Package 'Data2LD'

January 24, 2019

Type Package

Version 2.0.0

Date 2019-1-24

Title Functional Data Analysis with Linear Differential Equations

Author James Ramsay [aut, cre]

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

Depends R (>= 2.10.0), fda, splines, Matrix, graphics, deSolve

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 a new demo, CruiseControl. 'Matlab' versions of the code and sample analyses are available by ftp from <http://www.psych.mcgill.ca/misc/fda/downloads/Data2LD/>. 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

Suggests knitr, rmarkdown

VignetteBuilder knitr

NeedsCompilation yes

Repository CRAN

Date/Publication 2019-01-24 22:00:12 UTC

R topics documented:

`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, coefList)
```

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.
`coefList`	A list object containing the specifications of one or more coefficient functions.

Details

A coefficient specification can be made manually, or can set up in a by a single invocation of function `make.coef`. Model specifications can also be set manually, or by an invocation of function `make.variable` for each linear equation in the system.

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, coefList)
```

Arguments

XbasisList	A list object of length equal to the number of equations in the system. Each member of this list is in turn a list specifying the structure of the equation.
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.
coefList	A list object containing the specifications of one or more coefficient functions.

Details

A coefficient specification can be made manually, or can set up in a by a single invocation of function `make.coef`. Variable specifications can also be set manually, or by an invocation of function `make.variable` for each linear differential equation in the system.

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, coefList)
```

Arguments

XbasisList	A list object of length equal to the number of equations in the system. Each member of this list is in turn a list specifying the structure of the equation.
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.
coefList	A list object containing the specifications of one or more coefficient functions.

Details

A coefficient specification can be made manually, or can set up in a by a single invocation of function `make.coef`. Variable specifications can also be set manually, or by an invocation of function `make.variable` for each linear differential equation in the system.

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)
```

---

coefCheck *Check a list object containing coefficient 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. 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 checks that each of the coefficient specifications has been properly set up using values that are appropriate. In addition, it also assembles a numeric vector containing the values of of the estimated parameters, and returns this as well as the number of values in a named list. It is easy to make mistakes in setting up lists of coefficient objects, and it is strongly suggested to invoke this function prior to invoking any function using this list.

Usage

```
coefCheck(coefList, report=TRUE)
```

Arguments

coefList	A list object containing the specifications of one or more coefficient functions.
report	If TRUE, display a report on the structures of the coefficient functions.

Details

A coefficient specification can be made manually, or can set up in a by a single invocation of function `make.coef`. This function checks that all coefficient specifications encountered when processing a model List, whether or not the coefficients are to be estimated, are consistent with what is required to specify a coefficient. If certain values are not supplied, default values are assigned. For example, if the member with name `estimate` is not supplied, it is set to TRUE. If one or more parameter values are not specified, they are set to zeros.

Value

A named list object. Member `coefList` is an image of that argument where all default values are assigned. This should be defined as the version to be used in subsequent calculations. Member `theta` is a numeric vector containing the values of all parameters requiring estimation. Member `ntheta` is the length of this parameter vector.

References

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

See Also

`make.Coef`, `make.Xterm`, `make.Fterm`, `make.Variable`, `make.Model`

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, coefList, 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.
`coefList`	A list each member of which specifies one of the coefficients that multiply one or more terms in the system. These can be constructed using function `make.Coef`. 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

make.Model, coefCheck

Examples

```
#
#  Example 1:  The refinery data
#
# The single equation requires two coefficients:
# 1. A constant coefficient for the speed of reaction
# 2. A constant coefficient for the single forcing function
#  Set up the observation times and range
TimeData <- RefineryData[,"Time"]
TimeRng  <- range(TimeData)
# Defne a constant basis object
conbasis <- create.constant.basis(TimeRng)
```

```
#  Define the two coefficient functions and
#  store these in a coefficient function list
TrayCoefList <- vector("list",2)
# Both coefficients are estimated and both have initial value 0.
TrayCoefList[[1]] <- make.Coef(fun=conbasis, parvec=0, estimate=TRUE)
TrayCoefList[[2]] <- make.Coef(fun=conbasis, parvec=0, estimate=TRUE)
#  Run a check on the coefficient List array,
#  which also counts the number of estimated parameters
TraycoefResult <- coefCheck(TrayCoefList)
TrayCoefList   <- TraycoefResult$coefList
TrayNtheta     <- TraycoefResult$ntheta
print(paste("ntheta = ",TrayNtheta))
#  Define single homogeneous term with order zero derivative,
#  constant multiplier -1, and defined by the first coeffcient
XTerm <- make.Xterm(variable=1, derivative=0, ncoef=1, factor=-1)
XList <- vector("list", 1)
XList[[1]] <- XTerm
#  Define the step forcing function for the valve setting
Valvebreaks <- c(0,67,193)
Valvenbasis <- 2
Valvenorder <- 1
ValveBasis  <- create.bspline.basis(TimeRng, Valvenbasis, Valvenorder, Valvebreaks)
#  smooth the valve setting data to define the step forcing function
ValveData <- RefineryData[,"Valve.setting"]
Valvefd   <- smooth.basis(TimeData, ValveData, ValveBasis)$fd
#  Define the FList object for the single forcing function
FTerm <- make.Fterm(ncoef=2, Ufd=Valvefd)
FList <- vector("list", 1)
FList[[1]] <- FTerm
#  Define the single differential equation in the first order model
TrayVariable <- make.Variable(name="Tray level", order=1, XList=XList, FList=FList)
#  check the object for internal consistency
TrayList=vector("list",1)
TrayList[[1]] <- TrayVariable
# 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
#  check the object for internal consistency
TrayModelList <- make.Model(TrayBasisList, TrayList, TrayCoefList)
#  Define the List array containing the tray data
Tray.dataList <- list(argvals=RefineryData[,"Time"],
                      y         =RefineryData[,"Tray.level"])
TrayDataList  <- vector("list",1)
TrayDataList[[1]] <- Tray.dataList
#  Set smoothing constant rho to a value specifying light smoothing
```

```
rhoVec <- 0.5
#  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.
Data2LDList <- Data2LD(TrayDataList, TrayBasisList, TrayModelList, TrayCoefList, 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
#
#  This is a second order model forced by a single pulse function
#  All coefficients are positive
ImpactTime <- RefineryData[,2]  #  time in milliseconds
Impact     <- RefineryData[,3]  #  time in milliseconds
ImpactRng  <- c(0,60) # Define range time for estimated model
# Set up the constant basis
conbasis <- create.constant.basis(ImpactRng)
# Define the three constant coefficient functions
coef1 <- make.Coef(fun=conbasis, parvec=0, estimate=TRUE)
coef2 <- make.Coef(fun=conbasis, parvec=0, estimate=TRUE)
coef3 <- make.Coef(fun=conbasis, parvec=0, estimate=TRUE)
#  Set up the coefficient list
coefList  <- vector("list",3)
coefList[[1]] <- coef1
coefList[[2]] <- coef2
coefList[[3]] <- coef3
#  check coefficient list
coefCheckList <- coefCheck(coefList)
coefList      <- coefCheckList$coefList
ntheta        <- coefCheckList$ntheta
print(paste("ntheta = ",ntheta))
# Define the two terms in the homogeneous part of the equation
Xterm0 <- make.Xterm(variable=1, derivative=0, ncoef=1, factor=-1)
Xterm1 <- make.Xterm(variable=1, derivative=1, ncoef=2, factor=-1)
# Set up the XList vector of length two
XList <- vector("list",2)
XList[[1]] <- Xterm0
XList[[2]] <- Xterm1
# 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)
# Define the forcing term
Fterm      <- make.Fterm(ncoef=3, Ufd=Pulsefd, factor=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(order=2, XList=XList, FList=FList)
#  Define list of length one containing the equation definition
ImpactList=vector("list",1)
ImpactList[[1]] <- ImpactVariable
#  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)
#  set up basis list
ImpactBasisList <- vector("list",1)
ImpactBasisList[[1]] <- ImpactBasis
#  check the object for internal consistency
ImpactModelList <-
        make.Model(ImpactBasisList, ImpactList, coefList)
#  Define the List array containing the data
dataList1 <- list(argvals=ImpactTime, y=Impact)
dataList <- vector("list",1)
dataList[[1]] <- dataList1
# An evaluation of the criterion at the initial values
rhoVec <- 0.5  #  light smoothing
#  This command causes Data2LD to set up and save the tensors
# Data2LDResult <- Data2LD(dataList, ImpactBasisList,
#         ImpactModelList, coefList, rhoVec)
# MSE  <- Data2LDList$MSE    # Mean squared error for fit to data
# DMSE <- Data2LDList$DpMSE  #  gradient with respect to parameter values
```

`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, coefList, rhoMat,
            convcrit = 1e-4, iterlim = 20, dbglev = 1,
            active, parMap = diag(rep(1,npar)))
```

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.

`coefList` A list each member of which specifies one of the coefficients that multiply one or more terms in the system. These can be constructed using function `make.Coef`. 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.

`active` The indices of the parameters that are to be estimated. If there are parameters not in this set of indices, these will be held fixed even though their specification in `coefList` specifies that they are to be estimated.

`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 optimal parameter values.

`coefList.opt:` The coefficient list containing the optimal parameter values.

References

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

See Also

Data2LD

Examples

```
#
#  Example 1:  The refinery data
#
# The single equation requires two coefficients:
# 1. A constant coefficient for the speed of reaction
# 2. A constant coefficient for the single forcing function
#  Set up the observation times and range
TimeData <- RefineryData[,"Time"]
TimeRng  <- range(TimeData)
# Defne a constant basis object
conbasis <- create.constant.basis(TimeRng)
#  Define the two coefficient functions and
#  store these in a coefficient function list
TrayCoefList <- vector("list",2)
# Both coefficients are estimated and both have initial value 0.
TrayCoefList[[1]] <- make.Coef(fun=conbasis, parvec=0, estimate=TRUE)
TrayCoefList[[2]] <- make.Coef(fun=conbasis, parvec=0, estimate=TRUE)
#  Run a check on the coefficient List array,
#  which also counts the number of estimated parameters
TraycoefResult <- coefCheck(TrayCoefList)
TrayCoefList   <- TraycoefResult$coefList
TrayNtheta     <- TraycoefResult$ntheta
print(paste("ntheta = ",TrayNtheta))
#  Define single homogeneous term with order zero derivative,
#  constant multiplier -1, and defined by the first coeffcient
XTerm <- make.Xterm(variable=1, derivative=0, ncoef=1, factor=-1)
XList <- vector("list", 1)
XList[[1]] <- XTerm
#  Define the step forcing function for the valve setting
Valvebreaks <- c(0,67,193)
Valvenbasis <- 2
Valvenorder <- 1
ValveBasis  <- create.bspline.basis(TimeRng, Valvenbasis, Valvenorder, Valvebreaks)
#  smooth the valve setting data to define the step forcing function
ValveData <- RefineryData[,"Valve.setting"]
Valvefd   <- smooth.basis(TimeData, ValveData, ValveBasis)$fd
#  Define the FList object for the single forcing function
FTerm <- make.Fterm(ncoef=2, Ufd=Valvefd)
FList <- vector("list", 1)
FList[[1]] <- FTerm
#  Define the single differential equation in the first order model
TrayVariable <- make.Variable(name="Tray level", order=1, XList=XList, FList=FList)
#  check the object for internal consistency
```

```
TrayVariableList=vector("list",1)
TrayVariableList[[1]] <- TrayVariable
# 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
#  check the object for internal consistency and make modelList
TrayModelList <- make.Model(TrayBasisList,TrayVariableList, TrayCoefList)
#  Define the List array containing the tray data
Tray.yList <- list(argvals=RefineryData[,"Time"],
                   y      =RefineryData[,"Tray.level"])
TrayDataList  <- vector("list",1)
TrayDataList[[1]] <- Tray.yList
#  Set smoothing constant rho to a value specifying light smoothing
rhoVec <- 0.5
#  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.
Data2LDList <- Data2LD(TrayDataList, TrayBasisList, TrayModelList, TrayCoefList, 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,TrayNtheta)
#  This initializes the list object containing coefficient estimates
TrayCoefList.opt <- TrayCoefList
# 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, TrayCoefList.opt,
                rhoi, convrg, iterlim, dbglev)
  theta.opti <- Data2LDResult$thetastore
  TrayCoefList.opti <- modelVec2List(theta.opti, TrayCoefList)
  Data2LDList <- Data2LD(TrayDataList, TrayBasisList, TrayModelList,
```

```
                        TrayCoefList.opti, 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`

getForceTerm *Retrieve the specification of a forcing term in a linear differential equation.*

Description

A forcing term in a linear differential equation is a term on the right side of the equation that consists of an external variable that is multiplied by a coefficient function, and, possibly, is also multiplied by a fixed scalar constant. The coefficient function may be constant, or may vary with time itself, or with any external variable. It may not, however, depend on the value of either any of the variables in the system or on any derivative of any variable. Function `getHomoTerm` retrieves and returns as a named list the objects that specify the homogeneous term. The named list is detailed below.

Usage

```
getForceTerm(FtermList, coefList)
```

Arguments

`FtermList`	A named list containing the values defining the term, with fields and properties as follows: 1. `ncoef`: the the index in the coefficient list object of the variable in the term., 2. `factor`: a fixed scalar multiplier, such as -1, for the term. The default is 1. 3. `derivative`: the order of the derivative of the variable in the term. The default is 0.
`coefList`	A named list containing specifications all of the coefficient functions used in the model. These specifications can be set up using function `make.coef`.

Value

A named list containing the argument values.

Author(s)

J. O. Ramsay

References

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

See Also

`coefCheck`, `make.Coef`, `make.Fterm`, `make.Model` `make.Variable`, `make.Xterm`,

getHomoTerm *Retrieve the specification of a homogeneous term in a linear differential equation.*

Description

A homogeneous term in a linear differential equation is a term on the right side of the equation that consists of a derivative of a variable in one of the equations in the system that is multiplied by a coefficient function, and, possibly, is also multiplied by a fixed scalar constant. The coefficient function may be constant, or may vary with time itself, or with any external variable. It may not, however, depend on the value of either any of the variables in the system or on any derivative of any variable. Function `getHomoTerm` retrieves and returns as a named list the objects that specify the homogeneous term. The named list is detailed below.

Usage

```
getHomoTerm(XtermList, coefList)
```

Arguments

`XtermList`	A named list containing the values defining the term, with fields and properties as follows: 1. `variable`: the index in the model list object of the variable in the term. 2. `ncoef`: the the index in the coefficient list object of the variable in the term., 3. `factor`: a fixed scalar multiplier, such as -1, for the term. The default is 1. 4. `derivative`: the order of the derivative of the variable in the term. The default is 0.
`coefList`	A named list containing specifications all of the coefficient functions used in the model. These specifications can be set up using function `make.coef`.

Value

A named list containing the argument values.

Author(s)

J. O. Ramsay

References

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

See Also

`coefCheck`, `make.Coef`, `make.Fterm`, `make.Model` `make.Variable`, `make.Xterm`,

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.Coef *Check each specification in 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 thesystem. 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.Coef(funobj, parvec, estimate=TRUE)
```

Arguments

funobj	This is the definition coefficient function. This may be specified in two ways: most commonly as a functional data or a functional parameter object, or alternative by a list containing as members a function object for evaluating the function and a function object for evaluating its gradient with respect to the parameters defining it.
parvec	A numeric constant or vector of values of parameters defining the function defined by argument funobj.
estimate	A logical value indicating whether the parameters are to be estimated (TRUE) or regarded as fixed (FALSE).

Details

This function checks that all supplied conform to what is required. In the case where a user-defined coefficient function is provided, argument funobj is a list containing four named members: fd is a function object returning the value of the coefficient function, Dfd returns its gradient with respect to the parameters, theta is a numeric vector containing values for the parameters defining the function, and more is a list containing any other objects required by fd and Dfd.

Value

A named list object defining a coefficient object.

References

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

See Also

make.Xterm, make.Fterm, make.Variable, coefCheck

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
```

#

`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(ncoef, Ufd, factor=1, name=NULL)
```

Arguments

`ncoef`	An integer specifying the position of the coefficient specification in a coefficient list.
`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.
`name`	A name for the term, or for the forcing function itself.

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

`make.Coef`, `make.Model` `make.Variable`, `make.Xterm`,

Examples

```
#  For examples of the use of this function,
#  see these examples in the description of the function 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
#
```

`make.Model` *Check 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 checks that all linear differential equation specifications conform to what is required. Default values are supplied where needed. New named members are also supplied. 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

```
make.Model(XbasisList, variableList, coefList, report=TRUE)
```

Arguments

`XbasisList`	A list object containing the functional basis objects for representing the variables as functional data objects.
`variableList`	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.
`coefList`	A list object containing the specifications of one or more coefficient functions.
`report`	If TRUE, display a report on the structure of the dynamic system.

Details

A model list object containing variable specifications can be made manually, or it can set up in a by a single invocation of function `make.variable`. This function checks that all variable specifications encountered when processing a model List, whether or not the coefficients are to be estimated, are consistent with what is required to specify a linear differential equation. 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 `make.Model` is also invoked inside `Data2LD` therefore a use of `make.Model` is essential prior to using this function.

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

`coefCheck`, `make.Coef`, `make.Fterm`, `make.Variable`, `make.Xterm`,

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
#
```

`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)
```

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.

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

coefCheck

Examples

```
#
#  Example 1:  The refinery data
#
# The single equation requires two coefficients:
# 1. A constant coefficient for the speed of reaction
# 2. A constant coefficient for the single forcing function
```

```
#  Set up the observation times and range
TimeData <- RefineryData[,"Time"]
TimeRng  <- range(TimeData)
# DefIne a constant basis object
conbasis <- create.constant.basis(TimeRng)
#  Define the two coefficient functions and
#  store these in a coefficient function list
TrayCoefList <- vector("list",2)
# Both coefficients are estimated and both have initial value 0.
TrayCoefList[[1]] <- make.Coef(fun=conbasis, parvec=0, estimate=TRUE)
TrayCoefList[[2]] <- make.Coef(fun=conbasis, parvec=0, estimate=TRUE)
#  Run a check on the coefficient List array,
#  which also counts the number of estimated parameters
TraycoefResult <- coefCheck(TrayCoefList)
TrayCoefList   <- TraycoefResult$coefList
TrayNtheta     <- TraycoefResult$ntheta
print(paste("ntheta = ",TrayNtheta))
#  Define single homogeneous term with order zero derivative,
#  constant multiplier -1, and defined by the first coeffcient
XTerm <- make.Xterm(variable=1, derivative=0, ncoef=1, factor=-1, name="reaction")
XList <- vector("list", 1)
XList[[1]] <- XTerm
#  Define the step forcing function for the valve setting
Valvebreaks <- c(0,67,193)
Valvenbasis <- 2
Valvenorder <- 1
ValveBasis  <- create.bspline.basis(TimeRng, Valvenbasis, Valvenorder, Valvebreaks)
#  smooth the valve setting data to define the step forcing function
ValveData <- RefineryData[,"Valve.setting"]
Valvefd   <- smooth.basis(TimeData, ValveData, ValveBasis)$fd
#  Define the FList object for the single forcing function
FTerm <- make.Fterm(ncoef=2, Ufd=Valvefd, name="Valve")
FList <- vector("list", 1)
FList[[1]] <- FTerm
#  Define the single differential equation in the first order model
TrayVariable <- make.Variable(name="Tray level", order=1, XList=XList, FList=FList)
TrayList=vector("list",1)
TrayList[[1]] <- TrayVariable
#  Set up the basis list for the tray variable
# 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)
TrayBasisList    <- vector("list",1)
TrayBasisList[[1]] <- TrayBasis
#  check the object for internal consistency and make model list
TrayModelList <- make.Model(TrayBasisList, TrayList, TrayCoefList)
#
#  Example 2:  The head impact data
```

```
#
#  This is a second order model forced by a single pulse function
#  All coefficients are positive
HeadImpactTime <- HeadImpactData[,2]  #  time in milliseconds
HeadImpactRng  <- c(0,60) # Define range time for estimated model
# Set up the constant basis
conbasis <- create.constant.basis(HeadImpactRng)
# Define the three constant coefficient functions
coef1 <- make.Coef(fun=conbasis, parvec=0, estimate=TRUE)
coef2 <- make.Coef(fun=conbasis, parvec=0, estimate=TRUE)
coef3 <- make.Coef(fun=conbasis, parvec=0, estimate=TRUE)
#  Set up the coefficient list
coefList  <- vector("list",3)
coefList[[1]] <- coef1
coefList[[2]] <- coef2
coefList[[3]] <- coef3
#  check coefficient list
coefCheckList <- coefCheck(coefList)
coefList      <- coefCheckList$coefList
ntheta        <- coefCheckList$ntheta
print(paste("ntheta = ",ntheta))
# Define the two terms in the homogeneous part of the equation
Xterm0 <- make.Xterm(variable=1, derivative=0, ncoef=1, factor=-1, name="stiffness")
Xterm1 <- make.Xterm(variable=1, derivative=1, ncoef=2, factor=-1, name="viscosity")
# Set up the XList vector of length two
XList <- vector("list",2)
XList[[1]] <- Xterm0
XList[[2]] <- Xterm1
# Define a unit pulse function located at times 14-15
Pulsebasis <- create.bspline.basis(HeadImpactRng, 3, 1, c(0,14,15,60))
Pulsefd    <- fd(matrix(c(0,1,0),3,1),Pulsebasis)
# Define the forcing term
Fterm      <- make.Fterm(ncoef=3, Ufd=Pulsefd, factor=1, name="pulse")
# Set up the forcing term list of length one
FList      <- vector("list",1)
FList[[1]] <- Fterm
#  Define the single differential equation in the model
HeadImpactVariable <- make.Variable(order=2, XList=XList, FList=FList)
#  Define list of length one containing the equation definition
HeadImpactList=vector("list",1)
HeadImpactList[[1]] <- HeadImpactVariable
#  Construct the basis for representing the variable as a
#  functional data object
knots     <- c(0,14,14,14,15,15,seq(15,60,len=11))
norder    <- 6
nbasis    <- 21
HeadImpactBasis <- create.bspline.basis(HeadImpactRng,nbasis,norder,knots)
#  set up basis list
XbasisList <- vector("list",1)
XbasisList[[1]] <- HeadImpactBasis
#  check the object for internal consistency
HeadImpactList <- make.Model(XbasisList, HeadImpactList, coefList)
#
```

```
#  Example 3:  The X coordinate of the fda script data
#
#  The model requires a constant coefficient for the reaction speed
#  and a spline function for the coefficient of the forcing function,
#  which in this model is the unit function.
#  The estimated coefficient function is a step function,
#  or a spline function of order one, with 20 steps.
#  Define the observation times in 100ths of a second
centisec <- seq(0,2.3,len=1401)*100
fdarange <- range(centisec)
#  Define a constant basis
conbasis <- create.constant.basis(fdarange)
#  Define the order one Bspline basis for the coefficient
#  of the forcing term.
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)
#  Define the two coefficient  functions in this model
coef1 <- make.Coef(conbasis,  0.04,                TRUE)
coef2 <- make.Coef(Abasisobj, matrix(1,nAbasis,1), TRUE)
# List array containing the coefficient lists
coefList <- vector("list",2)
coefList[[1]] <- coef1
coefList[[2]] <- coef2
# Check the coefficients
coefResult <- coefCheck(coefList)
coefList   <- coefResult$coefList
ntheta     <- coefResult$ntheta
print(paste("ntheta = ",ntheta))
#  Set up single homogeneous term in D^2x = -beta x
Xterm <- make.Xterm(variable=1, derivative=0, ncoef=1, factor=-1,
                    name="reaction speed")
XList <- vector("list",1)
XList[[1]] <- Xterm
#  Set up coefficient for forcing term \alpha(t)*1
confd <- fd(1,conbasis)
Fterm <- make.Fterm(ncoef=2, Ufd=confd, factor=1, name="steps")
FList <- vector("list", 1)
FList[[1]] <- Fterm
#  make variable for X coefficient of script
Xvariable <- make.Variable(name="X", order=2, XList=XList, FList=FList)
#  List array for the whole system
fdaXList = vector("list",1)
fdaXList[[1]] <- Xvariable
#  Set up a list of basis objects for representing each
#  coordinate as a functional data object.
nbasis   <- 79
norder   <-  6 #  order 6 for a smooth 2nd deriv.
fdabasis <- create.bspline.basis(fdarange, nbasis, norder)
XbasisList <- vector("list",2)
```

```
XbasisList[[1]] <- fdabasis
XbasisList[[2]] <- fdabasis
#  check the system specification for consistency
fdaXList <- make.Model(XbasisList, fdaXList, coefList)
#
#  Example 4:  Average temperature in Montreal
#
#  In this example, a coefficient is nonconstant and also
#  a nonlinear function of its parameters.  The reaction speed
#  with which average temperature in Montreal responds to
#  solar forcing is required to be positive.
#  set up five-day block averages with winter centering
#  Here we use 73 five-day block averages as the data with the block
#  centers as the time points in order to speed up computation
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$dailyAv[,,"Temperature.C"]
tempav73 <- matrix(0,73,35)
m2 <- 0
for (  i in 1 : 73 ) {
  m1 <- m2 + 1
  m2 <- m2 + 5
  tempavi <- apply(tempav[m1:m2,1:35],2,mean)
  tempav73[i,] <- tempavi
}
#  center the time of the year on winter
winterind73  <- c( 37:73,1: 36)
tempav73 <- tempav73[winterind73,]
#  select the data for Montreal
station <- 12
#  Define the two forcing functions:
#  constant function Ufd for constant forcing
Uconbasis <- create.constant.basis(dayrange)
Uconfd    <- fd(1, Uconbasis)
Uconvec   <- matrix(1,73,1)
#  cosine function for solar radiation forcing
uvec      <- -cos((2*pi/365)*(daytime73+10+182))
Ucosbasis <- create.fourier.basis(dayrange, 3)
Ucosfd    <- smooth.basis(daytime73, uvec, Ucosbasis)$fd
#  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)
#  constant basis for the cosine 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 coefficient
coefListW  <-list(fun=linfun, parvec=matrix(0,nWbasis,1),
                  estimate=TRUE)
#  Coefficient for constant forcing
coefListA1 <- make.Coef(fun=AbasisobjC, parvec=1, estimate=TRUE)
#  Coeffcient for cosine forcing
coefListA2 <- make.Coef(fun=AbasisobjF, parvec=1, estimate=TRUE)
#  coefList constructed
coefList <- vector("list",3)
coefList[[1]] <- coefListW
coefList[[2]] <- coefListA1
coefList[[3]] <- coefListA2
#  check the coefficient list
coefResult <- coefCheck(coefList)
coefList <- coefResult$coefList
ntheta   <- coefResult$ntheta
print(paste("ntheta = ",ntheta))
#  define homogeneous term and list container
Xterm <- make.Xterm(variable=1, derivative=0, ncoef=1, factor=-1,
                    name="reaction speed")
XList <- vector("list", 1)
XList[[1]] <- Xterm
#  define two forcing terms
Fterm1 <- make.Fterm(Ufd=Uconfd, ncoef=2, factor=1, name="constant")
Fterm2 <- make.Fterm(Ufd=Ucosfd, ncoef=3, factor=1, name="cosine")
#  forcing term container
FList <- vector("list", 2)
FList[[1]] <- Fterm1
FList[[2]] <- Fterm2
#  model list object
TempVar <- make.Variable(name="temperature", order=1, XList=XList, FList=FList)
#  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)
dailyBasisList <- vector("list",1)
dailyBasisList[[1]] <- daybasis
# set up dailyList
dailyModelList <- vector("list",1)
dailyModelList[[1]] <- TempVar
dailyModelList <- make.Model(dailyBasisList, dailyModelList, coefList)
```

`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(variable, ncoef, derivative=0, factor=1, name=NULL)
```

Arguments

variable	An integer specifying the variable within which this is a term.
ncoef	An integer specifying the position of the coefficient specification in a coefficient list.
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.
name	A name for the term, such as "reaction" for the term involving the variable.

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.Coef, make.Fterm, make.Model 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
#
```

`modelList2Vec` *Extract a parameter vector from a modelList 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, coefList)
```

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.
`coefList`	A list object containing the specifications of one or more coefficient functions.

Details

A coefficient specification can be made manually, or can set up in a by a single invocation of function `make.coef`. Model specifications can also be set manually, or by an invocation of function `make.variable` for each linear equation in the system.

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(thetavec, coefList)
```

Arguments

`thetavec`	A numeric vector containing values for all the parameters in a linear differential system that requires estimation.
`coefList`	A list object containing the specifications of one or more coefficient functions.

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`

`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