1),Pulsebasis)
#  Construct basis for output x(t),
#  multiple knots at times 14 and 15.
#  Order 6 spline basis, with three knots at the impact point and
#  three knots at impact + delta to permit discontinuous first
#  derivatives at these points
knots     <- c(0,14,14,14,15,15,seq(15,60,len=11))
norder    <- 6
nbasis    <- 21
ImpactBasis <- create.bspline.basis(ImpactRng,nbasis,norder,knots)
#  plot the basis
ImpactTimefine <- seq(0,60,len=201)
ImpactBasisfine <- eval.basis(ImpactTimefine, ImpactBasis)
#  set up basis list
ImpactBasisList <- vector("list",1)
ImpactBasisList[[1]] <- ImpactBasis
# Set up the constant basis, make three coefficients and check them
conbasis <- create.constant.basis(ImpactRng)
# Define the terms in the second order linear equation
# Define the two terms in the homogeneous part of the equation
#  Xterm Fields:     funobj     parvec  estimate  variable deriv. factor
Xterm1 <- make.Xterm(conbasis,  1,      TRUE,     1,       0,     -1)
Xterm2 <- make.Xterm(conbasis,  1,      TRUE,     1,       1,     -1)
# Set up the XList vector of length two
XList <- vector("list",2)
XList[[1]] <- Xterm1
XList[[2]] <- Xterm2
# Define the forcing term
#  Fterm Fields:    funobj    parvec  estimate  Ufd      factor
Fterm <- make.Fterm(conbasis, 1.0,   TRUE,     Pulsefd, 1)
# Set up the forcing term list of length one
FList      <- vector("list",1)
```

```
FList[[1]] <- Fterm
#  Define the single differential equation in the model
ImpactVariable <- make.Variable(name="Impact", order=2, XList=XList, FList=FList)
#  Define list of length one containing the equation definition
ImpactModelList=vector("list",1)
ImpactModelList[[1]] <- ImpactVariable
#  Check the object for internal consistency and
#  set up the model object
#  Run a check on TrayVariableList and make the modelList object
checkModelList  <- checkModel(ImpactBasisList, ImpactModelList, summarywrd=TRUE)
ImpactModelList <- checkModelList$model
nparam          <- checkModelList$nparam
#
#  Example 3:  The X coordinate of the fda script data
#
fdascriptX <- as.matrix(fdascript[,1,1])
fdascriptY <- as.matrix(fdascript[,1,2])
#  Define the observation times in 100ths of a second
centisec <- seq(0,2.3,len=1401)*100
fdarange <- range(centisec)
fda.dataList = vector("list", 2)
fda.dataList[[1]]$argvals <- centisec
fda.dataList[[1]]$y       <- fdascriptX
fda.dataList[[2]]$argvals <- centisec
fda.dataList[[2]]$y       <- fdascriptY
#  ------------------------------------------------------------------------
#   Set up the basis object for representing the X and Y coordinate
#   functions
#  ------------------------------------------------------------------------
#  The basis functions for each coordinate will be order 6 B-splines,
#  32 basis functions per second, 4 per 8 herz cycle
#  2.3 seconds 2.3*32=73.6.
#  Order 6 is used because we want to estimate a smooth second
#  derivative.
#  This implies norder + no. of interior knots <- 74 - 1 + 6 <- 79
#  basis functions.
nbasis   <- 79
norder   <-  6 #  order 6 for a smooth 2nd deriv.
fdabasis <- create.bspline.basis(fdarange, nbasis, norder)
fdaBasisList <- vector("list",2)
fdaBasisList[[1]] <- fdabasis
fdaBasisList[[2]] <- fdabasis
#  ------------------------------------------------------------------------
#   Use the basis to estimate the curve functions and their
#   second derivatives
#  ------------------------------------------------------------------------
ResultX <- smooth.basis(centisec, fdascriptX, fdabasis)
fdascriptfdX <- ResultX$fd
D2fdascriptX <- eval.fd(centisec, fdascriptfdX, 2)
ResultY <- smooth.basis(centisec, fdascriptY, fdabasis)
fdascriptfdY <- ResultY$fd
D2fdascriptY <- eval.fd(centisec, fdascriptfdY, 2)
#  ------------------------------------------------------------------------
```

```
#   Set up the basis objects for representing the coefficient
#   functions
#  ------------------------------------------------------------------------
#  Define a constant basis and function over the 230 centisecond range
conbasis <- create.constant.basis(fdarange)
confd    <- fd(1,conbasis)
confdPar <- fdPar(confd)
#  Define the order one Bspline basis for the coefficient
#  of the forcing coefficients
nevent    <- 20
nAorder   <- 1
nAbasis   <- nevent
nAknots   <- nAbasis + 1
Aknots    <- seq(fdarange[1],fdarange[2],len=nAknots)
Abasisobj <- create.bspline.basis(fdarange,nAbasis,nAorder,Aknots)
#  ------------------------------------------------------------------------
#  Set up single homogeneous terms in the homogeneous portion of the
#  equations  D^2x = -beta_x x and D^2y = -beta_y y.
#  Each term involves a scalar constant multiplier -1.
#  ------------------------------------------------------------------------
Xterm <- make.Xterm(confdPar, 0.04, TRUE, variable=1, derivative=0, factor=-1)
Yterm <- make.Xterm(confdPar, 0.04, TRUE, variable=2, derivative=0, factor=-1)
#  Set up to list arrays to hold these term specifications
XListX <- vector("list",1)
XListX[[1]] <- Xterm
XListY <- vector("list",1)
XListY[[1]] <- Yterm
#  ------------------------------------------------------------------------
#  Set up coefficients for the forcing terms
#  \alpha_x(t)*1 and \alpha_y(t)*1.
#  The forcing functions are the unit constant function.
#  ------------------------------------------------------------------------
parvecA <- matrix(0,nAbasis,1)
FtermX <- make.Fterm(Abasisobj, parvecA, TRUE, Ufd=confd, factor=1)
FtermY <- make.Fterm(Abasisobj, parvecA, TRUE, Ufd=confd, factor=1)
#  Set up list arrays to hold forcing terms specs
FListX <- vector("list", 1)
FListX[[1]] <- FtermX
FListY <- vector("list", 1)
FListY[[1]] <- FtermY
#  ------------------------------------------------------------------------
#  make variables for X and Y coordinates and system object fdaVariableList
#  ------------------------------------------------------------------------
Xvariable <- make.Variable(name="X", order=2, XList=XListX, FList=FListX)
Yvariable <- make.Variable(name="Y", order=2, XList=XListY, FList=FListY)
#  List array for the whole system
fdaVariableList <- vector("list",2)
fdaVariableList[[1]] <- Xvariable
fdaVariableList[[2]] <- Yvariable
#  check the system specification for consistency,  This is a mandatory
#  step because objects are set up in the output list that are needed
#  in the analysis.  A summary is displayed.
checkModelList <- checkModel(fdaBasisList, fdaVariableList, TRUE)
```

```
fda.modelList <- checkModelList$modelList
nparam        <- checkModelList$nparam
#
#  Example 4:  Average temperature in Montreal
#
daytime73 <- seq(2.5,362.5,len=73)
dayrange  <- c(0,365)
#  set up block averages for Canadian temperature data,
#  available from the fda package
tempav <- CanadianWeather$dailyAv[,,"Temperature.C"]
tempav73 <- matrix(0,73,35)
m2 <- 0
for (  i in 1 : 73 ) {
  m1 <- m2 + 1
  m2 <- m2 + 5
  tempavi <- apply(tempav[m1:m2,1:35],2,mean)
  tempav73[i,] <- tempavi
}
#  center the time of the year on winter
winterind73  <- c( 37:73,1: 36)
tempav73 <- tempav73[winterind73,]
#  select the data for Montreal
station <- 12
#  set up TempDataList
TempDataList1 <- list(argvals=as.matrix(daytime73), y=as.matrix(tempav73[,station]))
TempDataList  <- vector("list",1)
TempDataList[[1]]    <- TempDataList1
#  basis for 5-day block averages
norder    <- 5
daybreaks <- seq(0,365,5)
nbreaks   <- length(daybreaks)
nbasis    <- norder + nbreaks - 2
daybasis  <- create.bspline.basis(dayrange, nbasis)
TempBasisList <- vector("list",1)
TempBasisList[[1]] <- daybasis
#  Define the two forcing functions:
#  constant function Ufd for constant forcing
Uconbasis <- create.constant.basis(dayrange)
Uconfd    <- fd(1, Uconbasis)
Uconvec   <- matrix(1,73,1)
#  cosine function for solar radiation forcing
uvec      <- -cos((2*pi/365)*(daytime73+10+182))
Ucosbasis <- create.fourier.basis(dayrange, 3)
Ucosfd    <- smooth.basis(daytime73, uvec, Ucosbasis)$fd
#  Define three basis functional objects for coefficients
#  A fourier basis for defining the positive homogeneous
#  coefficient
nWbasis   <- 7
Wbasisobj <- create.fourier.basis(dayrange, nWbasis)
#  constant forcing coefficient for constant forcing
nAbasisC   <- 1
AbasisobjC <- create.constant.basis(dayrange)
#  fourier coefficint function for radiative forcing
```

```
nAbasisF   <- 1
AbasisobjF <- create.constant.basis(dayrange)
#  Set up the list object for the positive coefficient for
#  the homogeneous term.
linfun <- list(fd=fun.explinear, Dfd=fun.Dexplinear, more=Wbasisobj)
#  Define the variable for temperature
#  define homogeneous term and list container
estimate = rep(TRUE,7)
estimate[7] <- FALSE
parvec <- matrix(0,7,1)
# parvec[7] <- 3
Xterm <- make.Xterm(funobj=linfun, parvec=parvec, estimate=estimate,
                    variable=1, derivative=0, factor= -1)
XList <- vector("list", 1)
XList[[1]] <- Xterm
#  define two forcing terms
Fterm1 <- make.Fterm(funobj=AbasisobjC, parvec=1.0, estimate=TRUE,
                     Ufd=Uconfd, factor=1)
Fterm2 <- make.Fterm(funobj=AbasisobjF, parvec=1.0, estimate=TRUE,
                     Ufd=Ucosfd, factor=1)
#  forcing term container
FList <- vector("list", 2)
FList[[1]] <- Fterm1
FList[[2]] <- Fterm2
#  set up variable list object
TempVariableList1 <- make.Variable(name="Temperature", order=1, XList=XList, FList=FList)
# set up TempVariableList
TempModelList <- vector("list",1)
TempModelList[[1]] <- TempVariableList1
# define the model list object
TempModelcheck <- checkModel(TempBasisList, TempModelList, TRUE)
TempModelList <- TempModelcheck$modelList
nparam <- TempModelcheck$nparam
```

`make.Xterm` *Check homogeneous term specifications for a linear differential equation.*

Description

A system of linear differential equations is defined by a list of length equal to the number of variables in the system. Each member of this list defines a single linear differential equation. Within each equation there are typically one or more terms involving a coefficient function multiplying either a derivative of one of the variables in the system. These terms often involve a fixed constant multiplier, which is frequently either 1 or -1. This function sets up a list object that specifies the structure of a single term.

Usage

```
make.Xterm(funobj, parvec, estimate, variable, derivative=0, factor=1)
```

Arguments

funobj	A specification of the coefficient function. If funobj is of class basisobj, fd or fdPar, the coefficient function is a functional data object for a single function. If the class is list, its members define a function, its derivative, and other information that may be needed in the definition.
parvec	The vector of parameters defining coefficient function.
estimate	A vector of the same length as parvec containing logical values defining which parameteres are to be estimated (TRUE) or kept fixed (FALSE). If all values are the same, a single TRUE or FALSE is sufficient.
variable	An integer specifying the variable within which this is a term.
derivative	The order of the derivative of the variable.
factor	A real number that is treated as fixed. For example, it is frequently the case that a variable will appear in two or more places in a system of equations, and sometimes multipled by -1.

Details

This function checks that all supplied terms conform to what is required.

Value

A named list object defining a homogeous term in a linear differential equation.

References

J. O. Ramsay and G. Hooker (2017) *Dynamic Data Analysis*. Springer.

See Also

make.Fterm, make.Variable, printModel

Examples

```
#  Here is the code that sets up the single homogeneous term for the
#  refinery data examnple:
conbasis  <- create.constant.basis(c(0,193))
betafdPar <- fdPar(conbasis)
#  Xterm Fields:    funobj     parvec  estimate  variable deriv. factor
XTerm <- make.Xterm(betafdPar, 0.04,  TRUE,      1,       0,     -1)
#  Enter this list object in a list of length one.
XList <- vector("list", 1)
XList[[1]] <- XTerm
#  Here is the code that sets up the single homogeneous term involving a
#  non-functional data function for the average temperature for Montreal.
#  Functions \code{fun.explinear} and \code{fun.Dexplinear} are in the
#  package, and compute the positive value of the exponential transform
#  of a functional data object and its derivative.  The basis object
#  Set up the list object for the positive coefficient for
#  the homogeneous term.
```

```
#  \code{Wbasisobj} is also supplied in the list object \code{linfun}.
nWbasis   <- 7
Wbasisobj <- create.fourier.basis(c(0,365), nWbasis)
linfun <- list(fd=fun.explinear, Dfd=fun.Dexplinear, more=Wbasisobj)
estimate = rep(TRUE,7)
estimate[7] <- FALSE
parvec <- matrix(0,7,1)
Xterm <- make.Xterm(funobj=linfun, parvec=parvec, estimate=estimate,
                    variable=1, derivative=0, factor= -1)
XList <- vector("list", 1)
XList[[1]] <- Xterm
#
#  For other examples of the use of this function,
#  see these examples in the description of the function \code{make.Variable}.
#
#  The single homogeneous term of the head impact data
#
#  The single zero derivative terms in the two second order equations
#  for the "fda" script model.
#
#  The speed and control terms of the cruise control data
#
```

`modelList2Vec` *Extract the master parameter vector from a model list object.*

Description

A system of linear differential equations is defined by a list of length equal to the number of variables in the system. Each member of this list defines a single linear differential equation. Within each equation there are typically one or more terms involving a coefficient function multiplying either a derivative of one of the variables, or multiplying a forcing function. The coefficient functions in term are defined by one or more parameters. For some coefficients, its parameters are regarded as fixed, but for others, they require estimation from the data. This function extracts all the parameters requiring estimation and returns them as a one-dimensional vector.

Usage

```
modelList2Vec(modelList, nparam)
```

Arguments

`modelList`	A list object containing the specification of a Data2LD model. Each member of this list contains a list object that defines a single linear differential equation.
`nparam`	The total number of parameters in the master parameter vector, which contains all parameters, whether estimated or not.

Details

The parameters in the master parameter vector are organized as follows. First come the parameters defining the homogeneous coefficient functions, in the order in which the appear in the equation system. Next come the parameters defining the forcing coefficient functions, also in the order in which they appear.

Normally the user will have a limited need to deal with the master parameter vector. Both the initial and final parameter values are displayed by the function `printModel` in a much more readable form.

Value

A numerical vector object containing values of the parameters to be estimated. The order of these parameters is the order in which they occur as each member of a model list object in turn is processed.

References

J. O. Ramsay and G. Hooker (2017) *Dynamic Data Analysis*. Springer.

See Also

`modelVec2List`

`modelVec2List` *Put values in a parameter vector into the coefficient specifications within a model List object.*

Description

A system of linear differential equations is defined by a list of length equal to the number of variables in the system. Each member of this list defines a single linear differential equation. Within each equation there are typically one or more terms involving a coefficient function multiplying either a derivative of one of the variables, or multiplying a forcing function. The coefficient functions in term are defined by one or more parameters. For some coefficients, its parameters are regarded as fixed, but for others, they require estimation from the data. This function replaces all the parameters in the coefficient functions in a coefficient list object requiring estimation by values within a one-dimensional vector.

Usage

```
modelVec2List(modelList, thetavec)
```

Arguments

`modelList`	A list object containing the specifications of the model.
`thetavec`	A numeric vector containing values for all the parameters in a linear differential system that requires estimation.

Details

A coefficient specification can be made manually, or can set up in a by a single invocation of function `make.coef`.

Value

A numerical vector object containing values of the parameters to be estimated. The order of these parameters is the order in which they occur as each member of a model list object in turn is processed.

References

J. O. Ramsay and G. Hooker (2017) *Dynamic Data Analysis*. Springer.

See Also

`modelList2Vec`

`printModel` *Display the definition of each variable in a system of linear differential equations.*

Description

The definitions of the linear differential equations in a system are containing in an un-named list object of the same length as the number of equations. Each member of the list is displayed in turn. For each equation, each of the homogenous terms in turn are displayed, and then each forcing term displayed next.

Usage

```
printModel(modelList, titlestr=NULL)
```

Arguments

`modelList`	This is a list object defining each equation in a system of linear differential equations.
`titlestr`	A character string used as a title for the equation system.

Details

For each equation in turn, the details of the homogenous terms are displayed first, and then the details of the forcing terms.

The homogeneous terms are those involving a product of a coefficient function and a derivative of a varible in the system.

The coefficient function is defined by a set of parameters, some or all of which can be either fixed or estimated. The display of the homogenous coefficient function is of: (1) the positions in the master

parameter vector of these parameters, (2) the values of these parameters, (3) TRUE/FALSE value(s) indicating which parameters are fixed and which are estimated, and (4) the type of the coefficient function. The coefficient function may be a functional data basis object, a functional data object, or a functional parameter object, in which case the parmeters define a linear combination of functional basis objects. Or, alternatively, the coefficient function may be named list whose members contain (1) `fd:` the definition of an R function, (2) `Dfd:` the definition of its derivative with respect to the parameters and (3) `more:` a list object containing any other information that the coefficient needs to define the values of itself and its derivative.

The homogeneous term is also defined by the number of the variable that the coefficient function multiplies, the order of the derivative of that function, and a fixed constant factor multiplying the product, which defaults to one; but, for example, is often -1.

A forcing term is the product of a coefficient function and a forcing function. Each forcing function is a fixed fixed functional data object of class `fd`. The function may be single, or involve a number of replications. The forcing function also involves a fixed multiplying constant.

References

J. O. Ramsay and G. Hooker (2017) *Dynamic Data Analysis*. Springer.

`RefineryData` *Reflux and tray level in a refinery*

Description

194 observations on reflux and "tray 47 level" in a distallation column in an oil refinery.

Usage

```
RefineryData
```

Format

A data.frame with the following components:

- Time: observation time 0:193
- Tray.level: reflux flow centered on the mean of the first 60 observations
- Valve.setting: tray 47 level centered on the mean of the first 60 observations

Source

Ramsay, James O., and Silverman, Bernard W. (2005), *Functional Data Analysis, 2nd ed.*, Springer, New York, p. 4, Figure 1.4, and chapter 17.

Examples

```
#  input the data
TimeData <- RefineryData[,1]
TrayData <- RefineryData[,2]
ValvData <- RefineryData[,3]
#  plot the data
par(mfrow=c(2,1))
plot(TimeData, TrayData, type="p")
lines(c(67,67), c(0,4.0), type="l")
plot(TimeData, ValvData, type="p")
lines(c(67,67), c(0,0.5), type="l")
```

Index