

# Package ‘refund’

February 15, 2012

**Type** Package

**Title** Regression with Functional Data

**Version** 0.1-5

**Date** 2011-10-18

**Author** Ciprian Crainiceanu <ccrainic@jhsph.edu> and Philip Reiss  
<phil.reiss@nyumc.org> (coordinating authors)

**Contributors** Jeff Goldsmith <jgoldsmi@jhsph.edu>, Sonja Greven  
<sonja.greven@stat.uni-muenchen.de>, Lei Huang  
<huangracer@gmail.com>, and Fabian Scheipl <fabian.scheipl@stat.uni-muenchen.de>

**Maintainer** Lei Huang <huangracer@gmail.com>

**Depends** R (>= 2.9.0), mgcv (>= 1.7-8), fda

**Description** Functions for regression with functional data. The principal methods currently implemented regress (i) scalar responses on functional predictors; (ii) functional responses on scalar predictors; and (iii) functional responses on functional predictors.

**License** GPL (>= 2)

**LazyLoad** yes

**Repository** CRAN

**Date/Publication** 2011-10-20 15:35:10

## R topics documented:

refund-package	2
amc	3
coef.pffr	4
DTI	5
expand.call	6

ff . . . . .	7
fitted.pffr . . . . .	8
fosr . . . . .	9
fosr.perm . . . . .	11
fpcr . . . . .	14
gasoline . . . . .	16
lofocv . . . . .	17
lpfr . . . . .	18
model.matrix.pffr . . . . .	20
pffr . . . . .	21
pfr . . . . .	23
plot.fosr . . . . .	26
plot.fpcr . . . . .	27
plot.pffr . . . . .	28
predict.pffr . . . . .	29
print.summary.pffr . . . . .	30
pwcv . . . . .	31
residuals.pffr . . . . .	32
summary.pffr . . . . .	33

<b>Index</b>	<b>34</b>
--------------	-----------

---

refund-package	<i>Regression with Functional Data</i>
----------------	--

---

## Description

Functions for regression with functional data. The methods currently implemented regress (i) scalar responses on functional predictors (`pfr` [Goldsmith et al., 2011], its longitudinal extension `lpfr` [Goldsmith et al., 2010], and `fpcr` [Reiss and Ogden, 2007]); (ii) functional responses on scalar predictors (`fosr` [Reiss et al., 2010]); and (iii) functional responses on functional predictors (`pffr` [Ivanescu et al., 2011]).

## Details

For a complete list of functions type `library(help=refund)`.

## Author(s)

Coordinating authors:

Ciprian Crainiceanu <ccrainic@jhspsh.edu>

Philip Reiss <phil.reiss@nyumc.org>

Contributors:

Jeff Goldsmith <jgoldsmi@jhspsh.edu>

Sonja Greven <sonja.greven@stat.uni-muenchen.de>

Lei Huang <huangracer@gmail.com>

Fabian Scheipl <fabian.scheipl@stat.uni-muenchen.de>

Maintainer: Lei Huang <huangracer@gmail.com>

## References

Goldsmith, J., Bobb, J., Crainiceanu, C., Caffo, B., and Reich, D. (2011). Penalized functional regression. *Journal of Computational and Graphical Statistics*, to appear.

Goldsmith, J., Crainiceanu, C., Caffo, B., and Reich, D. (2012). Longitudinal penalized functional regression for cognitive outcomes on neuronal tract measurements. *Journal of the Royal Statistical Society: Series C*, to appear.

Ivanescu, A. E., Staicu, A.-M., Greven, S., Scheipl, F., and Crainiceanu, C. M. (2011). Penalized function-on-function regression. Submitted.

Reiss, P. T., Huang, L., and Mennes, M. (2010). Fast function-on-scalar regression with penalized basis expansions. *International Journal of Biostatistics*, 6(1), article 28. Available at [http://works.bepress.com/phil\\_reiss/16/](http://works.bepress.com/phil_reiss/16/)

Reiss, P. T., and Ogden, R. T. (2007). Functional principal component regression and functional partial least squares. *Journal of the American Statistical Association*, 102, 984–996.

---

amc

*Additive model with constraints*

---

## Description

An internal function, called by `fosr()`, that fits additive models with linear constraints via a call to the `gam()` function in the `mgcv` package.

## Usage

```
amc(y, Xmat, S, gam.method = "REML", C = NULL, lambda = NULL)
```

## Arguments

<code>y</code>	response vector.
<code>Xmat</code>	design matrix.
<code>S</code>	list of penalty matrices.
<code>gam.method</code>	smoothing parameter selection method: "REML" for restricted maximum likelihood, "GCV.Cp" for generalized cross-validation.
<code>C</code>	matrix of linear constraints. Dimension should be number of constraints times <code>ncol(Xmat)</code> .
<code>lambda</code>	smoothing parameter value. If NULL, the smoothing parameter(s) will be estimated.

**Value**

A list with the following elements:

gam	the gam object returned by gam().
coefficients	coefficients with respect to design matrix Xmat, derived from the gam() fit.
Vp, GinvXt	outputs used by fosr().
method	the gam.method argument of the call to amc().

**Author(s)**

Philip Reiss <phil.reiss@nyumc.org>

**See Also**

[fosr](#)

---

coef.pffr

*Get estimated coefficients from a pffr*

---

**Description**

Returns estimated coefficient functions/surfaces  $\beta(t)$ ,  $\beta(s, t)$  and estimated smooth effects  $f(z)$ ,  $f(x, z)$  or  $f(x, z, t)$  and their standard errors. Not implemented for smooths in more than 3 dimensions.

**Usage**

```
## S3 method for class 'pffr'
coef(object, raw = FALSE, se = TRUE, freq
      = FALSE, n1 = 100, n2 = 40, n3 = 20, ...)
```

**Arguments**

object	a fitted pffr-object
raw	logical, defaults to FALSE. If TRUE, the function simply returns object\$coefficients
se	logical, defaults to TRUE. Return estimated standard error of the estimates?
freq	logical, defaults to FALSE. If FALSE, use posterior variance object\$Vp for variability estimates, else use object\$Ve. See <a href="#">gamObject</a>
n1, n2, n3	these parameters give the number of gridpoints for 1-/2-/3-dimensional smooth terms for the marginal equidistant grids over the range of the covariates at which the estimated effects are evaluated.
...	other arguments, not used.

**Value**

If `raw==FALSE`, a list containing

- `pterms` a matrix containing the parametric / non-functional coefficients (and, optionally, their `se`'s)
- `smterms` a named list with one entry for each smooth term in the model. Each entry contains
  - `coef` a matrix giving the grid values over the covariates, the estimated effect (and, optionally, the `se`'s). The first covariate varies the fastest.
  - `x`, `y`, `z` the unique gridpoints used to evaluate the smooth/coefficient function/coefficient surface
  - `xlim`, `ylim`, `zlim` the extent of the `x/y/z`-axes
  - `xlab`, `ylab`, `zlab` the names of the covariates for the `x/y/z`-axes
  - `dim` the dimensionality of the effect
  - `main` the label of the smooth term (a short label, same as the one used in `summary.pffr`)

**Author(s)**

Fabian Scheipl

---

DTI

*Diffusion Tensor Imaging: tract profiles and outcomes*

---

**Description**

Fractional anisotropy tract profiles for the corpus callosum (`cca`) and the right corticospinal tract (`rcst`). Accompanying the tract profiles are the subject ID numbers, visit number, total number of scans, multiple sclerosis case status and Paced Auditory Serial Addition Test (`pasat`) score. We thank Dr. Daniel Reich for making this dataset available.

**Usage**

`data(DTI)`

**Format**

A data frame made up of

`cca` A 382 x 93 matrix of fractional anisotropy tract profiles from the corpus callosum;  
`rcst` A 382 x 55 matrix of fractional anisotropy tract profiles from the right corticospinal tract;  
`ID` Numeric vector of subject ID numbers;  
`visit` Numeric vector of the subject-specific visit numbers;  
`Nscans` Numeric vector indicating the total number of visits for each subject;  
`case` Numeric vector of multiple sclerosis case status: 0 - healthy control, 1 - MS case;  
`pasat` Numeric vector containing the PASAT score at each visit.

## References

Goldsmith, J., Bobb, J., Crainiceanu, C., Caffo, B., and Reich, D. (2011). Penalized functional regression. *Journal of Computational and Graphical Statistics*, to appear.

Goldsmith, J., Crainiceanu, C., Caffo, B., and Reich, D. (2012). Longitudinal penalized functional regression for cognitive outcomes on neuronal tract measurements. *Journal of the Royal Statistical Society: Series C*, to appear.

---

expand.call	<i>Return call with all possible arguments</i>
-------------	--

---

## Description

Return a call in which all of the arguments which were supplied or have presets are specified by their full names and their supplied or default values.

## Usage

```
expand.call(definition = NULL, call =  
sys.call(sys.parent(1)), expand.dots = TRUE)
```

## Arguments

definition	a function. See <a href="#">match.call</a> .
call	an unevaluated call to the function specified by definition. See <a href="#">match.call</a> .
expand.dots	logical. Should arguments matching ... in the call be included or left as a ... argument? See <a href="#">match.call</a> .

## Value

An object of mode "[call](#)".

## Author(s)

Fabian Scheipl

## See Also

[match.call](#)

---

ff *Construct a function-on-function regression term*

---

### Description

Defines a term  $\int_{s_{lo,i}}^{s_{hi,i}} X_i(s)\beta(t,s)ds$  for inclusion in an `mgcv::gam`-formula (or `bam` or `gamm` or `gamm4::gamm`) as constructed by `pfpr`. Defaults to a cubic tensor product B-spline with marginal first difference penalties for  $\beta(t,s)$  and integration over the entire range  $[s_{lo,i}, s_{hi,i}] = [\min(s_i), \max(s_i)]$ . Can't deal with any missing  $X(s)$ , unequal lengths of  $X_i(s)$  not (yet?) possible. Unequal ranges for different  $X_i(s)$  should work.  $X_i(s)$  is assumed to be numeric.

### Usage

```
ff(X, yind, xind = seq(0, 1, l = ncol(X)), basistype =
c("te", "t2", "s"), integration = c("simpson",
"trapezoidal"), L = NULL, limits = NULL, splinepars =
list(bs = "ps", m = c(2, 1)))
```

### Arguments

X	an n by ncol(xind) matrix of function evaluations $X_i(s_{i1}), \dots, X_i(s_{iS}); i = 1, \dots, n$ .
yind	matrix (or vector) of indices of evaluations of $Y_i(t)$
xind	matrix (or vector) of indices of evaluations of $X_i(s)$ ; i.e. matrix with rows $(s_{i1}, \dots, s_{iS})$
basistype	defaults to "te", i.e. a tensor product spline to represent $\beta(t,s)$ . Alternatively, use "s" for bivariate basis functions (see <code>mgcv</code> 's <code>s</code> ) or "t2" for an alternative parameterization of tensor product splines (see <code>mgcv</code> 's <code>t2</code> ).
integration	method used for numerical integration. Defaults to "simpson"'s rule for calculating entries in L. Alternatively and for non-equidistant grids, "trapezoidal".
L	optional: an n by ncol(xind) matrix giving the weights for the numerical integration over $s$ .
limits	(NOT YET IMPLEMENTED)
splinepars	optional arguments supplied to the basistype-term. Defaults to a cubic tensor product B-spline with marginal first difference penalties, i.e. <code>list(bs="ps", m=c(2, 1))</code> See <code>te</code> or <code>s</code> in <code>mgcv</code> for details

### Value

a list containing

- call a "call" to `te` (or `s`, `t2`) using the appropriately constructed covariate and weight matrices
- data a list containing the necessary covariate and weight matrices

**Author(s)**

Fabian Scheipl, Sonja Greven

**See Also**

mgcv's [linear.functional.terms](#)

---

fitted.pffr

*Obtain fitted values for a pffr fit*

---

**Description**

Returns the linear predictor for the model data. See [predict.pffr](#) for alternative options.

**Usage**

```
## S3 method for class 'pffr'  
fitted(object, reformat = TRUE, ...)
```

**Arguments**

object	a fitted pffr-object
reformat	logical, defaults to TRUE. Should residuals be returned in $n \times y$ index matrix form (default) or in the long vector shape returned by <code>fitted.default()</code> ?
...	not used

**Value**

A matrix or vector of fitted values

**Author(s)**

Fabian Scheipl

---

fosr *Function-on-scalar regression*

---

**Description**

Fit linear regression with functional responses and scalar predictors, with efficient selection of optimal smoothing parameters.

**Usage**

```
fosr(fdobj, Z, L = NULL, eval.pts = seq(min(fdobj$basis$range),
max(fdobj$basis$range), length.out = 201), method = "OLS",
gam.method = "REML", lambda = NULL, multi.sp = FALSE, max.iter = 1,
maxlam = NULL, cv1 = FALSE, scale = FALSE)
```

**Arguments**

<code>fdobj</code>	a functional data object (class <code>fd</code> ) giving the functional responses.
<code>Z</code>	the model matrix, whose columns represent scalar predictors.
<code>L</code>	a row vector or matrix of linear contrasts of the coefficient functions, to be restricted to equal zero.
<code>eval.pts</code>	argument values at which the coefficient functions will be evaluated.
<code>method</code>	estimation method: "OLS" for penalized ordinary least squares, "GLS" for penalized generalized least squares.
<code>gam.method</code>	smoothing parameter selection method, to be passed to <code>gam()</code> : "REML" for restricted maximum likelihood, "GCV.Cp" for generalized cross-validation.
<code>lambda</code>	smoothing parameter value. If <code>NULL</code> , the smoothing parameter(s) will be estimated. See Details.
<code>multi.sp</code>	a logical value indicating whether separate smoothing parameters should be estimated for each coefficient function. Currently must be <code>FALSE</code> if <code>method = "OLS"</code> .
<code>max.iter</code>	maximum number of iterations if <code>method = "GLS"</code> .
<code>maxlam</code>	maximum smoothing parameter value to consider (when <code>lamvec=NULL</code> ; see <a href="#">lofocv</a> )
<code>cv1</code>	logical value indicating whether a cross-validation score should be computed even if a <code>fixedlambda</code> is specified (when <code>method = "OLS"</code> ).
<code>scale</code>	logical value or vector determining scaling of the matrix <code>Z</code> (see <a href="#">scale</a> , to which the value of this argument is passed).

**Details**

There are three types of values for argument `lambda`:

1. if `NULL`, the smoothing parameter is estimated by `gam` (package `mgcv`) if `method = "GLS"`, or by `optimize` if `method = "OLS"`;

2. if a scalar, this value is used as the smoothing parameter (but only for the initial model, if `method = "GLS"`);
3. if a vector, this is used as a grid of values for optimizing the cross-validation score (provided `method = "OLS"`; otherwise an error message is issued).

Please note that currently, if `multi.sp = TRUE`, then `lambda` must be `NULL` and `method` must be `"GLS"`.

### Value

An object of class `fosr`, which is a list with the following elements:

<code>B</code>	matrix of basis coefficients for the estimated coefficient functions.
<code>yhat</code>	an object of class <code>fd</code> giving the fitted values for the functional responses.
<code>est.func</code>	matrix of values of the coefficient function estimates at the points given by <code>eval.pts</code> .
<code>se.func</code>	matrix of values of the standard error estimates for the coefficient functions, at the points given by <code>eval.pts</code> .
<code>eval.pts</code>	points at which the coefficient functions are evaluated.
<code>fit</code>	fit object outputted by <code>amc()</code> .
<code>edf</code>	effective degrees of freedom of the fit.
<code>lambda</code>	smoothing parameter, or vector of smoothing parameters.
<code>cv</code>	cross-validated integrated squared error if <code>method="OLS"</code> , otherwise <code>NULL</code> .
<code>roughness</code>	value of the roughness penalty.

### Author(s)

Philip Reiss <[phil.reiss@nyumc.org](mailto:phil.reiss@nyumc.org)>

### References

Ramsay, J. O., and Silverman, B. W. (2005). *Functional Data Analysis*, 2nd ed., Chapter 13. New York: Springer.

Reiss, P. T., Huang, L., and Mennes, M. (2010). Fast function-on-scalar regression with penalized basis expansions. *International Journal of Biostatistics*, 6(1), article 28. Available at [http://works.bepress.com/phil\\_reiss/16/](http://works.bepress.com/phil_reiss/16/)

### See Also

[plot.fosr](#)

**Examples**

```

# The first two lines, adapted from help(fRegress) in package fda,
# set up a functional data object representing daily average
# temperatures at 35 sites in Canada
daybasis25 <- create.fourier.basis(rangeval=c(0, 365), nbasis=25,
                                axes=list('axesIntervals'))
Temp.fd <- with(CanadianWeather, smooth.basisPar(day.5,
                                                dailyAv[,,'Temperature.C'], daybasis25)$fd)

modmat = cbind(1, model.matrix(~ factor(CanadianWeather$region) - 1))
constraints = matrix(c(0,1,1,1,1), 1)

# Penalized OLS with smoothing parameter chosen by grid search
olsmod = foser(Temp.fd, modmat, constraints, method="OLS", lambda=100*10:30)
plot(olsmod, 1)

# Penalized GLS
glsmod = foser(Temp.fd, modmat, constraints, method="GLS")
plot(glsmod, 1)

```

foser.perm

*Permutation testing for function-on-scalar regression***Description**

foser.perm() is a wrapper function calling foser.perm.fit(), which fits models to permuted data, followed by foser.perm.test(), which performs the actual simultaneous hypothesis test. Calling the latter two functions separately may be useful for performing tests at different significance levels. By default, foser.perm() produces a plot using the plot function for class foser.perm.

**Usage**

```

foser.perm(yfdobj, Z, L = NULL, Z0 = NULL, L0 = NULL,
          eval.pts = seq(min(yfdobj$basis$range), max(yfdobj$basis$range),
                        length.out = 201),
          lambda = NULL, lambda0 = NULL, multi.sp = FALSE, nperm,
          level = 0.05, plot = TRUE, xlabel = "",
          title = NULL, prelim = 15, ...)

foser.perm.fit(yfdobj, Z, L = NULL, Z0 = NULL, L0 = NULL,
              eval.pts = seq(min(yfdobj$basis$range),
                            max(yfdobj$basis$range), length.out = 201),
              lambda = NULL, lambda0 = NULL, multi.sp = FALSE,
              nperm, prelim, ...)

foser.perm.test(x, level=.05)

## S3 method for class 'foser.perm'
plot(x, level = .05, xlabel = "", title = NULL, ...)

```

**Arguments**

<code>yfdobj</code>	a functional data object (class <code>fd</code> ) giving the functional responses.
<code>Z</code>	the design matrix, whose columns represent scalar predictors.
<code>L</code>	a row vector or matrix of linear contrasts of the coefficient functions, to be restricted to equal zero.
<code>Z0</code>	design matrix for the null-hypothesis model. If <code>NULL</code> , the null hypothesis is the intercept-only model.
<code>L0</code>	linear constraints for the null-hypothesis model.
<code>eval.pts</code>	argument values at which the coefficient functions will be evaluated.
<code>lambda</code>	smoothing parameter value. If <code>NULL</code> , the smoothing parameter(s) will be estimated. See <a href="#">fosr</a> for details.
<code>lambda0</code>	smoothing parameter for null-hypothesis model.
<code>multi.sp</code>	a logical value indicating whether separate smoothing parameters should be estimated for each coefficient function. Currently must be <code>FALSE</code> if <code>method = "OLS"</code> .
<code>nperm</code>	number of permutations.
<code>prelim</code>	number of preliminary permutations. The smoothing parameter in the main permutations will be fixed to the median value from these preliminary permutations. If <code>prelim=0</code> , this is not done.
<code>level</code>	significance level for the simultaneous test.
<code>plot</code>	logical value indicating whether to plot the real- and permuted-data pointwise F-type statistics.
<code>xlabel</code>	x-axis label for plots.
<code>title</code>	title for plot.
<code>x</code>	object of class <code>fosr.perm</code> , outputted by <code>fosr.perm</code> , <code>fosr.perm.fit</code> , or <code>fosr.perm.test</code> .
<code>...</code>	for <code>fosr.perm</code> and <code>fosr.perm.fit</code> , additional arguments passed to <a href="#">fosr</a> . These arguments may include <code>max.iter</code> , <code>method</code> , <code>gam.method</code> , and <code>scale</code> . For <code>plot.fosr.perm</code> , graphical parameters (see <a href="#">par</a> ) for the plot.

**Value**

`fosr.perm` or `fosr.perm.test` produces an object of class `fosr.perm`, which is a list with the elements below. `fosr.perm.fit` also outputs an object of this class, but without the last five elements.

<code>F</code>	pointwise F-type statistics at each of the points given by <code>eval.pts</code> .
<code>F.perm</code>	a matrix, each of whose rows gives the pointwise F-type statistics for a permuted data set.
<code>eval.pts</code>	points at which F-type statistics are computed.
<code>lambda.real</code>	smoothing parameter(s) for the real-data fit.
<code>lambda.prelim</code>	smoothing parameter(s) for preliminary permuted-data fits.
<code>lambda.perm</code>	smoothing parameter(s) for main permuted-data fits.

lambda0.real, lambda0.prelim, lambda0.perm	as above, but for null hypothesis models.
level	significance level of the test.
critval	critical value for the test.
signif	vector of logical values indicating whether significance is attained at each of the points eval.pts.
n2s	subset of $\{1, \dots, \text{length}(\text{eval.pts})\}$ identifying the points at which the test statistic changes from non-significant to significant.
s2n	points at which the test statistic changes from significant to non-significant.

**Author(s)**

Philip Reiss <phil.reiss@nyumc.org>

**References**

Reiss, P. T., Huang, L., and Mennes, M. (2010). Fast function-on-scalar regression with penalized basis expansions. *International Journal of Biostatistics*, 6(1), article 28. Available at [http://works.bepress.com/phil\\_reiss/16/](http://works.bepress.com/phil_reiss/16/)

**See Also**

[fosr](#)

**Examples**

```
# Test effect of region on mean temperature in the Canadian weather data
# The next two lines are taken from the fRegress.CV help file (package fda)
smallbasis <- create.fourier.basis(c(0, 365), 25)
tempfd <- smooth.basis(day.5,
  CanadianWeather$dailyAv[,,"Temperature.C"], smallbasis)$fd

Zreg = cbind(1, model.matrix(~factor(CanadianWeather$region)-1))
Lreg = matrix(c(0,1,1,1,1), 1) # constrain region effects to sum to 0

# This is for illustration only; for a real test, must increase nperm
# (and probably prelim as well)
regionperm = fosr.perm(tempfd, Zreg, Lreg, method="OLS", nperm=10, prelim=3)

# Redo the plot, using axisIntervals() from the fda package
plot(regionperm, axes=FALSE, xlab="")
box()
axis(2)
axisIntervals(1)
```

fpcr

*Functional principal component regression***Description**

Implements functional principal component regression (Reiss and Ogden, 2007, 2010) for generalized linear models with scalar responses and functional predictors.

**Usage**

```
fpcr(y, sigmat = NULL, basismat = NULL, penmats = NULL,
     fdobj = NULL, argvals = NULL, nc = NULL, covt = NULL,
     mean.signal.term = FALSE, nbasis = NULL,
     spline.order = NULL, pen.order = NULL,
     family = gaussian(), method="REML", sp=NULL, ...)
```

**Arguments**

<code>y</code>	scalar outcome vector.
<code>sigmat</code>	matrix of signals: $n \times N$ , where $n$ is the length of <code>y</code> and $N$ is the number of sites at which each signal is defined.
<code>basismat</code>	$N \times K$ matrix of values of $K$ basis functions at the $N$ sites
<code>penmats</code>	a list, each of whose components is a $K \times K$ matrix defining a penalty on the basis coefficients.
<code>fdobj</code>	functional data object (class <code>fd</code> ) giving the functional predictors.
<code>argvals</code>	points at which the signals (functional predictors) and the coefficient function are evaluated. By default, if the functional predictors are given by the $n \times N$ matrix <code>sigmat</code> , <code>argvals</code> is set to $N$ equally spaced points from 0 to 1; if they are given by <code>fdobj</code> , <code>argvals</code> is set to 401 equally spaced points spanning the domain of the given functions.
<code>nc</code>	number of principal components. By default, the smallest number accounting for at least 99% of the variance in the predictors.
<code>covt</code>	covariates: an $n$ -row matrix, or a vector of length $n$ .
<code>mean.signal.term</code>	logical: should the mean of each subject's signal be included as a covariate?
<code>nbasis</code>	number of basis functions. Ignored if <code>fdobj</code> is supplied. If <code>fdobj</code> is <i>not</i> supplied, this defaults to 40, i.e., 40 B-spline basis functions are used.
<code>spline.order</code>	order of B-splines used, if <code>fdobj</code> is not supplied; defaults to 4, i.e., cubic B-splines.
<code>pen.order</code>	order of derivative penalty applied when estimating the coefficient function; defaults to 2.
<code>family</code>	generalized linear model family.

method	smoothing parameter selection method, passed to function <code>gam</code> in package <code>mgcv</code> ; see the <code>gam</code> documentation for details.
sp	a fixed smoothing parameter; if <code>NULL</code> , an optimal value is chosen (see <code>method</code> ).
...	other arguments passed to function <code>gam</code> in package <code>mgcv</code> .

### Details

The functional predictors are given either in functional data object form, using argument `fdobj` (see the `fda` package of Ramsay, Hooker and Graves, 2009, and Method 1 in the example below), or explicitly, using `sigmat` (see Method 2 in the example). In the latter case, arguments `basismat` and `penmats` can also be used to specify the basis and/or penalty matrices (see Method 3).

### Value

An object of class `gam` (see `gamObject` in the `mgcv` package documentation), with additional components `nc` and `argvals` (same as the arguments of the same names) as well as `fhat` (coefficient function estimate) and `se` (pointwise Bayesian standard error).

### Author(s)

Philip Reiss <[phil.reiss@nyumc.org](mailto:phil.reiss@nyumc.org)>

### References

- Ramsay, J. O., Hooker, G., and Graves, S. (2009). *Functional Data Analysis with R and MATLAB*. New York: Springer.
- Reiss, P. T., and Ogden, R. T. (2007). Functional principal component regression and functional partial least squares. *Journal of the American Statistical Association*, 102, 984–996.
- Reiss, P. T., and Ogden, R. T. (2010). Functional generalized linear models with images as predictors. *Biometrics*, 66, 61–69.
- Wood, S. N. (2006). *Generalized Additive Models: An Introduction with R*. Boca Raton, FL: Chapman & Hall.

### Examples

```
##### Octane data example #####
data(gasoline)

# Create the requisite functional data objects
bbasis = create.bspline.basis(c(900, 1700), 40)
wavelengths = 2*450:850
nir = matrix(NA, 401, 60)
for (i in 1:60) nir[, i] = gasoline$NIR[i, ]
# Why not just take transpose of gasoline$NIR above?
# Because for some reason it leads to an error in the following statement
gas.fd = smooth.basisPar(wavelengths, nir, bbasis)$fd

# Method 1: Call fpcr with fdobj argument
gasmod1 = fpcr(gasoline$octane, fdobj = gas.fd, nc = 30)
```

```
# Method 2: Call fpcr with explicit signal matrix
gasmod2 = fpcr(gasoline$octane, sigmat = gasoline$NIR, nc = 30)
# Method 3: Call fpcr with explicit signal, basis, and penalty matrices
gasmod3 = fpcr(gasoline$octane, sigmat = gasoline$NIR,
               basismat = eval.basis(wavelengths, bbasis),
               penmats = list(getbasispenalty(bbasis)), nc = 30)

plot(gasmod1, xlab='Wavelength')

# Check that all 3 calls yield essentially identical estimates
all.equal(gasmod1$fhat, gasmod2$fhat, gasmod3$fhat)
# But note that, in general, you'd have to specify argvals in Method 1
# to get the same coefficient function values as with Methods 2 & 3.
```

---

gasoline

*Octane numbers and NIR spectra of gasoline*

---

### Description

Near-infrared reflectance spectra and octane numbers of 60 gasoline samples. Each NIR spectrum consists of  $\log(1/\text{reflectance})$  measurements at 401 wavelengths, in 2-nm intervals from 900 nm to 1700 nm. We thank Prof. John Kalivas for making this data set available.

### Usage

```
data(gasoline)
```

### Format

A data frame comprising

octane a numeric vector of octane numbers for the 60 samples.

NIR a 60 x 401 matrix of NIR spectra.

### Source

Kalivas, John H. (1997). Two data sets of near infrared spectra. *Chemometrics and Intelligent Laboratory Systems*, 37, 255–259.

### References

For applications of functional principal component regression to this data set:

Reiss, P. T., and Ogden, R. T. (2007). Functional principal component regression and functional partial least squares. *Journal of the American Statistical Association*, 102, 984–996.

Reiss, P. T., and Ogden, R. T. (2009). Smoothing parameter selection for a class of semiparametric linear models. *Journal of the Royal Statistical Society, Series B*, 71(2), 505–523.

### See Also

[fpcr](#)

---

lofocv	<i>Leave-one-function-out cross-validation</i>
--------	--

---

### Description

This internal function, called by `foser()` when `method="OLS"`, performs efficient leave-one-function-out cross-validation using Demmler-Reinsch orthogonalization to choose the smoothing parameter.

### Usage

```
lofocv(Y, X, S1, lamvec = NULL, constr = NULL, maxlam = NULL)
```

### Arguments

Y	matrix of responses, e.g. with columns corresponding to basis function coefficients.
X	model matrix.
S1	penalty matrix.
lamvec	vector of candidate smoothing parameter values. If NULL, smoothing parameter is chosen by <a href="#">optimize</a> .
constr	matrix of linear constraints.
maxlam	maximum smoothing parameter value to consider (when lamvec=NULL).

### Value

if lamvec=NULL, a list (returned by [optimize](#)) with elements `minimum` and `objective` giving, respectively, the chosen smoothing parameter and the associated cross-validation score. Otherwise a 2-column table with the candidate smoothing parameters in the first column and the corresponding cross-validation scores in the second.

### Author(s)

Philip Reiss <[phil.reiss@nyumc.org](mailto:phil.reiss@nyumc.org)>

### See Also

[foser](#), [pwcv](#)

**Description**

Implements longitudinal penalized functional regression (Goldsmith et al., 2010) for generalized linear functional models with scalar outcomes and subject-specific random intercepts.

**Usage**

```
lpfr(Y, subj, covariates = NULL, funcs, kz = 30, kb = 30,
      smooth.cov = FALSE, family = "gaussian", method = "REML", ...)
```

**Arguments**

Y	Vector of all outcomes over all visits
subj	Vector containing the subject number for each observation
covariates	Matrix of scalar covariates
funcs	Matrix or list of matrices containing observed functional predictors as rows. NA values are allowed.
kz	Dimension of principal components basis for the observed functional predictors
kb	Dimension of the truncated power series spline basis for the coefficient function
smooth.cov	Logical; do you wish to smooth the covariance matrix of observed functions? Increases computation time, but results in smooth principal components
family	Generalized linear model family
method	Method for estimating the smoothing parameters; defaults to REML
...	Additional arguments passed to the gam function to fit the regression model.

**Details**

Functional predictors are entered as a matrix or, in the case of multiple functional predictors, as a list of matrices using the funcs argument. Missing values are allowed in the functional predictors, but it is assumed that they are observed over the same grid. Functional coefficients and confidence bounds are returned as lists in the same order as provided in the funcs argument, as are principal component and spline bases.

**Value**

fit	The result of the call to gam
fitted.vals	Predicted outcomes
betaHat	List of estimated coefficient functions
beta.covariates	Parameter estimates for scalar covariates
ranef	Vector of subject-specific random intercepts

X	Design matrix used in the model fit
phi	List of truncated power series spline bases for the coefficient functions
psi	List of principal components basis for the functional predictors
varBetaHat	List containing covariance matrices for the estimated coefficient functions
Bounds	List of bounds of a 95% confidence interval for the estimated coefficient functions

### Author(s)

Jeff Goldsmith <jgoldsmi@jhsph.edu>

### References

Goldsmith, J., Crainiceanu, C., Caffo, B., and Reich, D. (2012). Longitudinal penalized functional regression for cognitive outcomes on neuronal tract measurements. *Journal of the Royal Statistical Society: Series C*, to appear.

### Examples

```
#####
# use longitudinal data to regress continuous outcomes on
# functional predictors (continuous outcomes only recorded for
# case == 1)
#####

data(DTI)

# subset data as needed for this example
cca = DTI$cca[which(DTI$case == 1),]
rcst = DTI$rcst[which(DTI$case == 1),]
DTI = DTI[which(DTI$case == 1),]

# note there is missingness in the functional predictors
apply(is.na(cca), 2, mean)
apply(is.na(rcst), 2, mean)

# fit two models with single functional predictors and plot the results
fit.cca = lpfr(Y=DTI$pasat, subj=DTI$ID, funcs = cca, smooth.cov=FALSE)
fit.rcst = lpfr(Y=DTI$pasat, subj=DTI$ID, funcs = rcst, smooth.cov=FALSE)

par(mfrow = c(1,2))
matplot(cbind(fit.cca$BetaHat[[1]], fit.cca$Bounds[[1]]),
        type = 'l', lty = c(1,2,2), col = c(1,2,2), ylab = "BetaHat",
        main = "CCA")
matplot(cbind(fit.rcst$BetaHat[[1]], fit.rcst$Bounds[[1]]),
        type = 'l', lty = c(1,2,2), col = c(1,2,2), ylab = "BetaHat",
        main = "RCST")
```

```
# fit a model with two functional predictors and plot the results
fit.cca.rcst = lpfr(Y=DTI$pasat, subj=DTI$ID, funcs = list(cca,rcst),
  smooth.cov=FALSE)

par(mfrow = c(1,2))
matplot(cbind(fit.cca.rcst$BetaHat[[1]], fit.cca.rcst$Bounds[[1]]),
  type = 'l', lty = c(1,2,2), col = c(1,2,2), ylab = "BetaHat",
  main = "CCA")
matplot(cbind(fit.cca.rcst$BetaHat[[2]], fit.cca.rcst$Bounds[[2]]),
  type = 'l', lty = c(1,2,2), col = c(1,2,2), ylab = "BetaHat",
  main = "RCST")
```

---

model.matrix.pffr      *Obtain model matrix for a pffr fit*

---

### Description

Obtain model matrix for a pffr fit

### Usage

```
## S3 method for class 'pffr'
model.matrix(object, ...)
```

### Arguments

object            a fitted pffr-object  
...                other arguments, passed to [predict.gam](#).

### Value

A model matrix

### Author(s)

Fabian Scheipl

pffr

*Penalized function-on-function regression***Description**

Implements additive regression for functional and scalar covariates and functional responses. This function is a wrapper for mgcv's [gam](#) and its siblings to fit models of the general form

$$E(Y_i(t)) = g(\mu(t) + \int X_i(s)\beta(s,t)ds + f(z_{1i}, t) + f(z_{2i}) + z_{3i}\beta_3(t) + \dots)$$

with a functional (but not necessarily continuous) response  $Y(t)$ , response function  $g$ , (optional) smooth intercept  $\mu(t)$ , (multiple) functional covariates  $X(t)$  and scalar covariates  $z_1, z_2$ , etc.

**Usage**

```
pffr(formula, yind, fitter = NA, method = "REML",
     bsy.default = list(bs = "ps", m = c(2, 1)), ...)
```

**Arguments**

formula	a formula with special terms as for <a href="#">gam</a> , with additional special terms <a href="#">ff()</a> and <a href="#">c()</a>
yind	a vector with length equal to the number of columns of the matrix of functional responses giving the vector of evaluation points $(t_1, \dots, t_G)$ . If formula contains an <a href="#">ff</a> -term which specifies yind this is used. If neither is given, yind is <code>1:ncol(&lt;response&gt;)</code> .
fitter	the name of the function used to estimate the model. Defaults to <a href="#">gam</a> if the matrix of functional responses has less than 2e5 data points and to <a href="#">bam</a> if not. "gamm" (see <a href="#">gamm</a> ) and "gamm4" (see <a href="#">gamm4</a> ) are valid options as well.
method	Defaults to "REML"-estimation, including of unknown scale. See <a href="#">gam</a> for details.
bsy.default	a named (!) list giving the parameters for spline bases on the index of the functional response. Defaults to <code>list(bs="ps", m=c(2, 1))</code> , i.e. a cubic B-spline basis with first order difference penalty. Only arguments <code>bs</code> , <code>k</code> , <code>m</code> are used, see <a href="#">s</a> for details.
...	additional arguments that are valid for <a href="#">gam</a> or <a href="#">bam</a> . <code>weights</code> , <code>subset</code> , <code>offset</code> are not yet implemented!

**Value**

a fitted pffr-object, which is a [gam](#)-object with some additional information in an pffr-entry. If fitter is "gamm" or "gamm4", only the \$gam part of the returned list is modified in this way.

**Details**

The routine can estimate

1. (nonlinear, and possibly multivariate) effects of (one or multiple) scalar covariates  $z$  that vary smoothly over the index  $t$  of  $Y(t)$  (e.g.  $f(z_{1i}, t)$ , specified in the formula simply as `~s(z1)`),

2. (nonlinear) effects of scalar covariates that are constant over  $t$  (e.g.  $f(z_{2i})$ , specified as  $\sim c(s(z2))$ , or  $\beta_2 z_{2i}$ , specified as  $\sim c(z2)$ ),
3. linear functional effects of scalar (numeric or factor) covariates that vary smoothly over  $t$  (e.g.  $z_{3i} \beta_3(t)$ , specified as  $\sim z3$ ),
4. function-on-function regression terms (e.g.  $\int X_i(s) \beta(s, t) ds$ , specified as  $\sim ff(X, yindex=t, xindex=s)$ , see [ff](#)).

Use the  $c()$ -notation to denote model terms that are constant over the index of the functional response.

Internally, univariate smooth terms without a  $c()$ -wrapper are expanded into bivariate smooth terms in the original covariate and the index of the functional response. Bivariate smooth terms ( $s()$ ,  $te()$  or  $t2()$ ) without a  $c()$ -wrapper are expanded into trivariate smooth terms in the original covariates and the index of the functional response. Linear terms for scalar covariates or categorical covariates are expanded into varying coefficient terms, varying smoothly over the index of the functional response.

Use  $\sim -1 + c(1) + \dots$  to specify a model with a constant instead of a functional intercept.

The functional response and functional covariates have to be supplied as  $n$  by  $\langle \text{no. of evaluations} \rangle$  matrices, i.e. each row is one functional observation. The model is then fitted with the data in long format, i.e., the rows of the matrix of the functional response evaluations  $Y_i(t)$  are stacked into one long vector and the covariates are expanded/repeated correspondingly. This means the models get quite big fairly fast, since the effective number of rows in the design matrix is number of observations times number of evaluations of  $Y(t)$  per observation.

Note that only default identifiability constraints as implemented in [gam.side](#) are used, i.e.

$\sum_{i,t} \hat{f}(z_i, t) = 0$ . ([gam.side](#) may impose other identifiability constraints as well but doesn't give interpretable results.) These may not be well suited for models with multiple functional effects varying across  $t$ .

Centering estimated effects  $\hat{f}(z_i, t)$  ( $\sim s(z)$ ) across observations so that  $\sum_i \hat{f}(z_i, t) = 0$  for all  $t$ , i.e., transforming  $\hat{f}(z_i, t) \rightarrow \hat{f}(z_i, t) - n^{-1} \sum_i \hat{f}(z_i, t)$  and adding the mean function  $n^{-1} \sum_i \hat{f}(z_i, t)$  to the global functional intercept can improve the interpretability of results. We also recommend using centered scalar covariates for terms like  $z\beta(t)$  ( $\sim z$ ) and centered functional covariates with  $\sum_i X_i(t) = 0$  for all  $t$  in [ff](#)-terms.

### Author(s)

Fabian Scheipl, Sonja Greven

### References

Ivanescu, A. E., Staicu, A.-M., Greven, S., Scheipl, F., and Crainiceanu, C. M. (2011). Penalized function-on-function regression. Submitted.

### Examples

```

data(DTI, package="refund")
DTI$case <- factor(DTI$case)
DTI$ID <- factor(DTI$ID)
DTI$visit <- factor(DTI$visit)
rcst.complete <- apply(DTI, 1, function(x) !any(is.na(x)))
DTIcomplete <- DTI[rcst.complete,]

## split into test and training data
set.seed(2213213)
trainind <- sort(sample(1:sum(rcst.complete), 150))
train <- DTIcomplete[trainind,]
test <- DTIcomplete[-trainind,]
# drop subjects not in the train data
test <- test[test$ID %in% unique(train$ID),]

rcstind <- 1:55
# constant random intercepts for ID,
# smooth effect of pasat varying over index of rcst
# constant effect of visit
# m1 adds a functional effect for cca
summary(m0 <- pffr(rcst ~ c(s(ID, bs="re")) + s(pasat) + c(visit),
  yind=rcstind, data=train))
summary(m1 <- pffr(rcst ~ c(s(ID, bs="re")) + s(pasat) +
  c(visit) + ff(cca, yind=rcstind),
  yind=rcstind, data=train))
AIC(m0, m1)
plot(m1, pers=TRUE, pages=1, ticktype="detailed")
str(coefs.m1 <- coef(m1))

#get predicted trajectories for test data
pr.m0 <- predict(m0, newdata=test)
pr.m1 <- predict(m1, newdata=test)
# (riemann integrated) squared prediction error:
(IMSE <- c(m0 = sum((pr.m0-test$rcst)^2),
  m1 = sum((pr.m1-test$rcst)^2)))

layout(t(1:3))
matplot(t(test$rcst), type="l", lty=1, col=rgb(0,0,0,.1), main="observed rcst",
  ylim=range(test$rcst, pr.m0, pr.m1))
matplot(t(pr.m0),type="l", lty=1, col=rgb(0,0,0,.1), main="predicted: m0",
  ylim=range(test$rcst, pr.m0, pr.m1))
matplot(t(pr.m1),type="l", lty=1, col=rgb(0,0,0,.1), main="predicted: m1",
  ylim=range(test$rcst, pr.m0, pr.m1))

```

**Description**

Implements penalized functional regression (Goldsmith et al., 2011) for generalized linear functional models with scalar outcomes.

**Usage**

```
pfr(Y, covariates = NULL, funcs, kz = 30, kb = 30,
    smooth.cov = FALSE, family = "gaussian", method = "REML", ...)
```

**Arguments**

Y	Vector of all outcomes over all visits
covariates	Matrix of scalar covariates
funcs	Matrix or list of matrices containing observed functional predictors as rows. NA values are allowed.
kz	Dimension of principal components basis for the observed functional predictors
kb	Dimension of the truncated power series spline basis for the coefficient function
smooth.cov	Logical; do you wish to smooth the covariance matrix of observed functions? Increases computation time, but results in smooth principal components
family	Generalized linear model family
method	Method for estimating the smoothing parameters; defaults to REML
...	Additional arguments passed to the gam function to fit the regression model.

**Details**

Functional predictors are entered as a matrix or, in the case of multiple functional predictors, as a list of matrices using the `funcs` argument. Missing values are allowed in the functional predictors, but it is assumed that they are observed over the same grid. Functional coefficients and confidence bounds are returned as lists in the same order as provided in the `funcs` argument, as are principal component and spline bases.

**Value**

<code>fit</code>	The result of the call to <code>gam</code>
<code>fitted.vals</code>	Predicted outcomes
<code>betaHat</code>	List of estimated coefficient functions
<code>beta.covariates</code>	Parameter estimates for scalar covariates
<code>X</code>	Design matrix used in the model fit
<code>phi</code>	List of truncated power series spline bases for the coefficient functions
<code>psi</code>	List of principal components basis for the functional predictors
<code>varBetaHat</code>	List containing covariance matrices for the estimated coefficient functions
<code>Bounds</code>	List of bounds of a 95% confidence interval for the estimated coefficient functions

**Author(s)**

Jeff Goldsmith <jgoldsmi@jhsph.edu>

## References

Goldsmith, J., Bobb, J., Crainiceanu, C., Caffo, B., and Reich, D. (2011). Penalized functional regression. *Journal of Computational and Graphical Statistics*, to appear.

## Examples

```
#####
#####          DTI Data Example          #####
#####

#####
# use baseline data to regress continuous outcomes on functional
# predictors (continuous outcomes only recorded for case == 1)
#####

data(DTI)

# subset data as needed for this example
cca = DTI$cca[which(DTI$visit ==1 & DTI$case == 1),]
rcst = DTI$rcst[which(DTI$visit ==1 & DTI$case == 1),]
DTI = DTI[which(DTI$visit ==1 & DTI$case == 1),]

# note there is missingness in the functional predictors
apply(is.na(cca), 2, mean)
apply(is.na(rcst), 2, mean)

# fit two models with single functional predictors and plot the results
fit.cca = pfr(Y=DTI$pasat, funcs = cca, smooth.cov=FALSE)
fit.rcst = pfr(Y=DTI$pasat, funcs = rcst, smooth.cov=FALSE)

par(mfrow = c(1,2))
matplot(cbind(fit.cca$BetaHat[[1]], fit.cca$Bounds[[1]]),
        type = 'l', lty = c(1,2,2), col = c(1,2,2), ylab = "BetaHat",
        main = "CCA")
matplot(cbind(fit.rcst$BetaHat[[1]], fit.rcst$Bounds[[1]]),
        type = 'l', lty = c(1,2,2), col = c(1,2,2), ylab = "BetaHat",
        main = "RCST")

# fit a model with two functional predictors and plot the results
fit.cca.rcst = pfr(Y=DTI$pasat, funcs = list(cca, rcst),
                  smooth.cov=FALSE)

par(mfrow = c(1,2))
matplot(cbind(fit.cca.rcst$BetaHat[[1]], fit.cca.rcst$Bounds[[1]]),
        type = 'l', lty = c(1,2,2), col = c(1,2,2), ylab = "BetaHat",
        main = "CCA")
matplot(cbind(fit.cca.rcst$BetaHat[[2]], fit.cca.rcst$Bounds[[2]]),
        type = 'l', lty = c(1,2,2), col = c(1,2,2), ylab = "BetaHat",
        main = "RCST")
```

```
#####
# use baseline data to regress binary case-status outcomes on
# functional predictors
#####

data(DTI)

# subset data as needed for this example
cca = DTI$cca[which(DTI$visit == 1),]
rcst = DTI$rcst[which(DTI$visit == 1),]
DTI = DTI[which(DTI$visit == 1),]

# fit two models with single functional predictors and plot the results
fit.cca = pfr(Y=DTI$case, funcs = cca, smooth.cov=FALSE,
  family = "binomial")
fit.rcst = pfr(Y=DTI$case, funcs = rcst, smooth.cov=FALSE,
  family = "binomial")

par(mfrow = c(1,2))
matplot(cbind(fit.cca$BetaHat[[1]], fit.cca$Bounds[[1]]),
  type = 'l', lty = c(1,2,2), col = c(1,2,2), ylab = "BetaHat",
  main = "CCA")
matplot(cbind(fit.rcst$BetaHat[[1]], fit.rcst$Bounds[[1]]),
  type = 'l', lty = c(1,2,2), col = c(1,2,2), ylab = "BetaHat",
  main = "RCST")

#####
#####          Octane Data Example          #####
#####

data(gasoline)
Y = gasoline$octane
#detach("package:refund")
funcs = gasoline$NIR
wavelengths = as.matrix(2*450:850)

# fit the model using pfr
fit = pfr(Y=Y, funcs=funcs, kz=50, kb=50)

# plot the estimated coefficient function
matplot(wavelengths, cbind(fit$BetaHat[[1]], fit$Bounds[[1]]),
  type='l', lwd=c(2,1,1), lty=c(1,2,2), xlab = "Wavelengths",
  ylab = "Coefficient Function", col=1)
```

**Description**

Plots the coefficient function estimates produced by `fosr()`.

**Usage**

```
## S3 method for class 'fosr'
plot(x, split = NULL, titles = NULL, xlabel = "",
     ylabel = "Coefficient function", set.par = TRUE, ...)
```

**Arguments**

<code>x</code>	an object of class <code>fosr</code> as produced by <code>fosr()</code> .
<code>split</code>	value, or vector of values, at which to divide the set of coefficient functions into groups, each plotted on a different scale. E.g., if set to 1, the first function is plotted on one scale, and all others on a different (common) scale. If <code>NULL</code> , all functions are plotted on the same scale.
<code>titles</code>	character vector of titles for the plots produced, e.g., names of the corresponding scalar predictors.
<code>xlabel</code>	label for the x-axes of the plots.
<code>ylabel</code>	label for the y-axes of the plots.
<code>set.par</code>	logical value: if <code>TRUE</code> , the function will try to set an appropriate value of the <code>mfrac</code> parameter for the plots. Otherwise you may wish to set <code>mfrac</code> outside the function call.
<code>...</code>	graphical parameters (see <a href="#">par</a> ) for the plot.

**Author(s)**

Philip Reiss <[phil.reiss@nyumc.org](mailto:phil.reiss@nyumc.org)>

**See Also**

[fosr](#), which includes examples.

---

plot.fpcr

*Default plotting for functional principal component regression output*

---

**Description**

Inputs an object created by [fpcr](#), and plots the estimated coefficient function.

**Usage**

```
## S3 method for class 'fpcr'
plot(x, se=TRUE, col=1, lty=c(1,2,2), xlab="",
     ylab="Coefficient function", ...)
```

**Arguments**

x	an object created by <a href="#">fpcr</a> .
se	If TRUE (the default), upper and lower lines are added at 2 standard errors (in the Bayesian sense; see Wood, 2006) above and below the coefficient function estimate. If a positive number is supplied, the standard error is instead multiplied by this number.
col	color for the line(s). This should be either a number, or a vector of length 3 for the coefficient function estimate, lower bound, and upper bound, respectively.
lty	line type(s) for the coefficient function estimate, lower bound, and upper bound.
xlab, ylab	x- and y-axis labels.
...	other arguments passed to the underlying plotting function.

**Value**

None; only a plot is produced.

**Author(s)**

Philip Reiss <[phil.reiss@nyumc.org](mailto:phil.reiss@nyumc.org)>

**References**

Wood, S. N. (2006). *Generalized Additive Models: An Introduction with R*. Boca Raton, FL: Chapman & Hall.

**See Also**

[fpcr](#), which includes an example.

---

plot.pffr

*Plot a pffr fit*

---

**Description**

Plot a fitted pffr-object. Simply dispatches to [plot.gam](#).

**Usage**

```
## S3 method for class 'pffr'
plot(x, ...)
```

**Arguments**

x	a fitted pffr-object
...	arguments handed over to <a href="#">plot.gam</a>

**Value**

This function only generates plots.

**Author(s)**

Fabian Scheipl

---

predict.pffr

*Prediction for penalized function-on-function regression*

---

**Description**

Takes a fitted pffr-object produced by `pffr()` and produces predictions given a new set of values for the model covariates or the original values used for the model fit. Predictions can be accompanied by standard errors, based on the posterior distribution of the model coefficients. This is a wrapper function for `predict.gam()`

**Usage**

```
## S3 method for class 'pffr'
predict(object, newdata, reformat = TRUE,
        type = "link", se.fit = FALSE, terms = NULL, ...)
```

**Arguments**

object	a fitted pffr-object
newdata	A named list containing the values of the model covariates at which predictions are required. If this is not provided then predictions corresponding to the original data are returned. If newdata is provided then it should contain all the variables needed for prediction, in the format supplied to pffr, i.e., functional predictors must be supplied as matrices with each row corresponding to one observed function. Index variables for the functional covariates are reused from the fitted model object and cannot be supplied with newdata. Prediction is always for the entire range of $Y(t)$ as defined in the original fit.
reformat	logical, defaults to TRUE. Should predictions be returned in matrix form (default) or in the long vector shape returned by <code>predict.gam()</code> ?
type	see <code>predict.gam()</code> for details. Note that <code>type == "lpmatrix"</code> will force reformat to FALSE.
se.fit	see <code>predict.gam()</code>
terms	If <code>type=="terms"</code> or <code>"iterms"</code> then only results for the terms given in this array will be returned. Note that these are the term-labels used in the gam-fit, not those in the original pffr-model specification – these labels can be requested via <code>names(object\$smooth)</code> . See <code>predict.gam()</code> for details. <NOT YET IMPLEMENTED>
...	additional arguments passed on to <code>predict.gam()</code>

**Value**

If `type == "lpmatrix"`, the design matrix for the supplied covariate values in long format. If `se == TRUE`, a list with entries `fit` and `se.fit` containing fits and standard errors, respectively. If `type == "terms"` or `"iterms"` each of these lists is a list of matrices of the same dimension as the response for `newdata` containing the linear predictor and its `se` for each term.

**Author(s)**

Fabian Scheipl

**See Also**

[predict.gam\(\)](#)

---

`print.summary.pffr`      *Print method for summary of a pffr fit*

---

**Description**

Pretty printing for a `summary.pffr`-object. See [print.summary.gam\(\)](#) for details.

**Usage**

```
## S3 method for class 'summary.pffr'  
print(x, digits = max(3,  
  getOption("digits") - 3), signif.stars =  
  getOption("show.signif.stars"), ...)
```

**Arguments**

<code>x</code>	a fitted <code>pffr</code> -object
<code>digits</code>	controls number of digits printed in output.
<code>signif.stars</code>	Should significance stars be printed alongside output?
<code>...</code>	not used

**Value**

A `summary.pffr` object

**Author(s)**

Fabian Scheipl, adapted from [print.summary.gam\(\)](#) by Simon Wood, Henric Nilsson

## Description

Estimates prediction error for a function-on-scalar regression model by leave-one-function-out cross-validation (CV), at each of a specified set of points.

## Usage

```
pwcv(fdobj, Z, L = NULL, lambda,
     eval.pts=seq(min(fdobj$basis$range), max(fdobj$basis$range),
                  length.out=201),
     scale = FALSE)
```

## Arguments

<code>fdobj</code>	a functional data object (class <code>fd</code> ) giving the functional responses.
<code>Z</code>	the model matrix, whose columns represent scalar predictors.
<code>L</code>	a row vector or matrix of linear contrasts of the coefficient functions, to be restricted to equal zero.
<code>lambda</code>	smoothing parameter: either a nonnegative scalar or a vector, of length <code>ncol(Z)</code> , of nonnegative values.
<code>eval.pts</code>	argument values at which the CV score is to be evaluated.
<code>scale</code>	logical value or vector determining scaling of the matrix <code>Z</code> (see <code>scale</code> , to which the value of this argument is passed).

## Details

Integrating the pointwise CV estimate over the function domain yields the *cross-validated integrated squared error*, the standard overall model fit score returned by `lofocv`.

It may be desirable to derive the value of `lambda` from an appropriate call to `fosr`, as in the example below.

## Value

A vector of the same length as `eval.pts` giving the CV scores.

## Author(s)

Philip Reiss <[phil.reiss@nyumc.org](mailto:phil.reiss@nyumc.org)>

## References

Reiss, P. T., Huang, L., and Mennes, M. (2010). Fast function-on-scalar regression with penalized basis expansions. *International Journal of Biostatistics*, 6(1), article 28. Available at [http://works.bepress.com/phil\\_reiss/16/](http://works.bepress.com/phil_reiss/16/)

**See Also**

[fcsr](#), [lofocv](#)

**Examples**

```
# Canadian weather example from Reiss et al. (2010).
# The first two lines are taken from the fRegress.CV help file (package fda)
smallbasis <- create.fourier.basis(c(0, 365), 25)
tempfd <- smooth.basis(day.5,
  CanadianWeather$dailyAv[, "Temperature.C"], smallbasis)$fd

# Model matrices for "latitude" and "region" models
Zreg = cbind(1, model.matrix(~factor(CanadianWeather$region)-1))
Zlat = model.matrix(~CanadianWeather$coord[,1])

# Fit each model using fcsr() to obtain lambda values for pwcv()
Lreg = matrix(c(0,1,1,1,1), 1) # constraint for region model
regionmod = fcsr(tempfd, Zreg, Lreg, method="OLS")
cv.region = pwcv(tempfd, Zreg, Lreg, regionmod$lambda)

latmod = fcsr(tempfd, Zlat, method="OLS")
cv.lat = pwcv(tempfd, Zlat, lambda=latmod$lambda)

# The following plots may require a wide graphics window to show up correctly
par(mfrow=1:2)
# Plot the functional data
plot(tempfd, col=1, lty=1, axes=list("axesIntervals"), xlab="", ylab="Mean temperature", main="Temperatures at 35
box()

# Plot the two models' pointwise CV
matplot(regionmod$eval.pts, cbind(cv.region, cv.lat), type='l', col=1, axes=FALSE, xlab="", ylab="MSE.CV", main="
legend(250, 40, c('Region', 'Latitude'), lty=1:2)
box()
axis(2)
axisIntervals(1)
```

---

residuals.pffr

*Obtain residuals for a pffr fit*

---

**Description**

Obtain residuals for a pffr fit

**Usage**

```
## S3 method for class 'pffr'
residuals(object, reformat = TRUE, ...)
```

**Arguments**

object            a fitted pffr-object  
 reformat        logical, defaults to TRUE. Should residuals be returned in n x y index matrix form (default) or in the long vector shape returned by resid.gam()?  
 ...             other arguments, passed to [residuals.gam](#).

**Value**

A matrix or vector of residuals

**Author(s)**

Fabian Scheipl

---

summary.pffr	<i>Summary for a pffr fit</i>
--------------	-------------------------------

---

**Description**

Take a fitted pffr-object and produce summaries from it. See [summary.gam\(\)](#) for details.

**Usage**

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

**Arguments**

object            a fitted pffr-object  
 ...             see [summary.gam\(\)](#) for options.

**Value**

A list with summary information, see [summary.gam\(\)](#)

**Author(s)**

Fabian Scheipl, adapted from [summary.gam\(\)](#) by Simon Wood, Henric Nilsson

# Index

- \*Topic **datasets**
  - gasoline, 16
- \*Topic **hplot**
  - plot.fosr, 26
- \*Topic **models**
  - amc, 3
  - fosr, 9
  - fosr.perm, 11
  - fpcr, 14
  - lofocv, 17
  - plot.fpcr, 27
  - pwcv, 31
- \*Topic **package**
  - refund-package, 2
- \*Topic **regression**
  - amc, 3
  - fosr, 9
  - fosr.perm, 11
  - fpcr, 14
  - lofocv, 17
  - plot.fpcr, 27
  - pwcv, 31
- \*Topic **smooth**
  - amc, 3
  - fosr, 9
  - fosr.perm, 11
  - fpcr, 14
  - lofocv, 17
  - plot.fpcr, 27
  - pwcv, 31
- amc, 3
- bam, 21
- call, 6
- coef.pffr, 4
- DTI, 5
- expand.call, 6
- ff, 7, 21, 22
- fitted.pffr, 8
- fosr, 2, 4, 9, 12, 13, 17, 27, 31, 32
- fosr.perm, 11
- fpcr, 2, 14, 16, 27, 28
- gam, 15, 21
- gam.side, 22
- gamm, 21
- gamm4, 21
- gamObject, 4, 15
- gasoline, 16
- linear.functional.terms, 8
- lofocv, 9, 17, 31, 32
- lpfr, 2, 18
- match.call, 6
- model.matrix.pffr, 20
- optimize, 17
- par, 12, 27
- pffr, 2, 7, 21, 29
- pfr, 2, 23
- plot.fosr, 10, 26
- plot.fosr.perm (fosr.perm), 11
- plot.fpcr, 27
- plot.gam, 28
- plot.pffr, 28
- predict.gam, 20, 29, 30
- predict.pffr, 8, 29
- print.summary.gam, 30
- print.summary.pffr, 30
- pwcv, 17, 31
- refund (refund-package), 2
- refund-package, 2
- residuals.gam, 33
- residuals.pffr, 32

*s*, 7, 21

*scale*, 9, 31

*summary.gam*, 33

*summary.pffr*, 30, 33

*t2*, 7

*te*, 7