

Package: funcreg (via r-universe)

September 25, 2026

Version 0.0-1

Date 2016-10-18

Title Functional Regression Estimation

Author Pierre Chausse <pchausse@uwaterloo.ca>

Maintainer Pierre Chausse <pchausse@uwaterloo.ca>

Description This package is complementary to the fda package. It includes function to choose smoothing parameters and to estimate models with multiple functional regressors.

Depends R (>= 3.2.5), fda, lattice

URL <https://codeberg.org/pchausse/funcreg>
<https://pchausse.r-universe.dev/funcreg>

BugReports <https://codeberg.org/pchausse/funcreg/issues>

Imports stats, methods, graphics

License GPL (>= 2)

Config/pak/sysreqs make

Repository <https://pchausse.r-universe.dev>

Date/Publication 2026-09-25 14:10:37 UTC

RemoteUrl <https://codeberg.org/pchausse/funcreg>

RemoteRef HEAD

RemoteSha 42abb7190c40ac0f5767b5e35a3710526245e23f

Contents

coefEst	2
crossval	4
fdaCV	6
funcreg	8
GDPv56	11
getLandK	14

myfdamethod	16
plotCoef	17
simData	19
vcov	20

Index	22
--------------	-----------

coefEst	<i>Functional Data Fit</i>
---------	----------------------------

Description

These are the main functions to fit a functional data object to a dataset. The model can be linear or nonlinear. The smoothing parameter and the number of basis is assumed to be known.

Usage

```
coefEst(y, t, lam, k, L=2,
        create_basis=create.bspline.basis, ...)

nlCoefEst(y, t, lam, k, L=2,
          create_basis=create.bspline.basis, maxit=100, tol=1e-8,
          typels=c("brent","grid"), addDat=FALSE, rangeval=NULL, ...)

makeLinFda(obj, npoints=200)

logFda(obj)

## S3 method for class 'myfda'
as.fd(x, fdnames=NULL, npoints=200, ...)

## S3 method for class 'myfda'
fitted(object, t=NULL, ...)
```

Arguments

y	A $T \times n$ matrix of data points, when T is the number of periods and n the number of individuals.
t	A numeric vector of time units. In <code>coefEst</code> and <code>nlCoefEst</code> , the number of elements is equal to the number of rows of <code>y</code> . For the <code>fitted</code> method, it can be a fine grid, and the range must be inside the range of the basis.
lam	The regularization parameter that penalizes for the absence of smoothness.
k	The number of basis.
L	Either a nonnegative integer defining an order of a derivative or a linear differential operator (see eval.penalty).
create_basis	The function used to create the basis object (see create.bspline.basis).

maxit	The maximum number of iteration for the Newton method
tol	The tolerance parameter for the stopping rule of the Newton method
...	Other argument that is passed to create_basis (see create.bspline.basis), or to other methods in the case of <code>as.fda.myfda</code> .
obj	Object of class "myfda"
npoints	Number of points to fit the linear FDA to the non-linear one
x	Object of class "myfda"
object	Object of class "myfda"
fdnames	Names of elements of the "fd" object (see fd)
typels	The nlCoefEst is based on Newton with line search. The argument refers to the method used for the line search
addDat	If TRUE, fake observations are added before and after assuming stationarity. for the estimation, the time span is expanded accordingly.
rangeval	This can be set if the basis is defined on a range other than <code>range(t)</code> .

Details

In `coefEst`, the functional data is a linear combination of the basis, while in `nlCoefEst` it is the exponential of the linear combination of the basis. The latter is used when we want to force the functional data curve to be positive,

`makeLinFda` will transform the nonlinear fit to a linear FDA by fitting the nonlinear curve to a linear combination of the same basis. The curve is almost identical and it is transformed this way by [funcreg](#)

`logFDA` transforms the non-linear FDA into a linear FDA. It only redefine the link equation and take the log of the observed data.

Value

It returns an object of 'class' "myfda"

The object of class "myfda" is a list containing at least:

coefficients	$k \times n$ matrix of coefficients.
y	The dataset matrix y.
t	The input vector t.
basis	The basis object
link	The function that links y to the linear combination of the basis.
lambda	The value of λ .
convergence	Information on the convergence of the Newton method.

References

Ramsay, James O., & Silverman, Bernard W. (2005), *Functional Data Analysis*, Springer, New York.

Examples

```
data(GDPv56)
t <- seq(0,1,len=nrow(GDPv56))

## Linear estimation
res <- coefEst(y=GDPv56, t=t, lam=0.003, k=15)
res

cbind(data=GDPv56[,1], fitted=fitted(res)[,1])

## Nonlinear Estimation
res <- nlCoefEst(y=GDPv56, t=t, lam=0.003, k=15)
res

resLin <- makeLinFda(res)
resLin

## Take the logarithm of the non-linear functional data
logres <- logFda(res)
plot(res,1:2)

## Transform the myfda object into an fd object
resfd <- as.fd(resLin)

## Nonlinear Estimation with zeros
data(simData)
t <- seq(0,1,length.out=5)
res <- nlCoefEst(y=simData, t=t, lam=0.003, k=7)
res
res <- nlCoefEst(y=simData, t=t, lam=0.003, k=7, addDat=TRUE)
res
```

crossval

Leave-one-out cross-validation

Description

It computes the cross-validation for a given lambda and k or find the optimal lambda for a given k (see below).

Usage

```
funcregCV(form, create_basis=create.bspline.basis, LD=2, lambda,
           k, CstInt=FALSE, regularized=TRUE, alpha=1e-5, data=NULL,
           obj=NULL, ...)

getFuncregLam(form, create_basis=create.bspline.basis, LD=2, lam0,
```

```
k, regularized=TRUE, CstInt=FALSE, data=NULL, loglam=FALSE,
method="BFGS", alpha=1e-5, optimArg=list(), ...)
```

Arguments

obj	An object of class "funcreg". If NULL, all other arguments must be set.
form	A formula for the functional regression. Arguments are objects of class "myfda" obtained either by <code>coefEst</code> or <code>nlCoefEst</code> , and must be in the local environment.
create_basis	The function used to create the basis object (see <code>create.bspline.basis</code>) for the functional parameters.
LD	Either a nonnegative integer defining an order of a derivative or a linear differential operator (see <code>eval.penalty</code>).
data	An optional list of myfda objects, which are the functional data included in the formula form
lambda	A vector of regularization parameters that penalizes for the absence of smoothness. The dimension is 2 times the number of regressors plus 1 if there is an intercept and if <code>CstInt=FALSE</code> . The first is for the intercept, the second and third for the first regressor and so on. If it is a scalar, it is assumed that the smoothing parameter is the same for all functional parameters.
lam0	Initial value of lambda for the minimization of the cross-validation
k	A vector of integers representing the number of basis for each functional parameter. The first is for the intercept (if there is an intercept and that <code>CstInt=FALSE</code>), the second for the first functional parameter and so on. If it is a scalar, it is assumed that the number of basis is the same for all functional parameters.
regularized	If TRUE, the system is regularized by add alpha to the main diagonal of the regression matrix
CstInt	If TRUE, the intercept is assumed constant. If FALSE, it is a one dimensional functional parameter.
loglam	If TRUE, lam0 is $\log_{10}(\lambda)$ and the function minimizes $CV(10^{\lambda})$. It prevents the lambda's from being negative.
method	The algorithm used by <code>link{optim}</code>
optimArg	A list of additional arguments to pass to <code>link{optim}</code>
alpha	The regularization parameter. The value is added to the diagonal of the regression matrix to make sure it is positive definite.
...	Additional arguments to be passed to <code>create_basis</code> (see <code>create.bspline.basis</code>).

Value

'funregCV' returns the value of the leave-one-out cross-validation. The attributes 'convergence', returns convergence code for each estimation.

'getFuncregLam' returns the fitted object of class 'funcreg' using the optimal lambda. The argument 'optimRes' gives the `optim` output from the minimization of the cross-validation.

References

Ramsay, James O., & Silverman, Bernard W. (2005), *Functional Data Analysis*, Springer, New York.

Examples

```
data(GDPv56)

## We just create response and a covariate artificially from the GDP
## series

y <- GDPv56[,1:30]
x <- GDPv56[,31:60]
t <- seq(0,1,len=nrow(y))

## First we create the "myfda" objects
yfd <- coefEst(y, t, .0004, 15, 2)
xfd <- coefEst(x, t, .0004, 15, 2)

## with the object
res <- funcreg(yfd~xfd, k=c(5,5), lambda=c(.001,.001,.001))
funcregCV(obj=res)

## without object
funcregCV(yfd~xfd, k=c(5,5), lambda=c(.001,.001,.001))

## With the default values
getFuncregLam(yfd~xfd, k=c(5,5), lam0=c(.001,.001,.001))

## With upper and lower bound
getFuncregLam(yfd~xfd, k=c(5,5), lam0=c(.001,.001,.001), method="L-BFGS-B",
              optimArg=list(lower=c(0,0,0),upper=c(200,200,200)))

## With loglam
res <- getFuncregLam(yfd~xfd, k=c(5,5), lam0=c(-3,-3,-3), loglam=TRUE)
res

## Get info about the optim()
res$optimRes
```

Description

The functions compute the leave-one-out cross-validation of functional data. It applies also to already estimated models.

Usage

```
nlFdaCV(y, t, lamvec, kvec, L=2,
        create_basis=create.bspline.basis, maxit=100, tol=1e-8,
        obj=NULL, typels=c("brent", "grid"), addDat=FALSE, ...)

fdaCV(y, t, lamvec, kvec, L=2,
      create_basis=create.bspline.basis, obj=NULL, ...)
```

Arguments

<code>y</code>	A $T \times n$ matrix of data points, when T is the number of periods and n the number of individuals.
<code>t</code>	A numeric vector of time units. The number of elements is equal to the number of rows of <code>y</code> . It must have a range inside <code>rangeval</code> .
<code>lamvec</code>	A vector of regularization parameters that penalizes for the absence of smoothness.
<code>kvec</code>	A vector of integers that indicates the number of basis.
<code>L</code>	Either a nonnegative integer defining an order of a derivative or a linear differential operator (see eval.penalty).
<code>create_basis</code>	The function used to create the basis object (see create.bspline.basis).
<code>maxit</code>	The maximum number of iteration for the Newton method
<code>tol</code>	The tolerance parameter for the stopping rule of the Newton method
<code>obj</code>	Object of class "myfda" if we want the cross-validation of an already fitted model
<code>typels</code>	The <code>nlCoefEst</code> is based on Newton with line search. The argument refers to the method used for the line search
<code>addDat</code>	If TRUE, fake observations are added before and after assuming stationarity. for the estimation, the time span is expanded accordingly.
<code>...</code>	Other argument that is passed to <code>create_basis</code> (see create.bspline.basis).

Details

It returns a matrix of cross-validations. The rows are for the lambda, and the columns for the number of basis.

Value

`fdaCV` returns a matrix of cross-validations. The rows are for the lambda, and the columns for the number of basis.

`nlFdaCV` returns a list with the following items:

<code>cv</code>	The cross-validation matrix (see details)
<code>convergence</code>	A matrix returning TRUE if there was no convergence problem while computing the cross-validation.

kvec	The vector of integers that indicates the number of basis.
lamvec	The vector of regularization parameters.
info	An array of convergence codes for all estimations in the process of computing the cross-validations

References

Ramsay, James O., & Silverman, Bernard W. (2005), *Functional Data Analysis*, Springer, New York.

Examples

```
data(GDPv56)
t <- seq(0,1,len=nrow(GDPv56))

## Linear estimation
#####

# From an object
res <- coefEst(y=GDPv56, t=t, lam=0.003, k=15)
fdaCV(obj=res)

# CV with the estimation
cv <- fdaCV(y=GDPv56,t,10^seq(-5,1,len=4), k=5:7)
cv

## Nonlinear Estimation
#####

res <- nlCoefEst(y=GDPv56, t=t, lam=0.003, k=10)
nlFdaCV(obj=res)

# CV with the estimation
cv <- nlFdaCV(y=GDPv56,t,10^seq(-5,1,len=4), k=5:7)

## Nonlinear Estimation with zeros
#####

data(simData)
t <- seq(0,1,length.out=5)
cv <- nlFdaCV(y=simData, t=t, lam=c(1e-5, 1e-2), k=c(6:8), addDat=TRUE)
print(cv, conv=TRUE)
```

Description

Estimation of the functional parameters in a multiple functional regression. The smoothing parameters and the number of basis is assumed to be known.

Usage

```
funcreg(form, create_basis=create.bspline.basis, LD=2, lambda,
        k, regularized=TRUE, CstInt=FALSE, CV=FALSE, data=NULL,
        alpha=1e-5, optAlpha=FALSE,
        controlAlpha=list(), ...)

## S3 method for class 'funcreg'
print(x, ...)

## S3 method for class 'funcreg'
summary(object, ...)

## S3 method for class 'funcreg'
fitted(object, ...)

## S3 method for class 'funcreg'
residuals(object, ...)
```

Arguments

x	An object of class "funcreg"
object	An object of class "funcreg"
form	A formula for the functional regression. Arguments are objects of class "myfda" obtained either by <code>coefEst</code> or <code>nlCoefEst</code> , and must be in the local environment.
create_basis	The function used to create the basis object (see <code>create.bspline.basis</code>) for the functional parameters.
LD	Either a nonnegative integer defining an order of a derivative or a linear differential operator (see <code>eval.penalty</code>).
lambda	A vector of regularization parameters that penalizes for the absence of smoothness. The dimension is 2 times the number of regressors plus 1 if there is an intercept and if <code>CstInt=FALSE</code> . The first is for the intercept, the second and third for the first regressor and so on. If it is a scalar, it is assumed that the smoothing parameter is the same for all functional parameters.
k	A vector of integers representing the number of basis for each functional parameter. The first is for the intercept (if there is an intercept and that <code>CstInt=FALSE</code>), the second for the first functional parameter and so on. If it is a scalar, it is assumed that the number of basis is the same for all functional parameters.
regularized	If TRUE, the system is regularized by add alpha to the main diagonal of the regression matrix
CstInt	If TRUE, the intercept is assumed constant. If FALSE, it is a one dimensional functional parameter.

CV	if TRUE, it computes the cross-validation.
data	An optional list of myfda objects, which are the functional data included in the formula form
alpha	The regularization parameter. The value is added to the diagonal of the regression matrix to make sure it is positive definite.
optAlpha	If TRUE, the regularization parameter is the one that minimizes the cross-validation using the Brent method
controlAlpha	A list of tuning parameter for the choice of the optimal alpha. The default parameters are <code>maxit=100</code> , <code>tol=1e-5</code> , and <code>intAlpha=c(1e-11, 1e-4)</code> .
...	For <code>funcreg</code> , it is other argument that is passed to <code>create_basis</code> (see create.bspline.basis). For the methods, it is the other arguments to pass to other objects.

Value

It returns an object of 'class' "funcreg"

The object of class "funcreg" is a list containing at least:

X	The list of functional regressors.
Y	The functional response.
data	A list with element "t" for time units of the observed data and "Y" for the matrix of observed response variable.

References

Ramsay, James O., & Silverman, Bernard W. (2005), *Functional Data Analysis*, Springer, New York.

Examples

```
data(GDPv56)

## We just create response and a covariate artificially from the GDP
## series

y <- GDPv56[,1:30]
x <- GDPv56[,31:60]
t <- seq(0,1,len=nrow(y))

## First we create the "myfda" objects
yfd <- coefEst(y, t, .0004, 15, 2)
xfd <- coefEst(x, t, .0004, 15, 2)

## we just set lambda and k to arbitrary values
res <- funcreg(yfd~xfd, k=c(5,5), lambda=c(.001,.001,.001))
res

summary(res)
```

```

plot(fitted(res))

plot(residuals(res))

## Getting the optimal regularization parameter
res <- funcreg(yfd~xfd, k=c(5,5), lambda=c(.001,.001,.001),
              optAlpha=TRUE)
res

```

GDPv56

International Gross Domestic Product

Description

The dataset includes GDP's of 90 countries from 1960 to 1980. The frequency is yearly.

Usage

```
data(GDPv56)
```

Format

A matrix containing 90 columns, one per country. Dates are reported as rownames().

ARGENTINA GDP of ARGENTINA

AUSTRALIA GDP of AUSTRALIA

AUSTRIA GDP of AUSTRIA

BURUNDI GDP of BURUNDI

BELGIUM GDP of BELGIUM

BENIN GDP of BENIN

BURKINA FASO GDP of BURKINA FASO

BOLIVIA GDP of BOLIVIA

BRAZIL GDP of BRAZIL

CANADA GDP of CANADA

SWITZERLAND GDP of SWITZERLAND

CHILE GDP of CHILE

COTE D'IVOIRE GDP of COTE D'IVOIRE

CAMEROON GDP of CAMEROON

CONGO REPUBLIC OF GDP of CONGO REPUBLIC OF

COLOMBIA GDP of COLOMBIA

COMOROS GDP of COMOROS

CAPE VERDE GDP of CAPE VERDE
COSTA RICA GDP of COSTA RICA
DENMARK GDP of DENMARK
DOMINICAN REPUBLIC GDP of DOMINICAN REPUBLIC
ALGERIA GDP of ALGERIA
ECUADOR GDP of ECUADOR
EGYPT GDP of EGYPT
SPAIN GDP of SPAIN
FINLAND GDP of FINLAND
FRANCE GDP of FRANCE
GABON GDP of GABON
UNITED KINGDOM GDP of UNITED KINGDOM
GHANA GDP of GHANA
GUINEA GDP of GUINEA
GAMBIA THE GDP of GAMBIA THE
GUINEA-BISSAU GDP of GUINEA-BISSAU
GREECE GDP of GREECE
GUATEMALA GDP of GUATEMALA
HONG KONG GDP of HONG KONG
HONDURAS GDP of HONDURAS
INDONESIA GDP of INDONESIA
INDIA GDP of INDIA
IRELAND GDP of IRELAND
IRAN GDP of IRAN
ICELAND GDP of ICELAND
ISRAEL GDP of ISRAEL
ITALY GDP of ITALY
JAMAICA GDP of JAMAICA
JORDAN GDP of JORDAN
JAPAN GDP of JAPAN
KENYA GDP of KENYA
KOREA REPUBLIC OF GDP of KOREA REPUBLIC OF
SRI LANKA GDP of SRI LANKA
LESOTHO GDP of LESOTHO
LUXEMBOURG GDP of LUXEMBOURG
MOROCCO GDP of MOROCCO
MADAGASCAR GDP of MADAGASCAR

MEXICO GDP of MEXICO
MALI GDP of MALI
MOZAMBIQUE GDP of MOZAMBIQUE
MAURITIUS GDP of MAURITIUS
MALAWI GDP of MALAWI
MALAYSIA GDP of MALAYSIA
NIGERIA GDP of NIGERIA
NICARAGUA GDP of NICARAGUA
NETHERLANDS GDP of NETHERLANDS
NORWAY GDP of NORWAY
NEW ZEALAND GDP of NEW ZEALAND
PAKISTAN GDP of PAKISTAN
PANAMA GDP of PANAMA
PERU GDP of PERU
PHILIPPINES GDP of PHILIPPINES
PORTUGAL GDP of PORTUGAL
PARAGUAY GDP of PARAGUAY
RWANDA GDP of RWANDA
SENEGAL GDP of SENEGAL
SINGAPORE GDP of SINGAPORE
EL SALVADOR GDP of EL SALVADOR
SWEDEN GDP of SWEDEN
SYRIA GDP of SYRIA
CHAD GDP of CHAD
TOGO GDP of TOGO
THAILAND GDP of THAILAND
TRINIDAD AND TOBAGO GDP of TRINIDAD AND TOBAGO
TURKEY GDP of TURKEY
TAIWAN GDP of TAIWAN
UGANDA GDP of UGANDA
URUGUAY GDP of URUGUAY
UNITED STATES GDP of UNITED STATES
VENEZUELA GDP of VENEZUELA
SOUTH AFRICA GDP of SOUTH AFRICA
ZAMBIA GDP of ZAMBIA
ZIMBABWE GDP of ZIMBABWE

Source

TBA

getLandK

*Cross-Validation***Description**

Several functions that computes the leave-one-out cross-validation of functional data. They include cases of linear and nonlinear fit.

Usage

```
nlGetLam(y, t, lamInt, k, L=2,
         create_basis=create.bspline.basis, maxit=100, tol=1e-8,
         maxitalgo=100, tolalgo=1e-7, type=c("Brent", "GS"),
         typels=c("brent", "grid"), addDat=FALSE, ...)

nlGetLandKopt(y, t, lamInt, kvec, L=2,
              create_basis=create.bspline.basis, maxit=100, tol=1e-7,
              maxitalgo=100, tolalgo=1e-6, type=c("Brent", "GS"),
              typels=c("brent", "grid"), addDat=FALSE, ...)

nlGetLandK(y, t, lamvec, kvec, L=2,
           create_basis=create.bspline.basis, maxit=100, tol=1e-8,
           typels=c("brent", "grid"), addDat=FALSE, ...)

getLam(y, t, lamInt, k, L=2,
       create_basis=create.bspline.basis,
       maxitalgo=100, tolalgo=1e-7, type=c("Brent", "GS"), ...)

getLandK(y, t, lamvec, kvec, L=2,
         create_basis=create.bspline.basis, ...)

getLandKopt(y, t, lamInt, kvec, L=2,
            create_basis=create.bspline.basis,
            maxitalgo=100, tolalgo=1e-6, type=c("Brent", "GS"), ...)
```

Arguments

y	A $T \times n$ matrix of data points, when T is the number of periods and n the number of individuals.
t	A numeric vector of time units. The number of elements is equal to the number of rows of y. It must have a range inside rangeval.
lamvec	A vector of regularization parameters that penalizes for the absence of smoothness.
lamInt	A vector that contains the lower and upper bounds for Lambda that is passed to the Brent or Golden Section method.
kvec	A vector of integers that indicates the number of basis.

k	A scalar that indicates the number of basis.
L	Either a nonnegative integer defining an order of a derivative or a linear differential operator (see eval.penalty).
create_basis	The function used to create the basis object (see create.bspline.basis).
maxit	The maximum number of iteration for the Newton method
tol	The tolerance parameter for the stopping rule of the Newton method
maxitalgo	Maximum number of iteration for the brent and Golden-Section methods
tolalgo	Tolerance level for the stopping rule of the Brent or Golden-Section method.
type	Which algorithm should we use to get the lambda.
typels	The nlCoefEst is based on Newton with line search. The argument refers to the method used for the line search
addDat	If TRUE, fake observations are added before and after assuming stationarity. for the estimation, the time span is expanded accordingly.
...	Other argument that is passed to create_basis

(see [create.bspline.basis](#)).

Details

Function that starts with 'nl' are the nonlinear functional data (see [coefEst](#)).

getLam and nlGetLam, finds the Lambda that minimizes the cross-validation for a given k using either the Brent method or the Golden-Section method.

nlGetLandK and getLandK find the optimal Lambda and k using a grid search. The grid is provided by the user through lamvec and kvec.

getLandKopt and nlGetLandKopt find the optimal Lambda and k using a grid search for k only. Lambda is obtained using the Brent or Golden-Section method.

Value

getLam and nlGetLam returns a list with the following items:

lam	The optimal Lambda
info	A convergence code for the optimization method (0 for normal convergence).
iter	The number of iterations for the Brent or Golden-Section method.
cv	The minimum cross-validation

getLandK, codenlGetLandK, getLandKopt, and codenlGetLandKopt, returns an object of class "myfda" (see [coefEst](#)). In addition to the usual element, the list includes information about the search.

References

Ramsay, James O., & Silverman, Bernard W. (2005), *Functional Data Analysis*, Springer, New York.

Examples

```

data(GDPv56)
t <- seq(0,1,len=nrow(GDPv56))

## Linear estimation
#####

# Using a grid
nlam <- 5
lamvec <- 10^seq(-5,1,length.out=nlam)
kvec <- 5:7
res1 <- getLandK(GDPv56, t, lamvec, kvec)
res1

# Using Brent
res2 <- getLandKopt(GDPv56, t, lamvec, kvec, tolalgo=1e-5)
res2

## Nonlinear estimation
#####

# Using a grid
res3 <- nlGetLandK(GDPv56, t, lamvec, kvec)
res3

# Using Brent
res4 <- nlGetLandKopt(GDPv56, t, lamvec, kvec, tolalgo=1e-5)
res3

```

myfdamethod

Methods for Functional Data Fit

Description

These are the plot and print methods which are associated with the objects of class "myfda".

Usage

```

## S3 method for class 'myfda'
plot(x, which=NULL, addpoints=TRUE, npoints=100,
     add=FALSE, ...)

## S3 method for class 'myfda'
print(x, ...)

## S3 method for class 'myfdaCV'
print(x, digits=4, conv=FALSE, ...)

```

```
## S3 method for class 'myfda'
x[i=NULL]
```

Arguments

x	Object of class "myfda" (see coefEst).
which	Which curve shall we plot
addpoints	If TRUE and which is not null, observations are added to the graph
npoints	The number of points used to plot the curves
digits	The number of digits to be printed
conv	Shall the information about convergence be printed
add	If TRUE, the line associated with "which" is added to the existing graph
i	The selection index.
...	Other argument that is passed to the plot or print methods

Examples

```
data(GDPv56)
t <- seq(0,1,len=nrow(GDPv56))

## Linear estimation
res <- coefEst(y=GDPv56, t=t, lam=0.003, k=15)
print(res)
plot(res)
plot(res,which=1:3)

## Using the operator []
plot(res[1:4])

res2 <- coefEst(y=GDPv56, t=t, lam=0.3, k=15)
plot(res,which=1)
plot(res2,which=1, add=TRUE, col=2, lty=3)
```

Description

It plot the estimated functional coefficients with confidence bands. The coefficients are estimated by [funcreg](#).

Usage

```
plotCoef(obj, which, n = c(50, 50), fixeds=NULL, fixedt=NULL, plotWF=TRUE, ...)

plotConfInt(obj, which, type = c("beta_t", "beta_s", "beta_st"),
            n = c(50, 50), level = 0.95, plotWF = TRUE, beta=NULL)
```

Arguments

obj	An object of class "funcreg"
which	Which functional coefficient we want to plot. 0 for the intercept, 1 for the first functional slope and so on.
n	The number of grid points for the plot. The first element is for the t dimension and the second for the s dimension.
fixeds	If NULL, a three dimensional plot is produced. Alternatively, we can set a value for s and plot beta(t,s) for a given s. It has no effect when which=0.
fixedt	If NULL, a three dimensional plot is produced. Alternatively, we can set a value for t and plot beta(t,s) for a given t. It has no effect when which=0.
plotWF	If FALSE, the wireframe is not shown. In that case, only the output of the function is returned. Only applicable for three dimensional plot.
type	The type of plot (see details).
level	The level for the confidence interval
beta	A one or two argument function (see details).
...	Other argument. to pass to either plot or wireframe.

Details

If type="beta_t", the default, the integral with respect to s is shown with pointwise confidence bands. It is simply the functional intercept with its pointwise confidence bands if which=0.

If type="beta_s" (not applicable for which=0), the integral with respect to t is shown with its pointwise confidence bands.

If type="beta_st" (not applicable for which=0), the whole three dimensional curve is shown with its pointwise confidence surfaces.

If beta is provided, the estimated curve is replaced by the one produced by the function beta, but the same confidence bands are produced. Useful for pointwise hypothesis testing. If type="beta_s" or type="beta_t", beta must be a function of t or s. If type="beta_st", it must be a function of both.

Value

If plotWF=FALSE and the expected plot is a surface, it returns an object of class "trellis" produced by [wireframe](#).

References

Ramsay, James O., & Silverman, Bernard W. (2005), *Functional Data Analysis*, Springer, New York.

Examples

```

data(GDPv56)

## We just create response and a covariate artificially from the GDP
## series

y <- GDPv56[,1:30]
x <- GDPv56[,31:60]
t <- seq(0,1,len=nrow(y))

## First we create the "myfda" objects
yfd <- coefEst(y, t, .0004, 15, 2)
xfd <- coefEst(x, t, .0004, 15, 2)

## we just set lambda and k to arbitrary values
res <- funcreg(yfd~xfd, k=c(5,5), lambda=c(.001,.001,.001))

plotCoef(res, 1, type="beta_st")

plotConfInt(res, 1, type="beta_t")

## Assume we what to do pointwise hypothesis testing
plotConfInt(res, 1, type="beta_t", beta=function(t) .6-.7*t)

## Nice 3-D confidence interval
plotConfInt(res, 1, type="beta_st")

```

simData

Simulated panel with some zeros

Description

This dataset is used to show how the nonlinear fit can be used when some observations are zero.

Usage

```
data(simData)
```

Format

A matrix containing 293 columns, and 5 time units.

vcov

*Variance of functional coefficients***Description**

It computes the functional variance of the estimated functional parameters obtained by [funcreg](#)

Usage

```
## S3 method for class 'funcreg'
vcov(object, which, type = c("beta", "beta_t", "beta_s", "beta_st"),
      s = NULL, t = NULL, ...)
```

Arguments

<code>object</code>	An object of class "funcreg"
<code>which</code>	The variance of which functional parameters shall we compute? 0 for the intercept, 1 for the first functional slope and so on.
<code>type</code>	What type of variance to compute (see details).
<code>s</code>	If type is "beta_s", the variance is computed at each element of s. If NULL, it is computed at the same time units as for the observed observations.
<code>t</code>	If type is "beta_t", the variance is computed at each element of s. If NULL, it is computed at the same time units as for the observed observations.
<code>...</code>	Argument to pass to other objects

Details

The function computed the variance of the integral of the functional parameters (single for the intercept and double for the slopes) when type='beta'.

If type='beta_t', it is the pointwise variance of the functional intercept if which=0 and the pointwise variance of the integral of the slopes which respect to s. By default (t=NULL) the time units of the observed data is used. Alternatively, we can provide a vector of points.

If type='beta_s', (not available for which=0) it is the pointwise variance of the integral of the slopes which respect to y. By default (s=NULL), equally spaced grid points are generated using the range of s-basis and the number of time units equals to the one from the observed observations.

If type='beta_st' (not available for which=0), a two dimensional pointwise variance is computed.

For the last three cases, vcov returns either a one dimension functional data object or a two-dimensional one.

Value

It returns a list containing at least:

<code>V</code>	Either a numeric value when type='beta' or an 'fda' object when type='beta_t' or type='beta_s', or a matrix when type='beta_st'.
----------------	--

References

Ramsay, James O., & Silverman, Bernard W. (2005), *Functional Data Analysis*, Springer, New York.

Examples

```
data(GDPv56)

## We just create response and a covariate artificially from the GDP
## series

y <- GDPv56[,1:30]
x <- GDPv56[,31:60]
t <- seq(0,1,len=nrow(y))

## First we create the "myfda" objects
yfd <- coefEst(y, t, .0004, 15, 2)
xfd <- coefEst(x, t, .0004, 15, 2)

## we just set lambda and k to arbitrary values
res <- funcreg(yfd~xfd, k=c(5,5), lambda=c(.001,.001,.001))

vcov(res, which=0)

plot(vcov(res, 1, type="beta_t")$V)
```

Index

* datasets

- GDPv56, [11](#)
- simData, [19](#)
- [.myfda (myfdamethod), [16](#)
- as.fd.myfda (coefEst), [2](#)
- coefEst, [2](#), [5](#), [9](#), [15](#), [17](#)
- create.bspline.basis, [2](#), [3](#), [5](#), [7](#), [9](#), [10](#), [15](#)
- crossval, [4](#)
- eval.penalty, [2](#), [5](#), [7](#), [9](#), [15](#)
- fd, [3](#)
- fdaCV, [6](#)
- fitted.funcreg (funcreg), [8](#)
- fitted.myfda (coefEst), [2](#)
- funcreg, [3](#), [8](#), [17](#), [20](#)
- funcregCV (crossval), [4](#)
- GDPv56, [11](#)
- getFuncregLam (crossval), [4](#)
- getLam (getLandK), [14](#)
- getLandK, [14](#)
- getLandKopt (getLandK), [14](#)
- logFda (coefEst), [2](#)
- makeLinFda (coefEst), [2](#)
- myfdamethod, [16](#)
- nlCoefEst, [5](#), [9](#)
- nlCoefEst (coefEst), [2](#)
- nlFdaCV (fdaCV), [6](#)
- nlGetLam (getLandK), [14](#)
- nlGetLandK (getLandK), [14](#)
- nlGetLandKopt (getLandK), [14](#)
- optim, [5](#)
- plot.myfda (myfdamethod), [16](#)

- plotCoef, [17](#)
- plotConfInt (plotCoef), [17](#)
- print.funcreg (funcreg), [8](#)
- print.myfda (myfdamethod), [16](#)
- print.myfdaCV (myfdamethod), [16](#)
- residuals.funcreg (funcreg), [8](#)
- simData, [19](#)
- summary.funcreg (funcreg), [8](#)
- vcov, [20](#)
- wireframe, [18](#)