Package 'refund'

July 2, 2014

Type Package

Title Regression with Functional Data

Version 0.1-11

Date 2014-06-28

Author Ciprian Crainiceanu [aut], Philip Reiss [aut], Jeff Goldsmith [aut],Lei Huang [aut, cre], Lan Huo [aut], Fabian Scheipl [aut], Bruce Swihart [ctb], Sonja Greven [ctb], Jaroslaw Harezlak [ctb], Madan Gopal Kundu [ctb], Yihong Zhao [ctb], Matt McLean [ctb], Luo Xiao [ctb]

Maintainer Lei Huang <huangracer@gmail.com>

Depends R (>= 2.14.0), fda,

Imports lattice, boot, mgcv (>= 1.7-28), MASS, glmnet, magic, nlme,wavethresh, Matrix, gamm4, matrixStats, lme4, splines, RLRsim

Description Methods for regression for functional data, including function-on-scalar, scalar-on-function, and function-on-function regression. Some of the functions are applicable to image data.

License GPL (>= 2)

LazyLoad yes

Repository CRAN

Collate 'Omegas.R' 'af.R' 'amc.R' 'ccb.fpc.R' 'decomp.R' 'decomp2d.R' 'decomp3d.R' 'fbps.R' 'fgam.R' 'fosr.R' 'fosr.perm.R' 'fosr.perm.fit.R' 'fosr.perm.test.R' 'fosr2s.R' 'fpca.sc.R' 'fpca.face.R' 'fpcr.R' 'fpcr.setup.R' 'lf.R' 'lofocv.R' 'lpeer.R' 'lpfr.R' 'lw.test.R' 'osplinepen2d.R' 'peer.R' 'pffr-ff.R' 'pffr-ffpc.R' 'pffr-methods.R' 'pffr-pcre.R' 'pffr-robust.R' 'pffr-sff.R' 'pffr-utilities.R' 'pffr.R' 'pfr.R' 'plot.fosr.R' 'plot.fosr.perm.R' 'plot.fpcr.R' 'plot.lpeer.R' 'plot.peer.R' 'predict.fgam.R' 'predict.wnet.R' 'pspline.setting.R' 'pwcv.R' 'reconstr.R' 'reconstr2d.R' 'reconstr3d.R' 'rlrt.pfr.R' 'fpca2s.R' 'fpca.ssvd.R'

'vis.fgam.R' 'wcr.R' 'wcr.perm.R' 'wnet.R' 'wnet.perm.R'
'parse.predict.pfr.R' 'postprocess.pfr.R' 'predict.pfr.R'
'preprocess.pfr.R' 'wUtil.R' 'plot.wnet-and-plot.wcr.R' 'irreg2mat.R'

NeedsCompilation no

Date/Publication 2014-06-28 18:49:01

R topics documented:

`refund-package` *Regression with Functional Data*

Description

Methods for regression with functional data. Various approaches to regression with scalar responses and functional predictors are implemented by `pfr`, `peer`, `lpeer`, `fpcr`, `wcr` and `wnet`. For regression with functional responses, see `pffr`, `fosr`, and `fosr2s`.

Details

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

Author(s)

Authors: Ciprian Crainiceanu <ccrainic@jhsph.edu>, Philip Reiss <phil.reiss@nyumc.org>, Jeff Goldsmith <jeff.goldsmith@columbia.edu>, Lei Huang <huangracer@gmail.com>, Lan Huo <lan.huo@nyumc.org>, Fabian Scheipl <fabian.scheipl@stat.uni-muenchen.de>.

Contributors: Bruce Swihart, Luo Xiao, Sonja Greven, Jarek Harezlak, Madan Gopal Kundu, Matt McLean, Yihong Zhao

Maintainer: Lei Huang <huangracer@gmail.com>

af *Construct an FGAM regression term*

Description

Defines a term $\int_T F(X_i(t), t)dt$ for inclusion in an `mgcv::gam-formula` (or `bam` or `gamm` or `gamm4:::gamm`) as constructed by `fgam`, where $F(x, t)$\$ is an unknown smooth bivariate function and $X_i(t)$ is a functional predictor on the closed interval T. Defaults to a cubic tensor product B-spline with marginal second-order difference penalties for estimating $F(x, t)$. The functional predictor must be fully observed on a regular grid.

Usage

```
af(X, xind = seq(0, 1, l = ncol(X)), basistype = c("te","t2", "s"),
   integration = c("simpson", "trapezoidal", "riemann"),
   L = NULL, splinepars = list(bs = "ps",
   k = c(min(ceiling(nrow(X)/5),20),min(ceiling(ncol(X)/5),20)),
  m = list(c(2, 2), c(2, 2))), presmooth = TRUE, Xrange=range(X), Qtransform=FALSE)
```

Arguments

`X`	an `N` by `J=ncol(xind)` matrix of function evaluations $X_i(t_{i1}), ., X_i(t_{iJ}); i = 1, ., N$.
`xind`	matrix (or vector) of indices of evaluations of $X_i(t)$; i.e. a matrix with *i*th row $(t_{i1}, ., t_{iJ})$.
`basistype`	defaults to "`te`", i.e. a tensor product spline to represent $F(x, t)$ Alternatively, use "`s`" for bivariate basis functions (see mgcv's `s`) or "`t2`" for an alternative parameterization of tensor product splines (see mgcv's `t2`).
`integration`	method used for numerical integration. Defaults to "`simpson`"'s rule for calculating entries in `L`. Alternatively and for non-equidistant grids, "`trapezoidal`" or "`riemann`". "`riemann`" integration is always used if `limits` is specified.
`L`	optional: an `N` by `ncol(xind)` matrix giving the weights for the numerical integration over `t`.
`splinepars`	optional arguments specifying options for representing and penalizing the function $F(x, t)$. Defaults to a cubic tensor product B-spline with marginal second-order difference penalties, i.e. `list(bs="ps", m=list(c(2, 2), c(2, 2))`, see `te` or `s` in `mgcv` for details.
`presmooth`	If true, the functional predictor is pre-smoothed prior to fitting. See `smooth.basisPar` in package `fda`
`Xrange`	Range to use when specifying the marginal basis for the x-axis. It may be desired to increase this slightly over the default of `range(X)` if concerned about predicting for future observed curves that take values outside of `range(X)`.
`Qtransform`	Should the functional be transformed using the empirical cdf and applying a quantile transformation on each column of `X` prior to fitting? This ensures `Xrange=c(0,1)`. If `Qtransform=TRUE` and `presmooth=TRUE`, presmoothing is done prior to transforming the functional predictor.

Value

A list with the following entries:

call	a "call" to te (or s, t2) using the appropriately constructed covariate and weight matrices.
xind	the xind argument supplied to af.
L	the matrix of weights used for the integration.
xindname	the name used for the functional predictor variable in the formula used by mgcv.
tindname	the name used for xind variable in the formula used by mgcv.
Lname	the name used for the L variable in the formula used by mgcv.
presmooth	the presmooth argument supplied to af.
Qtranform	the Qtransform argument supplied to af.
Xrange	the Xrange argument supplied to af.
ecdflist	a list containing one empirical cdf function from applying ecdf to each (possibly presmoothed) column of X. Only present if Qtransform=TRUE.
Xfd	an fd object from presmoothing the functional predictors using smooth.basisPar. Only present if presmooth=TRUE. See fd.

Author(s)

Mathew W. McLean <mwm79@cornell.edu> and Fabian Scheipl

References

McLean, M. W., Hooker, G., Staicu, A.-M., Scheipl, F., and Ruppert, D. (2013). Functional generalized additive models. *Journal of Computational and Graphical Statistics*, to appear. Available at http://courses2.cit.cornell.edu/mwmclean

See Also

fgam, lf, mgcv's linear.functional.terms, fgam for examples.

---

amc *Additive model with constraints*

Description

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

Usage

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

Arguments

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

Details

The additive model is fitted using gam, unless there are more than 10000 responses; in that case bam is used.

Value

A list with the following elements:

gam	the gam object returned by gam or bam.
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

---

ccb.fpc *Corrected confidence bands using functional principal components*

Description

Uses iterated expectation and variances to obtain corrected estimates and inference for functional expansions.

Usage

```
ccb.fpc(Y, nbasis = 10, pve = .99, n.boot = 100, simul = FALSE, sim.alpha = .95)
```

Arguments

Y	matrix of observed functions for which estimates and covariance matrices are desired.
nbasis	number of splines used in the estimation of the mean function and the bivariate smoothing of the covariance matrix
pve	proportion of variance explained used to choose the number of principal components to be included in the expansion.
n.boot	number of bootstrap iterations used to estimate the distribution of FPC decomposition objects.
simul	TRUE or FALSE, indicating whether critical values for simultaneous confidence intervals should be estimated
sim.alpha	alpha level of the simultaneous intervals.

Details

To obtain corrected curve estimates and variances, this function accounts for uncertainty in FPC decomposition objects. Observed curves are resampled, and a FPC decomposition for each sample is constructed. A mixed-model framework is used to estimate curves and variances conditional on each decomposition, and iterated expectation and variances combines both model-based and decomposition-based uncertainty.

Value

Yhat	a matrix whose rows are the estimates of the curves in Y.
Yhat.boot	a list containing the estimated curves within each bootstrap iteration.
diag.var	diagonal elements of the covariance matrices for each estimated curve.
VarMats	a list containing the estimated covariance matrices for each curve in Y.
crit.val	estimated critical values for constructing simultaneous confidence intervals.

Author(s)

Jeff Goldsmith <jeff.goldsmith@columbia.edu>

References

Goldsmith, J., Greven, S., and Crainiceanu, C. (2013). Corrected confidence bands for functional data using principal components. *Biometrics*, 69(1), 41–51.

Examples

```
## Not run:
data(cd4)

# obtain a subsample of the data with 25 subjects
set.seed(1236)
sample = sample(1:dim(cd4)[1], 25)
Y.sub = cd4[sample,]
```

```
# obtain a mixed-model based FPCA decomposition
Fit.MM = fpca.sc(Y.sub, var = TRUE, simul = TRUE)

# use iterated variance to obtain curve estimates and variances
Fit.IV = ccb.fpc(Y.sub, n.boot = 25, simul = TRUE)

# for one subject, examine curve estimates, pointwise and simultaneous itervals
EX = 2
EX.IV =  cbind(Fit.IV$Yhat[EX,],
      Fit.IV$Yhat[EX,] + 1.96 * sqrt(Fit.IV$diag.var[EX,]),
      Fit.IV$Yhat[EX,] - 1.96 * sqrt(Fit.IV$diag.var[EX,]),
      Fit.IV$Yhat[EX,] + Fit.IV$crit.val[EX] * sqrt(Fit.IV$diag.var[EX,]),
      Fit.IV$Yhat[EX,] - Fit.IV$crit.val[EX] * sqrt(Fit.IV$diag.var[EX,]))

EX.MM =  cbind(Fit.MM$Yhat[EX,],
      Fit.MM$Yhat[EX,] + 1.96 * sqrt(Fit.MM$diag.var[EX,]),
      Fit.MM$Yhat[EX,] - 1.96 * sqrt(Fit.MM$diag.var[EX,]),
      Fit.MM$Yhat[EX,] + Fit.MM$crit.val[EX] * sqrt(Fit.MM$diag.var[EX,]),
      Fit.MM$Yhat[EX,] - Fit.MM$crit.val[EX] * sqrt(Fit.MM$diag.var[EX,]))

# plot data for one subject, with curve and interval estimates
d = as.numeric(colnames(cd4))
plot(d[which(!is.na(Y.sub[EX,]))], Y.sub[EX,which(!is.na(Y.sub[EX,]))], type = 'o',
  pch = 19, cex=.75, ylim = range(0, 3400), xlim = range(d),
    xlab = "Months since seroconversion", lwd = 1.2, ylab = "Total CD4 Cell Count",
      main = "Est. & CI - Sampled Data")

matpoints(d, EX.IV, col = 2, type = 'l', lwd = c(2, 1, 1, 1, 1), lty = c(1,1,1,2,2))
matpoints(d, EX.MM, col = 4, type = 'l', lwd = c(2, 1, 1, 1, 1), lty = c(1,1,1,2,2))

legend("topright", c("IV Est", "IV PW Int", "IV Simul Int",
    expression(paste("MM - ", hat(theta), " Est", sep = "")),
    expression(paste("MM - ", hat(theta), " PW Int", sep = "")),
    expression(paste("MM - ", hat(theta), " Simul Int", sep = ""))),
    lty=c(1,1,2,1,1,2), lwd = c(2.5,.75,.75,2.5,.75,.75),
    col = c("red","red","red","blue","blue","blue"))

## End(Not run)
```

---

cd4 *Observed CD4 cell counts*

Description

CD4 cell counts for 366 subjects between months -18 and 42 since seroconversion. Each subject's observations are contained in a single row.

Usage

```
data(cd4)
```

Format

A data frame made up of

`cd4` A 366 x 61 matrix of CD4 cell counts.

References

Goldsmith, J., Greven, S., and Crainiceanu, C. (2013). Corrected confidence bands for functional data using principal components. *Biometrics*, 69(1), 41–51.

`coef.pffr` *Get estimated coefficients from a pffr fit*

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 point-wise estimated 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,
  sandwich = FALSE, seWithMean = TRUE, n1 = 100, n2 = 40, n3 = 20,
  Ktt = NULL, ...)
```

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 `gamObject`
`sandwich`	logical, defaults to FALSE. Use a Sandwich-estimator for approximate variances? See Details. THIS IS AN EXPERIMENTAL FEATURE, USE A YOUR OWN RISK.
`seWithMean`	logical, defaults to TRUE. Include uncertainty about the intercept/overall mean in standard errors returned for smooth components?
`n1`	see below
`n2`	see below
`n3`	`n1, n2, n3` give the number of gridpoints for 1-/2-/3-dimensional smooth terms used in the marginal equidistant grids over the range of the covariates at which the estimated effects are evaluated.

`Ktt` (optional) an estimate of the covariance operator of the residual process $\epsilon_i(t) \sim N(0, K(t,t'))$, evaluated on `yind` of `object`. If not supplied, this is estimated from the crossproduct matrices of the observed residual vectors. Only relevant for sandwich CIs.

`...` other arguments, not used.

Details

The `seWithMean`-option corresponds to the `"iterms"`-option in `predict.gam`. The `sandwich`-options works as follows: Assuming that the residual vectors $\epsilon_i(t), i = 1, \ldots, n$ are i.i.d. realizations of a mean zero Gaussian process with covariance $K(t,t')$, we can construct an estimator for $K(t,t')$ from the n replicates of the observed residual vectors. The covariance matrix of the stacked observations $\text{vec}(Y_i(t))$ is then given by a block-diagonal matrix with n copies of the estimated $K(t,t')$ on the diagonal. This block-diagonal matrix is used to construct the "meat" of a sandwich covariance estimator, similar to Chen et al. (2012), see reference below.

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

References

Chen H., Wang Y., Paik C.M., Choi A. (2012). A marginal approach to reduced-rank penalized spline smoothing for multilevel data. *Journal of the American Statistical Association*, under revision. http://www.columbia.edu/~yw2016/MarginalSpline6.pdf

See Also

`plot.gam`, `predict.gam` which this routine is based on.

`coefboot.pffr` *Simple bootstrap CIs for pffr*

Description

This function resamples observations in the data set to obtain approximate CIs for different terms and coefficient functions that correct for the effects of dependency and heteroskedasticity of the residuals along the index of the functional response, i.e., it aims for correct inference if the residuals along the index of the functional response are not i.i.d.

Usage

```
coefboot.pffr(object, n1 = 100, n2 = 40, n3 = 20, B = 100,
  ncpus = getOption("boot.ncpus", 1), parallel = c("no", "multicore",
  "snow"), cl = NULL, conf = c(0.9, 0.95), type = "percent",
  method = "resample", showProgress = TRUE, ...)
```

Arguments

`object`	a fitted `pffr`-model
`n1`	see `coef.pffr`
`n2`	see `coef.pffr`
`n3`	see `coef.pffr`
`B`	number of bootstrap replicates, defaults to (a measly) 100
`parallel`	see `boot`
`cl`	see `boot`
`ncpus`	see `boot`. Defaults to `getOption("boot.ncpus", 1L)` (like `boot`).
`conf`	desired levels of bootstrap CIs, defaults to 0.90 and 0.95
`type`	type of bootstrap interval, see `boot.ci`. Defaults to "percent" for percentile-based CIs.
`method`	either "resample" (default) to resample response trajectories, or "residual" to resample residual trajectories.
`showProgress`	TRUE/FALSE
`...`	not used

Value

a list with similar structure as the return value of `coef.pffr`, containing the original point estimates of the various terms along with their bootstrap CIs.

Author(s)

Fabian Scheipl

DTI *Diffusion Tensor Imaging: tract profiles and outcomes*

Description

Fractional anisotropy (FA) 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.

If you use this data as an example in written work, please include the following acknowledgment: "The MRI/DTI data were collected at Johns Hopkins University and the Kennedy-Krieger Institute"

DTI2 uses mean diffusivity of the the corpus callosum rather than FA, and parallel diffusivity of the rcst rather than FA. Please see the documentation for DTI2.

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*, 20, 830 - 851.

Goldsmith, J., Crainiceanu, C., Caffo, B., and Reich, D. (2010). Longitudinal Penalized Functional Regression for Cognitive Outcomes on Neuronal Tract Measurements . *Journal of the Royal Statistical Society: Series C*, 61, 453 - 469.

DTI2 *Diffusion Tensor Imaging: more fractional anisotropy profiles and outcomes*

Description

A diffusion tensor imaging dataset used in Swihart et al. (2012). Mean diffusivity profiles for the corpus callosum (cca) and parallel diffusivity for the right corticospinal tract (rcst). Accompanying the profiles are the subject ID numbers, visit number, and Paced Auditory Serial Addition Test (pasat) score. We thank Dr. Daniel Reich for making this dataset available.

If you use this data as an example in written work, please include the following acknowledgment: "The MRI/DTI data were collected at Johns Hopkins University and the Kennedy-Krieger Institute"

Note: DTI2 uses mean diffusivity of the the corpus callosum rather than fractional anisotropy (FA), and parallel diffusivity of the rcst rather than FA. Please see the documentation for DTI for more about the DTI dataset.

Usage

```
data(DTI2)
```

Format

A data frame made up of

`cca` a 340 x 93 matrix of fractional anisotropy profiles from the corpus callosum;

`rcst` a 340 x 55 matrix of fractional anisotropy profiles from the right corticospinal tract;

`id` numeric vector of subject ID numbers;

`visit` numeric vector of the subject-specific visit numbers;

`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*, 20(4), 830–851.

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*, 61(3), 453–469.

Swihart, B. J., Goldsmith, J., and Crainiceanu, C. M. (2012). Testing for functional effects. Johns Hopkins University Dept. of Biostatistics Working Paper 247. Available at http://biostats.bepress.com/jhubiostat/paper247

expand.call *Return call with all possible arguments*

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 match.call.
call	an unevaluated call to the function specified by definition. See match.call.
expand.dots	logical. Should arguments matching ... in the call be included or left as a ... argument? See match.call.

Value

An object of mode "call".

Author(s)

Fabian Scheipl

See Also

match.call

fbps *Sandwich smoother for matrix data*

Description

A fast bivariate *P*-spline method for smoothing matrix data.

Usage

```
fbps(data, covariates = NULL, knots = 35, p = 3, m = 2, lambda = NULL,
     alpha = 1, search.grid = T, search.length = 100, method = "L-BFGS-B",
     lower = -20, upper = 20, control = NULL)
```

Arguments

data	n1 by n2 data matrix without missing data
covariates	list of two vectors of covariates of lengthes n1 and n2; if NULL, then generates equidistant covariates
knots	list of two vectors of knots or number of equidistant knots for all dimensions; defaults to 35
p	degrees of B-splines; defaults to 3
m	order of differencing penalty; defaults to 2
lambda	user-specified smoothing parameters; defaults to NULL
alpha	the coefficient of the penalty for degrees of freedom in the GCV criterion; defaults to 1
search.grid	logical; defaults to TRUE, if FALSE, uses optim
search.length	number of equidistant (log scale) smoothing parameter; defaults to 100
method	see optim; defaults to L-BFGS-B
lower, upper	bounds for log smoothing parameter, passed to optim; defaults are -20 and 20.
control	see optim

Details

The smoothing parameter can be user-specified; otherwise, the function uses grid searching method or optim for selecting the smoothing parameter.

Value

A list with components

lambda	vector of length 2 of selected smoothing parameters
Yhat	fitted data
trace	trace of the overall smoothing matrix
gcv	value of generalized cross validation
Theta	matrix of estimated coefficients

Author(s)

Luo Xiao <lxiao@jhsph.edu>

References

Xiao, L., Li, Y., and Ruppert, D. (2013). Fast bivariate *P*-splines: the sandwich smoother. *Journal of the Royal Statistical Society: Series B*, 75(3), 577–599.

Examples

```
##########################
#### True function   #####
##########################
n1 <- 60
n2 <- 80
x <- (1: n1)/n1-1/2/n1
z <- (1: n2)/n2-1/2/n2
MY <- array(0,c(length(x),length(z)))

sigx <- .3
sigz <- .4
for(i in 1: length(x))
for(j in 1: length(z))
{
#MY[i,j] <- .75/(pi*sigx*sigz) *exp(-(x[i]-.2)^2/sigx^2-(z[j]-.3)^2/sigz^2)
#MY[i,j] <- MY[i,j] + .45/(pi*sigx*sigz) *exp(-(x[i]-.7)^2/sigx^2-(z[j]-.8)^2/sigz^2)
MY[i,j] = sin(2*pi*(x[i]-.5)^3)*cos(4*pi*z[j])
}
##########################
#### Observed data   #####
##########################
sigma <- 1
Y <- MY + sigma*rnorm(n1*n2,0,1)
##########################
####   Estimation    #####
##########################

est <- fbps(Y,list(x=x,z=z))
mse <- mean((est$Yhat-MY)^2)
cat("mse of fbps is",mse,"\n")
cat("The smoothing parameters are:",est$lambda,"\n")
#############################################################################
########## Compare the estimated surface with the true surface #########
#############################################################################

par(mfrow=c(1,2))
persp(x,z,MY,zlab="f(x,z)",zlim=c(-1,2.5), phi=30,theta=45,expand=0.8,r=4,
      col="blue",main="True surface")
persp(x,z,est$Yhat,zlab="f(x,z)",zlim=c(-1,2.5),phi=30,theta=45,
      expand=0.8,r=4,col="red",main="Estimated surface")
```

`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:::gamm4`) as constructed by `pffr`.

Defaults to a cubic tensor product B-spline with marginal first order differences penalties for $\beta(t,s)$ and numerical integration over the entire range $[s_{lo,i}, s_{hi,i}] = [\min(s_i), \max(s_i)]$ by using Simpson weights. Can't deal with any missing $X(s)$, unequal lengths of $X_i(s)$ not (yet?) possible. Unequal integration ranges for different $X_i(s)$ should work. $X_i(s)$ is assumed to be numeric (duh...).

Usage

```
ff(X, yind = NULL, xind = seq(0, 1, l = ncol(X)), basistype = c("te",
  "t2", "s"), integration = c("simpson", "trapezoidal", "riemann"),
  L = NULL, limits = NULL, splinepars = if (basistype != "s") {
  list(bs = "ps", m = list(c(2, 1), c(2, 1))) } else {     list(bs = "tp", m =
  NA) }, check.ident = TRUE)
```

Arguments

X	an n by ncol(xind) matrix of function evaluations $X_i(s_{i1}), \ldots, X_i(s_{iS})$; $i = 1, \ldots, n$.
yind	*DEPRECATED* used to supply matrix (or vector) of indices of evaluations of $Y_i(t)$, no longer used.
xind	matrix (or vector) of indices of evaluations of $X_i(s)$; i.e. matrix with rows $(s_{i1}, \ldots, s_{iS})$
basistype	defaults to "te", i.e. a tensor product spline to represent $\beta(t,s)$. Alternatively, use "s" for bivariate basis functions (see mgcv's s) or "t2" for an alternative parameterization of tensor product splines (see mgcv's t2).
integration	method used for numerical integration. Defaults to "simpson"'s rule for calculating entries in L. Alternatively and for non-equidistant grids, "trapezoidal" or "riemann". "riemann" integration is always used if limits is specified
L	optional: an n by ncol(xind) matrix giving the weights for the numerical integration over s.
limits	defaults to NULL for integration across the entire range of $X(s)$, otherwise specifies the integration limits $s_{hi,i}, s_{lo,i}$: either one of "s<t" or "s<=t" for $(s_{hi,i}, s_{lo,i}) = (0,t)$ or a function that takes s as the first and t as the second argument and returns TRUE for combinations of values (s,t) if s falls into the integration range for the given t. This is an experimental feature and not well tested yet; use at your own risk.
splinepars	optional arguments supplied to the basistype-term. Defaults to a cubic tensor product B-spline with marginal first difference penalties, i.e. list(bs="ps", m=list(c(2, 1), c(2,1)) See te or s in **mgcv** for details
check.ident	check identifiability of the model spec. See Details and References. Defaults to TRUE.

Details

If check.ident==TRUE and basistype!="s" (the default), the routine checks for overlap between the kernels of $\mathrm{Cov}(X(s))$ and the marginal penalty over s, as well as for sufficient rank of $\mathrm{Cov}(X(s))$. If there is overlap of the kernels, the penalty and basis for the covariate direction are changed to B-splines with a modified 'shrinkage' penalty, see smooth.construct.pss.smooth.spec.

A warning is given if the effective rank of $\mathrm{Cov}(X(s))$ (defined the number of eigenvalues accounting for at least 0.995 of the total variance in $X_i(s)$) is lower than the dimension of the basis for the covariate direction. See reference for details. Using an ffpc-term may be preferable if $X_i(s)$ is of very low rank.

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

References

For background on check.ident:
Scheipl, F., & Greven, S. (2012). Identifiability in penalized function-on-function regression models. LMU Munich, Department of Statistics: Technical Report 125. http://epub.ub.uni-muenchen.de/13060/

See Also

mgcv's linear.functional.terms

---

ffpc *Construct a PC-based function-on-function regression term*

Description

Defines a term $\int X_i(s)\beta(t,s)ds$ for inclusion in an mgcv::gam-formula (or bam or gamm or gamm4:::gamm4) as constructed by pffr.

Usage

```
ffpc(X, yind = NULL, xind = seq(0, 1, length = ncol(X)),
  splinepars = list(bs = "ps", m = c(2, 1), k = 8), decomppars = list(pve =
  0.99, useSymm = TRUE), npc.max = 15)
```

Arguments

X	an n by ncol(xind) matrix of function evaluations $X_i(s_{i1}), \ldots, X_i(s_{iS})$; $i = 1, \ldots, n$.
yind	*DEPRECATED* used to supply matrix (or vector) of indices of evaluations of $Y_i(t)$, no longer used.
xind	matrix (or vector) of indices of evaluations of $X_i(t)$, defaults to seq(0, 1, length=ncol(X)).
splinepars	optional arguments supplied to the basistype-term. Defaults to a cubic B-spline with first difference penalties and 8 basis functions for each $\tilde{\beta}_k(t)$.
decomppars	parameters for the FPCA performed with fpca.sc.
npc.max	maximal number K of FPCs to use, regardless of decomppars; defaults to 15

Details

In contrast to ff, ffpc does an FPCA decomposition $X(s) \approx \sum_{k=1}^{K} \xi_{ik}\Phi_k(s)$ using fpca.sc and represents $\beta(t,s)$ in the function space spanned by these $\Phi_k(s)$. That is, since

$$\int X_i(s)\beta(t,s)ds = \sum_{k=1}^{K} \xi_{ik} \int \Phi_k(s)\beta(s,t)ds = \sum_{k=1}^{K} \xi_{ik}\tilde{\beta}_k(t),$$

the function-on-function term can be represented as a sum of K univariate functions $\tilde{\beta}_k(t)$ in t each multiplied by the FPC loadings ξ_{ik}. The truncation parameter K is chosen as described in fpca.sc. Using this instead of ff() can be beneficial if the covariance operator of the $X_i(s)$ has low effective rank (i.e., if K is small). If the covariance operator of the $X_i(s)$ is of (very) high rank, i.e., if K is large, ffpc() will not be very efficient.

To reduce model complexity, the $\tilde{\beta}_k(t)$ all have a single joint smoothing parameter (in mgcv, they get the same id, see s).

Please see pffr for details on model specification and implementation.

Value

A list containing the necessary information to construct a term to be included in a mgcv::gam-formula.

Author(s)

Fabian Scheipl

Examples

```
## Not run:
set.seed(1122)
n <- 55
S <- 60
T <- 50
s <- seq(0,1, l=S)
t <- seq(0,1, l=T)
```

```
#generate X from a polynomial FPC-basis:
rankX <- 5
Phi <- cbind(1/sqrt(S), poly(s, degree=rankX-1))
lambda <- rankX:1
Xi <- sapply(lambda, function(l)
            scale(rnorm(n, sd=sqrt(l)), scale=FALSE))
X <- Xi %*% t(Phi)

beta.st <- outer(s, t, function(s, t) cos(2 * pi * s * t))

y <- (1/S*X) %*% beta.st + 0.1 * matrix(rnorm(n * T), nrow=n, ncol=T)

data <- list(y=y, X=X)
# set number of FPCs to true rank of process for this example:
m.pc <- pffr(y ~ c(1) + 0 + ffpc(X, yind=t, decomppars=list(npc=rankX)),
        data=data, yind=t)
summary(m.pc)
m.ff <- pffr(y ~ c(1) + 0 + ff(X, yind=t), data=data, yind=t)
summary(m.ff)

# fits are very similar:
all.equal(fitted(m.pc), fitted(m.ff))

# plot implied coefficient surfaces:
layout(t(1:3))
persp(t, s, t(beta.st), theta=50, phi=40, main="Truth",
    ticktype="detailed")
plot(m.ff, select=1, pers=TRUE, zlim=range(beta.st), theta=50, phi=40,
    ticktype="detailed")
title(main="ff()")
ffpcplot(m.pc, type="surf", auto.layout=FALSE, theta = 50, phi = 40)
title(main="ffpc()")

# show default ffpcplot:
ffpcplot(m.pc)

## End(Not run)
```

`ffpcplot` *Plot PC-based function-on-function regression terms*

Description

Convenience function for graphical summaries of `ffpc`-terms from a `pffr` fit.

Usage

```
ffpcplot(object, type = c("fpc+surf", "surf", "fpc"), pages = 1,
  se.mult = 2, ticktype = "detailed", theta = 30, phi = 30,
  plot = TRUE, auto.layout = TRUE)
```

Arguments

object	a fitted pffr-model
type	one of "fpc+surf", "surf" or "fpc": "surf" shows a perspective plot of the coefficient surface implied by the estimated effect functions of the FPC loadings, "fpc" shows three plots: 1) a scree-type plot of the estimated eigenvalues of the functional covariate, 2) the estimated eigenfunctions, and 3) the estimated coefficient functions associated with the FPC loadings. Defaults to showing both.
se.mult	display estimated coefficient functions associated with the FPC loadings with plus/minus this number time the estimated standard error. Defaults to 2.
pages	the number of pages over which to spread the output. Defaults to 1. (Irrelevant if auto.layout=FALSE.)
ticktype	see persp.
theta	see persp.
phi	see persp.
plot	produce plots or only return plotting data? Defaults to TRUE.
auto.layout	should the the function set a suitable layout automatically? Defaults to TRUE

Value

primarily produces plots, invisibly returns a list containing the data used for the plots.

Author(s)

Fabian Scheipl

Examples

```
## Not run:
 #see ?ffpc

## End(Not run)
```

---

fgam *Functional Generalized Additive Models*

Description

Implements functional generalized additive models for functional and scalar covariates and scalar responses. Additionally implements functional linear models. This function is a wrapper for mgcv's gam and its siblings to fit models of the general form

$$g(E(Y_i)) = \beta_0 + \int_{T_1} F(X_{i1}, t)dt + \int_{T_2} \beta(t)X_{i2}dt + f(z_{i1}) + f(z_{i2}, z_{i3}) + \ldots)$$

with a scalar (but not necessarily continuous) response Y, and link function g

Usage

```
fgam(formula,fitter=NA,tensortype=c('te','t2'),...)
```

Arguments

formula	a formula with special terms as for gam, with additional special terms af() and lf().
fitter	the name of the function used to estimate the model. Defaults to gam if the matrix of functional responses has less than 2e5 data points and to bam if not. "gamm" (see gamm) and "gamm4" (see gamm4) are valid options as well.
tensortype	which type of tenor product splines to use. One of "te" or "t2", defaults to te
...	additional arguments that are valid for gam or bam; for example, specify a gamma > 1 to increase amount of smoothing when using GCV to choose smoothing parameters or method="REML" to change to REML for estimation of smoothing parameters (default is GCV).

Value

a fitted fgam-object, which is a gam-object with some additional information in a fgam-entry. If fitter is "gamm" or "gamm4", only the $gam part of the returned list is modified in this way..

Warning

Binomial responses should be specified as a numeric vector rather than as a matrix or a factor.

Author(s)

Mathew W. McLean <mwm79@cornell.edu> and Fabian Scheipl

References

McLean, M. W., Hooker, G., Staicu, A.-M., Scheipl, F., and Ruppert, D. (2013). Functional generalized additive models. *Journal of Computational and Graphical Statistics*, to appear. Available at http://courses2.cit.cornell.edu/mwmclean

See Also

af, lf, predict.fgam, vis.fgam

Examples

```
################## DTI Example ####################
data(DTI)
## only consider first visit and cases (no PASAT scores for controls)
y <- DTI$pasat[DTI$visit==1 & DTI$case==1]
X <- DTI$cca[DTI$visit==1 & DTI$case==1,]
X_2 <- DTI$rcst[DTI$visit==1 & DTI$case==1,]

## remove samples containing missing data
```

```
ind <- rowSums(is.na(X))>0
ind2 <- rowSums(is.na(X_2))>0

y <- y[!(ind | ind2)]
X <- X[!(ind | ind2),]
X_2 <- X_2[!(ind | ind2),]

N <- length(y)

## fit fgam using FA measurements along corpus callosum
## as functional predictor with PASAT as response
## using 8 cubic B-splines for marginal bases with third
## order marginal difference penalties
## specifying gamma>1 enforces more smoothing when using
## GCV to choose smoothing parameters
fit <- fgam(y~af(X,splinepars=list(k=c(8,8),m=list(c(2,3),c(2,3)))),gamma=1.2)
vis.fgam(fit)


## fgam term for the cca measurements plus an flm term for the rcst measurements
## leave out 10 samples for prediction
test <- sample(N,10)
fit <- fgam(y~af(X,splinepars=list(k=c(7,7),m=list(c(2,2),c(2,2))))+
  lf(X_2,splinepars=list(k=7,m=c(2,2))),subset=(1:N)[-test])
## plot the fits
plot(fit)
## vis.fgam(fit,af.term='X')
vis.fgam(fit,af.term='X',xval=.6)
## predict the ten left outs samples
pred <- predict(fit,newdata=list(X=X[test,],X_2=X_2[test,]),type='response',PredOutOfRange=TRUE)
sqrt(mean((y[test]-pred)^2))

## Try to predict the binary response disease status (case or control)
##   using the quantile transformed measurements from the rcst tract
##   with a smooth component for a scalar covariate that is pure noise
y <- DTI$case[DTI$visit==1]
X <- DTI$cca[DTI$visit==1,]
X_2 <- DTI$rcst[DTI$visit==1,]

ind <- rowSums(is.na(X))>0
ind2 <- rowSums(is.na(X_2))>0

y <- y[!(ind | ind2)]
X <- X[!(ind | ind2),]
X_2 <- X_2[!(ind | ind2),]
z1 <- rnorm(length(y))

# select=TRUE allows terms to be zeroed out of model completely
fit <- fgam(y~s(z1,k=10)+af(X_2,splinepars =list(k=c(7,7),m=list(c(2,1),c(2,1))),Qtransform=TRUE),
            family=binomial(),select=TRUE)
plot(fit)
vis.fgam(fit,af.term='X_2',plot.type='contour')
```

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 x yindex` matrix form (default for regular grid data) or in the shape of the originally supplied `ydata` argument or (default for sparse/irregular data), or, if `FALSE`, simply in the long vector shape returned by `resid.gam()`?
`...`	not used

Value

A matrix (grid data) or `data.frame` (sparse/irregular data) or vector (if `!reformat`) 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(Y = NULL, fdobj = NULL, X, con = NULL, argvals = NULL,
     method = c("OLS", "GLS", "mix"),
     gam.method = c("REML", "ML", "GCV.Cp", "GACV.Cp", "P-REML", "P-ML"),
     cov.method = c("naive", "mod.chol"), lambda = NULL,
     nbasis = 15, norder = 4, pen.order = 2,
     multi.sp = ifelse(method == "OLS", FALSE, TRUE), pve = 0.99, max.iter = 1,
     maxlam = NULL, cv1 = FALSE, scale = FALSE)
```

Arguments

Y,fdobj	the functional responses, given as either an $n \times d$ matrix Y or a functional data object (class "fd") as in the **fda** package.
X	the model matrix, whose columns represent scalar predictors. Should ordinarily include a column of 1s.
con	a row vector or matrix of linear contrasts of the coefficient functions, to be constrained to equal zero.
argvals	the d argument values at which the coefficient functions will be evaluated.
method	estimation method: "OLS" for penalized ordinary least squares, "GLS" for penalized generalized least squares, "mix" for mixed effect models.
gam.method	smoothing parameter selection method, to be passed to gam: "REML" for restricted maximum likelihood, "GCV.Cp" for generalized cross-validation.
cov.method	covariance estimation method: the current options are naive or modified Cholesky. See Details.
lambda	smoothing parameter value. If NULL, the smoothing parameter(s) will be estimated. See Details.
nbasis, norder	number of basis functions, and order of splines (the default, 4, gives cubic splines), for the B-spline basis used to represent the coefficient functions. When the functional responses are supplied using fdobj, these arguments are ignored in favor of the values pertaining to the supplied object.
pen.order	order of derivative penalty.
multi.sp	a logical value indicating whether separate smoothing parameters should be estimated for each coefficient function. Currently must be FALSE if method = "OLS".
pve	if method = 'mix', the percentage of variance explained by the principal components; defaults to 0.99.
max.iter	maximum number of iterations if method = "GLS".
maxlam	maximum smoothing parameter value to consider (when lamvec=NULL; see lofocv).
cv1	logical value indicating whether a cross-validation score should be computed even if a single fixed lambda is specified (when method = "OLS").
scale	logical value or vector determining scaling of the matrix X (see scale, to which the value of this argument is passed).

Details

The GLS method requires estimating the residual covariance matrix, which has dimension $d \times d$ when the responses are given by Y, or $nbasis \times nbasis$ when they are given by fdobj. When cov.method = "naive", the ordinary sample covariance is used. But this will be singular, or non-singular but unstable, in high-dimensional settings, which are typical. cov.method = "mod.chol" implements the modified Cholesky method of Pourahmadi (1999) for estimation of covariance matrices whose inverse is banded. The number of bands is chosen to maximize the p-value for a sphericity test (Ledoit and Wolf, 2002) applied to the "prewhitened" residuals. Note, however, that the banded inverse covariance assumption is sometimes inappropriate, e.g., for periodic functional responses.

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:

fd	object of class "fd" representing the estimated coefficient functions. Its main components are a basis and a matrix of coefficients with respect to that basis.
pca.resid	if method = "mix", an object representing a functional PCA of the residuals, performed by fpca.sc if the responses are in raw form or by pca.fd if in functional-data-object form.
U	if method = "mix", an $n \times m$ matrix of random effects, where m is the number of functional PC's needed to explain proportion pve of the residual variance. These random effects can be interpreted as shrunken FPC scores.
yhat, resid	objects of the same form as the functional responses (see arguments Y and fdobj), giving the fitted values and residuals.
est.func	matrix of values of the coefficient function estimates at the points given by argvals.
se.func	matrix of values of the standard error estimates for the coefficient functions, at the points given by argvals.
argvals	points at which the coefficient functions are evaluated.
fit	fit object outputted by amc.
edf	effective degrees of freedom of the fit.
lambda	smoothing parameter, or vector of smoothing parameters.
cv	cross-validated integrated squared error if method="OLS", otherwise NULL.
roughness	value of the roughness penalty.
resp.type	"raw" or "fd", indicating whether the responses were supplied in raw or functional-data-object form.

Author(s)

Philip Reiss <phil.reiss@nyumc.org>, Lan Huo, and Fabian Scheipl

References

Ledoit, O., and Wolf, M. (2002). Some hypothesis tests for the covariance matrix when the dimension is large compared to the sample size. *Annals of Statistics*, 30(4), 1081–1102.

Pourahmadi, M. (1999). Joint mean-covariance models with applications to longitudinal data: unconstrained parameterisation. *Biometrika*, 86(3), 677–690.

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/

See Also

plot.fosr

Examples

```
require(fda)
# 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 = fosr(fdobj = Temp.fd, X = modmat, con = constraints, method="OLS", lambda=100*10:30)
plot(olsmod, 1)
## Not run:
# Penalized GLS
glsmod = fosr(fdobj = Temp.fd, X = modmat, con = constraints, method="GLS")
plot(glsmod, 1)

## End(Not run)
```

---

fosr.perm *Permutation testing for function-on-scalar regression*

Description

fosr.perm() is a wrapper function calling fosr.perm.fit(), which fits models to permuted data, followed by fosr.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, fosr.perm() produces a plot using the plot function for class fosr.perm.

Usage

```
fosr.perm(Y=NULL, fdobj=NULL, X, con = NULL, X0 = NULL, con0 = NULL,
          argvals = NULL, lambda = NULL, lambda0 = NULL, multi.sp = FALSE,
          nperm, level = 0.05, plot = TRUE, xlabel = "", title = NULL,
          prelim = 15, ...)

fosr.perm.fit(Y = NULL, fdobj = NULL, X, con = NULL, X0 = NULL,
              con0 = NULL, argvals = NULL, lambda = NULL, lambda0 = NULL,
              multi.sp = FALSE, nperm, prelim, ...)

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

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

Arguments

Y, fdobj	the functional responses, given as either an $n \times d$ matrix Y or a functional data object (class "fd") as in the **fda** package.
X	the design matrix, whose columns represent scalar predictors.
con	a row vector or matrix of linear contrasts of the coefficient functions, to be restricted to equal zero.
X0	design matrix for the null-hypothesis model. If NULL, the null hypothesis is the intercept-only model.
con0	linear constraints for the null-hypothesis model.
argvals	the d argument values at which the coefficient functions will be evaluated.
lambda	smoothing parameter value. If NULL, the smoothing parameter(s) will be estimated. See fosr for details.
lambda0	smoothing parameter for null-hypothesis model.
multi.sp	a logical value indicating whether separate smoothing parameters should be estimated for each coefficient function. Currently must be FALSE if method = "OLS".
nperm	number of permutations.
prelim	number of preliminary permutations. The smoothing parameter in the main permutations will be fixed to the median value from these preliminary permutations. If prelim=0, this is not done.
level	significance level for the simultaneous test.
plot	logical value indicating whether to plot the real- and permuted-data pointwise F-type statistics.
xlabel	x-axis label for plots.
title	title for plot.
x	object of class fosr.perm, outputted by fosr.perm, fosr.perm.fit, or fosr.perm.test.
...	for fosr.perm and fosr.perm.fit, additional arguments passed to fosr. These arguments may include max.iter, method, gam.method, and scale. For plot.fosr.perm, graphical parameters (see par) 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.

`F`	pointwise F-type statistics at each of the points given by `argvals`.
`F.perm`	a matrix, each of whose rows gives the pointwise F-type statistics for a permuted data set.
`argvals`	points at which F-type statistics are computed.
`lambda.real`	smoothing parameter(s) for the real-data fit.
`lambda.prelim`	smoothing parameter(s) for preliminary permuted-data fits.
`lambda.perm`	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 `argvals`.
`n2s`	subset of {1, ..., `length(argvals)`} 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> and Lan Huo

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/

See Also

fosr

Examples

```
## Not run:
# 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

Xreg = cbind(1, model.matrix(~factor(CanadianWeather$region)-1))
conreg = 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(fdobj=tempfd, X=Xreg, con=conreg, 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)

## End(Not run)
```

`fosr2s` *Two-step function-on-scalar regression*

Description

This function performs linear regression with functional responses and scalar predictors by (1) fitting a separate linear model at each point along the function, and then (2) smoothing the resulting coefficients to obtain coefficient functions.

Usage

```
fosr2s(Y, X, argvals = seq(0, 1, , ncol(Y)), nbasis = 15, norder = 4,
pen.order = norder - 2, basistype = "bspline")
```

Arguments

`Y`	the functional responses, given as an $n \times d$ matrix.
`X`	$n \times p$ model matrix, whose columns represent scalar predictors. Should ordinarily include a column of 1s.
`argvals`	the d argument values at which the functional responses are evaluated, and at which the coefficient functions will be evaluated.
`nbasis`	number of basis functions used to represent the coefficient functions.
`norder`	norder of the spline basis, when `basistype="bspline"` (the default, 4, gives cubic splines).
`pen.order`	order of derivative penalty.
`basistype`	type of basis used. The basis is created by an appropriate constructor function from the **fda** package; see `basisfd`. Only `"bspline"` and `"fourier"` are supported.

Details

Unlike `fosr` and `pffr`, which obtain smooth coefficient functions by minimizing a penalized criterion, this function introduces smoothing only as a second step. The idea was proposed by Fan and Zhang (2000), who employed local polynomials rather than roughness penalization for the smoothing step.

Value

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

`fd`	object of class `"fd"` representing the estimated coefficient functions. Its main components are a basis and a matrix of coefficients with respect to that basis.
`raw.coef`	$d \times p$ matrix of coefficient estimates from regressing on `X` separately at each point along the function.
`raw.se`	$d \times p$ matrix of standard errors of the raw coefficient estimates.
`yhat`	$n \times d$ matrix of fitted values.
`est.func`	$d \times p$ matrix of coefficient function estimates, obtained by smoothing the columns of `raw.coef`.
`se.func`	$d \times p$ matrix of coefficient function standard errors.
`argvals`	points at which the coefficient functions are evaluated.
`lambda`	smoothing parameters (chosen by REML) used to smooth the p coefficient functions with respect to the supplied basis.

Author(s)

Philip Reiss <phil.reiss@nyumc.org> and Lan Huo

References

Fan, J., and Zhang, J.-T. (2000). Two-step estimation of functional linear models with applications to longitudinal data. *Journal of the Royal Statistical Society, Series B*, 62(2), 303–322.

See Also

`fosr`, `pffr`

Examples

```
require(fda)

# Effect of latitude on daily mean temperatures
tempmat = t(CanadianWeather$dailyAv[,,1])
latmat = cbind(1, scale(CanadianWeather$coord[ , 1], TRUE, FALSE))  # centred!
fzmod <- fosr2s(tempmat, latmat, argvals=day.5, basistype="fourier", nbasis=25)

par(mfrow=1:2)
ylabs = c("Intercept", "Latitude effect")
for (k in 1:2) {
with(fzmod,matplot(day.5, cbind(raw.coef[,k],raw.coef[,k]-2*raw.se[,k],
     raw.coef[,k]+2*raw.se[,k],est.func[,k],est.func[,k]-2*se.func[,k],
     est.func[,k]+2*se.func[,k]), type=c("p","l","l","l","l","l"),pch=16,
     lty=c(1,2,2,1,2,2),col=c(1,1,1,2,2,2), cex=.5,axes=FALSE,xlab="",ylab=ylabs[k]))
    axesIntervals()
    box()
    if (k==1) legend("topleft", legend=c("Raw","Smoothed"), col=1:2, lty=2)
}
```

`fpca.face` *Functional principal component analysis with fast covariance estimation*

Description

A fast implementation of the sandwich smoother (Xiao et al., 2013) for covariance matrix smoothing. Pooled generalized cross validation at the data level is used for selecting the smoothing parameter.

Usage

```
fpca.face(Y, Y.pred=NULL, center = TRUE, argvals = NULL,
          knots = 35, p = 3, m = 2, lambda = NULL,pve = 0.99,
          npc=NULL,alpha = 1, score.method = "int", search.grid = TRUE,
          search.length = 100, method = "L-BFGS-B", lower = -20,
          upper = 20, control = NULL)
```

Arguments

`Y`	data matrix (rows: observations; columns: grid of eval. points); missing data allowed but caution is needed for very sparse data
`Y.pred`	if desired, a matrix of observed functions that will be estimated using the FPC decomposition of Y.
`center`	center `Y` so that its column-means are 0? Defaults to `TRUE`
`argvals`	vector of points at which the functions are observed.
`knots`	vector of knots or number of (interior) knots; defaults to 35.
`p`	degrees of B-splines; defaults to 3.
`m`	order of differencing penalty; defaults to 2.
`lambda`	user-specified smoothing parameter; defaults to NULL and uses grid-searching method or `optim`.
`pve`	percentage of variance explained for selecting the number of principal components; defaults to 0.99.
`npc`	number of extracted principal components; defaults to NULL and use `pve`.
`alpha`	the coefficient of the penalty for degrees of freedom in the GCV criterion; defaults to 1
`score.method`	method for predicting scores; `"int"` (the default) for numerical integration, or `"blup"` for best linear unbiased prediction.
`search.grid`	logical; defaults to TRUE, if FALSE, uses `optima`.
`search.length`	number of equidistant (log scale) smoothing parameters to search; defaults to 100.
`method`	see `optim`; defaults to `L-BFGS-B`.
`lower, upper`	bounds for log smoothing parameter, passed to `optim`; defaults are -20 and 20.
`control`	passed to `optim`.

Value

A list with components

Yhat	If Y.pred is specified, the smooth version of Y.pred. Otherwise, if Y.pred=NULL, the smooth version of Y.
scores	matrix of scores
mu	mean function
npc	number of principal components
eigenvectors	matrix of eigenvectors
eigenvalues	vector of eigenvalues

Note

We expect to merge this function into fpca.sc in future releases of the package.

Author(s)

Luo Xiao <lxiao@jhsph.edu>

References

Xiao, L., Li, Y., and Ruppert, D. (2013). Fast bivariate *P*-splines: the sandwich smoother, *Journal of the Royal Statistical Society: Series B*, 75(3), 577-599.

Xiao, L., Ruppert, D., Zipunnikov, V., and Crainiceanu, C., (2013). Fast covariance estimation for high-dimensional functional data. Submitted). Available at http://arxiv.org/abs/1306.5718.

See Also

fpca.sc for another covariance-estimate based smoothing of Y; fpca2s and fpca.ssvd for two SVD-based smoothings.

Examples

```
#### settings
I <- 50 # number of subjects
J <- 3000 # dimension of the data
t <- (1:J)/J # a regular grid on [0,1]
N <- 4 #number of eigenfunctions
sigma <- 2 ##standard deviation of random noises
lambdaTrue <- c(1,0.5,0.5^2,0.5^3) # True eigenvalues

case = 1
### True Eigenfunctions

if(case==1) phi <- sqrt(2)*cbind(sin(2*pi*t),cos(2*pi*t),
                                 sin(4*pi*t),cos(4*pi*t))
if(case==2) phi <- cbind(rep(1,J),sqrt(3)*(2*t-1),
                         sqrt(5)*(6*t^2-6*t+1),
```

```
                          sqrt(7)*(20*t^3-30*t^2+12*t-1))

  ##################################################
  ########     Generate Data            #############
  ##################################################
  xi <- matrix(rnorm(I*N),I,N);
  xi <- xi%*%diag(sqrt(lambdaTrue))
  X <- xi%*%t(phi); # of size I by J
  Y <- X + sigma*matrix(rnorm(I*J),I,J)

results <- fpca.face(Y,center = TRUE, argvals=t,knots=100,pve=0.99)
##################################################
####               FACE               ########
##################################################
Phi <- results$eigenvectors
eigenvalues <- results$eigenvalues

for(k in 1:N){
  if(Phi[,k]%*%phi[,k]< 0)
    Phi[,k] <- - Phi[,k]
}

### plot eigenfunctions
par(mfrow=c(N/2,2))
seq <- (1:(J/10))*10
for(k in 1:N){
  plot(t[seq],Phi[seq,k]*sqrt(J),type="l",lwd = 3,
       ylim = c(-2,2),col = "red",
       ylab = paste("Eigenfunction ",k,sep=""),
       xlab="t",main="FACE")

  lines(t[seq],phi[seq,k],lwd = 2, col = "black")
}
```

`fpca.sc` *Functional principal components analysis by smoothed covariance*

Description

Decomposes functional observations using functional principal components analysis. A mixed model framework is used to estimate scores and obtain variance estimates.

Usage

```
fpca.sc(Y=NULL, ydata = NULL, Y.pred=NULL, argvals = NULL,
        random.int = FALSE, nbasis = 10, pve = .99, npc = NULL,
        var = FALSE, simul = FALSE, sim.alpha = .95,
        useSymm = FALSE, makePD = FALSE, center=TRUE, cov.est.method = 2)
```

Arguments

Y, ydata	the user must supply either Y, a matrix of functions observed on a regular grid, or a data frame ydata representing irregularly observed functions. See Details.
Y.pred	if desired, a matrix of functions to be approximated using the FPC decomposition.
argvals	function argument.
random.int	If TRUE, the mean is estimated by gamm4 with random intercepts. If FALSE (the default), the mean is estimated by gam treating all the data as independent.
nbasis	number of B-spline basis functions used for estimation of the mean function and bivariate smoothing of the covariance surface.
pve	proportion of variance explained: used to choose the number of principal components.
npc	prespecified value for the number of principal components (if given, this overrides pve).
var	TRUE or FALSE indicating whether model-based estimates for the variance of FPCA expansions should be computed.
simul	logical: should critical values be estimated for simultaneous confidence intervals?
sim.alpha	1 - coverage probability of the simultaneous intervals.
useSymm	logical, indicating whether to smooth only the upper triangular part of the naive covariance (when cov.est.method==2). This can save computation time for large data sets, and allows for covariance surfaces that are very peaked on the diagonal.
makePD	logical: should positive definiteness be enforced for the covariance surface estimate?
center	logical: should an estimated mean function be subtracted from Y? Set to FALSE if you have already demeaned the data using your favorite mean function estimate.
cov.est.method	covariance estimation method. If set to 1, a one-step method that applies a bivariate smooth to the $y(s_1)y(s_2)$ values. This can be very slow. If set to 2 (the default), a two-step method that obtains a naive covariance estimate which is then smoothed.

Details

This function computes a FPC decomposition for a set of observed curves, which may be sparsely observed and/or measured with error. A mixed model framework is used to estimate curve-specific scores and variances.

FPCA via kernel smoothing of the covariance function, with the diagonal treated separately, was proposed in Staniswalis and Lee (1998) and much extended by Yao et al. (2005), who introduced the "PACE" method. fpca.sc uses penalized splines to smooth the covariance function, as developed by Di et al. (2009) and Goldsmith et al. (2013).

The functional data must be supplied as either

- an $n \times d$ matrix Y, each row of which is one functional observation, with missing values allowed; or

- a data frame ydata, with columns '.id' (which curve the point belongs to, say i), '.index' (function argument such as time point t), and '.value' (observed function value $Y_i(t)$).

Value

Yhat	FPC approximation (projection onto leading components) of Y.pred if specified, or else of Y.
scores	$n \times npc$ matrix of estimated FPC scores.
mu	estimated mean function (or a vector of zeroes if center==FALSE).
efunctions	$d \times npc$ matrix of estimated eigenfunctions of the functional covariance, i.e., the FPC basis functions.
evalues	estimated eigenvalues of the covariance operator, i.e., variances of FPC scores.
npc	number of FPCs: either the supplied npc, or the minimum number of basis functions needed to explain proportion pve of the variance in the observed curves.
sigma2	estimated measurement error variance.
diag.var	diagonal elements of the covariance matrices for each estimated curve.
VarMats	a list containing the estimated covariance matrices for each curve in Y.
crit.val	estimated critical values for constructing simultaneous confidence intervals.

Author(s)

Jeff Goldsmith <jeff.goldsmith@columbia.edu>, Sonja Greven <sonja.greven@stat.uni-muenchen.de>, Lan Huo <Lan.Huo@nyumc.org>, Lei Huang <huangracer@gmail.com>, and Philip Reiss <phil.reiss@nyumc.org>

References

Di, C., Crainiceanu, C., Caffo, B., and Punjabi, N. (2009). Multilevel functional principal component analysis. *Annals of Applied Statistics*, 3, 458–488.

Goldsmith, J., Greven, S., and Crainiceanu, C. (2013). Corrected confidence bands for functional data using principal components. *Biometrics*, 69(1), 41–51.

Staniswalis, J. G., and Lee, J. J. (1998). Nonparametric regression analysis of longitudinal data. *Journal of the American Statistical Association*, 93, 1403–1418.

Yao, F., Mueller, H.-G., and Wang, J.-L. (2005). Functional data analysis for sparse longitudinal data. *Journal of the American Statistical Association*, 100, 577–590.

Examples

```
## Not run:
data(cd4)

Fit.MM = fpca.sc(cd4, var = TRUE, simul = TRUE)

# for one subject, examine curve estimate, pointwise and simultaneous itervals
EX = 1
EX.MM =  cbind(Fit.MM$Yhat[EX,],
      Fit.MM$Yhat[EX,] + 1.96 * sqrt(Fit.MM$diag.var[EX,]),
```

```
      Fit.MM$Yhat[EX,] - 1.96 * sqrt(Fit.MM$diag.var[EX,]),
      Fit.MM$Yhat[EX,] + Fit.MM$crit.val[EX] * sqrt(Fit.MM$diag.var[EX,]),
      Fit.MM$Yhat[EX,] - Fit.MM$crit.val[EX] * sqrt(Fit.MM$diag.var[EX,]))


par(mfrow=c(1,3))

# plot data for one subject, with curve and interval estimates
d = as.numeric(colnames(cd4))
plot(d[which(!is.na(cd4[EX,]))], cd4[EX,which(!is.na(cd4[EX,]))], type = 'o', pch = 19,
  cex=.75, ylim = range(0, 3400), xlim = range(d), xlab = "Months since seroconversion",
    lwd = 1.2, ylab = "Total CD4 Cell Count", main = "Est. & CI - Sampled Data")

matpoints(d, EX.MM, col = 4, type = 'l', lwd = c(2, 1, 1, 1, 1), lty = c(1,1,1,2,2))

# plot estimated mean function
plot(d, Fit.MM$mu, type = 'l', xlab = "Months since seroconversion",
  ylim = range(0, 3400), ylab = "Total CD4 Cell Count", main = "Est. Mean Function")

# plot the first estimated basis function
plot(d, Fit.MM$efunctions[,1], type = 'l', xlab = "Months since seroconversion",
  ylab = "Total CD4 Cell Count", main = "First Est. Basis Function")


# input a dataframe instead of a matrix

nid <- 20
nobs <- sample(10:20, nid, rep=TRUE)
ydata <- data.frame(
    .id = rep(1:nid, nobs),
    .index = round(runif(sum(nobs), 0, 1), 3))
ydata$.value <- unlist(tapply(ydata$.index,
                              ydata$.id,
                              function(x)
                                  runif(1, -.5, .5) +
                                  dbeta(x, runif(1, 6, 8), runif(1, 3, 5))
                              )
                       )
Fit.MM = fpca.sc(ydata=ydata, var = TRUE, simul = FALSE)
matplot(Fit.MM$efunctions[,1:2])

## End(Not run)
```

`fpca.ssvd` *Smoothed FPCA via iterative penalized rank one SVDs.*

Description

Implements the algorithm of Huang, Shen, Buja (2008) for finding smooth right singular vectors of a matrix `X` containing (contaminated) evaluations of functional random variables on a regular,

equidistant grid. If the number of smooth SVs to extract is not specified, the function hazards a guess for the appropriate number based on the asymptotically optimal truncation threshold under the assumption of a low rank matrix contaminated with i.i.d. Gaussian noise with unknown variance derived in Donoho, Gavish (2013). Please note that Donoho, Gavish (2013) should be regarded as experimental for functional PCA, and will typically not work well if you have more observations than grid points.

Usage

```
fpca.ssvd(Y, npc = NA, center = TRUE, maxiter = 15, tol = 1e-04,
  diffpen = 3, gridsearch = TRUE, alphagrid = 1.5^(-20:40),
  lower.alpha = 1e-05, upper.alpha = 1e+07, verbose = FALSE)
```

Arguments

Y	data matrix (rows: observations; columns: grid of eval. points)
npc	how many smooth SVs to try to extract, if NA (the default) the hard thresholding rule of Donoho, Gavish (2013) is used (see Details, References).
center	center Y so that its column-means are 0? Defaults to TRUE
maxiter	how many iterations of the power algorithm to perform at most (defaults to 15)
tol	convergence tolerance for power algorithm (defaults to 1e-4)
diffpen	difference penalty order controlling the desired smoothness of the right singular vectors, defaults to 3 (i.e., deviations from local quadratic polynomials).
gridsearch	use optimize or a grid search to find GCV-optimal smoothing parameters? defaults to TRUE.
alphagrid	grid of smoothing parameter values for grid search
lower.alpha	lower limit for for smoothing parameter if !gridsearch
upper.alpha	upper limit for smoothing parameter if !gridsearch
verbose	generate graphical summary of progress and diagnostic messages? defaults to FALSE

Value

a list like the returned object from fpca.sc, with entries Yhat, the smoothed trajectories, scores, the estimated FPC loadings, mu, the column means of Y (or a vector of zeroes if !center), efunctions, the estimated smooth FPCs (note that these are orthonormal vectors, not evaluations of orthonormal functions...), evalues, their associated eigenvalues, and npc, the number of smooth components that were extracted.

Author(s)

Fabian Scheipl

References

Huang, J. Z., Shen, H., and Buja, A. (2008). Functional principal components analysis via penalized rank one approximation. *Electronic Journal of Statistics*, 2, 678-695

Donoho, D.L., and Gavish, M. (2013). The Optimal Hard Threshold for Singular Values is 4/sqrt(3). eprint arXiv:1305.5870. Available from http://arxiv.org/abs/1305.5870.

See Also

fpca.sc and fpca.face for FPCA based on smoothing a covariance estimate; fpca2s for a faster SVD-based approach.

Examples

```
## as in Sec. 6.2 of Huang, Shen, Buja (2008):
 set.seed(2678695)
 n <- 101
 m <- 101
 s1 <- 20
 s2 <- 10
 s <- 4
 t <- seq(-1, 1, l=m)
 v1 <- t + sin(pi*t)
 v2 <- cos(3*pi*t)
 V <- cbind(v1/sqrt(sum(v1^2)), v2/sqrt(sum(v2^2)))
 U <- matrix(rnorm(n*2), n, 2)
 D <- diag(c(s1^2, s2^2))
 eps <- matrix(rnorm(m*n, sd=s), n, m)
 Y <- U%*%D%*%t(V) + eps

 smoothSV <- fpca.ssvd(Y, verbose=TRUE)

 layout(t(matrix(1:4, nr=2)))
 clrs <- sapply(rainbow(n), function(c)
           do.call(rgb, as.list(c(col2rgb(c)/255, .1))))
 matplot(V, type="l", lty=1, col=1:2, xlab="",
         main="FPCs: true", bty="n")
 matplot(smoothSV$efunctions, type="l", lty=1, col=1:5, xlab="",
         main="FPCs: estimate", bty="n")
 matplot(1:m, t(U%*%D%*%t(V)), type="l", lty=1, col=clrs, xlab="", ylab="",
         main="true smooth Y", bty="n")
 matplot(1:m, t(smoothSV$Yhat), xlab="", ylab="",
         type="l", lty=1,col=clrs, main="estimated smooth Y", bty="n")
```

---

fpca2s *Functional principal component analysis by a two-stage method*

Description

This function performs functional PCA by performing an ordinary singular value decomposition on the functional data matrix, then smoothing the right singular vectors by smoothing splines.

Usage

```
fpca2s(Y, npc=NA, center = TRUE, argvals = NULL, smooth=TRUE)
```

Arguments

Y	data matrix (rows: observations; columns: grid of eval. points)
npc	how many smooth SVs to try to extract. If NA (the default), the hard thresholding rule of Donoho and Gavish (2013) is used. Application of this rule to functional PCA should be regarded as experimental.
center	center Y so that its column-means are 0? Defaults to TRUE
argvals	index vector of J entries for data in X; defaults to a sequence from 0 to 1.
smooth	logical; defaults to TRUE, if NULL, no smoothing of eigenvectors.

Details

The eigenvalues are the same as those obtained from eigendecomposition of the sample covariance matrix. Please note that we expect to merge this function into fpca.ssvd in future versions of the package.

Value

A list with components

Yhat	predicted data matrix
scores	matrix of scores
mu	mean function
npc	number of principal components
eigenvectors	matrix of eigenvectors
eigenvalues	vector of eigenvalues

Author(s)

Luo Xiao <lxiao@jhsph.edu>

References

Xiao, L., Ruppert, D., Zipunnikov, V., and Crainiceanu, C., (2013), Fast covariance estimation for high-dimensional functional data. (submitted) http://arxiv.org/abs/1306.5718.

Donoho, D.L. and Gavish, M. (2013). The optimal hard threshold for singular values is 4/sqrt(3). eprint arXiv:1305.5870. Available from http://arxiv.org/abs/1305.5870.

See Also

fpca.sc and fpca.face for FPCA based on smoothing a covariance estimate; fpca.ssvd for another SVD-based approach.

Examples

```
#### settings
I <- 50 # number of subjects
J <- 3000 # dimension of the data
t <- (1:J)/J # a regular grid on [0,1]
N <- 4 #number of eigenfunctions
sigma <- 2 ##standard deviation of random noises
lambdaTrue <- c(1,0.5,0.5^2,0.5^3) # True eigenvalues

case = 1
### True Eigenfunctions

if(case==1) phi <- sqrt(2)*cbind(sin(2*pi*t),cos(2*pi*t),
                                 sin(4*pi*t),cos(4*pi*t))
if(case==2) phi <- cbind(rep(1,J),sqrt(3)*(2*t-1),
                         sqrt(5)*(6*t^2-6*t+1),
                         sqrt(7)*(20*t^3-30*t^2+12*t-1))

#####################################################
########     Generate Data            #############
#####################################################
xi <- matrix(rnorm(I*N),I,N);
xi <- xi%*%diag(sqrt(lambdaTrue))
X <- xi%*%t(phi); # of size I by J
Y <- X + sigma*matrix(rnorm(I*J),I,J)

results <- fpca2s(Y,npc=4,argvals=t)
#####################################################
####               SVDS              ########
#####################################################
Phi <- results$eigenvectors
eigenvalues <- results$eigenvalues

for(k in 1:N){
  if(Phi[,k]%*%phi[,k]< 0)
    Phi[,k] <- - Phi[,k]
}

### plot eigenfunctions
par(mfrow=c(N/2,2))
seq <- (1:(J/10))*10
for(k in 1:N){
     plot(t[seq],Phi[seq,k]*sqrt(J),type="l",lwd = 3,
          ylim = c(-2,2),col = "red",
          ylab = paste("Eigenfunction ",k,sep=""),
          xlab="t",main="SVDS")

     lines(t[seq],phi[seq,k],lwd = 2, col = "black")
     }
```

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, xfuncs = NULL, fdobj = NULL, ncomp = NULL, pve = 0.99, nbasis = NULL,
     basismat = NULL, penmat = NULL, argvals = NULL, covt = NULL,
     mean.signal.term = FALSE, spline.order = NULL, family = gaussian,
     method = "REML", sp = NULL, pen.order = 2, cv1 = FALSE, nfold = 5,
     store.cv = FALSE, store.gam = TRUE, ...)
```

Arguments

y	scalar outcome vector.
xfuncs	for 1D predictors, an $n \times d$ matrix of signals/functional predictors, where n is the length of y and d is the number of sites at which each signal is defined. For 2D predictors, an $n \times d1 \times d2$ array representing n images of dimension $d1 \times d2$.
fdobj	functional data object (class "fd") giving the functional predictors. Allowed only for 1D functional predictors.
ncomp	number of principal components. If NULL, this is chosen by pve.
pve	proportion of variance explained: used to choose the number of principal components.
nbasis	number(s) of B-spline basis functions: either a scalar, or a vector of values to be compared. For 2D predictors, tensor product B-splines are used, with the given basis dimension(s) in each direction; alternatively, nbasis can be given in the form list(v1,v2), in which case cross-validation will be performed for each combination of the first-dimension basis sizes in v1 and the second-dimension basis sizes in v2. Ignored if fdobj is supplied. If fdobj is *not* supplied, this defaults to 40 (i.e., 40 B-spline basis functions) for 1D predictors, and 15 (i.e., tensor product B-splines with 15 functions per dimension) for 2D predictors.
basismat	a $d \times K$ matrix of values of K basis functions at the d sites.
penmat	a $K \times K$ matrix defining a penalty on the basis coefficients.
argvals	points at which the functional predictors and the coefficient function are evaluated. By default, if 1D functional predictors are given by the $n \times d$ matrix xfuncs, argvals is set to d equally spaced points from 0 to 1; if they are given by fdobj, argvals is set to 401 equally spaced points spanning the domain of the given functions. For 2D (image) predictors supplied as an $n \times d1 \times d2$ array, argvals defaults to a list of (1) $d1$ equally spaced points from 0 to 1; (2) $d2$ equally spaced points from 0 to 1.

covt	covariates: an n-row matrix, or a vector of length n.
mean.signal.term	logical: should the mean of each subject's signal be included as a covariate?
spline.order	order of B-splines used, if fdobj is not supplied; defaults to 4, i.e., cubic B-splines.
family	generalized linear model family. Current version supports "gaussian" (the default) and "binomial".
method	smoothing parameter selection method, passed to function gam; see the gam documentation for details.
sp	a fixed smoothing parameter; if NULL, an optimal value is chosen (see method).
pen.order	order of derivative penalty applied when estimating the coefficient function; defaults to 2.
cv1	logical: should cross-validation be performed to select the best model if only one set of tuning parameters provided? By default, FALSE. Note that, if there are multiple sets of tuning parameters provided, cv is always performed.
nfold	the number of validation sets ("folds") into which the data are divided; by default, 5.
store.cv	logical: should a CV result table be in the output? By default, FALSE.
store.gam	logical: should the gam object be included in the output? Defaults to TRUE.
...	other arguments passed to function gam.

Details

One-dimensional functional predictors can be 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 xfuncs (see Method 2 in the example). In the latter case, arguments basismat and penmat can also be used to specify the basis and/or penalty matrices (see Method 3).

For two-dimensional predictors, functional data object form is not supported. Instead of radial B-splines as in Reiss and Ogden (2010), this implementation employs tensor product cubic B-splines, sometimes known as bivariate O-splines (Ormerod, 2008).

For purposes of interpreting the fitted coefficients, please note that the functional predictors are decorrelated from the scalar predictors before fitting the model (when there are no scalar predictors other than the intercept, this just means the columns of the functional predictor matrix are demeaned); see section 3.2 of Reiss (2006) for details.

Value

A list with components

gamObject	if store.gam = TRUE, an object of class gam (see gamObject in the **mgcv** package documentation).
fhat	coefficient function estimate.
se	pointwise Bayesian standard error.
undecor.coef	undecorrelated coefficient for covariates.

argvals	the supplied value of argvals.
cv.table	a table giving the CV criterion for each combination of nbasis and ncomp, if store.cv = TRUE; otherwise, the CV criterion only for the optimized combination of these parameters. Set to NULL if CV is not performed.
nbasis, ncomp	when CV is performed, the values of nbasis and comp that minimize the CV criterion.

Author(s)

Philip Reiss <phil.reiss@nyumc.org>, Lan Huo <lan.huo@nyumc.org>, and Lei Huang <huangracer@gmail.com>

References

Ormerod, J. T. (2008). On semiparametric regression and data mining. Ph.D. dissertation, School of Mathematics and Statistics, University of New South Wales.

Ramsay, J. O., Hooker, G., and Graves, S. (2009). *Functional Data Analysis with R and MATLAB*. New York: Springer.

Reiss, P. T. (2006). Regression with signals and images as predictors. Ph.D. dissertation, Department of Biostatistics, Columbia University. Available at http://works.bepress.com/phil_reiss/11/.

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.

See Also

wcr, wnet

Examples

```
require(fda)
### 1D functional predictor example ###

######### 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$octane, fdobj = gas.fd, ncomp = 30)
plot(gasmod1, xlab="Wavelength")
## Not run:
# Method 2: Call fpcr with explicit signal matrix
gasmod2 = fpcr(gasoline$octane, xfuncs = gasoline$NIR, ncomp = 30)
# Method 3: Call fpcr with explicit signal, basis, and penalty matrices
gasmod3 = fpcr(gasoline$octane, xfuncs = gasoline$NIR,
               basismat = eval.basis(wavelengths, bbasis),
               penmat = getbasispenalty(bbasis), ncomp = 30)

# 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.

## End(Not run)

### 2D functional predictor example ###

n = 200; d = 70

# Create true coefficient function
ftrue = matrix(0,d,d)
ftrue[40:46,34:38] = 1

# Generate random functional predictors, and scalar responses
ii = array(rnorm(n*d^2), dim=c(n,d,d))
iimat = ii; dim(iimat) = c(n,d^2)
yy = iimat %*% as.vector(ftrue) + rnorm(n, sd=.3)

mm = fpcr(yy, ii, ncomp=40)

image(ftrue)
contour(mm$fhat, add=TRUE)

## Not run:
### Cross-validation ###
cv.gas = fpcr(gasoline$octane, xfuncs = gasoline$NIR,
                 nbasis=seq(20,40,5), ncomp = seq(10,20,5), store.cv = TRUE)
image(seq(20,40,5), seq(10,20,5), cv.gas$cv.table, xlab="Basis size",
      ylab="Number of PCs", xaxp=c(20,40,4), yaxp=c(10,20,2))


## End(Not run)
```

---

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/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`, `wcr`, `wnet`

---

lf *Construct an FLM regression term*

Description

Defines a term $\int_T \beta(t)X_i(t)dt$ for inclusion in an `mgcv::gam-formula` (or `bam` or `gamm` or `gamm4:::gamm`) as constructed by `fgam`, where $\beta(t)$ is an unknown coefficient function and $X_i(t)$ is a functional predictor on the closed interval T. Defaults to a cubic B-spline with second-order difference penalties for estimating $\beta(t)$. The functional predictor must be fully observed on a regular grid.

Usage

```
lf(X, xind = seq(0, 1, l = ncol(X)),
   integration = c("simpson", "trapezoidal", "riemann"),
   L = NULL,
   splinepars = list(bs = "ps", k= min(ceiling(n/4), 40), m = c(2, 2)),
   presmooth = TRUE)
```

Arguments

X	an N by J=ncol(xind) matrix of function evaluations $X_i(t_{i1}), ., X_i(t_{iJ}); i = 1, ., N$.
xind	matrix (or vector) of indices of evaluations of $X_i(t)$; i.e. a matrix with *i*th row $(t_{i1}, ., t_{iJ})$.
integration	method used for numerical integration. Defaults to "simpson"'s rule for calculating entries in L. Alternatively and for non-equidistant grids, "trapezoidal" or "riemann". "riemann" integration is always used if limits is specified.
L	optional: an N by ncol(xind) matrix giving the weights for the numerical integration over t.
splinepars	optional arguments specifying options for representing and penalizing the functional coefficient $\beta(t)$. Defaults to a cubic B-spline with second-order difference penalties, i.e. list(bs="ps", m=c(2, 1)) See te or s in mgcv for details.
presmooth	If true, the functional predictor is pre-smoothed prior to fitting. See smooth.basisPar in package fda

Value

a list with the following entries

call	a "call" to te (or s, t2) using the appropriately constructed covariate and weight matrices.
xind	the xind argument supplied to lf.
L	the matrix of weights used for the integration.
xindname	the name used for the functional predictor variable in the formula used by mgcv.
tindname	the name used for xind variable in the formula used by mgcv.
LXname	the name used for the L variable in the formula used by mgcv.
presmooth	the presmooth argument supplied to lf.
Xfd	an fd object from presmoothing the functional predictors using smooth.basisPar. Only present if presmooth=TRUE. See fd.

Author(s)

Mathew W. McLean <mwm79@cornell.edu> and Fabian Scheipl

See Also

fgam, af, mgcv's linear.functional.terms, fgam for examples.

lofocv *Leave-one-function-out cross-validation*

Description

This internal function, called by `fosr()` 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 `optimize`.
`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>

See Also

`fosr`, `pwcv`

lpeer *Longitudinal Functional Models with Structured Penalties*

Description

Implements longitudinal functional model with structured penalties (Kundu et al., 2012) with scalar outcome, single functional predictor, one or more scalar covariates and subject-specific random intercepts through mixed model equivalence.

Usage

```
lpeer(Y, subj, t, funcs, covariates=NULL, comm.pen=TRUE, pentype='Ridge', L.user=NULL,
f_t=NULL, Q=NULL, phia=10^3, se=FALSE, ...)
```

Arguments

Y	vector of all outcomes over all visits or timepoints
subj	vector containing the subject number for each observation
t	vector containing the time information when the observation are taken
covariates	matrix of scalar covariates.
funcs	matrix containing observed functional predictors as rows. Rows with NA and Inf values will be deleted.
comm.pen	logical value indicating whether common penalty for all the components of regression function. Default is TRUE.
pentype	type of penalty: either decomposition based penalty ('DECOMP') or ridge ('RIDGE') or second-order difference penalty ('D2') or any user defined penalty ('USER'). For decomposition based penalty user need to specify Q matrix in Q argument (see details). For user defined penalty user need to specify L matrix in L argument (see details). For Ridge and second-order difference penalty, specification for arguments L and Q will be ignored. Default is 'RIDGE'.
f_t	vector or matrix with number of rows equal to number of total observations and number of columns equal to d (see details). If matrix then each column pertains to single function of time and the value in the column represents the realization corresponding to time vector t. The column with intercept or multiple of intercept will be dropped. A NULL value refers to time-invariant regression function. Default value is NULL.
Q	Q matrix to derive decomposition based penalty. Need to be specified with pentype='DECOMP'. When comm.pen=TRUE, number of columns must equal number of columns of matrix specified to funcs. When comm.pen=FALSE, Number of columns need to be equal with the number of columns of matrix specified to funcs times the number of components of regression function. Each row represents a basis function where functional predictor is expected lie according to prior belief. This argument will be ignored when value of pentype is other than 'DECOMP'.

L.user	penalty matrix. Need to be specified with pentype='USER'. When comm.pen=TRUE, Number of columns need to be equal with number of columns of matrix specified to funcs. When comm.pen=FALSE, Number of columns need to be equal with the number of columns of matrix specified to funcs times the number of components of regression function. Each row represents a constraint on functional predictor. This argument will be ignored when value of pentype is other than 'USER'.
phia	scalar value of a in decomposition based penalty. Needs to be specified with pentype='DECOMP'.
se	logical; calculate standard error when TRUE.
...	additional arguments passed to lme.

Details

If there are any missing or infinite values in Y, subj, t, covariates, funcs and f_t, the corresponding row (or observation) will be dropped, and infinite values are not allowed for these arguments. Neither Q nor L may contain missing or infinite values. lpeer() fits the following model:

$y_{i(t)} = X_{i(t)}^T \beta + \int W_{i(t)}(s)\gamma(t,s)ds + Z_{i(t)}u_i + \epsilon_{i(t)}$

where $\epsilon_{i(t)} \tilde{} N(0, \sigma^2)$ and $u_i \tilde{} N(0, \sigma_u^2)$. For all the observations, predictor function $W_{i(t)}(s)$ is evaluated at K sampling points. Here, regression function $\gamma(t,s)$ is represented in terms of (d+1) component functions $\gamma_0(s)$,..., $\gamma_d(s)$ as follows

$\gamma(t,s) = \gamma_0(s) + f_1(t)\gamma_1(s) + f_d(t)\gamma_d(s)$

Values of $y_{i(t)}, X_{i(t)}$ and $W_{i(t)}(s)$ are passed through argument Y, covariates and funcs, respectively. Number of elements or rows in Y, t, subj, covariates (if not NULL) and funcs need to be equal.

Values of $f_1(t), ..., f_d(t)$ are passed through f_t argument. The matrix passed through f_t argument should have d columns where each column represents one and only one of $f_1(t), ..., f_d(t)$.

The estimate of (d+1) component functions $\gamma_0(s)$,..., $\gamma_d(s)$ is obtained as penalized estimated. The following 3 types of penalties can be used for a component function:

i. Ridge: I_K

ii. Second-order difference: $[d_{i,j}]$ with $d_{i,i} = d_{i,i+2} = 1, d_{i,i+1} = -2$, otherwise $d_{i,j} = 0$

iii. Decomposition based penalty: $bP_Q + a(I - P_Q)$ where $P_Q = Q^T(QQ^T)^{-1}Q$

For Decomposition based penalty the user must specify pentype= 'DECOMP' and the associated Q matrix must be passed through the Q argument. Alternatively, one can directly specify the penalty matrix by setting pentype= 'USER' and using the L argument to supply the associated L matrix.

If Q (or L) matrix is similar for all the component functions then argument comm.pen should have value TRUE and in that case specified matrix to argument Q (or L) should have K columns. When Q (or L) matrix is different for all the component functions then argument comm.pen should have value FALSE and in that case specified matrix to argument Q (or L) should have K(d+1) columns. Here first K columns pertains to first component function, second K columns pertains to second component functions, and so on.

Default penalty is Ridge penalty for all the component functions and user needs to specify 'RIDGE'. For second-order difference penalty, user needs to specify 'D2'. When pentype is 'RIDGE' or 'D2' the value of comm.pen is always TRUE and comm.pen=FALSE will be ignored.

Value

A list containing:

fit	result of the call to lme
fitted.vals	predicted outcomes
BetaHat	parameter estimates for scalar covariates including intercept
se.Beta	standard error of parameter estimates for scalar covariates including intercept
Beta	parameter estimates with standard error for scalar covariates including intercept
GammaHat	estimates of components of regression functions. Each column represents one component function.
Se.Gamma	standard error associated with GammaHat
AIC	AIC value of fit (smaller is better)
BIC	BIC value of fit (smaller is better)
logLik	(restricted) log-likelihood at convergence
lambda	list of estimated smoothing parameters associated with each component function
V	conditional variance of Y treating only random intercept as random one.
V1	unconditional variance of Y
N	number of subjects
K	number of Sampling points in functional predictor
TotalObs	total number of observations over all subjects
Sigma.u	estimated sd of random intercept.
sigma	estimated within-group error standard deviation.

Author(s)

Madan Gopal Kundu <mgkundu@iupui.edu>

References

Kundu, M. G., Harezlak, J., and Randolph, T. W. (2012). Longitudinal functional models with structured penalties (arXiv:1211.4763 [stat.AP]).

Randolph, T. W., Harezlak, J, and Feng, Z. (2012). Structured penalties for functional linear models - partially empirical eigenvectors for regression. *Electronic Journal of Statistics*, 6, 323–353.

See Also

peer, plot.lpeer

Examples

```
## Not run:
#-------------------------------------------------------------------------
# Example 1: Estimation with Ridge penalty
#-------------------------------------------------------------------------

##Load Data
data(DTI)

## Extract values for arguments for lpeer() from given data
cca = DTI$cca[which(DTI$case == 1),]
DTI = DTI[which(DTI$case == 1),]

##1.1 Fit the model with single component function
##    gamma(t,s)=gamm0(s)
t<- DTI$visit
fit.cca.lpeer1 = lpeer(Y=DTI$pasat, t=t, subj=DTI$ID, funcs = cca)
plot(fit.cca.lpeer1)

##1.2 Fit the model with two component function
##    gamma(t,s)=gamm0(s) + t*gamma1(s)
fit.cca.lpeer2 = lpeer(Y=DTI$pasat, t=t, subj=DTI$ID, funcs = cca,
                       f_t=t, se=TRUE)
plot(fit.cca.lpeer2)

#-------------------------------------------------------------------------
# Example 2: Estimation with structured penalty (need structural
#            information about regression function or predictor function)
#-------------------------------------------------------------------------

##Load Data
data(PEER.Sim)

## Extract values for arguments for lpeer() from given data
K<- 100
W<- PEER.Sim[,c(3:(K+2))]
Y<- PEER.Sim[,K+3]
t<- PEER.Sim[,2]
id<- PEER.Sim[,1]

##Load Q matrix containing structural information
data(Q)

##2.1 Fit the model with two component function
##    gamma(t,s)=gamm0(s) + t*gamma1(s)
Fit1<- lpeer(Y=Y, subj=id, t=t, covariates=cbind(t), funcs=W,
    pentype='DECOMP', f_t=cbind(1,t), Q=Q, se=TRUE)

Fit1$Beta
plot(Fit1)

##2.2 Fit the model with three component function
```

```
##    gamma(t,s)=gamm0(s) + t*gamma1(s) + t^2*gamma1(s)
Fit2<- lpeer(Y=Y, subj=id, t=t, covariates=cbind(t), funcs=W,
     pentype='DECOMP', f_t=cbind(1,t, t^2), Q=Q, se=TRUE)

Fit2$Beta
plot(Fit2)

##2.3 Fit the model with two component function with different penalties
##    gamma(t,s)=gamm0(s) + t*gamma1(s)
Q1<- cbind(Q, Q)
Fit3<- lpeer(Y=Y, subj=id, t=t, covariates=cbind(t), comm.pen=FALSE, funcs=W,
     pentype='DECOMP', f_t=cbind(1,t), Q=Q1, se=TRUE)

##2.4 Fit the model with two component function with user defined penalties
##    gamma(t,s)=gamm0(s) + t*gamma1(s)
phia<- 10^3
P_Q <- t(Q)%*%solve(Q%*%t(Q))%*% Q
L<- phia*(diag(K)- P_Q) + 1*P_Q
Fit4<- lpeer(Y=Y, subj=id, t=t, covariates=cbind(t), funcs=W,
     pentype='USER', f_t=cbind(1,t), L=L, se=TRUE)

L1<- adiag(L, L)
Fit5<- lpeer(Y=Y, subj=id, t=t, covariates=cbind(t), comm.pen=FALSE, funcs=W,
     pentype='USER', f_t=cbind(1,t), L=L1, se=TRUE)

## End(Not run)
```

`lpfr` *Longitudinal penalized functional regression*

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 gam 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	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 <jeff.goldsmith@columbia.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*, 61(3), 453–469.

Examples

```
## Not run:
######################################################################
# 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")

## End(Not run)
```

`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

`pcre` *pffr-constructor for functional principal component-based functional random intercepts.*

Description

pffr-constructor for functional principal component-based functional random intercepts.

Usage

```
pcre(id, efunctions, evalues, yind, ...)
```

Arguments

id	grouping variable a factor
efunctions	matrix of eigenfunction evaluations on gridpoints yind (<length of yind> x <no. of used eigenfunctions>)
evalues	eigenvalues associated with efunctions
yind	vector of gridpoints on which efunctions are evaluated.
...	not used

Value

a list used internally for constructing an appropriate call to mgcv::gam

Details

Fits functional random intercepts $B_i(t)$ for a grouping variable `id` using as a basis the functions $\phi_m(t)$ in `efunctions` with variances λ_m in `evalues`: $B_i(t) \approx \sum_m^M \phi_m(t)\delta_{im}$ with independent $\delta_{im} \sim N(0, \sigma^2\lambda_m)$, where σ^2 is (usually) estimated and controls the overall contribution of the $B_i(t)$ while the relative importance of the M basisfunctions is controlled by the supplied variances `lambda_m`. Can be used to model smooth residuals if `id` is simply an index of observations.

`efunctions` and `evalues` are typically eigenfunctions and eigenvalues of an estimated covariance operator for the functional process to be modeled, i.e., they are a functional principal components basis.

Author(s)

Fabian Scheipl

Examples

```
## Not run:
residualfunction <- function(t){
#generate quintic polynomial error functions
    drop(poly(t, 5)%*%rnorm(5, sd=sqrt(2:6)))
}
# generate data Y(t) = mu(t) + E(t) + white noise
set.seed(1122)
n <- 50
T <- 30
t <- seq(0,1, l=T)
# E(t): smooth residual functions
E <- t(replicate(n, residualfunction(t)))
int <- matrix(scale(3*dnorm(t, m=.5, sd=.5) - dbeta(t, 5, 2)), byrow=T, n, T)
Y <- int + E + matrix(.2*rnorm(n*T), n, T)
data <- data.frame(Y=I(Y))
# fit model under independence assumption:
summary(m0 <- pffr(Y ~ 1, yind=t, data=data))
# get first 5 eigenfunctions of residual covariance
# (i.e. first 5 functional PCs of empirical residual process)
Ehat <- resid(m0)
fpcE <- fpca.sc(Ehat, npc=5)
efunctions <- fpcE$efunctions
evalues <- fpcE$evalues
data$id <- factor(1:nrow(data))
# refit model with fpc-based residuals
m1 <- pffr(Y ~ 1 + pcre(id=id, efunctions=efunctions, evalues=evalues, yind=t), yind=t, data=data)
t1 <- predict(m1, type="terms")
summary(m1)
#compare squared errors
mean((int-fitted(m0))^2)
mean((int-t1[[1]])^2)
mean((E-t1[[2]])^2)
# compare fitted & true smooth residuals and fitted intercept functions:
layout(t(matrix(1:4,2,2)))
matplot(t(E), lty=1, type="l", ylim=range(E, t1[[2]]))
```

```
matplot(t(t1[[2]]), lty=1, type="l", ylim=range(E, t1[[2]]))
plot(m1, select=1, main="m1", ylim=range(Y))
lines(t, int[1,], col=rgb(1,0,0,.5))
plot(m0, select=1, main="m0", ylim=range(Y))
lines(t, int[1,], col=rgb(1,0,0,.5))

## End(Not run)
```

---

peer *Functional Models with Structured Penalties*

Description

Implements functional model with structured penalties (Randloph et al., 2012) with scalar outcome and single functional predictor through mixed model equivalence.

Usage

```
peer(Y, funcs, pentype='Ridge', L.user=NULL, Q=NULL, phia=10^3, se=FALSE, ...)
```

Arguments

Y	vector of all outcomes
funcs	matrix containing observed functional predictors as rows. Rows with NA and Inf values will be deleted.
pentype	type of penalty. It can be either decomposition based penalty (DECOMP) or ridge (RIDGE) or second-order difference penalty (D2) or any user defined penalty (USER). For decomposition based penalty user need to specify Q matrix in Q argument (see details). For user defined penalty user need to specify L matrix in L argument (see details). For Ridge and second-order difference penalty, specification for arguments L and Q will be ignored. Default is RIDGE.
L.user	penalty matrix. Need to be specified with pentype='USER'. Number of columns need to be equal with number of columns of matrix specified to funcs. Each row represents a constraint on functional predictor. This argument will be ignored when value of pentype is other than USER.
Q	Q matrix to derive decomposition based penalty. Need to be specified with pentype='DECOMP'. Number of columns need to be equal with number of columns of matrix specified to funcs. Each row represents a basis function where functional predictor is expected lie according to prior belief. This argument will be ignored when value of pentype is other than DECOMP.
phia	Scalar value of a in decomposition based penalty. Need to be specified with pentype='DECOMP'.
se	logical; calculate standard error when TRUE.
...	additional arguments passed to the lme function.

Details

If there are any missing or infinite values in Y, and funcs, the corresponding row (or observation) will be dropped. Neither Q nor L may contain missing or infinite values.

peer() fits the following model:

$y_i = \int W_i(s)\gamma(s)ds + \epsilon_i$

where $\epsilon_i \tilde{} N(0, \sigma^2)$. For all the observations, predictor function $W_i(s)$ is evaluated at K sampling points. Here, $\gamma(s)$ denotes the regression function.

Values of y_i and $W_i(s)$are passed through argument Y and funcs, respectively. Number of elements or rows in Y and funcs need to be equal.

The estimate of regression functions $\gamma(s)$ is obtained as penalized estimated. Following 3 types of penalties can be used:

i. Ridge: I_K

ii. Second-order difference: $[d_{i,j}]$ with $d_{i,i} = d_{i,i+2} = 1, d_{i,i+1} = -2$, otherwise $d_{i,i} = 0$

iii. Decomposition based penalty: $bP_Q + a(I - P_Q)$ where $P_Q = Q^T(QQ^T)^{-1}Q$

For Decomposition based penalty user need to specify pentype='DECOMP' and associated Q matrix need to be passed through Q argument.

Alternatively, user can pass directly penalty matrix through argument L. For this user need to specify pentype='USER' and associated L matrix need to be passed through L argument.

Default penalty is Ridge penalty and user needs to specify RIDGE. For second-order difference penalty, user needs to specify D2.

Value

a list containing:

fit	result of the call to lme
fitted.vals	predicted outcomes
Gamma	estimates with standard error for regression function
GammaHat	estimates of regression function
se.Gamma	standard error associated with GammaHat
AIC	AIC value of fit (smaller is better)
BIC	BIC value of fit (smaller is better)
logLik	(restricted) log-likelihood at convergence
lambda	estimates of smoothing parameter
N	number of subjects
K	number of Sampling points in functional predictor
sigma	estimated within-group error standard deviation.

Author(s)

Madan Gopal Kundu <mgkundu@iupui.edu>

References

Kundu, M. G., Harezlak, J., and Randolph, T. W. (2012). Longitudinal functional models with structured penalties (arXiv:1211.4763 [stat.AP]).

Randolph, T. W., Harezlak, J, and Feng, Z. (2012). Structured penalties for functional linear models - partially empirical eigenvectors for regression. *Electronic Journal of Statistics*, 6, 323–353.

See Also

`lpeer`, `plot.peer`

Examples

```
## Not run:
#-----------------------------------------------------------------------
# Example 1: Estimation with D2 penalty
#-----------------------------------------------------------------------

## Load Data
data(DTI)

## Extract values for arguments for peer() from given data
cca = DTI$cca[which(DTI$case == 1),]
DTI = DTI[which(DTI$case == 1),]

##1.1 Fit the model
fit.cca.peer1 = peer(Y=DTI$pasat, funcs = cca, pentype='D2', se=TRUE)
plot(fit.cca.peer1)

#-----------------------------------------------------------------------
# Example 2: Estimation with structured penalty (need structural
#            information about regression function or predictor function)
#-----------------------------------------------------------------------

## Load Data
data(PEER.Sim)

## Extract values for arguments for peer() from given data
PEER.Sim1<- subset(PEER.Sim, t==0)
K<- 100
W<- PEER.Sim1[,c(3:(K+2))]
Y<- PEER.Sim1[,K+3]

##Load Q matrix containing structural information
data(Q)

##2.1 Fit the model
Fit1<- peer(Y=Y, funcs=W, pentype='Decomp', Q=Q, se=TRUE)
plot(Fit1)

## End(Not run)
```

PEER.Sim, Q — *Simulated longitudinal data with functional predictor and scalar response, and structural information associated with predictor function*

Description

PEER.Sim contains simulated observations from 100 subjects, each observed at 4 distinct timepoints. At each timepoint bumpy predictor profile is generated randomly and the scalar response variable is generated considering a time-varying regression function and subject intercept. Accompanying the functional predictor and scalar response are the subject ID numbers and time of measurements.

Q represents the 7 x 100 matrix where each row provides structural information about the functional predictor profile for data PEER.Sim. For specific details about the simulation and Q matrix, please refer to Kundu et. al. (2012).

Usage

```
data(PEER.Sim)
data(Q)
```

Format

The data frame PEER.Sim is made up of subject ID number(id), subject-specific time of measurement (t), functional predictor profile (W.1-W.100) and scalar response (Y)

References

Kundu, M. G., Harezlak, J., and Randolph, T. W. (2012). Longitudinal functional models with structured penalties. (please contact J. Harezlak at harezlak@iupui.edu)

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
$E(Y_i(t)) = g(\mu(t) + \int X_i(s)\beta(s,t)ds + f(z_{1i},t) + f(z_{2i}) + z_{3i}\beta_3(t) + \ldots)$
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, data = NULL, ydata = NULL, algorithm = NA,
  method = "REML", tensortype = c("ti", "t2"), bs.yindex = list(bs = "ps",
  k = 5, m = c(2, 1)), bs.int = list(bs = "ps", k = 20, m = c(2, 1)), ...)
```

Arguments

formula	a formula with special terms as for gam, with additional special terms ff(), sff(), ffpc(), pcre() and c().
yind	a vector with length equal to the number of columns of the matrix of functional responses giving the vector of evaluation points $(t_1, \ldots, t_G)$. If not supplied, yind is set to 1:ncol(<response>).
algorithm	the name of the function used to estimate the model. Defaults to gam if the matrix of functional responses has less than 2e5 data points and to bam if not. 'gamm' and 'gamm4' are valid options as well. For 'gamm4', see Details.
data	an (optional) data.frame or a named list containing the data. Functional covariates have to be supplied as n by <no. of evaluations> 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.
ydata	an (optional) data.frame supplying functional responses that are not observed on a regular grid. See Details.
method	Defaults to "REML"-estimation, including of unknown scale. See gam for details.
bs.yindex	a named (!) list giving the parameters for spline bases on the index of the functional response. Defaults to list(bs="ps", k=5, m=c(2, 1)), i.e. 5 cubic B-splines bases with first order difference penalty.
bs.int	a named (!) list giving the parameters for the spline basis for the global functional intercept. Defaults to list(bs="ps", k=20, m=c(2, 1)), i.e. 20 cubic B-splines bases with first order difference penalty.
tensortype	which typ of tensor product splines to use. One of "ti" or "t2", defaults to ti. t2-type terms do not enforce the more suitable special constraints for functional regression, see Details.
...	additional arguments that are valid for gam or bam. weights, subset, offset are not yet implemented!

Value

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

Details

The routine can estimate

1. linear functional effects of scalar (numeric or factor) covariates that vary smoothly over t (e.g. $z_{1i}\beta_1(t)$, specified as ~z1),

2. nonlinear, and possibly multivariate functional effects of (one or multiple) scalar covariates z that vary smoothly over the index t of $Y(t)$ (e.g. $f(z_{2i}, t)$, specified in the `formula` simply as `~s(z2)`)
3. (nonlinear) effects of scalar covariates that are constant over t (e.g. $f(z_{3i})$, specified as `~c(s(z3))`, or $\beta_3 z_{3i}$, specified as `~c(z3)`),
4. function-on-function regression terms (e.g. $\int X_i(s)\beta(s,t)ds$, specified as `~ff(X, yindex=t, xindex=s)`, see `ff`). Terms given by `sff` and `ffpc` provide nonlinear and FPC-based effects of functional covariates, respectively.
5. concurrent effects of functional covariates `X` measured on the same grid as the response are specified as follows: `~s(x)` for a smooth, index-varying effect $f(X(t), t)$, `~x` for a linear index-varying effect $X(t)\beta(t)$, `~c(s(x))` for a constant nonlinear effect $f(X(t))$, `~c(x)` for a constant linear effect $X(t)\beta$.
6. Smooth functional random intercepts $b_{0g(i)}(t)$ for a grouping variable `g` with levels $g(i)$ can be specified via `~s(g, bs="re")`), functional random slopes $u_i b_{1g(i)}(t)$ in a numeric variable `u` via `~s(g, u, bs="re")`). Scheipl, Staicu, Greven (2013) contains code examples for modeling correlated functional random intercepts using `mrf`-terms.

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. For factor variables, a separate smooth function with its own smoothing parameter is estimated for each level of the factor.

The marginal spline basis used for the index of the the functional response is specified via the *global* argument `bs.yindex`. If necessary, this can be overriden for any specific term by supplying a `bs.yindex`-argument to that term in the formula, e.g. `~s(x, bs.yindex=list(bs="tp", k=7))` would yield a tensor product spline for which the marginal basis for the index of the response are 7 cubic thin-plate spline functions overriding the global default for the basis and penalty on the index of the response given by the *global* `bs.yindex`-argument .
Use `~-1 + c(1) + ...` to specify a model with only a constant and no functional intercept.

The functional covariates have to be supplied as a n by <no. of evaluations> matrices, i.e. each row is one functional observation. For data on a regular grid, the functional response is supplied in the same format, i.e. as a matrix-valued entry in `data`, which can contain missing values. If the functional responses are sparse or irregular (i.e., not evaluated on the same evaluation points across all observations), the `ydata`-argument can be used to specify the responses: `ydata` must be a `data.frame` with 3 columns called `'.obs'`, `'.index'`, `'.value'` which specify which curve the point belongs to (`'.obs'`=i), at which t it was observed (`'.index'`=t), and the observed value (`'.value'`=$Y_i(t)$). Note that the vector of unique sorted entries in `ydata$.obs` must be equal to `rownames(data)` to ensure the correct association of entries in `ydata` to the corresponding rows of `data`. For both regular and irregular functional responses, the model is then fitted with the data in long format, i.e., for data on a grid 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 `pffr` does not use `mgcv`'s default identifiability constraints ($\sum_{i,t} \hat{f}(z_i, x_i, t) = 0$ or $\sum_{i,t} \hat{f}(x_i, t) = 0$ for tensor product terms whose marginals include the index t of the functional response. Instead, $\sum_i \hat{f}(z_i, x_i, t) = 0$ for all t is enforced, so that effects varying over t can be interpreted as local deviations from the global functional intercept. This is achieved by using `ti`-terms with a suitably modified `mc`-argument. Note that this is not possible if `algorithm='gamm4'` since only `t2`-type terms can then be used and these modified constraints are not available for `t2`. We recommend using centered scalar covariates for terms like $z\beta(t)$ (`~z`) and centered functional covariates with $\sum_i X_i(t) = 0$ for all t in `ff`-terms so that the global functional intercept can be interpreted as the global mean function.

Author(s)

Fabian Scheipl, Sonja Greven

References

Ivanescu, A., Staicu, A.-M., Scheipl, F. and Greven, S. (2013). Penalized function-on-function regression. (under revision) http://biostats.bepress.com/jhubiostat/paper254/

Scheipl, F., Staicu, A.-M. and Greven, S. (2013). Functional Additive Mixed Models. (under revision) http://arxiv.org/abs/1207.5947

See Also

`smooth.terms` for details of `mgcv` syntax and available spline bases and penalties.

Examples

```
###############################################################################
# univariate model:
# Y(t) = f(t)  + \int X1(s)\beta(s,t)ds + eps
set.seed(2121)
data1 <- pffrSim(scenario="ff", n=40)
t <- attr(data1, "yindex")
s <- attr(data1, "xindex")
m1 <- pffr(Y ~ ff(X1, xind=s), yind=t, data=data1)
summary(m1)
plot(m1, pers=TRUE, pages=1)

## Not run:
###############################################################################
# multivariate model:
# Y(t) = f0(t)  + \int X1(s)\beta1(s,t)ds + \int X2(s)\beta2(s,t)ds +
#   xlin \beta3(t) + f1(xte1, xte2) + f2(xsmoo, t) + beta4 xconst + eps
data2 <- pffrSim(scenario="all", n=200)
t <- attr(data2, "yindex")
s <- attr(data2, "xindex")
```

```
m2 <- pffr(Y ~  ff(X1, xind=s) + #linear function-on-function
                ff(X2, xind=s) + #linear function-on-function
                xlin  +  #varying coefficient term
             c(te(xte1, xte2)) + #bivariate smooth term in xte1 & xte2, const. over Y-index
                s(xsmoo) + #smooth effect of xsmoo varying over Y-index
                c(xconst), # linear effect of xconst constant over Y-index
        yind=t,
        data=data2)
summary(m2)
plot(m2, pers=TRUE)
str(coef(m2))
# convenience functions:
preddata <- pffrSim(scenario="all", n=20)
str(predict(m2, newdata=preddata))
str(predict(m2, type="terms"))
cm2 <- coef(m2)
cm2$pterms
str(cm2$smterms, 2)
str(cm2$smterms[["s(xsmoo)"]]$coef)

###############################################################################
# sparse data (80% missing on a regular grid):
set.seed(88182004)
data3 <- pffrSim(scenario=c("int", "smoo"), n=100, propmissing=0.8)
t <- attr(data3, "yindex")
m3.sparse <- pffr(Y ~ s(xsmoo), data=data3$data, ydata=data3$ydata, yind=t)
summary(m3.sparse)
plot(m3.sparse, pers=TRUE, pages=1)

## End(Not run)
```

`pffrGLS` *Penalized function-on-function regression with non-i.i.d. residuals*

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 $Y_i(t) = \mu(t) + \int X_i(s)\beta(s,t)ds + f(z_{1i},t) + f(z_{2i}) + z_{3i}\beta_3(t) + \ldots + E_i(t))$ with a functional (but not necessarily continuous) response $Y(t)$, (optional) smooth intercept $\mu(t)$, (multiple) functional covariates $X(t)$ and scalar covariates z_1, z_2, etc. The residual functions $E_i(t) \sim GP(0, K(t,t'))$ are assumed to be i.i.d. realizations of a Gaussian process. An estimate of the covariance operator $K(t,t')$ evaluated on `yind` has to be supplied in the `hatSigma`-argument.

Usage

```
pffrGLS(formula, yind, hatSigma, algorithm = NA, method = "REML",
  tensortype = c("te", "t2"), bs.yindex = list(bs = "ps", k = 5, m = c(2,
  1)), bs.int = list(bs = "ps", k = 20, m = c(2, 1)), cond.cutoff = 500,
  ...)
```

Arguments

formula	a formula with special terms as for gam, with additional special terms ff() and c(). See pffr.
yind	a vector with length equal to the number of columns of the matrix of functional responses giving the vector of evaluation points $(t_1, \ldots, t_G)$. see pffr
algorithm	the name of the function used to estimate the model. Defaults to gam if the matrix of functional responses has less than 2e5 data points and to bam if not. "gamm" (see gamm) and "gamm4" (see gamm4) are valid options as well.
hatSigma	(an estimate of) the within-observation covariance (along the responses' index), evaluated at yind. See Details.
method	See pffr
bs.yindex	See pffr
bs.int	See pffr
tensortype	See pffr
cond.cutoff	if the condition number of hatSigma is greater than this, hatSigma is made "more" positive-definite via nearPD to ensure a condition number equal to cond.cutoff. Defaults to 500.
...	additional arguments that are valid for gam or bam. See pffr.

Value

a fitted pffr-object, see pffr.

Details

Note that hatSigma has to be positive definite. If hatSigma is close to positive *semi*-definite or badly conditioned, estimated standard errors become unstable (typically much too small). pffrGLS will try to diagnose this and issue a warning. The danger is especially big if the number of functional observations is smaller than the number of gridpoints (i.e, length(yind)), since the raw covariance estimate will not have full rank.
Please see pffr for details on model specification and implementation.
THIS IS AN EXPERIMENTAL VERSION AND NOT WELL TESTED YET – USE AT YOUR OWN RISK.

Author(s)

Fabian Scheipl

See Also

pffr, fpca.sc

pffrSim *Simulate example data for pffr*

Description

Simulates example data for pffr from a variety of terms. Scenario "all" generates data from a complex multivariate model

$$Y_i(t) = \mu(t) + \int X_{1i}(s)\beta_1(s,t)ds + xlin\beta_3(t) + f(xte1, xte2) + f(xsmoo, t) + \beta_4 xconst + \epsilon_i(t)$$

. Scenarios "int", "ff", "lin", "te", "smoo", "const" generate data from simpler models containing only the respective term(s) in the model equation given above. Specifiying a vector-valued scenario will generate data from a combination of the respective terms. Sparse/irregular response trajectories can be generated by setting `propmissing` to something greater than 0 (and smaller than 1). The return object then also includes a `ydata`-item with the sparsified data.

Usage

```
pffrSim(scenario = "all", n = 100, nxgrid = 40, nygrid = 60, SNR = 10,
  propmissing = 0, limits = NULL)
```

Arguments

`scenario`	see Description
`n`	number of observations
`nxgrid`	number of evaluation points of functional covariates
`nygrid`	number of evaluation points of the functional response
`SNR`	the signal-to-noise ratio for the generated data: empirical variance of the additive predictor divided by variance of the errors.
`propmissing`	proportion of missing data in the response, default = 0. See Details.
`limits`	a function that defines an integration range, see `ff`

Details

See source code for details.

Value

a named list with the simulated data, and the true components of the predictor etc as attributes.

pfr *Penalized functional regression and Longitudinal penalized functional regression*

Description

Implements penalized functional regression (Goldsmith et al., 2011) for generalized linear functional models with scalar outcomes, as well as longitudinal penalized functional regression (Goldsmith et al., 2012) for generalized linear functional models with scalar outcomes and subject-specific random intercepts. The function refund::lpfr is no longer supported.

Usage

```
pfr(Y, subj=NULL, covariates = NULL, funcs, kz = 10, kb = 30, nbasis=10,
family = "gaussian", method="REML", smooth.option="fpca.sc", pve=0.99, ...)
```

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	can be NULL; can be a scalar, in which case this will be the dimension of principal components basis for each and every observed functional predictors; can be a vector of length equal to the number of functional predictors, in which case each element will correspond to the dimension of principal components basis for the corresponding observed functional predictors
kb	dimension of the B-spline basis for the coefficient function (note: this is a change from versions 0.1-7 and previous)
nbasis	passed to refund::fpca.sc (note: using fpca.sc is a change from versions 0.1-7 and previous)
family	generalized linear model family
method	method for estimating the smoothing parameters; defaults to REML
smooth.option	method to do FPC decomposition on the predictors. Two options available – "fpca.sc" or "fpca.face". If using "fpca.sc", a number less than 35 for nbasis should be used while if using "fpca.face",35 or more is recommended.
pve	proportion of variance explained used to choose the number of principal components to be included in the expansion.
...	additional arguments passed to gam 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. Increasing values of `nbasis` will increase computational time and the values of `nbasis`, `kz`, and `kb` in relation to each other may need to be adjusted in application-specific ways.

Value

`fit`	result of the call to `gam`
`fitted.vals`	predicted outcomes
`fitted.vals.level.0`	predicted outcomes at population level
`fitted.vals.level.1`	predicted outcomes at subject-specific level (if applicable)
`betaHat`	list of estimated coefficient functions
`beta.covariates`	parameter estimates for scalar covariates
`varBetaHat`	list containing covariance matrices for the estimated coefficient functions
`Bounds`	list of bounds of a pointwise 95% confidence interval for the estimated coefficient functions
`X`	design matrix used in the model fit
`D`	penalty matrix used in the model fit
`phi`	list of B-spline bases for the coefficient functions
`psi`	list of principal components basis for the functional predictors
`C`	stacked row-specific prinicipal component scores
`J`	transpose of psi matrix multiplied by phi
`CJ`	C matrix multiplied J
`Z1`	design matrix of random intercepts
`subj`	subject identifiers as specified by user
`fixed.mat`	the fixed effects design matrix of the pfr as a mixed model
`rand.mat`	the fixed effects design matrix of the pfr as a mixed model
`N_subj`	the number of unique subjects, if subj is specified
`p`	number of scalar covariates
`N.pred`	number of functional covariates
`kz`	as specified
`kz.adj`	For smooth.option="fpca.sc", will be same as kz (or a vector of repeated values of the specified scalar kz). For smooth.option="fpca.face", will be the corresponding number of principal components for each functional predictor as determined by fpca.face; will be less than or equal to kz on an elemental-wise level.

kb	as specified
nbasis	as specified
totD	number of penalty matrices created for mgcv::gam
funcs	as specified
covariates	as specified
smooth.option	as specified

Author(s)

Jeff Goldsmith <jeff.goldsmith@columbia.edu> and Bruce Swihart <bswihart@jhsph.edu>

References

Goldsmith, J., Bobb, J., Crainiceanu, C., Caffo, B., and Reich, D. (2011). Penalized functional regression. *Journal of Computational and Graphical Statistics*, 20(4), 830–851.

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*, 61(3), 453–469.

Swihart, Bruce J., Goldsmith, Jeff; and Crainiceanu, Ciprian M. (July 2012). Testing for functional effects. Johns Hopkins University Dept. of Biostatistics Working Paper 247, available at http://biostats.bepress.com/jhubiostat/paper247

See Also

rlrt.pfr, predict.pfr

Examples

```
## Not run:
#########################################################################
#########               DTI Data Example                #########
#########################################################################

#########################################################################
# For more about this example, see Swihart et al. 2013
#########################################################################

## load and reassign the data;
data(DTI2)
Y  <- DTI2$pasat ## PASAT outcome
id <- DTI2$id    ## subject id
W1 <- DTI2$cca   ## Corpus Callosum
W2 <- DTI2$rcst  ## Right corticospinal
V  <- DTI2$visit ## visit

## prep scalar covariate
visit.1.rest <- matrix(as.numeric(V > 1), ncol=1)
covar.in <- visit.1.rest
```

```
## note there is missingness in the functional predictors
apply(is.na(W1), 2, mean)
apply(is.na(W2), 2, mean)

## fit two univariate models
pfr.obj.t1 <- pfr(Y = Y, covariates=covar.in, funcs = list(W1),    subj = id, kz = 10, kb = 50)
pfr.obj.t2 <- pfr(Y = Y, covariates=covar.in, funcs = list(W2),    subj = id, kz = 10, kb = 50)

### one model with two functional predictors using "smooth.face"
###  for smoothing predictors
pfr.obj.t3 <- pfr(Y = Y, covariates=covar.in, funcs = list(W1, W2),
                  subj = id, kz = 10, kb = 50, nbasis=35,smooth.option="fpca.face")

## plot the coefficient function and bounds
dev.new()
par(mfrow=c(2,2))
ran <- c(-2,.5)
matplot(cbind(pfr.obj.t1$BetaHat[[1]], pfr.obj.t1$Bounds[[1]]),
  type = 'l', lty = c(1,2,2), col = c(1,2,2), ylab = "BetaHat",
  main = "CCA", xlab="Location", ylim=ran)
abline(h=0, col="blue")
matplot(cbind(pfr.obj.t2$BetaHat[[1]], pfr.obj.t2$Bounds[[1]]),
  type = 'l', lty = c(1,2,2), col = c(1,2,2), ylab = "BetaHat",
  main = "RCST", xlab="Location", ylim=ran)
abline(h=0, col="blue")
matplot(cbind(pfr.obj.t3$BetaHat[[1]], pfr.obj.t3$Bounds[[1]]),
  type = 'l', lty = c(1,2,2), col = c(1,2,2), ylab = "BetaHat",
  main = "CCA  - mult.", xlab="Location", ylim=ran)
abline(h=0, col="blue")
matplot(cbind(pfr.obj.t3$BetaHat[[2]], pfr.obj.t3$Bounds[[2]]),
  type = 'l', lty = c(1,2,2), col = c(1,2,2), ylab = "BetaHat",
  main = "RCST - mult.", xlab="Location", ylim=ran)
abline(h=0, col="blue")


     ######################################################################
     # 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$pasat, funcs = cca, kz=10, kb=50, nbasis=20)
      fit.rcst = pfr(Y=DTI$pasat, funcs = rcst, kz=10, kb=50, nbasis=20)

      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), kz=10, kb=30, nbasis=20)


      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$case, funcs = cca, family = "binomial")
fit.rcst = pfr(Y=DTI$case, funcs = rcst, 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$octane
funcs = gasoline$NIR
wavelengths = as.matrix(2*450:850)

# fit the model using pfr and the smoothing option "fpca.face"
fit = pfr(Y=Y, funcs=funcs, kz=15, kb=50,nbasis=35,smooth.option="fpca.face")

# 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)

## End(Not run)
```

`plot.fosr` *Default plotting of function-on-scalar regression objects*

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.mfrow = TRUE, ...)
```

Arguments

`x`	an object of class "`fosr`".
`split`	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 `NULL`, all functions are plotted on the same scale.
`titles`	character vector of titles for the plots produced, e.g., names of the corresponding scalar predictors.
`xlabel`	label for the x-axes of the plots.
`ylabel`	label for the y-axes of the plots.
`set.mfrow`	logical value: if `TRUE`, the function will try to set an appropriate value of the `mfrow` parameter for the plots. Otherwise you may wish to set `mfrow` outside the function call.
`...`	graphical parameters (see `par`) for the plot.

Author(s)

Philip Reiss <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 of class "fpcr".
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>

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.lpeer *Plotting of estimated regression functions obtained through* lpeer()

Description

Plots the estimate of components of estimated regression function obtained from an lpeer object along with pointwise confidence bands.

Usage

```
## S3 method for class 'lpeer'
plot(x, conf=0.95, ...)
```

Arguments

x	object of class "lpeer".
conf	pointwise confidence level.
...	additional arguments passed to plot.

Details

Pointwise confidence interval is displayed only if the user set se=T in the call to lpeer, and does not reflect any multiplicity correction.

Author(s)

Madan Gopal Kundu <mgkundu@iupui.edu>

References

Kundu, M. G., Harezlak, J., and Randolph, T. W. (2012). Longitudinal functional models with structured penalties. (Please contact J. Harezlak at <harezlak@iupui.edu>.)

Randolph, T. W., Harezlak, J, and Feng, Z. (2012). Structured penalties for functional linear models - partially empirical eigenvectors for regression. *Electronic Journal of Statistics*, 6, 323–353.

See Also

peer, lpeer, plot.peer

Examples

```
data(DTI)
cca = DTI$cca[which(DTI$case == 1),]
DTI = DTI[which(DTI$case == 1),]
fit.cca.lpeer1 = lpeer(Y=DTI$pasat, t=DTI$visit, subj=DTI$ID, funcs = cca)
plot(fit.cca.lpeer1)
```

plot.peer *Plotting of estimated regression functions obtained through* peer()

Description

Plots the estimate of components of estimated regression function obtained from a peer object along with pointwise confidence bands.

Usage

```
## S3 method for class 'peer'
plot(x, conf=0.95, ylab='Estimated regression function', main=expression(gamma), ...)
```

Arguments

x	object of class "peer".
conf	pointwise confidence level.
ylab	y-axis label.
main	title for the plot.
...	additional arguments passed to plot.

Details

Pointwise confidence interval is displayed only if the user set se=T in the call to peer, and does not reflect any multiplicity correction.

Author(s)

Madan Gopal Kundu <mgkundu@iupui.edu>

References

Kundu, M. G., Harezlak, J., and Randolph, T. W. (2012). Longitudinal functional models with structured penalties. (Please contact J. Harezlak at <harezlak@iupui.edu>.)

Randolph, T. W., Harezlak, J, and Feng, Z. (2012). Structured penalties for functional linear models - partially empirical eigenvectors for regression. *Electronic Journal of Statistics*, 6, 323–353.

See Also

peer, lpeer, plot.lpeer

Examples

```
data(DTI)
cca = DTI$cca[which(DTI$case == 1),]
DTI = DTI[which(DTI$case == 1),]
fit.cca.peer1 = peer(Y=DTI$pasat, funcs = cca)
plot(fit.cca.peer1)
```

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 `plot.gam`

Value

This function only generates plots.

Author(s)

Fabian Scheipl

plot.wcr/wnet *Default plotting for wavelet-domain scalar-on-function regression*

Description

Plots the coefficient function/image estimates produced by `wcr` and `wnet`.

Usage

```
## S3 method for class 'wcr'
plot(x, xlabel = "", ylabel = "Coefficient function", which.dim = 1, slices = NULL,
set.mfrow = TRUE, image.axes = FALSE, ...)
## S3 method for class 'wnet'
plot(x, xlabel = "", ylabel = "Coefficient function", which.dim = 1, slices = NULL,
set.mfrow = TRUE, image.axes = FALSE, ...)
```

Arguments

x	an object of class "wcr" or "wnet".
xlabel, ylabel	for 1D functional predictors, x- and y-axis labels.
which.dim, slices	for 3D image predictors, the dimension (1, 2 or 3) and slices to use for plotting; see Details.
set.mfrow	logical value: for 3D predictors, if TRUE (the default), the function will try to set an appropriate value of the mfrow plotting parameter to display the number of slices specified. Otherwise you may wish to set mfrow outside the function call.
image.axes	for 2D and 3D predictors, the axes argument passed to image.
...	additional parameters passed to plot or image.

Details

As an example of how which.dim and slices are used, suppose we set which.dim=2 and slices=7:9. Then three 2D slices of the coefficient image estimate x$fhat are displayed: x$fhat[, 7,], x$fhat[, 8,], and x$fhat[, 9,].

Author(s)

Lan Huo <lan.huo@nyumc.org>

See Also

wcr and wnet; the latter includes examples.

---

predict.fgam *Prediction from a fitted FGAM model*

Description

Takes a fitted fgam-object produced by fgam 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 'fgam'
## S3 method for class 'fgam'
predict(object, newdata, type = "response", se.fit = FALSE, terms = NULL,
PredOutOfRange = FALSE, ...)
```

Arguments

object	a fitted fgam object as produced by fgam
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. All variables provided to newdata should be in the format supplied to fgam, 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 or alternatively can be supplied as attributes of the matrix of functional predictor values. Any variables in the model not specified in newdata are set to their average values from the data supplied during fitting the model.
type	see predict.gam for details.
se.fit	see predict.gam for details.
terms	see predict.gam for details.
PredOutOfRange	if this argument is true then any functional predictor values in newdata corresponding to fgam terms that are greater[less] than the maximum[minimum] of the domain of the marginal basis for the rows of the tensor product smooth are set to the maximum[minimum] of the domain. If this argument is false, attempting to predict a value of the functional predictor outside the range of this basis produces an error.
...	additional arguments passed on to predict.gam.

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)

Mathew W. McLean <mwm79@cornell.edu> and Fabian Scheipl

See Also

fgam, predict.gam

Examples

```
require(splines)
######### Octane data example #########
data(gasoline)
N <- length(gasoline$octane)
wavelengths = 2*450:850
nir = matrix(NA, 60,401)
test <- sample(60,20)
for (i in 1:60) nir[i,] = gasoline$NIR[i, ] #changes class from AsIs to matrix
y <- gasoline$octane
```

```
fit <- fgam(y~af(nir,xind=wavelengths,splinepars=list(k=c(6,6),m=list(c(2,2),c(2,2)))),
             subset=(1:N)[-test])
preds <- predict(fit,newdata=list(nir=nir[test,]),type='response')
plot(preds,y[test])
abline(a=0,b=1)
```

`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, ...)
```

Arguments

`object`	a fitted `pffr`-object
`newdata`	A named list (or a `data.frame`) containing the values of the model covariates at which predictions are required. If no `newdata` is provided then predictions corresponding to the original data are returned. If `newdata` is provided then it must 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. See Details for more on index variables and prediction for models fit on irregular or sparse data.
`reformat`	logical, defaults to TRUE. Should predictions be returned in matrix form (default) or in the long vector shape returned by `predict.gam()`?
`type`	see `predict.gam()` for details. Note that `type == "lpmatrix"` will force `reformat` to FALSE.
`se.fit`	see `predict.gam()`
`...`	additional arguments passed on to `predict.gam()`

Details

Index variables (i.e., evaluation points) for the functional covariates are reused from the fitted model object and cannot be supplied with `newdata`. Prediction is always for the entire index range of the responses as defined in the original fit. If the original fit was performed on sparse or irregular, non-gridded response data supplied via `pffr`'s `ydata`-argument and no `newdata` was supplied, this function will simply return fitted values for the original evaluation points of the response (in list form). If the original fit was performed on sparse or irregular data and `newdata` *was* supplied, the function will return predictions on the grid of evaluation points given in `object$pffr$yind`.

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()`

`predict.pfr` *Prediction for penalized functional regression*

Description

Given a `pfr` object and new data, produces fitted values on both a population and subject-specific scale.

Usage

```
  ## S3 method for class 'pfr'
predict(object, new.data=NULL, levels=NULL, ...)
```

Arguments

`object`	an object returned by `pfr`.
`new.data`	the covariate and functional predictor values for which predicted values are desired.
`levels`	currently not supported; both population- and subject-level fitted values are returned.
`...`	additional arguments.

Details

Predicting for new data in a penalized functional regression setting takes care. Utilizing `fpca.sc` in both `pfr` and `predict.pfr` allows for the correct estimation of new scores using the original fit's basis functions. In the spirit of `predict.lme`, we can estimate for population and (if the subjects are in both the original and new data) subject-specific levels.

Value

A list with components

`fitted.vals.level.0`
: predicted outcomes at population level

`fitted.vals.level.1`
: predicted outcomes at subject-specific level (if applicable)

Author(s)

Jeff Goldsmith <jeff.goldsmith@columbia.edu> and Bruce Swihart <bswihart@jhsph.edu>

References

Goldsmith, J., Bobb, J., Crainiceanu, C., Caffo, B., and Reich, D. (2011). Penalized functional regression. *Journal of Computational and Graphical Statistics*, 20(4), 830-851.

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*, 61(3), 453–469.

Swihart, B. J., Goldsmith, J., and Crainiceanu, C. M. (2012). Testing for functional effects. Johns Hopkins University Dept. of Biostatistics Working Paper 247, available at http://biostats.bepress.com/jhubiostat/paper247

See Also

pfr, rlrt.pfr, predict.lme

Examples

```
## Not run:
#########################################################################
#########               DTI Data Example                #########
#########################################################################


## load and reassign the data;
data(DTI2)
Y  <- DTI2$pasat ## PASAT outcome
id <- DTI2$id    ## subject id
W1 <- DTI2$cca   ## Corpus Callosum
W2 <- DTI2$rcst  ## Right corticospinal
V  <- DTI2$visit ## visit

## prep scalar covariate
visit.1.rest <- matrix(as.numeric(V > 1), ncol=1)
covar.in <- visit.1.rest

## kz -- put it too high and the computation time bogs down.
pfr.obj <- pfr(Y = Y, covariates=covar.in, funcs = list(W1), subj = id, kz = 10, kb = 50)
## first column is not using random effects; second column is
```

```
head(cbind(pfr.obj$fitted.vals.level.0, pfr.obj$fitted.vals.level.1))

## do some predictions with predict.pfr()
## run predict.pfr() on all rows of data -- hope for same results as pfr fit above.
same.inputs <- predict.pfr(pfr.obj, new.data = list(subj=id, covariates=covar.in, funcs=list(W1)))
head(cbind(same.inputs[[1]], same.inputs[[2]]))
## run predict.pfr() on first 300 rows of data
subset.inputs <- predict.pfr(pfr.obj,
                             new.data = list(subj=head(id,300),
                                             covariates=head(covar.in,300),
                                             funcs=list(W1[1:300,])))
head(cbind(subset.inputs[[1]], subset.inputs[[2]]))
## compare the first 300 vs. the original full data
plot(subset.inputs[[1]], same.inputs[[1]][1:300]); abline(a=0,b=1,col="blue")
summary(subset.inputs[[1]] - same.inputs[[1]][1:300])

## try one where we have different subjects for predictions
## ids 7:12 are new
## and we just double-down the covariates and funcs from the first 6 to the second.
## check level.1 and level.0 predictions
subset.inputs <- predict.pfr(pfr.obj,
                             new.data = list(subj=c(head(id,6), 7:12),
                                    covariates=rbind(head(covar.in,6),head(covar.in,6)),
                                             funcs=list(rbind(W1[1:6,],W1[1:6,]))))
## first 6 have both levels; the last 6 do not have subject-specific
head(cbind(subset.inputs[[1]], subset.inputs[[2]]),12)

## see that whether predicted on subset or full data, the predicted
## values are the same for individuals regardless of who is in
## prediction set
## compare the first 6 vs. the original full data: 6 on the 45 degree line and 6 off.
plot(subset.inputs[[1]], same.inputs[[1]][1:12]); abline(a=0,b=1,col="blue")
summary(subset.inputs[[1]] - same.inputs[[1]][1:12])

## End(Not run)
```

`predict.wnet` *Prediction method for generalized elastic net in the wavelet domain*

Description

Produces predicted values for a `wnet` object, given new functional predictors and scalar covariates.

Usage

```
  ## S3 method for class 'wnet'
predict(object, newx, newcovt = NULL, ...)
```

Arguments

object	a fitted wnet object.
newx	matrix of new values for functional predictors.
newcovt	matrix of new covariate values.
...	not currently used.

Value

A vector of predicted values.

Author(s)

Lan Huo

See Also

wnet

---

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

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

Value

A summary.pffr object

Author(s)

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

pwcv *Pointwise cross-validation for function-on-scalar regression*

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

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

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/

See Also

fosr, lofocv

Examples

```
require(fda)
# 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
Xreg = cbind(1, model.matrix(~factor(CanadianWeather$region)-1))
Xlat = model.matrix(~CanadianWeather$coord[,1])

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

latmod = fosr(fdobj=tempfd, X=Xlat, method="OLS")
cv.lat = pwcv(tempfd, Xlat, 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 stations")
box()

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

---

refund-internal *Internal functions for the refund package*

Description

These functions are ordinarily not to be called by the user, but if you contact the package authors with any questions about them, we'll do our best to clarify matters.

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 yindex matrix form (default for regular grid data) or in the shape of the originally supplied ydata argument or (default for sparse/irregular data), or, if FALSE, simply 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

rlrt.pfr *Likelihood Ratio Test and Restricted Likelihood Ratio Test for inference of functional predictors*

Description

Given a pfr object of family="gaussian", tests whether the function is identically equal to its mean (constancy), or whether the functional predictor significantly improves the model (inclusion). Based on zero-variance-component work of Crainiceanu et al. (2004), Scheipl et al. (2008), and Swihart et al. (2012).

Usage

```
rlrt.pfr(pfr.obj=pfr.obj, test=NULL, ...)
```

Arguments

pfr.obj	an object returned by pfr
test	"constancy" will test functional form of the coefficient function of the last function listed in funcs in pfr.obj against the null of a constant line: the average of the functional predictor. "inclusion" will test functional form of the coefficient function of the last function listed in funcs in pfr.obj against the null of 0: that is, whether the functional predictor should be included in the model.
...	additional arguments

Details

A Penalized Functional Regression of family="gaussian" can be represented as a linear mixed model dependent on variance components. Testing whether certain variance components and (potentially) fixed effect coefficients are 0 correspond to tests of constancy and inclusion of functional predictors.

For rlrt.pfr, Restricted Likelihood Ratio Test is preferred for the constancy test as under the special B-splines implementation of pfr for the coefficient function basis the test involves only the variance component. Therefore, the constancy test is best for pfr objects with method="REML"; if the method was something else, a warning is printed and the model refit with "REML" and a test is then conducted.

For rlrt.pfr, the Likelihood Ratio Test is preferred for the inclusion test as under the special B-splines implementation of pfr for the coefficient function basis the test involves both the variance component and a fixed effect coefficient in the linear mixed model representation. Therefore, the inclusion test is best for pfr objects with method="ML"; if the method was something else, a warning is printed and the model refit with "ML" and a test is then conducted.

Value

p.val	the p-value for the full model (alternative) against the null specified by the test
test.stat	the test statistic, see Scheipl et al. 2008 and Swihart et al 2012
ma	the alternative model as fit with mgcv::gam
m0	the null model as fit with mgcv::gam
m	the model containing only the parameters being tested as fit with mgcv::gam

Author(s)

Jeff Goldsmith <jeff.goldsmith@columbia.edu> and Bruce Swihart <bswihart@jhsph.edu>

References

Goldsmith, J., Bobb, J., Crainiceanu, C., Caffo, B., and Reich, D. (2011). Penalized functional regression. *Journal of Computational and Graphical Statistics*, 20(4), 830–851.

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*, 61(3), 453–469.

Crainiceanu, C. and Ruppert, D. (2004) Likelihood ratio tests in linear mixed models with one variance component. *Journal of the Royal Statistical Society: Series B*, 66, 165–185.

Scheipl, F. (2007) Testing for nonparametric terms and random effects in structured additive regression. Diploma thesis.\ http://www.statistik.lmu.de/~scheipl/downloads/DIPLOM.zip.

Scheipl, F., Greven, S. and Kuechenhoff, H (2008) Size and power of tests for a zero random effect variance or polynomial regression in additive and linear mixed models. *Computational Statistics & Data Analysis*, 52(7), 3283–3299.

Swihart, Bruce J., Goldsmith, Jeff; and Crainiceanu, Ciprian M. (2012). Testing for functional effects. Johns Hopkins University Dept. of Biostatistics Working Paper 247. Available at http://biostats.bepress.com/jhubiostat/paper247

See Also

pfr, predict.pfr, package RLRsim

Examples

```
## Not run:
######################################################################
#########               DTI Data Example                  #########
######################################################################

######################################################################
# For more about this example, see Swihart et al. 2012
# Testing for Functional Effects
######################################################################

## load and reassign the data;
data(DTI2)
O  <- DTI2$pasat ## PASAT outcome
id <- DTI2$id    ## subject id
W1 <- DTI2$cca   ## Corpus Callosum
W2 <- DTI2$rcst  ## Right corticospinal
V  <- DTI2$visit ## visit

## prep scalar covariate
visit.1.rest <- matrix(as.numeric(V > 1), ncol=1)
covar.in <- visit.1.rest


## note there is missingness in the functional predictors
apply(is.na(W1), 2, mean)
apply(is.na(W2), 2, mean)

## fit two univariate models, then one model with both functional predictors
pfr.obj.t1 <- pfr(Y = O, covariates=covar.in, funcs = list(W1),     subj = id, kz = 10, kb = 50)
pfr.obj.t2 <- pfr(Y = O, covariates=covar.in, funcs = list(W2),     subj = id, kz = 10, kb = 50)
pfr.obj.t3 <- pfr(Y = O, covariates=covar.in, funcs = list(W1, W2), subj = id, kz = 10, kb = 50)

## plot the coefficient function and bounds
dev.new()
par(mfrow=c(2,2))
ran <- c(-2,.5)
```

```
matplot(cbind(pfr.obj.t1$BetaHat[[1]], pfr.obj.t1$Bounds[[1]]),
  type = 'l', lty = c(1,2,2), col = c(1,2,2), ylab = "BetaHat",
  main = "CCA", xlab="Location", ylim=ran)
abline(h=0, col="blue")
matplot(cbind(pfr.obj.t2$BetaHat[[1]], pfr.obj.t2$Bounds[[1]]),
  type = 'l', lty = c(1,2,2), col = c(1,2,2), ylab = "BetaHat",
  main = "RCST", xlab="Location", ylim=ran)
abline(h=0, col="blue")
matplot(cbind(pfr.obj.t3$BetaHat[[1]], pfr.obj.t3$Bounds[[1]]),
  type = 'l', lty = c(1,2,2), col = c(1,2,2), ylab = "BetaHat",
  main = "CCA  - mult.", xlab="Location", ylim=ran)
abline(h=0, col="blue")
matplot(cbind(pfr.obj.t3$BetaHat[[2]], pfr.obj.t3$Bounds[[2]]),
  type = 'l', lty = c(1,2,2), col = c(1,2,2), ylab = "BetaHat",
  main = "RCST - mult.", xlab="Location", ylim=ran)
abline(h=0, col="blue")

## do some testing
t1 <- rlrt.pfr(pfr.obj.t1, "constancy")
t2 <- rlrt.pfr(pfr.obj.t2, "constancy")
t3 <- rlrt.pfr(pfr.obj.t3, "inclusion")

t1$test.stat
t1$p.val

t2$test.stat
t2$p.val

t3$test.stat
t3$p.val


## do some testing with rlrt.pfr(); same as above but subj = NULL
pfr.obj.t1 <- pfr(Y = O, covariates=covar.in, funcs = list(W1),    subj = NULL, kz = 10, kb = 50)
pfr.obj.t2 <- pfr(Y = O, covariates=covar.in, funcs = list(W2),    subj = NULL, kz = 10, kb = 50)
pfr.obj.t3 <- pfr(Y = O, covariates=covar.in, funcs = list(W1, W2), subj = NULL, kz = 10, kb = 50)

t1 <- rlrt.pfr(pfr.obj.t1, "constancy")
t2 <- rlrt.pfr(pfr.obj.t2, "constancy")
t3 <- rlrt.pfr(pfr.obj.t3, "inclusion")

t1$test.stat
t1$p.val

t2$test.stat
t2$p.val

t3$test.stat
t3$p.val

## End(Not run)
```

sff *Construct a smooth function-on-function regression term*

Description

Defines a term $\int_{s_{lo,i}}^{s_{hi,i}} f(X_i(s), s, t)ds$ for inclusion in an `mgcv::gam`-formula (or `bam` or `gamm` or `gamm4:::gamm`) as constructed by `pffr`. Defaults to a cubic tensor product B-spline with marginal second differences penalties for $f(X_i(s), s, t)$ 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.
`sff()` IS AN EXPERIMENTAL FEATURE AND NOT WELL TESTED YET – USE AT YOUR OWN RISK.

Usage

```
sff(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, 2, 2)))
```

Arguments

`X`	an n by `ncol(xind)` matrix of function evaluations $X_i(s_{i1}), \ldots, X_i(s_{iS})$; $i = 1, \ldots, n$.
`yind`	*DEPRECATED* matrix (or vector) of indices of evaluations of $Y_i(t)$; i.e. matrix with rows $(t_{i1}, \ldots, t_{iT})$; no longer used.
`xind`	matrix (or vector) of indices of evaluations of $X_i(s)$; i.e. matrix with rows $(s_{i1}, \ldots, s_{iS})$
`basistype`	defaults to `"te"`, i.e. a tensor product spline to represent $f(X_i(s), t)$. Alternatively, use `"s"` for bivariate basis functions (see `s`) or `"t2"` for an alternative parameterization of tensor product splines (see `t2`).
`integration`	method used for numerical integration. Defaults to `"simpson"`'s rule. Alternatively and for non-equidistant grids, `"trapezoidal"`.
`L`	optional: an n by `ncol(xind)` giving the weights for the numerical integration over s.
`limits`	defaults to NULL for integration across the entire range of $X(s)$, otherwise specifies the integration limits $s_{hi,i}, s_{lo,i}$: either one of `"s<t"` or `"s<=t"` for $(s_{hi,i}, s_{lo,i}) = (0, t)$ or a function that takes `s` as the first and `t` as the second argument and returns TRUE for combinations of values `(s,t)` if `s` falls into the integration range for the given `t`. This is an experimental feature and not well tested yet; use at your own risk.
`splinepars`	optional arguments supplied to the `basistype`-term. Defaults to a cubic tensor product B-spline with marginal second differences, i.e. `list(bs="ps", m=c(2,2,2))`. See `te` or `s` for details

Value

a list containing

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

Author(s)

Fabian Scheipl, based on Sonja Greven's trick for fitting functional responses.

`smooth.construct.pcre.smooth.spec`

mgcv-style constructor for PC-basis functional random effects

---

Description

Sets up design matrix for functional random effects based on the PC loadings of the covariance operator of the random effect process. See `smooth.construct.re.smooth.spec` for more details on `mgcv`-style smoother specification and `pcre` for the corresponding `pffr()`-formula wrapper.

Usage

```
## S3 method for class 'pcre.smooth.spec'
smooth.construct(object, data, knots)
```

Arguments

`object`	a smooth specification object, see `smooth.construct`
`data`	see `smooth.construct`
`knots`	see `smooth.construct`

Value

An object of class `"random.effect"`. See `smooth.construct` for the elements that this object will contain.

Author(s)

Fabian Scheipl; adapted from 're' constructor by S.N. Wood.

`smooth.construct.pss.smooth.spec` *P-spline constructor with modified 'shrinkage' penalty*

Description

Construct a B-spline basis with a modified difference penalty of full rank (i.e., that also penalizes low-order polynomials).

Usage

```
## S3 method for class 'pss.smooth.spec'
smooth.construct(object, data, knots)
```

Arguments

`object`	see `smooth.construct`. The shrinkage factor can be speficied via `object$xt$shrink`
`data`	see `smooth.construct`.
`knots`	see `smooth.construct`.

Details

This penalty-basis combination is useful to avoid non-identifiability issues for `ff` terms. See 'ts' or 'cs' in `smooth.terms` for similar "shrinkage penalties" for thin plate and cubic regression splines. The basic idea is to replace the k-th zero eigenvalue of the original penalty by $s^k \nu_m$, where s is the shrinkage factor (defaults to 0.1) and ν_m is the smallest non-zero eigenvalue. See reference for the original idea, implementation follows that in the 'ts' and 'cs' constructors (see `smooth.terms`).

Author(s)

Fabian Scheipl; adapted from 'ts' and 'cs' constructors by S.N. Wood.

References

Marra, G., & Wood, S. N. (2011). Practical variable selection for generalized additive models. *Computational Statistics & Data Analysis*, 55(7), 2372-2387.

summary.pffr *Summary for a pffr fit*

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

vis.fgam *Visualization of FGAM objects*

Description

Produces perspective or contour plot views of an estimated surface corresponding to af terms fit using fgam or plots "slices"" of the estimated surface or estimated second derivative surface with one of its arguments fixed and corresponding twice-standard error "Bayesian" confidence bands constructed using the method in Marra and Wood (2012). See the details.

Usage

```
vis.fgam(object, af.term, xval = NULL, tval = NULL, deriv2 = FALSE, theta = 50,
plot.type = "persp", ticktype = "detailed", ...)
```

Arguments

object	an fgam object, produced by gam().
af.term	the name of the functional predictor to be plotted. Only important if multiple af terms are fit. Defaults to the first af term in object$call.
xval	a number in the range of functional predictor to be plotted. The surface will be plotted with the first argument of the estimated surface fixed at this value.
tval	a number in the domain of the functional predictor to be plotted. The surface will be plotted with the second argument of the estimated surface fixed at this value. Ignored if xval is specified.
deriv2	If TRUE, plot the estimated second derivative surface along with Bayesian confidence bands. Only implemented for the "slices" plot from either xval or tval being specified.
theta	
plot.type	one of "contour" or "persp". Ignored if either xval or tval is specified.
ticktype	how to draw the tick marks if plot.type="persp". Defaults to "detailed".
...	other options to be passed to persp, levelplot, or plot.

Details

The confidence bands used when plotting slices of the estimated surface or second derivative surface are the ones proposed in Marra and Wood (2012). These are a generalization of the "Bayesian" intevals of Wahba (1983) with an adjustment for the uncertainty about the model intercept. The estimated covariance matrix of the model parameters is obtained from assuming a particular Bayesian model on the parameters.

Value

Simply produces a plot.

Author(s)

Mathew W. McLean <mwm79@cornell.edu>

References

McLean, M. W., Hooker, G., Staicu, A.-M., Scheipl, F., and Ruppert, D. (2013). Functional generalized additive models. *Journal of Computational and Graphical Statistics*, to appear. Available at http://courses2.cit.cornell.edu/mwmclean

Marra, G., and Wood, S. N. (2012) Coverage properties of confidence intervals for generalized additive model components. *Scandinavian Journal of Statistics*, 39(1), 53–74.

Wabha, G. (1983) "Confidence intervals" for the cross-validated smoothing spline. *Journal of the Royal Statistical Society, Series B*, 45(1), 133–150.

See Also

vis.gam, plot.gam, fgam, persp, levelplot

Examples

```
require(splines)
################## DTI Example ####################
data(DTI)

## only consider first visit and cases (since no PASAT scores for controls)
y <- DTI$pasat[DTI$visit==1 & DTI$case==1]
X <- DTI$cca[DTI$visit==1 & DTI$case==1,]

## remove samples containing missing data
ind <- rowSums(is.na(X))>0

y <- y[!ind]
X <- X[!ind,]

## fit the fgam using FA measurements along corpus
## callosum as functional predictor with PASAT as response
## using 8 cubic B-splines for each marginal bases with
## third order marginal difference penalties
## specifying gamma>1 enforces more smoothing when using GCV
## to choose smoothing parameters
fit <- fgam(y~af(X,splinepars=list(k=c(8,8),m=list(c(2,3),c(2,3)))),gamma=1.2)

## contour plot of the fitted surface
vis.fgam(fit,plot.type='contour')

## similar to Figure 5 from McLean et al.
## Bands seem too conservative in some cases
xval=runif(1,min(fit$fgam$ft[[1]]$Xrange),max(fit$fgam$ft[[1]]$Xrange))
tval=runif(1,min(fit$fgam$ft[[1]]$xind),max(fit$fgam$ft[[1]]$xind))
par(mfrow=c(4,1))
vis.fgam(fit,af.term='X',deriv2=FALSE,xval=xval)
vis.fgam(fit,af.term='X',deriv2=FALSE,tval=tval)
vis.fgam(fit,af.term='X',deriv2=TRUE,xval=xval)
vis.fgam(fit,af.term='X',deriv2=TRUE,tval=tval)
```

`wcr` *Principal component regression and partial least squares in the wavelet domain*

Description

Performs generalized linear scalar-on-function or scalar-on-image regression in the wavelet domain, by sparse principal component regression (PCR) and sparse partial least squares (PLS).

Usage

```
wcr(y, xfuncs, min.scale, nfeatures, ncomp, method = c("pcr", "pls"),
```

```
mean.signal.term = FALSE, covt = NULL, filter.number = 10,
wavelet.family = "DaubLeAsymm", family = "gaussian", cv1 = FALSE, nfold = 5,
nsplit = 1, store.cv = FALSE, store.glm = FALSE, seed = NULL)
```

Arguments

y	scalar outcome vector.
xfuncs	functional predictors. For 1D predictors, an $n \times d$ matrix of signals, where n is the length of y and d is the number of sites at which each signal is defined. For 2D predictors, an $n \times d \times d$ array comprising n images of dimension $d \times d$. For 3D predictors, an $n \times d \times d \times d$ array comprising n images of dimension $d \times d \times d$. Note that d must be a power of 2.
min.scale	either a scalar, or a vector of values to be compared. Used to control the coarseness level of wavelet decomposition. Possible values are $0, 1, \ldots, log_2(d) - 1$.
nfeatures	number(s) of features, i.e. wavelet coefficients, to retain for prediction of y: either a scalar, or a vector of values to be compared.
ncomp	number(s) of principal components (if method="pcr") or PLS components (if method="pls"): either a scalar, or a vector of values to be compared.
method	either "pcr" (principal component regression) (the default) or "pls" (partial least squares).
mean.signal.term	logical: should the mean of each subject's signal be included as a covariate? By default, FALSE.
covt	covariates: an n-row matrix, or a vector of length n; defaults to NULL.
filter.number	argument passed to function wd, imwd, or wd3D in the **wavethresh** package. Used to select the smoothness of wavelet in the decomposition; defaults to 10.
wavelet.family	family of wavelets: passed to functions wd, imwd, orwd3D; defaults to "DaubLeAsymm".
family	generalized linear model family. Current version supports "gaussian" (the default) and "binomial".
cv1	logical: should cross-validation be performed (to estimate prediction error) even if a single value is provided for each of min.scale, nfeatures and ncomp? By default, FALSE. Note that whenever multiple candidate values are provided for one or more of these tuning parameters, CV is performed to select the best model.
nfold	the number of validation sets ("folds") into which the data are divided.
nsplit	number of splits into nfold validation sets; CV is computed by averaging over these splits.
store.cv	logical: should the output include a CV result table?
store.glm	logical: should the output include the fitted glm? Defaults to FALSE.
seed	the seed for random data division. If seed = NULL, a random seed is used.

Details

Briefly, the algorithm works by (1) applying the discrete wavelet transform (DWT) to the functional/image predictors; (2) retaining only the `nfeatures` wavelet coefficients having the highest variance (for PCR; cf. Johnstone and Lu, 2009) or highest covariance with `y` (for PLS); (3) regressing `y` on the leading `ncomp` PCs or PLS components, along with any scalar covariates; and (4) applying the inverse DWT to the result to obtain the coefficient function estimate `fhat`.

This function supports only the standard DWT (see argument `type` in `wd`) with periodic boundary handling (see argument `bc` in `wd`).

For 2D predictors, setting `min.scale=1` will lead to an error, due to a technical detail regarding `imwd`. Please contact the author if a workaround is needed.

See the Details for `fpcr` for a note regarding decorrelation.

Value

An object of class `"wcr"`. This is a list that, if `store.glm = TRUE`, includes all components of the fitted `glm` object. The following components are included even if `store.glm = FALSE`:

`fitted.values`	the fitted values.
`param.coef`	coefficients for covariates with decorrelation. The model is fitted after decorrelating the functional predictors from any scalar covariates; but for CV, one needs the "undecorrelated" coefficients from the training-set models.
`undecor.coef`	coefficients for covariates *without* decorrelation. See `param.coef`.
`fhat`	coefficient function estimate.
`Rsq`	coefficient of determination.
`tuning.params`	if CV is performed, 2×4 table the indices and values of `min.scale`, `nfeatures` and `ncomp` chosen by CV.
`cv.table`	a table giving the CV criterion for each combination of `min.scale`, `nfeatures` and `ncomp`, if `store.cv = TRUE`; otherwise, the CV criterion only for the optimized combination of these parameters. Set to `NULL` if CV is not performed.
`se.cv`	if `store.cv = TRUE`, the standard error of the CV estimate for each combination of `min.scale`, `nfeatures` and `ncomp`.
`family`	generalized linear model family.

Author(s)

Lan Huo <lan.huo@nyumc.org>

References

Johnstone, I. M., and Lu, Y. (2009). On consistency and sparsity for principal components analysis in high dimensions. *Journal of the American Statistical Association*, 104, 682–693.

See Also

`wnet`, `fpcr`

Examples

```
# See example for wnet
```

`wcr/wnet.perm` *Permutation test for wavelet-domain scalar-on-function regression*

Description

This function assesses statistical significance of a `wcr` or `wnet` fit by referring the cross-validation criterion to a permutation distribution.

Usage

```
wcr.perm(y, xfuncs, min.scale = 0, nfeatures, ncomp, method = c("pcr", "pls"),
         covt = NULL, nrep = 1, nsplit = 1, nfold = 5, nperm = 20,
         perm.method = NULL, family = "gaussian",
         seed.real = NULL, seed.perm = NULL, ...)
wnet.perm(y, xfuncs, min.scale = 0, nfeatures = NULL, alpha = 1, lambda,
         covt = NULL, nrep = 1, nsplit = 1, nfold = 5, nperm = 20,
         perm.method = NULL, family = "gaussian",
         seed.real = NULL, seed.perm = NULL, ...)
```

Arguments

`y, xfuncs, min.scale, nfeatures, method, covt, family, nsplit, nfold`	arguments passed to `wcr` or `wnet`.
`ncomp`	number of components; passed to `wcr`.
`alpha, lambda`	tuning parameters, passed to `wnet`.
`nrep`	number of replicates to average over, when computing the real-data CV criterion.
`nperm`	number of permutations. The default is suitable for toy applications only.
`perm.method`	either NULL or one of • `"responses"`: permute the response vector `y`. • `"y.residuals"`: permute the residuals upon regressing `y ~ covt`. • `"x.residuals"`: permute the residuals upon regressing `xfuncs ~ covt`. See Details.
`seed.real`	the seed for random data division for real data. If `seed.real = NULL`, a random seed is used.
`seed.perm`	the seed for random data division for permuted data. If `seed.perm = NULL`, a random seed is used.
`...`	other arguments passed to `wcr` or `wnet`.

Details

Permutation tests of this type are discussed, in a classification setting, by Ojala and Garriga (2010). Permuting the responses (`perm.method="responses"`) is appropriate when regressing on functions/images only, with no scalar covariates. For linear regression with covariates, it is preferable to first regress on the covariates, and then permute the residuals (`perm.method="y.residuals"`). For logistic regression this is not feasible; but, following Potter (2005), one can instead permute the residuals from a regression of the functions/images on the covariates (`perm.method="x.residuals"`). When `perm.method=NULL` (the default), `"responses"` is used if `covt` is `NULL`, and `"x.residuals"` otherwise.

Value

`cv`	CV value for the real data (averaged over `nrep` replications).
`cv.perm`	CV values for the permuted data.
`pvalue`	p-value for the permutation test.

Author(s)

Lan Huo <lan.huo@nyumc.org>

References

Ojala, M., and Garriga, G. C. (2010). Permutation tests for studying classifier performance. *Journal of Machine Learning Research*, 11, 1833–1863.

Potter, D. M. (2005). A permutation test for inference in logistic regression with small- and moderate-sized data sets. *Statistics in Medicine*, 24, 693–708.

See Also

wcr, wnet

Examples

```
## Not run:
n = 200; d = 64

# Create true coefficient function
ftrue = matrix(0,d,d)
ftrue[40:46,34:38] = 1

# Generate random functional predictors, and scalar responses
ii = array(rnorm(n*d^2), dim=c(n,d,d))
iimat = ii; dim(iimat) = c(n,d^2)
yy = iimat %*% as.vector(ftrue) + rnorm(n, sd=.3)

obj.wnet.perm <- wnet.perm(yy, xfuncs = ii, min.scale = 4, nfeatures = 200, alpha = 1,
                           nperm = 10)

obj.wcr.perm <- wcr.perm(yy, xfuncs = ii, min.scale = 4, nfeatures = 20, ncomp = 6,
```

```
                        cv1 = TRUE, method = "pls", nperm = 10)

## End(Not run)
```

---

wnet *Generalized elastic net in the wavelet domain*

Description

Performs generalized linear scalar-on-function or scalar-on-image regression in the wavelet domain, by (naive) elastic net.

Usage

```
wnet(y, xfuncs, covt = NULL, min.scale = 0, nfeatures = NULL, alpha = 1,
     lambda = NULL, standardize = FALSE, pen.covt = FALSE,
     filter.number = 10, wavelet.family = "DaubLeAsymm", family = "gaussian",
     nfold = 5, nsplit = 1, store.cv = FALSE, store.glmnet = FALSE,
     seed = NULL, ...)
```

Arguments

y	scalar outcome vector.
xfuncs	functional predictors. For 1D predictors, an $n \times d$ matrix of signals, where n is the length of y and d is the number of sites at which each signal is observed. For 2D predictors, an $n \times d \times d$ array comprising n images of dimension $d \times d$. For 3D predictors, an $n \times d \times d \times d$ array comprising n images of dimension $d \times d \times d$. Note that d must be a power of 2.
covt	covariates, if any: an n-row matrix, or a vector of length n.
min.scale	either a scalar, or a vector of candidate values to be compared. Used to control the coarseness level of the wavelet decomposition. Possible values are $0, 1, \ldots, log_2(d) - 1$.
nfeatures	number(s) of features, i.e. wavelet coefficients, to retain for prediction of y: either a scalar, or a vector of values to be compared.
alpha	elastic net mixing parameter, used by glmnet: either a scalar, or a vector of values to be compared. alpha=1 gives the lasso penalty, while alpha=0 yields the ridge penalty.
lambda	a vector of candidate regularization parameter values. If not supplied, glmnet automatically generates a sequence of candidate values.
standardize	logical, passed to glmnet: should the predictor variables be standardized? Defaults to FALSE, but either way, the coefficients are returned on the original scale.
pen.covt	logical: should the scalar covariates be penalized? If FALSE (the default), penalization is suppressed by setting the appropriate components of penalty.factor to 0 in the call to glmnet.

filter.number	passed to wd, imwd, or wd3D, in the **wavethresh** package, to select the smoothness of the wavelets; defaults to 10.
wavelet.family	family of wavelets: passed to wd, imwd, or wd3D. Defaults to "DaubLeAsymm".
family	generalized linear model family. Current version supports "gaussian" (the default) and "binomial".
nfold	the number of validation sets ("folds") into which the data are divided.
nsplit	number of splits into nfold validation sets; CV is computed by averaging over these splits.
store.cv	logical: should the output include a CV result table?
store.glmnet	logical: should the output include the fitted glmnet? Defaults to FALSE.
seed	the seed for random data division. If seed = NULL, a random seed is used.
...	other arguments passed to glmnet.

Details

This function supports only the standard discrete wavelet transform (see argument type in wd) with periodic boundary handling (see argument bc in wd).

For 2D predictors, setting min.scale=1 will lead to an error, due to a technical detail regarding imwd. Please contact the authors if a workaround is needed.

Value

An object of class "wnet", which is a list with the following components:

glmnet	if store.glmnet = TRUE, the object returned by glmnet.
fitted.value	the fitted values.
coef.param	parametric coefficient estimates, for the scalar covariates.
fhat	coefficient function estimate.
Rsq	coefficient of determination.
lambda.table	array giving the candidate lambda values, chosen automatically by glmnet, for each combination of the other tuning parameters.
tuning.params	a 2×4 table giving the indices and values of min.scale, nfeatures, alpha and lambda that minimize the CV. For example, if alpha=c(0,0.5,1) is specified and the CV-minimizing tuning parameter combination takes alpha to be the 2nd of these values, then the third column of the table is c(2, 0.5).
cv.table	if store.cv = TRUE, a table giving the CV criterion for each combination of min.scale, nfeatures, alpha and lambda. Otherwise, just the minimized CV criterion.
se.cv	if store.cv = TRUE, the standard error of the CV estimate for each combination of min.scale, nfeatures, alpha and lambda.
family	generalized linear model family.

Author(s)

Lan Huo <lan.huo@nyumc.org> and Yihong Zhao

References

Zhao, Y., Ogden, R. T., and Reiss, P. T. (2013). Wavelet-based LASSO in functional linear regression. *Journal of Computational and Graphical Statistics*, to appear.

See Also

wcr, wnet.perm

Examples

```
## Not run:
### 1D functional predictor example ###

data(gasoline)

# input a single value of each tuning parameters
gas.wnet1 <- wnet(gasoline$octane, xfuncs = gasoline$NIR[,1:256],
             nfeatures= 20, min.scale = 0, alpha = 1)
gas.wpcr1 <- wcr(gasoline$octane, xfuncs = gasoline$NIR[,1:256], min.scale = 0,
                 nfeatures = 20, ncomp = 15)
gas.wpls1 <- wcr(gasoline$octane, xfuncs = gasoline$NIR[,1:256], min.scale = 0,
                 nfeatures = 20, ncomp = 15, method = "pls")
plot(gas.wnet1)
plot(gas.wpcr1)
plot(gas.wpls1)

# input vectors of candidate tuning parameter values
gas.wnet2 <- wnet(gasoline$octane, xfuncs = gasoline$NIR[,1:256],
                  nfeatures= 20, min.scale = 0:3, alpha = c(0.9, 1))
gas.wpcr2 <- wcr(gasoline$octane, xfuncs = gasoline$NIR[,1:256], min.scale = 0:3,
                 nfeatures = c(16, 18, 20), ncomp = 10:15)
gas.wpls2 <- wcr(gasoline$octane, xfuncs = gasoline$NIR[,1:256], min.scale = 0:3,
                 nfeatures = c(16, 18, 20), ncomp = 10:15, method = "pls")
plot(gas.wnet2)
plot(gas.wpcr2)
plot(gas.wpls2)

### 2D functional predictor example ###

n = 200; d = 64

# Create true coefficient function
ftrue = matrix(0,d,d)
ftrue[40:46,34:38] = 1

# Generate random functional predictors, and scalar responses
ii = array(rnorm(n*d^2), dim=c(n,d,d))
iimat = ii; dim(iimat) = c(n,d^2)
yy = iimat %*% as.vector(ftrue) + rnorm(n, sd=.3)

mm.wnet <- wnet(yy, xfuncs = ii, min.scale = 4, alpha = 1)
```

```
mm.wpls <- wcr(yy, xfuncs = ii, min.scale = 4, nfeatures = 20, ncomp = 6,
               method = "pls")

plot(mm.wnet)
plot(mm.wpls)

### 3D functional predictor example ###

n = 200; d = 16

# Create true coefficient function
ftrue = array(0,dim = rep(d, 3))
ftrue[10:16,12:15, 4:8] = 1

# Generate random functional predictors, and scalar responses
ii = array(rnorm(n*d^3), dim=c(n,rep(d,3)))
iimat = ii; dim(iimat) = c(n,d^3)
yy = iimat %*% as.vector(ftrue) + rnorm(n, sd=.3)

mmm.wnet <- wnet(yy, xfuncs = ii, min.scale = 2, alpha = 1)

mmm.wpls <- wcr(yy, xfuncs = ii, min.scale = 2, nfeatures = 20, ncomp = 6,
               method = "pls")
plot(mmm.wnet)
plot(mmm.wpls)

## End(Not run)
```

Index