Package ‘refund’

September 4, 2020

Type Package

Title Regression with Functional Data

Version 0.1-22

Date 2020-08-31

Depends R (>= 3.00.0)

Imports fda, Matrix, lattice, boot, mgcv (>= 1.8-23), MASS, magic, nlme, gamm4, lme4, RLRsim, splines, grpreg, ggplot2, stats, pbs, methods

Suggests RColorBrewer, reshape2, testthat

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

LazyData true

Repository CRAN

Collate 'Omegas.R' 'af.R' 'af_old.R' 'ame.R' 'ccf.fpc.R'
'create.prep.func.R' 'coefficients.pfr.R' 'dt_basis.R'
'irreg2mat.R' 'fbps.R' 'fgam.R' 'fosl.R' 'fosl_perm.R'
'fosl_perm.fit.R' 'fosl_perm.test.R' 'fosl.vs.R' 'fosl2s.R'
'fpc.R' 'fPCA2s.R' 'fPCA.sc.R' 'fPCA_face.R' 'fPCA_svd.R'
'fpcr.R' 'fpcr_setup.R' 'lf.R' 'lf_old.R' 'lf_vd.R' 'lofocv.R'
'lpeer.R' 'lpfr.R' 'quadWeights.R' 'lw.test.R' 'osplinepen2d.R'
'parse.predict.pfr.R' 'peer.R' 'peer_old.R' 'pfr-fz.R'
'pfr-fpc.R' 'pfr-methods.R' 'pfr-perc.R' 'pfr-robust.R'
'pfr-sfl.R' 'pfr-utilities.R' 'pfr.R' 'pfr_old.R'
'pi_basis.R' 'plot.fosr.R' 'plot.fosr_perm.R' 'plot.fosr_vs.R'
'plot.fpcr.R' 'plot.lpeer.R' 'plot.peer.R' 'plot.pfr.R'
'poridge.R' 'postprocess.pfr.R' 'predict.fgam.R'
'predict.fosl.R' 'predict.pfr.R' 'predict.pfr_old.R'
'preprocess.pfr.R' 'pspline.setting.R' 'pwcv.R' 'summary.pfr.R'
R topics documented:

're.R' 'rlrt.pfr.R' 'vis.fgam.R' 'predict.fosr.vs.R'
'CD4-data.R' 'DTI-data.R' 'DTI2-data.R' 'PEER.Sim-data.R'
'gasoline-data.R' 'vis.pfr.R' 'GLS_CS.R' 'Gibbs_CS_FPCA.R'
'Gibbs_CS_Wish.R' 'Gibbs_Mult_FPCA.R' 'Gibbs_Mult_Wish.R'
'OLS_CS.R' 'VB_CS_FPCA.R' 'VB_CS_Wish.R' 'VB_Mult_FPCA.R'
'VB_Mult_Wish.R' 'XtSiginvX.R' 'bayes_fosr.R' 'f_sum.R'
'f_sum2.R' 'f_sum4.R' 'f_trace.R' 'mfpca.sc.R' 'fpca.lfda.R'
'predict.fbps.R' 'select_knots.R'

RoxygenNote 7.1.0

NeedsCompilation no

Author Jeff Goldsmith [aut],
  Fabian Scheipl [aut],
  Lei Huang [aut],
  Julia Wrobel [aut, cre],
  Chongzhi Di [aut],
  Jonathan Gellar [aut],
  Jaroslaw Harezlak [aut],
  Mathew W. McLean [aut],
  Bruce Swihart [aut],
  Luo Xiao [aut],
  Ciprian Crainiceanu [aut],
  Philip T. Reiss [aut],
  Yakuan Chen [ctb],
  Sonja Greven [ctb],
  Lan Huo [ctb],
  Madan Gopal Kundu [ctb],
  So Young Park [ctb],
  David L. Miller [ctb],
  Ana-Maria Staicu [ctb]

Maintainer Julia Wrobel <julia.wrobel@cuanschutz.edu>

Date/Publication 2020-09-03 22:20:06 UTC

R topics documented:

  refund-package ................................................. 5
  af .......................................................... 5
  af_old ....................................................... 8
  bayes_fosr .................................................. 10
  ccb.fpc ..................................................... 12
  cd4 ........................................................ 14
  cmdscale_lanczos ............................................ 15
  coef.pffr ................................................... 16
  coefboot.pffr ............................................... 18
  coefficients.pfr ............................................ 19
  create.prep.func ........................................... 21
  DTI .......................................................... 22
### R topics documented:

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>DTI2</td>
<td>23</td>
</tr>
<tr>
<td>expand.call</td>
<td>24</td>
</tr>
<tr>
<td>fbps</td>
<td>24</td>
</tr>
<tr>
<td>ff</td>
<td>27</td>
</tr>
<tr>
<td>ffpc</td>
<td>29</td>
</tr>
<tr>
<td>ffpcplot</td>
<td>31</td>
</tr>
<tr>
<td>fgam</td>
<td>32</td>
</tr>
<tr>
<td>fosr</td>
<td>34</td>
</tr>
<tr>
<td>fosr.perm</td>
<td>38</td>
</tr>
<tr>
<td>fosr.vs</td>
<td>41</td>
</tr>
<tr>
<td>fosr2s</td>
<td>43</td>
</tr>
<tr>
<td>fpc</td>
<td>45</td>
</tr>
<tr>
<td>fpca.face</td>
<td>48</td>
</tr>
<tr>
<td>fpca.lfda</td>
<td>51</td>
</tr>
<tr>
<td>fpca.sc</td>
<td>58</td>
</tr>
<tr>
<td>fpca.ssvd</td>
<td>61</td>
</tr>
<tr>
<td>fpca2s</td>
<td>64</td>
</tr>
<tr>
<td>fpcr</td>
<td>66</td>
</tr>
<tr>
<td>f_sum</td>
<td>70</td>
</tr>
<tr>
<td>f_sum2</td>
<td>71</td>
</tr>
<tr>
<td>f_sum4</td>
<td>72</td>
</tr>
<tr>
<td>f_trace</td>
<td>72</td>
</tr>
<tr>
<td>gasoline</td>
<td>73</td>
</tr>
<tr>
<td>gibbs_cs_fpca</td>
<td>73</td>
</tr>
<tr>
<td>gibbs_cs_wish</td>
<td>75</td>
</tr>
<tr>
<td>gibbs_mult_fpca</td>
<td>76</td>
</tr>
<tr>
<td>gibbs_mult_wish</td>
<td>77</td>
</tr>
<tr>
<td>gls_cs</td>
<td>79</td>
</tr>
<tr>
<td>lf</td>
<td>80</td>
</tr>
<tr>
<td>lf.vd</td>
<td>82</td>
</tr>
<tr>
<td>lf_old</td>
<td>85</td>
</tr>
<tr>
<td>lpeer</td>
<td>87</td>
</tr>
<tr>
<td>lpfr</td>
<td>91</td>
</tr>
<tr>
<td>mfpca.sc</td>
<td>94</td>
</tr>
<tr>
<td>model.matrix.pffr</td>
<td>96</td>
</tr>
<tr>
<td>ols_cs</td>
<td>97</td>
</tr>
<tr>
<td>pco_predict_preprocess</td>
<td>98</td>
</tr>
<tr>
<td>pcre</td>
<td>99</td>
</tr>
<tr>
<td>peer</td>
<td>100</td>
</tr>
<tr>
<td>PEER.Sim</td>
<td>103</td>
</tr>
<tr>
<td>peer_old</td>
<td>103</td>
</tr>
<tr>
<td>pffr</td>
<td>106</td>
</tr>
<tr>
<td>pffr.check</td>
<td>111</td>
</tr>
<tr>
<td>pffrGLS</td>
<td>112</td>
</tr>
<tr>
<td>pffrSim</td>
<td>113</td>
</tr>
<tr>
<td>pfr</td>
<td>114</td>
</tr>
<tr>
<td>pfr_old</td>
<td>117</td>
</tr>
<tr>
<td>plot.fosr</td>
<td>122</td>
</tr>
</tbody>
</table>
R topics documented:

plot.fosr.vs ......................................................... 123
plot.fpcr .......................................................... 125
plot.lpeer .......................................................... 126
plot.peer ........................................................... 127
plot.pffr ........................................................... 128
plot.pfr ............................................................. 129
predict.fbps ......................................................... 130
predict.fgam ......................................................... 132
predict.fosr ........................................................ 133
predict.fosr.vs ....................................................... 135
Predict.matrix.dt.smooth ........................................... 136
Predict.matrix.fpc.smooth ......................................... 137
Predict.matrix.pcre.random.effect ................................ 137
Predict.matrix.peer.smooth ........................................ 138
Predict.matrix.pi.smooth .......................................... 139
predict.pffr ......................................................... 139
predict.pfr .......................................................... 141
print.summary.pffr .................................................. 142
pwcv ................................................................. 143
qq.pffr ............................................................... 145
quadWeights ......................................................... 146
re ...................................................................... 146
refund-internal ....................................................... 147
residuals.pffr ......................................................... 147
rlrt.pfr ............................................................... 148
sff ..................................................................... 151
smooth.construct.dt.smooth.spec .................................. 152
smooth.construct.fpc.smooth.spec ................................ 154
smooth.construct.pco.smooth.spec ................................. 156
smooth.construct.pcre.smooth.spec ................................ 158
smooth.construct.peer.smooth.spec ................................ 159
smooth.construct.pi.smooth.spec ................................ 160
smooth.construct.pss.smooth.spec ................................ 162
sofa ................................................................. 163
summary.pffr ......................................................... 164
summary.pfr .......................................................... 164
vb_cs_fPCA .......................................................... 165
vb_cs_wish ............................................................ 166
vb_mult_fPCA ........................................................ 167
vb_mult_wish ........................................................ 168
vis.fgam ............................................................. 170
vis.pfr ................................................................. 172
Xt_siginv_X ........................................................... 174

Index 175
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 and fpcri. For regression with functional responses, see pffr, fosr, and fosr2s.

Details

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

Author(s)

Authors: Jeff Goldsmith <jeff.goldsmith@columbia.edu>, Fabian Scheipl <fabian.scheipl@stat.uni-muenchen.de>, Lei Huang <huangracer@gmail.com>, Julia Wrobel <jw3134@cumc.columbia.edu>, Jonathan Gellar <jgellar1@jhu.edu>, Jaroslaw Harezlak, Mathew W. McLean <mathew.w.mclean@gmail.com>, Bruce Swihart <bswichart@jhspih.edu>, Luo Xiao <lxiao@jhspih.edu>, Ciprian Crainiceanu <ccrainic@jhspih.edu>, Philip Reiss <phil.reiss@nymc.org>.

Contributors: Yakuan Chen, Sonja Greven, Lan Huo, Madan Gopal Kundu, Ana-Maria Staicu, David L. Miller, So Young Park

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 pfr, where $F(x,t)$ is an unknown smooth bivariate function and $X_i(t)$ is a functional predictor on the closed interval $T$. See smooth.terms for a list of bivariate basis and penalty options; the default is a tensor product basis with marginal cubic regression splines for estimating $F(x,t)$.

Usage

af(
  X, 
  argvals = NULL, 
  xind = NULL, 
  basistype = c("te", "t2", "s"), 
  integration = c("simpson", "trapezoidal", "riemann"), 
  L = NULL, 
  presmooth = NULL, 
)
presmooth.opts = NULL,
Xrange = range(X, na.rm = T),
Qtransform = FALSE,
...
)

**Arguments**

- **X**
  functional predictors, typically expressed as an N by J matrix, where N is the number of columns and J is the number of evaluation points. May include missing/sparse functions, which are indicated by NA values. Alternatively, can be an object of class "fd"; see fd.

- **argvs**
  indices of evaluation of X, i.e. \((t_{i1}, \ldots, t_{iJ})\) for subject \(i\). May be entered as either a length-J vector, or as an N by J matrix. Indices may be unequally spaced. Entering as a matrix allows for different observations times for each subject. If NULL, defaults to an equally-spaced grid between 0 or 1 (or within X$basis$rangeval if X is a fd object.)

- **xind**
  same as argvs. It will not be supported in the next version of refund.

- **basistype**
  defaults to "te", i.e. a tensor product spline to represent \(F(x,t)\). 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 for calculating entries in L. Alternatively and for non-equidistant grids, "trapezoidal" or "riemann".

- **L**
  an optional N by ncol(argvs) matrix giving the weights for the numerical integration over t. If present, overrides integration.

- **presmooth**
  string indicating the method to be used for preprocessing functional predictor prior to fitting. Options are fpca.sc, fpca.face, fpca.ssvd, fpca.bspline, and fpca.interpolate. Defaults to NULL indicating no preprocessing. See create.prep.func.

- **presmooth.opts**
  list including options passed to preprocessing method create.prep.func.

- **Xrange**
  numeric; 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**
  logical; should the functional be transformed using the empirical cdf and applying a quantile transformation on each column of X prior to fitting?

- **...**
  optional arguments for basis and penalization to be passed to the function indicated by basistype. These could include, for example, "bs", "k", "m", etc. See te or s for details.

**Value**

A list with the following entries:

- **call**
  a "call" to te (or s, t2) using the appropriately constructed covariate and weight matrices.
The `af` function is part of the `mgcv` package and is used for functional additive models. It takes several arguments:

- `argvals` - the `argvals` 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 `argvals` 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`
- `Xrange` - the `Xrange` argument supplied to `af`
- `prep.func` - a function that preprocesses data based on the preprocessing method specified in `presmooth`. See `create.prep.func`.

**Author(s)**
Mathew W. McLean <mathew.w.mclean@gmail.com>, Fabian Scheipl, and Jonathan Gellar

**References**

**See Also**
pfr, lf, mgcv's linear.functional.terms, pfr for examples

**Examples**
```r
## Not run:
data(DTI)
data(DTI)
DTI1 <- DTI[DTI$visit==1 & DTI$case==1,]
DTI2 <- DTI[complete.cases(DTI1),]

## 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
fit1 <- pfr(pasat ~ af(cca, k=c(8,8), m=list(c(2,3), c(2,3)),
               presmooth="bspline", bs="ps"),
               method="GCV.Cp", gamma=1.2, data=DTI2)
plot(fit1, scheme=2)
vis.pfr(fit1)

## af term for the cca measurements plus an lf term for the rcst measurements
## leave out 10 samples for prediction
test <- sample(nrow(DTI2), 10)
fit2 <- pfr(pasat ~ af(cca, k=c(7,7), m=list(c(2,2), c(2,2)), bs="ps",
               presmooth="fpca.face") +
```

```
Construct an FGAM regression term

Description

Defines a term \( \int_T F(X(t), t) \, dt \) for inclusion in an \texttt{mgcv::gam}-formula (or \texttt{bam} or \texttt{gamm} or \texttt{gamm4::gamm}) as constructed by \texttt{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

```r
af_old(
  X,
  argvals = seq(0, 1, l = ncol(X)),
  xind = NULL,
  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 \(N \times J=\text{ncol(argvals)}\) matrix of function evaluations \(X_i(t_{i1}), \ldots, X_i(t_{iJ}); i = 1, \ldots, N\).
- **argvals**: matrix (or vector) of indices of evaluations of \(X_i(t)\); i.e. a matrix with \(i\)th row \((t_{i1}, \ldots, t_{iJ})\).
- **xind**: Same as \text{argvals}. It will discard this argument in the next version of \text{refund}.
- **basistype**: defaults to "te", i.e. a tensor product spline to represent \(F(x, t)\). Alternatively, use "s" for bivariate basis functions (see \text{s}) or "t2" for an alternative parameterization of tensor product splines (see \text{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 \(L\) is specified.
- **L**: optional weight matrix for the linear functional.
- **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. \text{list(bs="ps", m=list(c(2,2), c(2,2)))}, see \text{te} or \text{s} for details.
- **presmooth**: logical; if true, the functional predictor is pre-smoothed prior to fitting; see \text{smooth.basisPar}.
- **Xrange**: numeric; range to use when specifying the marginal basis for the \(x\)-axis. It may be desired to increase this slightly over the default of \text{range(X)} if concerned about predicting for future observed curves that take values outside of \text{range(X)}.
- **Qtransform**: logical; should the functional be transformed using the empirical cdf and applying a quantile transformation on each column of \(X\) prior to fitting? This ensures \(Xrange=[0,1]\). If \text{Qtransform=TRUE} and \text{presmooth=TRUE}, presmoothing is done prior to transforming the functional predictor.

Value

A list with the following entries:
1. call - a "call" to te (or s, t2) using the appropriately constructed covariate and weight matrices.

2. argvals - the argvals argument supplied to af

3. 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 argvals 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 Qtransform - 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 <mathew.w.mclean@gmail.com> and Fabian Scheipl

References


See Also

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

bayes_fosr  Bayesian Function-on-scalar regression

Description

Wrapper function that implements several approaches to Bayesian function-on-scalar regression. Currently handles real-valued response curves; models can include subject-level random effects in a multilevel framework. The residual curve error structure can be estimated using Bayesian FPCA or a Wishart prior. Model parameters can be estimated using a Gibbs sampler or variational Bayes.

Usage

bayes_fosr(formula, data = NULL, est.method = "VB", cov.method = "FPCA", ...)

Arguments

formula a formula indicating the structure of the proposed model. Random intercepts are designated using re().

data an optional data frame, list or environment containing the variables in the model. If not found in data, the variables are taken from environment(formula), typically the environment from which the function is called.
bayes_fosr

est.method method used to estimate model parameters. Options are "VB", "Gibbs", and
"GLS" with "VB" as default. Variational Bayes is a fast approximation to the
full posterior and often provides good point estimates, but may be unreliable for
inference. "GLS" doesn’t do anything Bayesian – just fits an unpenalized GLS
estimator for the specified model.

cov.method method used to estimate the residual covariance structure. Options are "FPCA"
and "Wishart", with default "FPCA"

... additional arguments that are passed to individual fitting functions.

Author(s)

Jeff Goldsmith <ajg2202@cumc.columbia.edu>

References

215-236.

Examples

## Not run:
library(reshape2)
library(dplyr)
library(ggplot2)

##### Cross-sectional real-data examples #####

## organize data
data(DTI)
DTI = subset(DTI, select = c(cca, case, pasat))
DTI = DTI[complete.cases(DTI),]
DTI$gender = factor(sample(c("male", "female"), dim(DTI)[1], replace = TRUE))
DTI$status = factor(sample(c("RRMS", "SPMS", "PPMS"), dim(DTI)[1], replace = TRUE))

## fit models
default = bayes_fosr(cca ~ pasat, data = DTI)
VB = bayes_fosr(cca ~ pasat, data = DTI, Kp = 4, Kt = 10)
Gibbs = bayes_fosr(cca ~ pasat, data = DTI, Kt = 10, est.method = "Gibbs", cov.method = "Wishart",
N.iter = 500, N.burn = 200)
OLS = bayes_fosr(cca ~ pasat, data = DTI, Kt = 10, est.method = "OLS")
GLS = bayes_fosr(cca ~ pasat, data = DTI, Kt = 10, est.method = "GLS")

## plot results
models = c("default", "VB", "Gibbs", "OLS", "GLS")
intercepts = sapply(models, function(u) get(u)$beta.hat[,1])
slopes = sapply(models, function(u) get(u)$beta.hat[,2])

plot.dat = melt(intercepts); colnames(plot.dat) = c("grid", "method", "value")
```r
ggplot(plot.dat, aes(x = grid, y = value, group = method, color = method)) + geom_path() + theme_bw()

plot.dat = melt(slopes); colnames(plot.dat) = c("grid", "method", "value")
ggplot(plot.dat, aes(x = grid, y = value, group = method, color = method)) + geom_path() + theme_bw()

## fit a model with an interaction
fosr.dti.interaction = bayes_fosr(cca ~ pasat*gender, data = DTI, Kp = 4, Kt = 10)

###### Longitudinal real-data examples ######

data(DTI2)
class(DTI2$cca) = class(DTI2$cca)[-1]
DTI2 = subset(DTI2, select = c(cca, id, pasat))
DTI2 = DTI2[complete.cases(DTI2),]
default = bayes_fosr(cca ~ pasat + re(id), data = DTI2)
VB = bayes_fosr(cca ~ pasat + re(id), data = DTI2, Kt = 10, cov.method = "Wishart")

## End(Not run)
```

---

cbb.fpc

**Corrected confidence bands using functional principal components**

**Description**

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

**Usage**

```r
cbb.fpc(Y, argvals = NULL, nbasis = 10, pve = 0.99, n.boot = 100, simul = FALSE, sim.alpha = 0.95)
```

**Arguments**

- `Y` matrix of observed functions for which estimates and covariance matrices are desired.
argvals numeric; function argument.
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

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 intervals
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
ex.d = as.numeric(colnames(cd4))
plot(ex.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(ex.d),
     xlab = "Months since seroconversion", lwd = 1.2, ylab = "Total CD4 Cell Count",
     main = "Est. & CI - Sampled Data")
matpoints(ex.d, EX.IV, col = 2, type = 'l', lwd = c(2, 1, 1, 1, 1), lty = c(1,1,1,2,2))
matpoints(ex.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", "MM - \hat{theta} Est", "MM - \hat{theta} PW Int", "MM - \hat{theta} 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(1,1,1,1,1), lwd = c(2.5,.75,.75,2.5,.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.

**Format**

A data frame made up of

- **list("cd4")**  A 366 x 61 matrix of CD4 cell counts.
  A 366 x 61 matrix of CD4 cell counts.
cmdscale_lanczos

References


---

**cmdscale_lanczos**  
*Faster multi-dimensional scaling*

**Description**

This is a modified version of `cmdscale` that uses the Lanczos procedure (`slanczos`) instead of `eigen`. Called by `smooth.construct.pco.smooth.spec`.

**Usage**

`cmdscale_lanczos(d, k = 2, eig = FALSE, add = FALSE, x.ret = FALSE)`

**Arguments**

- `d`: a distance structure as returned by `dist`, or a full symmetric matrix of distances or dissimilarities.
- `k`: the maximum dimension of the space which the data are to be represented in; must be in \{1,2,...,n-1\}.
- `eig`: logical indicating whether eigenvalues should be returned.
- `add`: logical indicating if the additive constant of Cailliez (1983) should be computed, and added to the non-diagonal dissimilarities such that the modified dissimilarities are Euclidean.
- `x.ret`: indicates whether the doubly centred symmetric distance matrix should be returned.

**Value**

as `cmdscale`

**Author(s)**

David L Miller, based on code by R Core.

**References**


**See Also**

`smooth.construct.pco.smooth.spec`
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

```r
## 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`, `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.
(optional) an estimate of the covariance operator of the residual process \( \epsilon_i(t) \sim N(0, K(t,t')) \), evaluated on \( \text{yind} \) of \text{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 \text{seWithMean}-option corresponds to the "iterms"-option in \text{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 \text{raw==FALSE}, a list containing

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

Author(s)

Fabian Scheipl

References


See Also

\text{plot.gam}, \text{predict.gam} which this routine is based on.
**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**

```r
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 = c("resample", "residual", "residual.c"),
  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
- **ncpus**: see `boot`. Defaults to `getOption("boot.ncpus", 1L)` (like `boot`).
- **parallel**: see `boot`
- **cl**: see `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 responses as fitted values plus residual trajectories or "residual.c" to resample responses as fitted values plus residual trajectories that are centered at zero for each gridpoint.
coefficients.pfr

showProgress TRUE/FALSE
...
not used

Value

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

Author(s)

Fabian Scheipl

coefficients.pfr Extract coefficient functions from a fitted pfr-object

Description

This function is used to extract a coefficient from a fitted ‘pfr’ model, in particular smooth functions resulting from including functional terms specified with lf, af, etc. It can also be used to extract smooths generated using mgcv’s s, te, or t2.

Usage

## S3 method for class 'pfr'
coefficients(
  object,
  select = 1,
  coords = NULL,
  n = NULL,
  se = ifelse(length(object$smooth) & select, TRUE, FALSE),
  seWithMean = FALSE,
  useVc = TRUE,
  Qtransform = FALSE,
  ...
)

## S3 method for class 'pfr'
coef(
  object,
  select = 1,
  coords = NULL,
  n = NULL,
  se = ifelse(length(object$smooth) & select, TRUE, FALSE),
  seWithMean = FALSE,
  useVc = TRUE,
  Qtransform = FALSE,
  ...
)
Arguments

object
select
coords
n
se
seWithMean
useVc
Qtransform

... these arguments are ignored

Value

a data frame containing the evaluation points, coefficient function values and optionally the SE’s for the term indicated by select.

Author(s)

Jonathan Gellar and Fabian Scheipl

References

**create.prep.func**

Construct a function for preprocessing functional predictors

**Description**

Prior to using functions $X$ as predictors in a scalar-on-function regression, it is often necessary to presmooth curves to remove measurement error or interpolate to a common grid. This function creates a function to do this preprocessing depending on the method specified.

**Usage**

```r
create.prep.func(
  X,
  argvals = seq(0, 1, length = ncol(X)),
  method = c("fpca.sc", "fpca.face", "fpca.ssvd", "bspline", "interpolate"),
  options = NULL
)
```

**Arguments**

- **X**
  - an N by J=ncol(argvals) matrix of function evaluations $X_i(t_{i1}),...,X_i(t_{iJ}); i = 1,..,N$. For FPCA-based processing methods, these functions are used to define the eigen decomposition used to preprocess current and future data (for example, in `predict.pfr`)
- **argvals**
  - matrix (or vector) of indices of evaluations of $X_i(t); i.e. a matrix with $i$th row $(t_{i1},..,t_{iJ})$
- **method**
  - character string indicating the preprocessing method. Options are "fpca.sc", "fpca.face", "fpca.ssvd", "bspline", and "interpolate". The first three use the corresponding existing function; "bspline" uses an (unpenalized) cubic bspline smoother with nbasis basis functions; "interpolate" uses linear interpolation.
- **options**
  - list of options passed to the preprocessing method; as an example, options for `fpca.sc` include pve, nbasis, and npc.

**Value**

a function that returns the preprocessed functional predictors, with arguments

- **newX**
  - The functional predictors to process
- **argvals**
  - Indices of evaluation of newX
- **options**
  - Any options needed to preprocess the predictor functions

**Author(s)**

Jeff Goldsmith <ajg2202@cumc.columbia.edu>
See Also

pfr, fpca.sc, fpca.face, fpca.ssvd

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.

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;
visit.time Numeric vector of the subject-specific visit time, measured in days since first visit;
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;
sex factor variable indicated subject’s sex;
pasat Numeric vector containing the PASAT score at each visit.

Details

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

References


**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.

**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.

**Details**

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 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.

**References**


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

```r
tbps(
  data,
  subj = NULL,
  covariates = NULL,
  knots = 35,
  knots.option = "equally-spaced",
  periodicity = c(FALSE, FALSE),
  p = 3,
  m = 2,
  lambda = NULL,
  selection = "GCV",
  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
subj vector of subject id (corresponding to the columns of data); defaults to NULL
covariates list of two vectors of covariates of lengths 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
knots.option knot selection method; defaults to "equally-spaced"
periodicity vector of two logical, indicating periodicity in the direction of row and column; defaults to c(FALSE, FALSE)
p degrees of B-splines; defaults to 3
m order of differencing penalty; defaults to 2
lambda user-specified smoothing parameters; defaults to NULL
selection selection of smoothing parameter; defaults to "GCV"
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**


**Examples**

```r
# 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*runif(n1*n2,0,1)

# Estimation
est <- fbps(Y,list(x=x,z=z))
 mse <- mean((est$Yhat-MY)^2)
```
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**

```r
ff(X, yind = NULL, xind = seq(0, 1, l = ncol(X)), basistype = c("te", "t2", "ti", "s", "tes"), integration = c("simpson", "trapezoidal", "riemann"), L = NULL, limits = NULL, splinepars = if (basistype != "s") { list(bs = "ps", m = list(c(2, 1), c(2, 1)), k = c(5, 5)) } else { list(bs = "tp", m = NA) }, check.ident = TRUE)
```

**Arguments**

- `X` an \( n \) by \( \text{ncol}(xind) \) matrix of function evaluations \( X_i(s_1), \ldots, X_i(s_S) \); \( i = 1, \ldots, n \).
- `yind` `DEPRECATED` used to supply matrix (or vector) of indices of evaluations of \( Y_i(t) \), no longer used.
- `xind` vector of indices of evaluations of \( X_i(s) \), i.e., \( (s_1, \ldots, s_S) \)
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 $n \times \text{ncol(xind)}$ 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 $s_{hi}(t), s_{lo}(t)$: either one of "s<t" or "s<=t" for $(s_{hi}(t), s_{lo}(t)) = (t, 0)$ or $[t, 0]$, respectively, 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 conditions for non-identifiability of the effect. This occurs if a) the marginal basis for the functional covariate is rank-deficient (typically because the functional covariate has lower rank than the spline basis along its index) and simultaneously b) the kernel of Cov$(X(s))$ is not disjunct from the kernel of the marginal penalty over $s$. In practice, a) occurs quite frequently, and b) occurs usually because curve-wise mean centering has removed all constant components from the functional covariate.

If there is kernel overlap, $\beta(t, s)$ is constrained to be orthogonal to functions in that overlap space (e.g., if the overlap contains constant functions, constraints $\int \beta(t, s) ds = 0$ for all $t$' are enforced). See reference for details.

A warning is always given if the effective rank of Cov$(X(s))$ (defined as the number of eigenvalues accounting for at least 0.995 of the total variance in $X_i(s)$) is lower than 4. If $X_i(s)$ is of very low rank, ffpc-term may be preferable.

Value

A list containing

call a "call" to te (or s or 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:

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**

```r
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 $\beta_k(t)$.
- `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 \texttt{ff}, \texttt{ffpc} does an FPCA decomposition \( X(s) \approx \sum_{k=1}^{K} \xi_{ik} \Phi_k(s) \) using \texttt{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 scores \( \xi_{ik} \). The truncation parameter \( K \) is chosen as described in \texttt{fpca.sc}. Using this instead of \texttt{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, \texttt{ffpc()} will not be very efficient.

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

Please see \texttt{pffr} for details on model specification and implementation.

Value

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

Author(s)

Fabian Scheipl

Examples

```r
## 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)
```
`ffpcplot`  

Plot PC-based function-on-function regression terms

Description

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

Usage

```r
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 scores, "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 scores. Defaults to showing both.

pages

the number of pages over which to spread the output. Defaults to 1. (Irrelevant if auto.layout=FALSE.)

se.mult

display estimated coefficient functions associated with the FPC scores with plus/minus this number time the estimated standard error. Defaults to 2.

ticktype

see persp.

theta

see persp.

phi

see persp.

plot

produce plots or only return plotting data? Defaults to TRUE.

auto.layout

should 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, tensor-type = c("te", "t2"), ...)
Arguments

- **formula**: a formula with special terms as for `gam`, with additional special terms `af()`, `lf()`, `re()`.
- **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**: defaults to `te`, other valid option is `t2`.
- ... 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 <mathew.w.mclean@gmail.com> and Fabian Scheipl

References


See Also

- `af`, `lf`, `predict.fgam`, `vis.fgam`

Examples

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
## plot the fits
## vis.fgam(fit, af.term = "X")
## 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')
**Description**

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

**Usage**

fosr(
    formula = NULL,
    Y = NULL,
    fdobj = NULL,
    data = 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**

- **formula**
  Formula for fitting fosr. If used, data argument must not be null.

- **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.

- **data**
  data frame containing the predictors and responses.

- **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.
The GLS method requires estimating the residual covariance matrix, which has dimension $d \times d$ when the responses are given by $Y$, or $\text{nbasis} \times \text{nbasis}$ 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".

The value of this argument is passed.

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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lambda</td>
<td>smoothing parameter value. If NULL, the smoothing parameter(s) will be estimated. See Details.</td>
</tr>
<tr>
<td>nbasis, norder</td>
<td>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.</td>
</tr>
<tr>
<td>pen.order</td>
<td>order of derivative penalty.</td>
</tr>
<tr>
<td>multi.sp</td>
<td>a logical value indicating whether separate smoothing parameters should be estimated for each coefficient function. Currently must be FALSE if method = &quot;OLS&quot;.</td>
</tr>
<tr>
<td>pve</td>
<td>if method = 'mix', the percentage of variance explained by the principal components; defaults to 0.99.</td>
</tr>
<tr>
<td>max.iter</td>
<td>maximum number of iterations if method = &quot;GLS&quot;.</td>
</tr>
<tr>
<td>maxlam</td>
<td>maximum smoothing parameter value to consider (when lamvec=NULL; see lofocv).</td>
</tr>
<tr>
<td>cv1</td>
<td>logical value indicating whether a cross-validation score should be computed even if a single fixed lambda is specified (when method = &quot;OLS&quot;).</td>
</tr>
<tr>
<td>scale</td>
<td>logical value or vector determining scaling of the matrix $X$ (see scale, to which the value of this argument is passed).</td>
</tr>
</tbody>
</table>

Details

The GLS method requires estimating the residual covariance matrix, which has dimension $d \times d$ when the responses are given by $Y$, or $\text{nbasis} \times \text{nbasis}$ 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".
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**


**See Also**

plot.fosr
Examples

## Not run:
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)

# Test use formula to fit fosr
set.seed(2121)
data1 <- pffrSim(scenario="ff", n=40)
formod = fosr(Y~xlin+xsmoo, data=data1)
plot(formod, 1)

# 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,
fosr.perm

argvals = NULL,
lambda = NULL,
lambda0 = NULL,
multi.sp = FALSE,
nperm,
level = 0.05,
plot = TRUE,
xlabel = "",
title = NULL,
prelim = if (multi.sp) 0 else 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 = 0.05)

## S3 method for class 'fosr.perm'
plot(x, level = 0.05, xlabel = "", title = NULL, ...)

Arguments

Y, fdobj the functional responses, given as either an n x 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.
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.
prelim  number of preliminary permutations. The smoothing parameter in the main per-
mutations will be fixed to the median value from these preliminary permutations. 
If prelim=0, this is not done. Preliminary permutations are not available when 
multi.sp = TRUE (hence the complicated default).

...  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.
x  object of class fosr.perm, outputted by fosr.perm, fosr.perm.fit, or fosr.perm.test.

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

See Also
	fosr

Examples

```r
## 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()

## End(Not run)
```

fosr.vs

Function-on Scalar Regression with variable selection

Description

Implements an iterative algorithm for function-on-scalar regression with variable selection by alternatively updating the coefficients and covariance structure.

Usage

```r
fosr.vs(
    formula,
    data,
    nbasis = 10,
)```
method = c("ls", "grLasso", "grMCP", "grSCAD"),
epsilon = 1e-05,
max.iter_num = 100
)

Arguments

formula an object of class "formula": an expression of the model to be fitted.
data a data frame that contains the variables in the model.
nbasis number of B-spline basis functions used.
method group variable selection method to be used ("grLasso", "grMCP", "grSCAD" refer to group Lasso, group MCP and group SCAD, respectively) or "ls" for least squares estimation.
epsilon the convergence criterion.
max.iter_num maximum number of iterations.

Value

A fitted fosr.vs-object, which is a list with the following elements:

formula an object of class "formula": an expression of the model to be fitted.
coefficients the estimated coefficient functions.
fitted.values the fitted curves.
residuals the residual curves.
vcov the estimated variance-covariance matrix when convergence is achieved.
method group variable selection method to be used or "ls" for least squares estimation.

Author(s)

Yakuan Chen <yc2641@cumc.columbia.edu>

References


See Also

ggrpreg

Examples

## Not run:
set.seed(100)

I = 100
p = 20
D = 50
grid = seq(0, 1, length = D)

beta.true = matrix(0, p, D)
beta.true[1,] = sin(2*grid*pi)
beta.true[2,] = cos(2*grid*pi)
beta.true[3,] = 2

psi.true = matrix(NA, 2, D)
psi.true[1,] = sin(4*grid*pi)
psi.true[2,] = cos(4*grid*pi)
lambda = c(3,1)

set.seed(100)

X = matrix(rnorm(I*p), I, p)
C = cbind(rnorm(I, mean = 0, sd = lambda[1]), rnorm(I, mean = 0, sd = lambda[2]))

fixef = X %*% beta.true
pcaef = C %*% psi.true
error = matrix(rnorm(I*D), I, D)

Yi.true = fixef
Yi.pca = fixef + pcaef
Yi.obs = fixef + pcaef + error

data = as.data.frame(X)
data$Y = Yi.obs
fit.fosr.vs = fosr.vs(Y~., data = data, method="grMCP")
plot(fit.fosr.vs)

## 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**

```r
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


See Also

forsr, pfrr

Examples

```r
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 <- fors2s(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","1","1","1","1","1"),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)
}
```

fpc

*Construct a FPC regression term*

Description

Constructs a functional principal component regression (Reiss and Ogden, 2007, 2010) term for inclusion in an mgcv::gam-formula (or bam or gamm or gamm4:::gamm) as constructed by pfr. Currently only one-dimensional functions are allowed.

Usage

```r
fpc(
X,
argvals = NULL,
method = c("svd", "fpca.sc", "fpca.face", "fpca.ssvd"),
ncomp = NULL,
pve = 0.99,
```

penalize = (method == "svd"),
bs = "ps",
k = 40,
...
)

Arguments

X         functional predictors, typically expressed as an N by J matrix, where N is the number of columns and J is the number of evaluation points. May include missing/sparse functions, which are indicated by NA values. Alternatively, can be an object of class "fd"; see fd.
argvals   indices of evaluation of X, i.e. \((t_{i1}, \ldots, t_{iJ})\) for subject \(i\). May be entered as either a length-J vector, or as an N by J matrix. Indices may be unequally spaced. Entering as a matrix allows for different observations times for each subject. If NULL, defaults to an equally-spaced grid between 0 or 1 (or within X$basis$rangeeval if X is a fd object.)
method    the method used for finding principal components. The default is an unconstrained SVD of the \(XB\) matrix. Alternatives include constrained (functional) principal components approaches
ncomp     number of principal components. if NULL, chosen by pve
pve       proportion of variance explained; used to choose the number of principal components
penalize  if TRUE, a roughness penalty is applied to the functional estimate. Defaults to TRUE if method=="svd" (corresponding to the FPCR_R method of Reiss and Ogden (2007)), and FALSE if method!="svd" (corresponding to FPCR_C).
bs         two letter character string indicating the mgcv-style basis to use for pre-smoothing X
k           the dimension of the pre-smoothing basis
...         additional options to be passed to lf. These include argvals, integration, and any additional options for the pre-smoothing basis (as constructed by mgcv::s), such as m.

details

fpc is a wrapper for 1f, which defines linear functional predictors for any type of basis for inclusion in a pfr formula. fpc simply calls 1f with the appropriate options for the fpc basis and penalty construction.

This function implements both the FPCR-R and FPCR-C methods of Reiss and Ogden (2007). Both methods consist of the following steps:

1. project X onto a spline basis \(B\)
2. perform a principal components decomposition of \(XB\)
3. use those PC’s as the basis in fitting a (generalized) functional linear model
This implementation provides options for each of these steps. The basis for in step 1 can be specified using the arguments bs and k, as well as other options via ...; see s for these options. The type of PC-decomposition is specified with method. And the FLM can be fit either penalized or unpenalized via penalize.

The default is FPCR-R, which uses a b-spline basis, an unconstrained principal components decomposition using svd, and the FLM fit with a second-order difference penalty. FPCR-C can be selected by using a different option for method, indicating a constrained ("functional") PC decomposition, and by default an unpenalized fit of the FLM.

FPCR-R is also implemented in fpcr; here we implement the method for inclusion in a pfr formula.

Value
The result of a call to lf.

NOTE
Unlike fpcr, fpc within a pfr formula does not automatically decorrelate the functional predictors from additional scalar covariates.

Author(s)
Jonathan Gellar <JGellar@mathematica-mpr.com>, Phil Reiss <phil.reiss@nyumc.org>, Lan Huo <lan.huo@nyumc.org>, and Lei Huang <huangracer@gmail.com>

References

See Also
lf, smooth.construct.fpc.smooth.spec

Examples
```r
data(gasoline)
par(mfrow=c(3,1))

# Fit PFCR_R
gasmod1 <- pfr(octane ~ fpc(NIR, ncomp=30), data=gasoline)
plot(gasmod1, rug=FALSE)
est1 <- coef(gasmod1)

# Fit FPCR_C with fpca.sc
gasmod2 <- pfr(octane ~ fpc(NIR, method="fpca.sc", ncomp=6), data=gasoline)
```
# Fit penalized model with fpca.face
gasmod3 <- pfr(octane ~ fpc(NIR, method="fpca.face", penalize=TRUE), data=gasoline)
plot(gasmod3, rug=FALSE)
est3 <- coef(gasmod3)

par(mfrow=c(1,1))
ylm <- range(est1$value)*1.35
plot(value ~ X.argvals, type="l", data=est1, ylim=ylm)
lines(value ~ X.argvals, col=2, data=est2)
lines(value ~ X.argvals, col=3, data=est3)

---

### 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

```r
fpca.face(Y = NULL,
ydata = NULL,
Y.pred = NULL,
argvals = NULL,
pve = 0.99,
npc = NULL,
var = FALSE,
simul = FALSE,
sim.alpha = 0.95,
center = TRUE,
knots = 35,
p = 3,
m = 2,
lambda = NULL,
alpha = 1,
search.grid = TRUE,
search.length = 100,
method = "L-BFGS-B",
lower = -20,
upper = 20,

```
control = NULL
)

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
numeric; function argument.
pve
proportion of variance explained: used to choose the number of principal components.
npc
how many smooth SVs to try to extract, if NA (the default) the hard thresholding rule of Gavish and Donoho (2014) is used (see Details, References).
var
logical; should an estimate of standard error be returned?
simul
logical; if TRUE curves will we simulated using Monte Carlo to obtain an estimate of the sim.alpha quantile at each argval; ignored if var == FALSE
sim.alpha
numeric; if simul==TRUE, quantile to estimate at each argval; ignored if var == FALSE
center
logical; center Y so that its column-means are 0? Defaults to TRUE
knots
number of knots to use or the vectors of knots; defaults to 35
p
integer; the degree of B-splines functions to use
m
integer; the order of difference penalty to use
lambda
smoothing parameter; if not specified smoothing parameter is chosen using optim or a grid search
alpha
numeric; tuning parameter for GCV; see parameter gamma in gam
search.grid
logical; should a grid search be used to find lambda? Otherwise, optim is used
search.length
integer; length of grid to use for grid search for lambda; ignored if search.grid is FALSE
method
method to use; see optim
lower
see optim
upper
see optim
control
see optim

Value

A list with components

1. Yhat - If Y.pred is specified, the smooth version of Y.pred. Otherwise, if Y.pred=NULL, the smooth version of Y.
2. scores - matrix of scores
3. mu - mean function
4. npc - number of principal components
5. efunctions - matrix of eigenvectors
6. evalues - vector of eigenvalues

if var == TRUE additional components are returned
1. sigma2 - estimate of the error variance
2. VarMats - list of covariance function estimate for each subject
3. diag.var - matrix containing the diagonals of each matrix in
4. crit.val - list of estimated quantiles; only returned if simul == TRUE

Author(s)
Luo Xiao

References

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

Examples
```R
#### 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))
```

```R
###################################################
######## Generate Data #############
###################################################
```

```R
###################################################

### Generate Data
```

```R
###################################################
```
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)

Phi <- results$efunctions
eigenvalues <- results$evalues

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.lfda  

Longitudinal Functional Data Analysis using FPCA

Description

Implements longitudinal functional data analysis (Park and Staicu, 2015). It decomposes longitudinally-observed functional observations in two steps. It first applies FPCA on a properly defined marginal covariance function and obtain estimated scores (mFPCA step). Then it further models the underlying process dynamics by applying another FPCA on a covariance of the estimated scores obtained in the mFPCA step. The function also allows to use a random effects model to study the underlying process dynamics instead of a KL expansion model in the second step. Scores in mFPCA step are estimated using numerical integration. Scores in sFPCA step are estimated under a mixed model framework.

Usage

fpca.lfda(  
  Y,  
  subject.index,  
  visit.index,
obsT = NULL,
funcArg = NULL,
numTEvalPoints = 41,
newdata = NULL,
fbps.knots = c(5, 10),
fbps.p = 3,
fbps.m = 2,
mFPCA.pve = 0.95,
mFPCA.knots = 35,
mFPCA.p = 3,
mFPCA.m = 2,
mFPCA.npc = NULL,
LongiModel.method = c("fpca.sc", "lme"),
sFPCA.pve = 0.95,
sFPCA.nbasis = 10,
sFPCA.npc = NULL,
gam.method = "REML",
gam.kT = 10
)

Arguments

Y
a matrix of which each row corresponds to one curve observed on a regular and dense grid (dimension of N by m; N = total number of observed functions; m = number of grid points)

subject.index
subject id; vector of length N with each element corresponding a row of Y

visit.index
index for visits (repeated measures); vector of length N with each element corresponding a row of Y

obsT
actual time of visits at which a function is observed; vector of length N with each element corresponding a row of Y

funcArg
numeric; function argument

numTEvalPoints
total number of evaluation time points for visits; used for pre-binning in sFPCA step; defaults to 41

newdata
an optional data frame providing predictors (i for subject id / Ltime for visit time) with which prediction is desired; defaults to NULL

fbps.knots
list of two vectors of knots or number of equidistant knots for all dimensions for a fast bivariate \( P \)-spline smoothing (fbps) method used to estimate a bivariate, smooth mean function; defaults to c(5,10); see fbps

fbps.p
integer; degrees of B-spline functions to use for a fbps method; defaults to 3; see fbps

fbps.m
integer; order of differencing penalty to use for a fbps method; defaults to 2; see fbps

mFPCA.pve
proportion of variance explained for a mFPCA step; used to choose the number of principal components (PCs); defaults to 0.95; see fpca.face

mFPCA.knots
number of knots to use or the vectors of knots in a mFPCA step; used for obtain a smooth estimate of a covariance function; defaults to 35; see fpca.face
mFPCA.p integer; the degree of B-spline functions to use in a mFPCA step; defaults to 3; see fpca.face
mFPCA.m integer; order of differencing penalty to use in a mFPCA step; defaults to 2; see fpca.face
mFPCA.npc pre-specified value for the number of principal components; if given, it overrides pve; defaults to NULL; see fpca.face
LongiModel.method model and estimation method for estimating covariance of estimated scores from a mFPCA step; either KL expansion model or random effects model; defaults to fpca.sc
sFPCA.pve proportion of variance explained for sFPCA step; used to choose the number of principal components; defaults to 0.95; see fpca.sc
sFPCA.nbasis number of B-spline basis functions used in sFPCA step for estimation of the mean function and bivariate smoothing of the covariance surface; defaults to 10; see fpca.sc
sFPCA.npc pre-specified value for the number of principal components; if given, it overrides pve; defaults to NULL; see fpca.sc
gam.method smoothing parameter estimation method when gam is used for predicting score functions at unobserved visit time, T; defaults to REML; see gam
gam.kT dimension of basis functions to use; see gam

Value
A list with components
obsData observed data (input)
i subject id
funcArg function argument
visitTime visit times
fitted.values fitted values (in-sample); of the same dimension as Y
fitted.values.all a list of which each component consists of a subject’s fitted values at all pairs of evaluation points (s and T)
predicted.values predicted values for variables provided in newdata
bivariateSmoothMeanFunc estimated bivariate smooth mean function
mFPCA.efunctions estimated eigenfunction in a mFPCA step
mFPCA.evalues estimated eigenvalues in a mFPCA step
mFPCA.npc number of principal components selected with pre-specified pve in a mFPCA step
mFPCA.scree.eval estimated eigenvalues obtained with pre-specified pve = 0.9999; for scree plot
sFPCA.xiHat.bySubj

a list of which each component consists of a subject's predicted score functions evaluated at equidistant grid in direction of visit time, T

sFPCA.npc

a vector of numbers of principal components selected in a sFPCA step with pre-specified pve; length of mFPCA.npc

mFPCA.covar

estimated marginal covariance

sFPCA.longDynCov.k

a list of estimated covariance of score function; length of mFPCA.npc

Details

A random effects model is recommended when a set of visit times for all subjects and visits is not dense in its range.

Author(s)

So Young Park <spark13@ncsu.edu>, Ana-Maria Staicu

References


Examples

## Not run:

########################################
### Illustration with real data ###
########################################

data(DTI)
MS <- subset(DTI, case == 1) # subset data with multiple sclerosis (MS) case

index.na <- which(is.na(MS$cca))
Y <- MS$cca; Y[index.na] <- fpca.sc(Y)$Yhat[index.na]; sum(is.na(Y))
id <- MS$ID
visit.index <- MS$visit
visit.time <- MS$visit.time/max(MS$visit.time)

lfpca.dti <- fpca.lfda(Y = Y, subject.index = id,
            visit.index = visit.index, obsT = visit.time,
            LongiModel.method = 'lme',
            mFPCA.pve = 0.95)

TT <- seq(0,1,length.out=41); ss = seq(0,1,length.out=93)

# estimated mean function
persp(x = ss, y = TT, z = t(lfpca.dti$bivariateSmoothMeanFunc),
    xlab="s", ylab="visit times", zlab="estimated mean fn", col='light blue')

# first three estimated marginal eigenfunctions
matplot(ss, lfpca.dti$mFPCA.efunctions[,1:3], type='l', xlab='s', ylab='estimated eigen fn')
# predicted scores function corresponding to first two marginal PCs
matplot(TT, do.call(cbind, lapply(lfpca.dti$sFPCA.xiHat.bySubj, function(a) a[,1])),
         xlab="visit time (T)", ylab="xi_hat(T)", main = "k = 1", type='l')
matplot(TT, do.call(cbind, lapply(lfpca.dti$sFPCA.xiHat.bySubj, function(a) a[,2])),
         xlab="visit time (T)", ylab="xi_hat(T)", main = "k = 2", type='l')

# prediction of cca of first two subjects at T = 0, 0.5 and 1 (black, red, green)
matplot(ss, t(lfpca.dti$fitted.values.all[[1]][c(1,21,41),]),
         type='l', lty = 1, ylab='', xlab="s", main = "Subject = 1")
matplot(ss, t(lfpca.dti$fitted.values.all[[2]][c(1,21,41),]),
         type='l', lty = 1, ylab='', xlab="s", main = "Subject = 2")

########################################
### Illustration with simulated data ###
########################################

###########################################################################################
# data generation
###########################################################################################
set.seed(1)
n <- 100 # number of subjects
ss <- seq(0,1,length.out=101)
TT <- seq(0, 1, length.out=41)
mi <- runif(n, min=6, max=15)
ij <- sapply(mi, function(a) sort(sample(1:41, size=a, replace=FALSE))

# error variances
sigma <- 0.1
sigma_wn <- 0.2

lambdaTrue <- c(1,0.5) # True eigenvalues
eta1True <- c(0.5, 0.5^2, 0.5^3) # True eigenvalues
eta2True <- c(0.5^2, 0.5^3) # True eigenvalues
phi <- sqrt(2)*cbind(sin(2*pi*ss),cos(2*pi*ss))
psi1 <- cbind(rep(1,length(TT)), sqrt(3)*(2*TT-1), sqrt(5)*(6*TT^2-6*TT+1))
psi2 <- sqrt(2)*cbind(sin(2*pi*TT),cos(2*pi*TT))
zeta1 <- sapply(eta1True, function(a) rnorm(n = n, mean = 0, sd = a))
zeta2 <- sapply(eta2True, function(a) rnorm(n = n, mean = 0, sd = a))

xi1 <- unlist(lapply(1:n, function(a) (zeta1 %*% t(psi1))[a,ij[[a]]] ))
xi2 <- unlist(lapply(1:n, function(a) (zeta2 %*% t(psi2))[a,ij[[a]]] ))
xi <- cbind(xi1, xi2)

Tij <- unlist(lapply(1:n, function(i) TT[[i]]))
i <- unlist(lapply(1:n, function(i) rep(i, length(ij[[i]])))))
j <- unlist(lapply(1:n, function(i) 1:length(ij[[i]]))]])

X <- xi %*% t(phi)
meanFn <- function(s,t){ 0.5*t + 1.5*s + 1.3*s*t}
mu <- matrix(meanFn(s = rep(ss, each=length(Tij)), t=rep(Tij, length(ss)) ), nrow=nrow(X))
Y <- mu + X +
matrix(rnorm(nrow(X)*ncol(phi), 0, sigma), nrow=nrow(X)) %*% t(phi) + #correlated error
matrix(rnorm(length(X), 0, sigma_wn), nrow=nrow(X)) # white noise

matplot(ss, t(Y[which(i==2),]), type='l', ylab='', xlab="functional argument",
main="observations from subject i = 2")

# END: data generation

Estimation I : when covariance of scores from a mFPCA step is estimated using fpca.sc

est <- fpca.lfda(Y = Y,
subject.index = i, visit.index = j, obsT = Tij,
funcArg = ss, numTEvalPoints = length(TT),
newdata = data.frame(i = c(1:3), Ltime = c(Tij[[1]], 0.2, 0.5)),
fbps.knots = 35, fbps.p = 3, fbps.m = 2,
LongiModel.method='fpca.sc',
mFPCA.pve = 0.95, mFPCA.knots = 35, mFPCA.p = 3, mFPCA.m = 2,
sFPCA.pve = 0.95, sFPCA.nbasis = 10, sFPCA.npc = NULL,
gam.method = 'REML', gam.kT = 10)

# mean function (true vs. estimated)
par(mfrow=c(1,2))
persp(x=TT, y = ss, z= t(sapply(TT, function(a) meanFn(s=ss, t = a))),
xlab="visit times", ylab="s", zlab="true mean fn")
persp(x = TT, y = ss, est$bivariateSmoothMeanFunc,
  xlab="visit times", ylab="s", zlab="estimated mean fn", col='light blue')

############### mFPCA step ###############
par(mfrow=c(1,2))

# marginal covariance fn (true vs. estimated)
image(phi%*%diag(lambdaTrue)%*%t(phi))
image(est$mFPCA.covar)

# eigenfunctions (true vs. estimated)
matplot(ss, phi, type='l')
matlines(ss, cbind(est$mFPCA.efunctions[,1], est$mFPCA.efunctions[,2]), type='l', lwd=2)

# scree plot
plot(cumsum(est$mFPCA.scree.eval)/sum(est$mFPCA.scree.eval), type='l',
  ylab = "Percentage of variance explained")
points(cumsum(est$mFPCA.scree.eval)/sum(est$mFPCA.scree.eval), pch=16)

############### sFPCA step ###############
par(mfrow=c(1,2))
print(est$sFPCA.npc)  # k = 2

# covariance of score functions for k = 1 (true vs. estimated)
image(psi1%*%diag(eta1True)%*%t(psi1), main='TRUE')
image(est$sFPCA.longDynCov.k[[1]], main='ESTIMATED')
# covariance of score functions for k = 2 (true vs. estimated)
image(psi2%*%diag(eta2True)%*%t(psi2))
image(est$sFPCA.longDynCov.k[[2]])

# estimated scores functions
matplot(TT, do.call(cbind,lapply(est$sFPCA.xiHat.bySubj, function(a) a[,1])),
       xlab="visit time", main="k=1", type='l', ylab='',
       col=rainbow(100, alpha = 1), lwd=1, lty=1)
matplot(TT, do.call(cbind,lapply(est$sFPCA.xiHat.bySubj, function(a) a[,2])),
       xlab="visit time", main="k=2", type='l', ylab='',
       col=rainbow(100, alpha = 1), lwd=1, lty=1)

##################################  In-sample and Out-of-sample Prediction  ##################################
par(mfrow=c(1,2))
# fitted
matplot(ss, t(Y[which(i==1),]), type='l', ylab='', xlab="functional argument")
matlines(ss, t(est$fitted.values[which(i==1),]), type='l', lwd=2)
# sanity check : expect fitted and predicted (obtained using info from newdata)
# values to be the same
plot(ss, est$fitted.values[1,], type='p', xlab='', ylab='', pch = 1, cex=1)
lines(ss, est$predicted.values[1,], type='l', lwd=2, col='blue')
all.equal(est$predicted.values[1,], est$fitted.values[1,])

####################################################################################################################
# Illustration II : when covariance of scores from a mFPCA step is estimated using lmer
######################################################################################################################
est.lme <- fpca.lfda(Y = Y,
                      subject.index = i, visit.index = j, obsT = Tij,
                      funcArg = ss, numTEvalPoints = length(TT),
                      newdata = data.frame(i = c(1:3), Ltime = c(Tij[1], 0.2, 0.5)),
                      fbps.knots = 35, fbps.p = 3, fbps.m = 2,
                      LongiModel.method='lme',
                      mFPCA.pve = 0.95, mFPCA.knots = 35, mFPCA.p = 3, mFPCA.m = 2,
                      gam.method = 'REML', gam.kT = 10)
par(mfrow=c(2,2))
# fpca.sc vs. lme (assumes linearity)
matplot(TT, do.call(cbind,lapply(est$sFPCA.xiHat.bySubj, function(a) a[,1])),
        xlab="visit time", main="k=1", type='l', ylab='',
        col=rainbow(100, alpha = 1), lwd=1, lty=1)
matplot(TT, do.call(cbind,lapply(est$sFPCA.xiHat.bySubj, function(a) a[,2])),
        xlab="visit time", main="k=2", type='l', ylab='',
        col=rainbow(100, alpha = 1), lwd=1, lty=1)
matplot(TT, do.call(cbind,lapply(est.lme$sFPCA.xiHat.bySubj, function(a) a[,1])),
        xlab="visit time", main="k=1", type='l', ylab='',
        col=rainbow(100, alpha = 1), lwd=1, lty=1)
matplot(TT, do.call(cbind,lapply(est.lme$sFPCA.xiHat.bySubj, function(a) a[,2])),
        xlab="visit time", main="k=2", type='l', ylab='',
        col=rainbow(100, alpha = 1), lwd=1, lty=1)
Description

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

Usage

```r
fpca.sc(
    Y = NULL,
    ydata = NULL,
    Y.pred = NULL,
    argvals = NULL,
    random.int = FALSE,
    nbasis = 10,
    pve = 0.99,
    npc = NULL,
    var = FALSE,
    simul = FALSE,
    sim.alpha = 0.95,
    useSymm = FALSE,
    makePD = FALSE,
    center = TRUE,
    cov.est.method = 2,
    integration = "trapezoidal"
)
```

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**: the argument values of the function evaluations in Y, defaults to a equidistant grid from 0 to 1.
- **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: 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.

integration quadrature method for numerical integration; only 'trapezoidal' is currently supported.

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

An object of class fpca containing:

\( Yhat \)   FPC approximation (projection onto leading components) of \( Y.pred \) if specified, or else of \( Y \).
the observed data
scores $n \times \text{npc}$ matrix of estimated FPC scores.
mu estimated mean function (or a vector of zeroes if center==FALSE).
efunctions $d \times \text{npc}$ 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 \text{npc}, or the minimum number of basis functions needed to explain proportion pve of the variance in the observed curves.
argvals argument values of eigenfunction evaluations
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

Examples
```r
# Not run:
library(ggplot2)
library(reshape2)
data(cd4)

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

Fit.mu = data.frame(mu = Fit.MM$mu,
                   d = as.numeric(colnames(cd4)))
Fit.basis = data.frame(phi = Fit.MM$efunctions,
                      d = as.numeric(colnames(cd4)))

## for one subject, examine curve estimate, pointwise and simultaneous intervals
EX = 1
```
EX.MM = data.frame(fitted = Fit.MM$Yhat[EX,],
ptwise.UB = Fit.MM$Yhat[EX,] + 1.96 * sqrt(Fit.MM$diag.var[EX,]),
ptwise.LB = Fit.MM$Yhat[EX,] - 1.96 * sqrt(Fit.MM$diag.var[EX,]),
simul.UB = Fit.MM$Yhat[EX,] + Fit.MM$crit.val[EX] * sqrt(Fit.MM$diag.var[EX,]),
simul.LB = Fit.MM$Yhat[EX,] - Fit.MM$crit.val[EX] * sqrt(Fit.MM$diag.var[EX,]),
d = as.numeric(colnames(cd4)))

## plot data for one subject, with curve and interval estimates
EX.MM.m = melt(EX.MM, id = 'd')
ggplot(EX.MM.m, aes(x = d, y = value, group = variable, color = variable, linetype = variable)) + geom_path() +
  scale_linetype_manual(values = c(fitted = 1, ptwise.UB = 2,
                             ptwise.LB = 2, simul.UB = 3, simul.LB = 3)) +
  scale_color_manual(values = c(fitted = 1, ptwise.UB = 2,
                              ptwise.LB = 2, simul.UB = 3, simul.LB = 3)) +
  labs(x = 'Months since seroconversion', y = 'Total CD4 Cell Count')

## plot estimated mean function
ggplot(Fit.mu, aes(x = d, y = mu)) + geom_path() +
  labs(x = 'Months since seroconversion', y = 'Total CD4 Cell Count')

## plot the first two estimated basis functions
Fit.basis.m = melt(Fit.basis, id = 'd')
ggplot(subset(Fit.basis.m, variable %in% c('phi.1', 'phi.2')), aes(x = d,
                                   y = value, group = variable, color = variable)) + geom_path()

## 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)

## End(Not run)
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

```r
fpca.ssvd(
  Y = NULL,
  ydata = NULL,
  argvals = NULL,
  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,
  integration = "trapezoidal"
)
```

Arguments

- `Y`: data matrix (rows: observations; columns: grid of eval. points)
- `ydata`: a data frame `ydata` representing irregularly observed functions. NOT IMPLEMENTED for this method.
- `argvals`: the argument values of the function evaluations in `Y`, defaults to an equidistant grid from 0 to 1. See Details.
- `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.
<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lower.alpha</td>
<td>lower limit for for smoothing parameter if !gridsearch</td>
</tr>
<tr>
<td>upper.alpha</td>
<td>upper limit for smoothing parameter if !gridsearch</td>
</tr>
<tr>
<td>verbose</td>
<td>generate graphical summary of progress and diagnostic messages? defaults to FALSE</td>
</tr>
<tr>
<td>integration</td>
<td>ignored, see Details.</td>
</tr>
</tbody>
</table>

**Details**

Note that `fpca.ssvd` computes smoothed orthonormal eigenvectors of the supplied function evaluations (and associated scores), not (!) evaluations of the smoothed orthonormal eigenfunctions. The smoothed orthonormal eigenvectors are then rescaled by the length of the domain defined by `argvals` to have a quadratic integral approximately equal to one (instead of crossproduct equal to one), so they approximate the behavior of smooth eigenfunctions. If `argvals` is not equidistant, `fpca.ssvd` will simply return the smoothed eigenvectors without rescaling, with a warning.

**Value**

an `fpca` object like that returned from `fpca.sc`, with entries `Yhat`, the smoothed trajectories, `Y`, the observed data, `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 if `argvals` is not equidistant), `evalues`, their associated eigenvalues, and `npc`, the number of smooth components that were extracted.

**Author(s)**

Fabian Scheipl

**References**


**See Also**

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

**Examples**

```r
## 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, 1/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 = NULL,
ydata = NULL,
argvals = NULL,
npc = NA,
center = TRUE,
smooth = TRUE
)

Arguments

Y data matrix (rows: observations; columns: grid of eval. points)
ydata a data frame ydata representing irregularly observed functions. NOT IMPLEMENTED for this method.
argvals the argument values of the function evaluations in Y, defaults to a equidistant grid from 0 to 1. See Details.
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

smooth

logical; defaults to TRUE, if NULL, no smoothing of eigenvectors.

Details

Note that fpca2s computes smoothed orthonormal eigenvectors of the supplied function evaluations (and associated scores), not (!) evaluations of the smoothed orthonormal eigenfunctions. The smoothed orthonormal eigenvectors are then rescaled by the length of the domain defined by argvals to have a quadratic integral approximately equal to one (instead of crossproduct equal to one), so they approximate the behavior of smooth eigenfunctions. If argvals is not equidistant, fpca2s will simply return the smoothed eigenvectors without rescaling, with a warning.

Value

an fpca object like that returned from fpca.sc, with entries Yhat, the smoothed trajectories, Y, the observed data, 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 if argvals is not equidistant), evalues, their associated eigenvalues, and npc, the number of smooth components that were extracted.

Author(s)

Luo Xiao <lxiao@jhsph.edu>, Fabian Scheipl

References


See Also

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

Examples

```r
### 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

```r
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

```r
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)
Phi <- results$efunctions
eigenvalues <- results$evalues
```

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

### plot eigenfunctions

```r
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**

Usage

```r
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.

cv 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


Examples

```r
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 <- t(gasoline$NIR)
gas.fd = smooth.basisPar(wavelengths, nir, bbasis)$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.

### 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)

### 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)
**f_sum**

*Sum computation 1*

**Description**
Internal function used compute a sum in FPCA-based covariance updates

**Usage**
f_sum(mu.q.c, sig.q.c, theta, obspts.mat)

**Arguments**
- mu.q.c: current value of mu.q.c
- sig.q.c: current value of sig.q.c
- theta: spline basis
- obspts.mat: matrix indicating the points on which data is observed

**Author(s)**
Jeff Goldsmith <ajg2202@cumc.columbia.edu>

---

**f_sum2**

*Sum computation 2*

**Description**
Internal function used compute a sum in FPCA-based covariance updates

**Usage**
f_sum2(y, fixef, mu.q.c, kt, theta)

**Arguments**
- y: outcome matrix
- fixef: current estimate of fixed effects
- mu.q.c: current value of mu.q.c
- kt: number of basis functions
- theta: spline basis

**Author(s)**
Jeff Goldsmith <ajg2202@cumc.columbia.edu>
f_sum4  
*Sum computation 2*

**Description**

Internal function used compute a sum in FPCA-based covariance updates

**Usage**

\[ f_{\text{sum4}}(\text{mu.q.c}, \text{sig.q.c}, \text{mu.q.bpsi}, \text{sig.q.bpsi}, \text{theta}, \text{obspts.mat}) \]

**Arguments**

- **mu.q.c**: current value of mu.q.c
- **sig.q.c**: current value of sig.q.c
- **mu.q.bpsi**: current value of mu.q.bpsi
- **sig.q.bpsi**: current value of sig.q.bpsi
- **theta**: current value of theta
- **obspts.mat**: matrix indicating where curves are observed

**Author(s)**

Jeff Goldsmith <ajg2202@cumc.columbia.edu>

---

f_trace  
*Trace computation*

**Description**

Internal function used compute a trace in FPCA-based covariance updates

**Usage**

\[ f_{\text{trace}}(\text{Theta\_i}, \text{Sig\_q\_Bpsi}, \text{Kp}, \text{Kt}) \]

**Arguments**

- **Theta\_i**: basis functions on observed grid points
- **Sig\_q\_Bpsi**: variance of FPC basis coefficients
- **Kp**: number of FPCs
- **Kt**: number of spline basis functions

**Author(s)**

Jeff Goldsmith <ajg2202@cumc.columbia.edu>
**Description**

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

**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**


**References**

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


**See Also**

- fpcr
Usage

gibbs_cs_fpca(
  formula,
  Kt = 5,
  Kp = 2,
  data = NULL,
  verbose = TRUE,
  N.iter = 5000,
  N.burn = 1000,
  SEED = NULL,
  sig2.me = 0.01,
  alpha = 0.1,
  Aw = NULL,
  Bw = NULL,
  Apsi = NULL,
  Bpsi = NULL
)

Arguments

formula a formula indicating the structure of the proposed model.
Kt number of spline basis functions used to estimate coefficient functions
Kp number of FPCA basis functions to be estimated
data an optional data frame, list or environment containing the variables in the model.
If not found in data, the variables are taken from environment(formula), typically the environment from which the function is called.
verbose logical defaulting to TRUE – should updates on progress be printed?
N.iter number of iterations used in the Gibbs sampler
N.burn number of iterations discarded as burn-in
SEED seed value to start the sampler; ensures reproducibility
sig2.me starting value for measurement error variance
alpha tuning parameter balancing second-derivative penalty and zeroth-derivative penalty (alpha = 0 is all second-derivative penalty)
Aw hyperparameter for inverse gamma controlling variance of spline terms for population-level effects
Bw hyperparameter for inverse gamma controlling variance of spline terms for population-level effects
Apsi hyperparameter for inverse gamma controlling variance of spline terms for FPC effects
Bpsi hyperparameter for inverse gamma controlling variance of spline terms for FPC effects

Author(s)

Jeff Goldsmith <ajg2202@columbia.edu>
References
215-236.

---

gibbs_cs_wish  Cross-sectional FoSR using a Gibbs sampler and Wishart prior

Description
Fitting function for function-on-scalar regression for cross-sectional data. This function estimates
model parameters using a Gibbs sampler and estimates the residual covariance surface using a
Wishart prior.

Usage
gibbs_cs_wish(
  formula,  
  Kt = 5, 
  data = NULL, 
  verbose = TRUE, 
  N.iter = 5000, 
  N.burn = 1000, 
  alpha = 0.1, 
  min.iter = 10, 
  max.iter = 50, 
  Aw = NULL, 
  Bw = NULL, 
  v = NULL, 
  SEED = NULL 
)

Arguments
  formula a formula indicating the structure of the proposed model.
  Kt number of spline basis functions used to estimate coefficient functions
  data an optional data frame, list or environment containing the variables in the model.
  If not found in data, the variables are taken from environment(formula), typi-
  cally the environment from which the function is called.
  verbose logical defaulting to TRUE – should updates on progress be printed?
  N.iter number of iterations used in the Gibbs sampler
  N.burn number of iterations discarded as burn-in
  alpha tuning parameter balancing second-derivative penalty and zeroth-derivative penalty
  (alpha = 0 is all second-derivative penalty)
gibbs_mult_fpca

min.iter  minimum number of iterations
max.iter  maximum number of iterations
$A_w$    hyperparameter for inverse gamma controlling variance of spline terms for population-level effects
$B_w$    hyperparameter for inverse gamma controlling variance of spline terms for population-level effects
$\nu$    hyperparameter for inverse Wishart prior on residual covariance
SEED    seed value to start the sampler; ensures reproducibility

Author(s)

Jeff Goldsmith <ajg2202@cumc.columbia.edu>

References


gibbs_mult_fpca  *Multilevel FoSR using a Gibbs sampler and FPCA*

Description

Fitting function for function-on-scalar regression for longitudinal data. This function estimates model parameters using a Gibbs sampler and estimates the residual covariance surface using FPCA.

Usage

gibbs_mult_fpca(
  formula,
  Kt = 5,
  Kp = 2,
  data = NULL,
  verbose = TRUE,
  N.iter = 5000,
  N.burn = 1000,
  sig2.me = 0.01,
  alpha = 0.1,
  SEED = NULL
)
gibbs_mult_wish

**Arguments**

- **formula**: a formula indicating the structure of the proposed model.
- **Kt**: number of spline basis functions used to estimate coefficient functions.
- **Kp**: number of FPCA basis functions to be estimated.
- **data**: an optional data frame, list or environment containing the variables in the model. If not found in data, the variables are taken from environment(formula), typically the environment from which the function is called.
- **verbose**: logical defaulting to TRUE – should updates on progress be printed?
- **N.iter**: number of iterations used in the Gibbs sampler.
- **N.burn**: number of iterations discarded as burn-in.
- **sig2.me**: starting value for measurement error variance.
- **alpha**: tuning parameter balancing second-derivative penalty and zeroth-derivative penalty (alpha = 0 is all second-derivative penalty).
- **SEED**: seed value to start the sampler; ensures reproducibility.

**Author(s)**

Jeff Goldsmith <ajg2202@cumc.columbia.edu>

**References**


---

**gibbs_mult_wish**

**Multilevel FoSR using a Gibbs sampler and Wishart prior**

**Description**

Fitting function for function-on-scalar regression for multilevel data. This function estimates model parameters using a Gibbs sampler and estimates the residual covariance surface using a Wishart prior.

**Usage**

```r
# Example

# gibbs_mult_wish(
  formula,  # a formula indicating the structure of the proposed model.
  Kt = 5,  # number of spline basis functions used to estimate coefficient functions.
  data = NULL,  # an optional data frame, list or environment containing the variables in the model. If not found in data, the variables are taken from environment(formula), typically the environment from which the function is called.
  verbose = TRUE,  # logical defaulting to TRUE – should updates on progress be printed?
  N.iter = 5000,  # number of iterations used in the Gibbs sampler.
  N.burn = 1000,  # number of iterations discarded as burn-in.
  sig2.me = starting value for measurement error variance.
  alpha = 0.1,  # tuning parameter balancing second-derivative penalty and zeroth-derivative penalty (alpha = 0 is all second-derivative penalty).
  SEED = seed value to start the sampler; ensures reproducibility.
)```
Az = NULL,
Bz = NULL,
Aw = NULL,
Bw = NULL,
v = NULL,
SEED = NULL
)

Arguments

formula  
a formula indicating the structure of the proposed model.
Kt     
number of spline basis functions used to estimate coefficient functions
data     
an optional data frame, list or environment containing the variables in the model.
If not found in data, the variables are taken from environment(formula), typically the environment from which the function is called.
verbose     
logical defaulting to TRUE – should updates on progress be printed?
N.iter     
number of iterations used in the Gibbs sampler
N.burn     
number of iterations discarded as burn-in
alpha     
tuning parameter balancing second-derivative penalty and zeroth-derivative penalty
(alpha = 0 is all second-derivative penalty)
Az     
hyperparameter for inverse gamma controlling variance of spline terms for subject-level effects
Bz     
hyperparameter for inverse gamma controlling variance of spline terms for subject-level effects
Aw     
hyperparameter for inverse gamma controlling variance of spline terms for population-level effects
Bw     
hyperparameter for inverse gamma controlling variance of spline terms for population-level effects
v     
hyperparameter for inverse Wishart prior on residual covariance
SEED     
seed value to start the sampler; ensures reproducibility

Author(s)

Jeff Goldsmith <ajg2202@cumc.columbia.edu>

References

**gls_cs**

*Cross-sectional FoSR using GLS*

**Description**

Fitting function for function-on-scalar regression for cross-sectional data. This function estimates model parameters using GLS: first, an OLS estimate of spline coefficients is estimated; second, the residual covariance is estimated using an FPC decomposition of the OLS residual curves; finally, a GLS estimate of spline coefficients is estimated. Although this is in the ‘BayesFoSR’ package, there is nothing Bayesian about this FoSR.

**Usage**

```r
gls_cs(
  formula,
  data = NULL,
  Kt = 5,
  basis = "bs",
  sigma = NULL,
  verbose = TRUE,
  CI.type = "pointwise"
)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>formula</td>
<td>a formula indicating the structure of the proposed model.</td>
</tr>
<tr>
<td>data</td>
<td>an optional data frame, list or environment containing the variables in the model. If not found in data, the variables are taken from environment(formula), typically the environment from which the function is called.</td>
</tr>
<tr>
<td>Kt</td>
<td>number of spline basis functions used to estimate coefficient functions</td>
</tr>
<tr>
<td>basis</td>
<td>basis type; options are &quot;bs&quot; for b-splines and &quot;pbs&quot; for periodic b-splines</td>
</tr>
<tr>
<td>sigma</td>
<td>optional covariance matrix used in GLS; if NULL, OLS will be used to estimated fixed effects, and the covariance matrix will be estimated from the residuals.</td>
</tr>
<tr>
<td>verbose</td>
<td>logical defaulting to TRUE – should updates on progress be printed?</td>
</tr>
<tr>
<td>CI.type</td>
<td>Indicates CI type for coefficient functions; options are &quot;pointwise&quot; and &quot;simultaneous&quot;</td>
</tr>
</tbody>
</table>

**Author(s)**

Jeff Goldsmith <ajg2202@columbia.edu>

**References**

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 pfr, where \( \beta(t) \) is an unknown coefficient function and \( X_i(t) \) is a functional predictor on the closed interval \( T \). See smooth.terms for a list of basis and penalty options; the default is thin-plate regression splines, as this is the default option for s.

Usage

\[
\text{lf}( \\
X, \\
\text{argvals} = \text{NULL}, \\
xind = \text{NULL}, \\
\text{integration} = \text{c("simpson", "trapezoidal", "riemann")}, \\
L = \text{NULL}, \\
presmooth = \text{NULL}, \\
presmooth.opts = \text{NULL}, \\
... \\
)
\]

Arguments

- **X**: functional predictors, typically expressed as an \( N \) by \( J \) matrix, where \( N \) is the number of columns and \( J \) is the number of evaluation points. May include missing/sparse functions, which are indicated by NA values. Alternatively, can be an object of class "fd"; see fd.
- **argvals**: indices of evaluation of \( X \), i.e. \( (t_{i1}, \ldots, t_{iJ}) \) for subject \( i \). May be entered as either a length-\( J \) vector, or as an \( N \) by \( J \) matrix. Indices may be unequally spaced. Entering as a matrix allows for different observations times for each subject. If NULL, defaults to an equally-spaced grid between 0 or 1 (or within \( X \)$basis$rangeval$ if \( X \) is a fd object.)
- **xind**: same as argvals. It will not be supported in the next version of refund.
- **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".
- **L**: an optional \( N \) by \( \text{ncol} \text{(argvals)} \) matrix giving the weights for the numerical integration over \( t \). If present, overrides integration.
- **presmooth**: string indicating the method to be used for preprocessing functional predictor prior to fitting. Options are fpca.sc, fpca.face, fpca.ssvd, fpca.bspline, and fpca.interpolate. Defaults to NULL indicating no preprocessing. See create.prep.func.
- **presmooth.opts**: list including options passed to preprocessing method create.prep.func.
optional arguments for basis and penalization to be passed to mgcv::s. These
could include, for example, "bs", "k", "m", etc. See s for details.

Value

a list with the following entries

call a call to te (or s, t2) using the appropriately constructed covariate and weight
matrices
argvals the argvals 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 argvals 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
prep.func a function that preprocesses data based on the preprocessing method specified
in presmooth. See create.prep.func

Author(s)

Mathew W. McLean <mathew.w.mclean@gmail.com>, Fabian Scheipl, and Jonathan Gellar

References


regression for cognitive outcomes on neuronal tract measurements. *Journal of the Royal Statistical

See Also

pfr, af, mgcv's smooth.terms and linear.functional.terms; pfr for additional examples

Examples

data(DTI)
DTI1 <- DTI[DTI$visit==1 & complete.cases(DTI),]

# We can apply various preprocessing options to the DTI data
fit1 <- pfr(pasat ~ lf(cca, k=30), data=DTI1)
fit2 <- pfr(pasat ~ lf(cca, k=30, presmooth="fpca.sc",
                   presmooth.opts=list(nbasis=8, pve=.975)), data=DTI1)
fit3 <- pfr(pasat ~ lf(cca, k=30, presmooth="fpca.face",
                   presmooth.opts=list(m=3, npc=9)), data=DTI1)
fit4 <- pfr(pasat ~ lf(cca, k=30, presmooth="fpca.ssvd"), data=DTI1)
fit5 <- pfr(pasat ~ lf(cca, k=30, presmooth="bspline",
                   presmooth.opts=list(nbasis=8)), data=DTI1)
```r
fit6 <- pfr(pasat ~ lf(cca, k=30, presmooth="interpolate"), data=DTI1)

# All models should result in similar fits
fits <- as.data.frame(lapply(1:6, function(i)
    get(paste0("fit",i))$fitted.values))
names(fits) <- c("none", "fpca.sc", "fpca.face", "fpca.ssvd", "bspline", "interpolate")
pairs(fits)
```

---

**lf.vd**

*Construct a VDFR regression term*

---

**Description**

This function defines the a variable-domain functional regression term for inclusion in an `gam`-formula (or `bam` or `gamm` or `gamm4::gamm` as constructed by `pfr`). These are functional predictors for which each function is observed over a domain of different width. The default is the term $\frac{1}{T_i} \int_0^{T_i} X_i(t) \beta(t,T_i) dt$, where $X_i(t)$ is a functional predictor of length $T_i$ and $\beta(t,T_i)$ is an unknown bivariate coefficient function. Various domain transformations are available, such as lagging or domain-standardizing the coordinates, or parameterizing the interactions; these often result in improved model fit. Basis choice is fully customizable using the options of `s` and `te`.

**Usage**

```r
lf.vd(
X,
argvals = seq(0, 1, 1 = ncol(X)),
v = NULL,
integration = c("simpson", "trapezoidal", "riemann"),
L = NULL,
basistype = c("s", "te", "t2"),
transform = NULL,
mp = TRUE,
...)
```

**Arguments**

- **X** matrix containing variable-domain functions. Should be $N \times J$, where $N$ is the number of subjects and $J$ is the maximum number of time points per subject. Most rows will have NA values in the right-most columns, corresponding to unobserved time points.
- **argvals** indices of evaluation of X, i.e. $t_{i1}, ..., t_{iJ}$ for subject $i$. May be entered as either a length-J vector, or as an N by J matrix. Indices may be unequally spaced. Entering as a matrix allows for different observations times for each subject.
- **vd** vector of values of containing the variable-domain width ($T_i$ above). Defaults to the argvals value corresponding to the last non-NA element of $X_i(t)$.  

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".

L  an optional N by ncol(argvals) matrix giving the weights for the numerical integration over t. If present, overrides integration.

basistype  character string indicating type of bivariate basis used. Options include "s" (the default), "te", and "t2", which correspond to mgcv::s, mgcv::te, and mgcv::t2.

transform  character string indicating an optional basis transformation; see Details for options.

mp  for transform="linear" or transform="quadratic", TRUE to use multiple penalties for the smooth (one for each marginal basis). If FALSE, penalties are concatenated into a single block-diagonal penalty matrix (with one smoothing parameter).

...  optional arguments for basis and penalization to be passed to the function indicated by basistype. These could include, for example, "bs", "k", "m", etc. See te or s for details.

Details

The variable-domain functional regression model uses the term \( \frac{1}{T_i} \int_0^{T_i} X_i(t) \beta(t, T_i) dt \) to incorporate a functional predictor with subject-specific domain width. This term imposes a smooth (non-parametric) interaction between t and T_i. The domain of the coefficient function is the triangular (or trapezoidal) surface defined by t, T_i : 0 ≤ t ≤ T_i. The default basis uses bivariate thin-plate regression splines.

Different basis transformations can result in different properties; see Gellar, et al. (2014) for a more complete description. We make five basis transformations easily accessible using the transform argument. This argument is a character string that can take one of the following values:

1. "lagged": transforms argvals to argvals -vd
2. "standardized": transforms argvals to argvals/vd, and then rescales vd linearly so it ranges from 0 to 1
3. "linear": first transforms the domain as in "standardized", then parameterizes the interaction with "vd" to be linear
4. "quadratic": first transforms the domain as in "standardized", then parameterizes the interaction with "vd" to be quadratic
5. "noInteraction": first transforms the domain as in "standardized", then reduces the bivariate basis to univariate with no effect of vd. This would be equivalent to using lf on the domain-standardized predictor functions.

The practical effect of using the "lagged" basis is to increase smoothness along the right (diagonal) edge of the resultant estimate. The practical effect of using a "standardized" basis is to allow for greater smoothness at high values of T_i compared to lower values.

These basis transformations rely on the basis constructors available in the mgcvTrans package. For more specific control over the transformations, you can use bs="dt" and/or bs="pi"; see
smooth.construct.dt.smooth.spec or smooth.construct.pi.smooth.spec for an explanation of the options (entered through the xt argument of lf.vd/s).

Note that tensor product bases are only recommended when a standardized transformation is used. Without this transformation, just under half of the "knots" used to define the basis will fall outside the range of the data and have no data available to estimate them. The penalty allows the corresponding coefficients to be estimated, but results may be unstable.

Value

da list with the following entries

call
data
L
argname
Tindname
LXname

Author(s)

Jonathan E. Gellar <JGellar@mathematica-mpr.com>

References


See Also

pfr, lf, mgcv's linear.functional.terms.

Examples

```r
## Not run:
data(sofa)
fit.vd1 <- pfr(death ~ lf.vd(SOFA) + age + los,
    family="binomial", data=sofa)
fit.vd2 <- pfr(death ~ lf.vd(SOFA, transform="lagged") + age + los,
    family="binomial", data=sofa)
fit.vd3 <- pfr(death ~ lf.vd(SOFA, transform="standardized") + age + los,
    family="binomial", data=sofa)
fit.vd4 <- pfr(death ~ lf.vd(SOFA, transform="standardized",
    basistype="te") + age + los,
    family="binomial", data=sofa)
fit.vd5 <- pfr(death ~ lf.vd(SOFA, transform="linear", bs="ps") + age + los,
    family="binomial", data=sofa)
fit.vd6 <- pfr(death ~ lf.vd(SOFA, transform="quadratic", bs="ps") + age + los,
```
lf_old

Construct an FLM regression term

Description

Defines a term $\int_T \beta(t)X_i(t)dt$ for inclusion in an \texttt{gam}-formula (or \texttt{bam} or \texttt{gamm} or \texttt{gamm4}) as constructed by \texttt{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

\begin{verbatim}
lf_old(X,
  argvals = seq(0, 1, l = ncol(X)),
  xind = NULL,
  integration = c("simpson", "trapezoidal", "riemann"),
  L = NULL,
)
\end{verbatim}
splinepars = list(bs = "ps", k = min(ceiling(n/4), 40), m = c(2, 2)),
    presmooth = TRUE)
)

Arguments

X an N by J=ncol(argvals) matrix of function evaluations \(X_i(t_{i1}), \ldots, X_i(t_{iJ}); j = 1, \ldots, N\).

argvals matrix (or vector) of indices of evaluations of \(X_i(t)\); i.e. a matrix with ith row \((t_{i1}, \ldots, t_{iJ})\).

xind same as argvals. It will not be supported in the next version of refund.

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 \(L\) is specified

\(L\) an optional N by ncol(argvals) 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. \(\text{list(bs="ps",m=c(2,1))}\) See \(\text{te}\) or \(\text{s}\) for details

presmooth logical; if true, the functional predictor is pre-smoothed prior to fitting. See \text{smooth.basisPar}

Value

a list with the following entries

1. call - a call to \(\text{te}\) (or \(\text{s}\), \(\text{t2}\)) using the appropriately constructed covariate and weight matrices
2. argvals - the \(\text{argvals}\) argument supplied to \(\text{lf}\)
3. \(L\) - the matrix of weights used for the integration
4. xindname - the name used for the functional predictor variable in the formula used by \text{mgcv}
5. tindname - the name used for \(\text{argvals}\) variable in the formula used by \text{mgcv}
6. \(LX\)name - the name used for the \(L\) variable in the formula used by \text{mgcv}
7. presmooth - the \(\text{presmooth}\) argument supplied to \(\text{lf}\)
8. \(Xfd\) - an \text{fd} object from presmoothing the functional predictors using \text{smooth.basisPar}. Only present if \(\text{presmooth}=\text{TRUE}\). See \text{fd}

Author(s)

Mathew W. McLean <mathew.w.mclean@gmail.com> and Fabian Scheipl

See Also

\(\text{fgam}, \text{af}, \text{mgcv's linear.functional.terms}, \text{fgam}\) for examples
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

```r
lpeer(
  Y,
  subj,
  t,
  funcs,
  argvals = NULL,
  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
- `funcs` matrix containing observed functional predictors as rows. Rows with NA and Inf values will be deleted.
- `argvals` matrix (or vector) of indices of evaluations of $X_i(t)$; i.e. a matrix with ith row $(t_{i1}, \ldots, t_{iJ})$
- `covariates` matrix of scalar covariates.
- `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".
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'.

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 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'.

scalar value of a in decomposition based penalty. Needs to be specified with pentype='DECOMP'.

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:

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

where \( \epsilon_{i(t)} \sim N(0, \sigma^2) \) and \( u_i \sim N(0, \sigma_a^2) \). 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 \texttt{pentype} = 'DECOMP' and the associated Q matrix must be passed through the Q argument. Alternatively, one can directly specify the penalty matrix by setting \texttt{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 \texttt{comm.pen} should have value \texttt{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 \texttt{comm.pen} should have value \texttt{FALSE} and in that case specified matrix to argument Q (or L) should have K(d+1) columns. Here first K columns pertain to first component function, second K columns pertain 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 \texttt{comm.pen} is always \texttt{TRUE} and \texttt{comm.pen=FALSE} will be ignored.

\textbf{Value}

A list containing:

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

\textbf{Author(s)}

Madan Gopal Kundu <mgkundu@iupui.edu>
References


See Also

peer, plot.lpeer

Examples

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

#Load Data
data(DTI)

## Extract values for arguments for lpeer() from given data
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
## \( \gamma(t,s) = \gamma_0(s) + t\gamma_1(s) \)

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
## \( \gamma(t,s) = \gamma_0(s) + t\gamma_1(s) + t^2\gamma_1(s) \)

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
## \( \gamma(t,s) = \gamma_0(s) + t\gamma_1(s) \)

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
## \( \gamma(t,s) = \gamma_0(s) + t\gamma_1(s) \)

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., 2012) 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
lpfr

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


Examples

```r
## Not run:
# use longitudinal data to regress continuous outcomes on  
# functional predictors (continuous outcomes only recorded for  
# case == 1)

# subset data as needed for this example
data(DTI)
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), 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), 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)

### mfpca.sc

**Multilevel 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**

```r
mfpca.sc(
  Y = NULL,
  id = NULL,
  visit = NULL,
  twoway = FALSE,
  argvals = NULL,
  nbasis = 10,
  pve = 0.99,
  npc = NULL,
  makePD = FALSE,
  center = TRUE,
  cov.est.method = 2,
  integration = "trapezoidal"
)
```

**Arguments**

- `Y`: The user must supply a matrix of functions on a regular grid
- `id`: Must be supplied, a vector containing the id information used to identify clusters
- `visit`: A vector containing information used to identify visits. Defaults to `NULL`.
twoway logical, indicating whether to carry out twoway ANOVA and calculate visit-specific means. Defaults to FALSE.

argvals function argument.

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.

rpc prespecified value for the number of principal components (if given, this overrides pve).

makePD logical: should positive definiteness be enforced for the covariance surface estimate? Defaults to FALSE. Only FALSE is currently supported.

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. 2 is currently supported.

integration quadrature method for numerical integration; only "trapezoidal" is currently supported.

Details

This function computes a multilevel 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 level 1 and level 2 scores.

MFPCA was proposed in Di et al. (2009), with variations for MFPCA with sparse data in Di et al. (2014). mfpca.sc uses penalized splines to smooth the covariance functions, as Described in Di et al. (2009) and Goldsmith et al. (2013).

Value

An object of class mfpca containing:

Yhat FPC approximation (projection onto leading components) of $Y$, estimated curves for all subjects and visits

Yhat.subject estimated subject specific curves for all subjects

Y the observed data

scores $n \times \text{rpc}$ matrix of estimated FPC scores for level1 and level2.

mu estimated mean function (or a vector of zeroes if center==FALSE).

efunctions $d \times \text{rpc}$ matrix of estimated eigenfunctions of the functional covariance, i.e., the FPC basis functions for levels 1 and 2.

evalues estimated eigenvalues of the covariance operator, i.e., variances of FPC scores for levels 1 and 2.
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.

Author(s)

Julia Wrobel <jw3134@cumc.columbia.edu>, Jeff Goldsmith <jeff.goldsmith@columbia.edu>, and Chongzhi Di

References


Examples

## Not run:
data(DTI)
DTI = subset(DTI, Nscans < 6) ## example where all subjects have 6 or fewer visits
id = DTI$ID
Y = DTI$cca
mf pca.DTI = mf pca.sc(Y=Y, id = id, twoway = TRUE)

## End(Not run)
ols_cs

Value

A model matrix

Author(s)

Fabian Scheipl

ols_cs Cross-sectional FoSR using GLS

Description

Fitting function for function-on-scalar regression for cross-sectional data. This function estimates model parameters using GLS: first, an OLS estimate of spline coefficients is estimated; second, the residual covariance is estimated using an FPC decomposition of the OLS residual curves; finally, a GLS estimate of spline coefficients is estimated. Although this is in the ‘BayesFoSR’ package, there is nothing Bayesian about this FoSR.

Usage

ols_cs(formula, data = NULL, Kt = 5, basis = "bs", verbose = TRUE)

Arguments

formula a formula indicating the structure of the proposed model.
data an optional data frame, list or environment containing the variables in the model. If not found in data, the variables are taken from environment(formula), typically the environment from which the function is called.
Kt number of spline basis functions used to estimate coefficient functions
basis basis type; options are "bs" for b-splines and "pbs" for periodic b-splines
verbose logical defaulting to TRUE – should updates on progress be printed?

Author(s)

Jeff Goldsmith <ajg2202@cumc.columbia.edu>

References

pco_predict_preprocess

*Make predictions using pco basis terms*

Description

This function performs the necessary preprocessing for making predictions with `gam` models that include `pco` basis terms. The function `pco_predict_preprocess` builds a `data.frame` (or augments an existing one) to be used with the usual `predict` function.

Usage

```r
pco_predict_preprocess(model, newdata = NULL, dist_list)
```

Arguments

- **model**: a fitted `gam` model with at least one term of class "pco.smooth".
- **newdata**: data frame including the new values for any non-`pco` terms in the original fit. If there were none, this can be left as `NULL`.
- **dist_list**: a list of $n \times n^*$ matrices, one per `pco` term in the model, giving the distances from the $n^*$ prediction points to the $n$ design points (original observations). List entry names should correspond to the names of the terms in the model (e.g., if the model includes a `s(x)` term, `dist_list` must include an element named "x").

Details

Models with `pco` basis terms are fitted by inputting distances among the observations and then regressing (with a ridge penalty) on leading principal coordinates arising from these distances. To perform prediction, we must input the distances from the new data points to the original points, and then "insert" the former into the principal coordinate space by the interpolation method of Gower (1968) (see also Miller, 2012).

An example of how to use this function in practice is shown in `smooth.construct.pco.smooth.spec`.

Value

A `data.frame` with the coordinates for the new data inserted into principal coordinate space, in addition to the supplied `newdata` if this was non-`NULL`. This can be used as the `newdata` argument in a call to `predict.gam`.

Author(s)

David L Miller
References


See Also

smooth.construct.pco.smooth.spec

Description

pffr-construct 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 \( \lambda_m \) 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 \( \sigma^2 \) 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 \( \lambda_{\text{m}} \). Can be used to model smooth residuals if \( id \) is simply an index of observations. Differing from random effects in mgcv, these effects are estimated under a "sum-to-zero-for-each-\( t^\)"-constraint.

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

```r
## 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)
```

Construct a PEER regression term in a pfr formula
Description

Defines a term $\int_0^T \beta(t) X_i(t) dt$ for inclusion in a pfr formula, where $\beta(t)$ is estimated with structured penalties (Randolph et al., 2012).

Usage

```r
peer(
  x,
  argvals = NULL,
  ptype = "RIDGE",
  Q = NULL,
  phia = 10^3,
  L = NULL,
  ...
)
```

Arguments

- **x**: functional predictors, typically expressed as an N by J matrix, where N is the number of columns and J is the number of evaluation points. May include missing/sparse functions, which are indicated by NA values. Alternatively, can be an object of class "fd"; see `fd`.
- **argvals**: indices of evaluation of X, i.e. $(t_{i1}, \ldots, t_{iJ})$ for subject i. May be entered as either a length-J vector, or as an N by J matrix. Indices may be unequally spaced. Entering as a matrix allows for different observations times for each subject. If NULL, defaults to an equally-spaced grid between 0 or 1 (or within X$\text{basis}$rangeval if X is a fd object.)
- **ptype**: the type of penalty to apply, one of "RIDGE", "D", "DECOMP", or "USER"; see Details.
- **Q**: matrix Q used for ptype="DECOMP"; see Details.
- **phia**: scalar $a$ used for ptype="DECOMP"; see Details.
- **L**: user-supplied penalty matrix for ptype="USER"; see Details.
- **...**: additional arguments to be passed to lf (and then possibly s). Arguments processed by lf include, for example, integration for specifying the method of numerical integration. Arguments processed by s include information related to basis and penalization, such as m for specifying the order of the difference penalty; See Details. xt-argument is not allowed for peer-terms and will cause an error.

Details

`peer` is a wrapper for lf, which defines linear functional predictors for any type of basis. It simply calls lf with the appropriate options for the peer basis and penalty construction. The type of penalty is determined by the ptype argument. There are four types of penalties available:

1. ptype="RIDGE" for a ridge penalty, the default
2. pentype=="D" for a difference penalty. The order of the difference penalty may be specified by supplying an \( m \) argument (default is 2).

3. pentype=="DECOMP" for a decomposition-based penalty, \( bPQ + a(I - PQ) \), where \( PQ = Q't(QQ't)^{-1}Q \). The \( Q \) matrix must be specified by \( Q \), and the scalar \( a \) by \( \text{phia} \). The number of columns of \( Q \) must be equal to the length of the data. Each row represents a basis function where the functional predictor is expected to lie, according to prior belief.

4. pentype=="USER" for a user-specified penalty matrix, supplied by the \( L \) argument.

The original stand-alone implementation by Madan Gopal Kundu is available in peer_old.

Author(s)

Jonathan Gellar <JGellar@mathematica-mpr.com> and Madan Gopal Kundu <mgkundu@iupui.edu>

References


See Also

pfr, smooth.construct.peer.smooth.spec

Examples

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

data(DTI)
DTI = DTI[which(DTI$case == 1),]
fit.D2 = pfr(pasat ~ peer(cca, pentype="D"), data=DTI)
plot(fit.D2)

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

data(PEER.Sim)
data(Q)
PEER.Sim1 <- subset(PEER.Sim, t==0)

# Setting k to max possible value
fit.decomp <- pfr(Y ~ peer(W, pentype="Decomp", Q=Q, k=99), data=PEER.Sim1)
plot(fit.decomp)
```
### PEER.Sim

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 time-points. 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.

#### 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).

#### Details

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).

#### 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>)

### peer_old

Functional Models with Structured Penalties

#### Description

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

peer_old(
  Y,
  funcs,
  argvals = NULL,
  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.
argvals matrix (or vector) of indices of evaluations of \( X_i(t) \); i.e. a matrix with \( i \)th row \((t_{i1}, .., t_{iJ})\)
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_old() fits the following model:
\[ y_i = \int W_i(s)\gamma(s)ds + \epsilon_i \]

where \( \epsilon_i \sim N(0, \sigma^2) \). 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 \( \text{pentype} = \text{DECOMP} \) 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 \( \text{pentype} = \text{USER} \) and associated \( L \) matrix need to be passed through \( L \) argument.

Default penalty is Ridge penalty and user needs to specify \( \text{RIDGE} \). 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


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, ptype="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)
W<- PEER.Sim1$W
Y<- PEER.Sim1$Y

#Load Q matrix containing structural information
data(Q)

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

## End(Not run)

pffr  
Penalized flexible functional 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

\[ E(Y(t)) = g(\mu(t)) + \int X_i(s) \beta(s,t) ds + f(z_{1i}, t) + f(z_{2i}) + z_{3i}\beta_3(t) + \ldots \]

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

```r
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>)`.
- `data` an (optional) `data.frame` containing the data. Can also be a named list for regular data. Functional covariates have to be supplied as `<no. of observations> by <no. of evaluations>` matrices, i.e. each row is one functional observation.
- `ydata` an (optional) `data.frame` supplying functional responses that are not observed on a regular grid. See Details.
- `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', 'gamm4' and 'jagam' are valid options as well. See Details for 'gamm4' and 'jagam'.
- `method` Defaults to "REML"-estimation, including of unknown scale. If `algorithm="bam"`, the default is switched to "fREML". See `gam` and `bam` for details.
- `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.
- `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.

... additional arguments that are valid for `gam`, `bam`, 'gamm4' or 'jagam'. subset is not 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. Available methods/functions to postprocess fitted models: `summary.pffr`, `plot.pffr`, `coef.pffr`, `fitted.pffr`, `residuals.pffr`, `predict.pffr`, `model.matrix.pffr`, `qq.pffr`, `pffr.check`. If algorithm is "jagam", only the location of the model file and the usual `jagam`-object are returned, you have to run the sampler yourself.

Details

The routine can estimate

1. linear functional effects of scalar (numeric or factor) covariates that vary smoothly over \(t\) (e.g. \(z_1 \beta_1(t)\), 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. \(f(z_2,t)\), specified in the formula simply as \(~s(z2)\))
3. (nonlinear) effects of scalar covariates that are constant over \(t\) (e.g. \(f(z_3)\), specified as \(~c(s(z3))\)), or \(\beta_3 z_3\), 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 functional response is specified via the `global` argument `bs.yindex`. If necessary, this can be overridden 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 over `x` and the index of the response in 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 an `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 (i.e., \( \sum_i \hat{f}(z_i, x_i, t) = 0 \) or \( \sum_i \hat{f}(x_i, t) = 0 \)) 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 \( \sum_i \hat{X}_i(t) = 0 \) for all `t` in `ff`-terms so that the global functional intercept can be interpreted as the global mean function.

The `family`-argument can be used to specify all of the response distributions and link functions described in `family.mgcv`. Note that `family = "gaulss"` is treated in a special way: Users can supply the formula for the variance by supplying a special argument `varformula`, but this is not modified in the way that the `formula`-argument is but handed over to the fitter directly, so this is for expert use only. If `varformula` is not given, `pffr` will use the parameters from argument `bs.int` to define a spline basis along the index of the response, i.e., a smooth variance function over `$t$` for responses `$Y(t)$`.

Author(s)

Fabian Scheipl, Sonja Greven
References


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)

# multivariate model:
# E(Y(t)) = \beta_0(t) + \int X1(s)\beta_1(s,t)ds + xlin \beta_3(t) +
# f_1(xte1, xte2) + f_2(xsmoo, t) + \beta_4 xconst
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
          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$pterms
str(cm2$pterms, 2)
```
```r
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)
```

### Description

Some diagnostics for a fitted `pffr` model

### Usage

```r
pffr.check(
b, 
old.style = FALSE,
type = c("deviance", "pearson", "response"),
k.sample = 5000,
k.rep = 200,
rep = 0,
level = 0.9,
rl.col = 2,
rep.col = "gray80",
... 
)
```

### Arguments

- **b**: a fitted `pffr`-object
- **old.style**: If you want old fashioned plots, exactly as in Wood, 2006, set to TRUE.
- **type**: type of residuals, see `residuals.gam`, used in all plots.
- **k.sample**: Above this k testing uses a random sub-sample of data.
- **k.rep**: how many re-shuffles to do to get p-value for k testing.
- **rep**: passed to `qq.gam` when `old.style` is FALSE.
- **level**: passed to `qq.gam` when `old.style` is FALSE.
- **rl.col**: passed to `qq.gam` when `old.style` is FALSE.
- **rep.col**: passed to `qq.gam` when `old.style` is FALSE.
- **...**: extra graphics parameters to pass to plotting functions.
**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`.
- `hatSigma` (an estimate of) the within-observation covariance (along the responses’ index), evaluated at `yind`. See Details.
- `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.
- `method` See `pffr`.
- `tensortype` See `pffr`.
- `bs.yindex` See `pffr`.
- `bs.int` 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

Description

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

\[ Y_i(t) = \mu(t) + \int X_{1i}(s)\beta_1(s,t)ds + xlin\beta_3(t) + f(xte1, xte2) + f(xsmoo, t) + \beta_4xconst + f(xfactor, t) + \epsilon_i(t) \]

Scenarios "int", "ff", "lin", "te", "smoo", "const", "factor", generate data from simpler models containing only the respective term(s) in the model equation given above. Specifying 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

Description

Implements various approaches to penalized scalar-on-function regression. These techniques include Penalized Functional Regression (Goldsmith et al., 2011), Longitudinal Penalized Functional Regression (Goldsmith, et al., 2012), Functional Principal Component Regression (Reiss and Ogden, 2007), Partially Empirical Eigenvectors for Regression (Randolph et al., 2012), Functional Generalized Additive Models (McLean et al., 2013), and Variable-Domain Functional Regression (Gellar et al., 2014). This function is a wrapper for mgcv’s **gam** and its siblings to fit models with a scalar (but not necessarily continuous) response.
Usage

pfr(formula = NULL, fitter = NA, method = "REML", ...)

Arguments

formula a formula that could contain any of the following special terms: 1f(), af(), 1f.vd(), peer(), fpc(), or re(); also mgcv’s s(), te(), or t2.
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.
method The smoothing parameter estimation method. Default is "REML". For options, see gam.
... additional arguments that are valid for gam or bam. These include data and family to specify the input data and outcome family, as well as many options to control the estimation.

Value

A fitted pfr-object, which is a gam-object with some additional information in a $pfr-element. 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)

Jonathan Gellar <JGellar@mathematica-mpr.com>, Mathew W. McLean, Jeff Goldsmith, and Fabian Scheipl

References

See Also

af, lf, lf.vd, fpc, peer, re.

Examples

# See lf(), lf.vd(), af(), fpc(), and peer() for additional examples
data(DTI)
DTI1 <- DTI[DTI$visit==1 & complete.cases(DTI),]
par(mfrow=c(1,2))

# Fit model with linear functional term for CCA
fit.lf <- pfr(pasat ~ lf(cca, k=30, bs="ps"), data=DTI1)
plot(fit.lf, ylab=expression(paste(beta(t))), xlab="t")
## Not run:
# Alternative way to plot
bhat.lf <- coef(fit.lf, n=101)
bhat.lf$upper <- bhat.lf$value + 1.96*bhat.lf$se
bhat.lf$lower <- bhat.lf$value - 1.96*bhat.lf$se
matplot(bhat.lf$cca.argvals, bhat.lf[,c("value", "upper", "lower")],
        type="l", lty=c(1,2,2), col=1,
ylab=expression(paste(beta(t))), xlab="t")

# Fit model with additive functional term for CCA, using tensor product basis
fit.af <- pfr(pasat ~ af(cca, Qtransform=TRUE, k=c(7,7)), data=DTI1)
plot(fit.af, scheme=2, xlab="t", ylab="cca(t)", main="Tensor Product")
plot(fit.af, scheme=2, Qtransform=TRUE,
        xlab="t", ylab="cca(t)", main="Tensor Product")

# Change basistype to thin-plate regression splines
fit.af.s <- pfr(pasat ~ af(cca, basistype="s", Qtransform=TRUE, k=50),
data(DTI1))
plot(fit.af.s, scheme=2, xlab="t", ylab="cca(t)", main="TPRS", rug=FALSE)
plot(fit.af.s, scheme=2, Qtransform=TRUE,
        xlab="t", ylab="cca(t)", main="TPRS", rug=FALSE)

# Visualize bivariate function at various values of x
par(mfrow=c(2,2))
vis.pfr(fit.af, xval=.2)
vis.pfr(fit.af, xval=.4)
vis.pfr(fit.af, xval=.6)
vis.pfr(fit.af, xval=.8)

# Include random intercept for subject
DTI.re <- DTI[complete.cases(DTI$cca),]
DTI.re$ID <- factor(DTI.re$ID)
fit.re <- pfr(pasat ~ lf(cca, k=30) + re(ID), data=DTI.re)
coef.re <- coef(fit.re)
par(mfrow=c(1,2))
plot(fit.re)

# FPCR_R Model
**pfr_old**

Penalized Functional Regression (old version)

**Description**

This code implements the function pfr() available in refund 0.1-11. It is included to maintain backwards compatibility.

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.

**Usage**

```r
pfr_old(
  Y,
  subj = NULL,
  covariates = NULL,
  funcs,
  kz = 10,
  kb = 30,
  nbasis = 10,
  family = "gaussian",
  method = "REML",
  smooth.option = "f pca.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.

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 principal component scores

J  transpose of psi matrix multiplied by phi
CJ 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 mgev::gam
funcs as specified
covariates as specified
smooth.option as specified

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

Author(s)
Bruce Swihart <bruce.swihart@gmail.com> and Jeff Goldsmith <jeff.goldsmith@columbia.edu>

References

See Also
rlrt.pfr, predict.pfr.
Examples

```r
## 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")

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

---

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,
    xlab = "",
    ylab = "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.

Description

Given a "fosr.vs" object, produces a figure of estimated coefficient functions.

Usage

```r
## S3 method for class 'fosr.vs'
plot(x, ...)
```

Arguments

- **x**: an object of class "fosr.vs".
- **...**: additional arguments.

Value

a figure of estimated coefficient functions.
## Examples

```r
## Not run:
I = 100
p = 20
D = 50
grid = seq(0, 1, length = D)

beta.true = matrix(0, p, D)
beta.true[1,] = sin(2*grid*pi)
beta.true[2,] = cos(2*grid*pi)
beta.true[3,] = 2

psi.true = matrix(NA, 2, D)
psi.true[1,] = sin(4*grid*pi)
psi.true[2,] = cos(4*grid*pi)
lambda = c(3, 1)
set.seed(100)
X = matrix(rnorm(I*p), I, p)
C = cbind(rnorm(I, mean = 0, sd = lambda[1]), rnorm(I, mean = 0, sd = lambda[2]))

fixef = X%*%beta.true
pcaef = C %*% psi.true
error = matrix(rnorm(I*D), I, D)

Yi.true = fixef
Yi.pca = fixef + pcaef
Yi.obs = fixef + pcaef + error
data = as.data.frame(X)
data$Y = Yi.obs
fit.mcp = fosr.vs(Y~., data = data[1:80,], method="grMCP")
plot(fit.mcp)

## End(Not run)
```
Description

Inputs an object created by `fpcr`, and plots the estimated coefficient function.

Usage

```r
## 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

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>.)


See Also

peer, lpeer, plot.peer
Examples

```r
## Not run:
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)
## End(Not run)
```

Description

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

Usage

```r
## 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>.)


See Also
peer, lpeer, plot.lpeer

Examples

```r
# See example in peer()
```

---

### `plot.pffr`

*Plot a pffr fit*

#### Description
Plot a fitted pffr-object. Simply dispatches to `plot.gam`.

#### Usage
```r
## 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.pfr

Plot a pfr object

Description

This function plots the smooth coefficients of a pfr object. These include functional coefficients as well as any smooths of scalar covariates. The function dispatches to pfr_plot.gam, which is our local copy of plot.gam with some minor changes.

Usage

```r
## S3 method for class 'pfr'
plot(x, Qtransform = FALSE, ...)
```

Arguments

- `x` a fitted pfr-object
- `Qtransform` For additive functional terms, TRUE indicates the coefficient should be plotted on the quantile-transformed scale, whereas FALSE indicates the scale of the original data. Note this is different from the Qtransform argument of af, which specifies the scale on which the term is fit.
- `...` arguments handed over to plot.gam

Value

This function’s main purpose is its side effect of generating plots. It also silently returns a list of the data used to produce the plots, which can be used to generate customized plots.

Author(s)

Jonathan Gellar

See Also

af, pfr
Description

Produces predictions given a fbps object and new data

Usage

## S3 method for class 'fbps'
predict(object, newdata, ...)

Arguments

- object: an object returned by fbps
- newdata: a data frame or list consisting of x and z values for which predicted values are desired. vectors of x and z need to be of the same length.
- ...: additional arguments.

Value

A list with components

- x: a vector of x given in newdata
- z: a vector of z given in newdata
- fitted.values: a vector of fitted values corresponding to x and z given in newdata

Author(s)

Luo Xiao <lxiao@jhsph.edu>

References


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)))
sigy <- .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), theta=45, r=4, col="blue", main="True surface")
persp(x,z,est$Yhat,zlab="f(x,z)",zlim=c(-1,2.5), theta=45, r=4, col="red", main="Estimated surface")

########################################################################
#### prediction ####
########################################################################
# 1. make prediction with predict.fbps() for all pairs of x and z given in the original data # (it's expected to have same results as Yhat obtained using fbps() above)
newdata <- list(x=rep(x, each=length(z)), z = rep(z, each=length(x)))
pred1 <- predict(est, newdata=newdata)$fitted.values
pred1.mat <- matrix(pred1, nrow=length(x))
par(mfrow=c(1,2))
image(pred1.mat); image(est$Yhat)
all.equal(as.numeric(pred1.mat), as.numeric(est$Yhat))

# 2. predict for pairs of first 10 x values and first 5 z values
newdata <- list(x=rep(x[1:10], 5), z = rep(z[1:5], each=10))
pred2 <- predict(est, newdata=newdata)$fitted.values
pred2.mat <- matrix(pred2, nrow=10)
par(mfrow=c(1,2))
image(pred2.mat); image(est$Yhat[1:10,1:5])
all.equal(as.numeric(pred2.mat), as.numeric(est$Yhat[1:10,1:5]))

# 3. predict for one pair
newdata <- list(x=x[5], z=z[3])
pred3 <- predict(est, newdata=newdata)$fitted.values
all.equal(as.numeric(pred3), as.numeric(est$Yhat[5,3]))

---

**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**

```r
## 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` character; see `predict.gam` for details
- `se.fit` logical; see `predict.gam` for details
- `terms` character see `predict.gam` for details
- `PredOutOfRange` logical; 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,
predict.fosr

Prediction from a fitted bayes_fosr model

Description

Takes a fitted fosr-object produced by bayes_fosr and produces predictions given a new set of values for the model covariates or the original values used for the model fit.

Usage

## S3 method for class 'fosr'
predict(object, newdata, ...)
predict.fosr

Arguments

object a fitted fosr object as produced by bayes_fosr
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 the model fitting function.
... additional (unused) arguments

Value

...

Author(s)

Jeff Goldsmith <jeff.goldsmith@columbia.edu>

See Also

bayes_fosr

Examples

## Not run:
library(reshape2)
library(dplyr)
library(ggplot2)

##### Cross-sectional real-data example #####

## organize data
data(DTI)
DTI = subset(DTI, select = c(cca, case, pasat))
DTI = DTI[complete.cases(DTI),]
DTI$gender = factor(sample(c("male","female"), dim(DTI)[1], replace = TRUE))
DTI$status = factor(sample(c("RRMS", "SPMS", "PPMS"), dim(DTI)[1], replace = TRUE))

## fit models
VB = bayes_fosr(cca ~ pasat, data = DTI, Kp = 4, Kt = 10)

## obtain predictions
pred = predict(VB, sample_n(DTI, 10))

## End(Not run)
**predict.fosr.vs**  
*Prediction for Function-on Scalar Regression with variable selection*

**Description**

Given a "fosr.vs" object and new data, produces fitted values.

**Usage**

```r
## S3 method for class 'fosr.vs'
predict(object, newdata = NULL, ...)
```

**Arguments**

- `object`: an object of class "fosr.vs".
- `newdata`: a data frame that contains the values of the model covariates at which predictors are required.
- `...`: additional arguments.

**Value**

fitted values.

**Author(s)**

Yakuan Chen <yc2641@cumc.columbia.edu>

**See Also**

- `fosr.vs`

**Examples**

```r
## Not run:
I = 100
p = 20
D = 50
g = seq(0, 1, length = D)

beta.true = matrix(0, p, D)
beta.true[1,] = sin(2*g*pi)
beta.true[2,] = cos(2*g*pi)
beta.true[3,] = 2

psi.true = matrix(NA, 2, D)
psi.true[1,] = sin(4*g*pi)
psi.true[2,] = cos(4*g*pi)
lambda = c(31)
```
set.seed(100)

X = matrix(rnorm(I*p), I, p)
C = cbind(rnorm(I, mean = 0, sd = lambda[1]), rnorm(I, mean = 0, sd = lambda[2]))

fixef = X%*%beta.true
pcaef = C %*% psi.true
error = matrix(rnorm(I*D), I, D)

Yi.true = fixef
Yi.pca = fixef + pcaef
Yi.obs = fixef + pcaef + error

data = as.data.frame(X)
data$Y = Yi.obs
fit.mcp = fosr.vs(Y~., data = data[1:80,], method="grMCP")
predicted.value = predict(fit.mcp, data[81:100,])

## End(Not run)

---

**Predict.matrix.dt.smooth**

*Predict.matrix method for dt basis*

**Description**

Predict.matrix method for dt basis

**Usage**

```r
## S3 method for class 'dt.smooth'
Predict.matrix(object, data)
```

**Arguments**

- `object`: a `dt.smooth` object created by `smooth.construct.dt.smooth.spec`, see `smooth.construct`
- `data`: see `smooth.construct`

**Value**

design matrix for domain-transformed terms

**Author(s)**

Jonathan Gellar
Predict.matrix.fpc.smooth

mgcv-style constructor for prediction of FPC terms

Description

mgcv-style constructor for prediction of FPC terms

Usage

## S3 method for class 'fpc.smooth'
Predict.matrix(object, data)

Arguments

object a fpc.smooth object created by smooth.construct.fpc.smooth.spec, see smooth.construct
data see smooth.construct

Value

design matrix for FPC terms

Author(s)

Jonathan Gellar

Predict.matrix.pcre.random.effect

mgcv-style constructor for prediction of PC-basis functional random effects

Description

mgcv-style constructor for prediction of PC-basis functional random effects

Usage

## S3 method for class 'pcre.random.effect'
Predict.matrix(object, data)

Arguments

object a smooth specification object, see smooth.construct
data see smooth.construct
**Predict.matrix.peer.smooth**

*mgcv-style constructor for prediction of PEER terms*

**Value**

design matrix for PC-based functional random effects

**Author(s)**

Fabian Scheipl; adapted from 'Predict.matrix.random.effect' by S.N. Wood.

**Description**

mgcv-style constructor for prediction of PEER terms

**Usage**

```r
## S3 method for class 'peer.smooth'
Predict.matrix(object, data)
```

**Arguments**

- `object` a peer.smooth object created by `smooth.construct.peer.smooth.spec`, see `smooth.construct`
- `data` see `smooth.construct`

**Value**

design matrix for PEER terms

**Author(s)**

Jonathan Gellar
Predict.matrix.pi.smooth

Predict.matrix method for pi basis

Description

Predict.matrix method for pi basis

Usage

## S3 method for class 'pi.smooth'
Predict.matrix(object, data)

Arguments

object a pi.smooth object created by smooth.construct.pi.smooth.spec, see smooth.construct
data see smooth.construct

Value

design matrix for PEER terms

Author(s)

Jonathan Gellar

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
newdata
reformat
type
se.fit
...  

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 \texttt{pfr}'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 \texttt{object$pfr$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

\texttt{predict.gam()}
predict.pfr Prediction from a fitted pfr model

**Description**

Takes a fitted pfr-object produced by pfr 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**

```r
## S3 method for class 'pfr'
predict(
  object,
  newdata,
  type = "response",
  se.fit = FALSE,
  terms = NULL,
  PredOutOfRange = FALSE,
  ...
)
```

**Arguments**

- `object` a fitted pfr object as produced by pfr
- `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 pfr, 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` character; see predict.gam for details
- `se.fit` logical; see predict.gam for details
- `terms` character see predict.gam for details
- `PredOutOfRange` logical; if this argument is true then any functional predictor values in newdata corresponding to pfr 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 <mathew.w.mclean@gmail.com> and Fabian Scheipl

See Also
pfr, predict.gam

Examples

```
######### 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 <- pfr(y~af(nir,argvals=wavelengths,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=test)
```

---

**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
```r
## 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

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

```r
pwcv(
  fdobj,  # a functional data object (class fd) giving the functional responses.
  Z,      # the model matrix, whose columns represent scalar predictors.
  L = NULL,  # 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 = seq(min(fdobj$basis$range), max(fdobj$basis$range), length.out = 201),  # argument values at which the CV score is to be evaluated.
  scale = FALSE  # logical value or vector determining scaling of the matrix Z (see scale, to which the value of this argument is passed).
)
```

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 \textit{cross-validated integrated squared error}, the standard overall model fit score returned by \texttt{lofocv}.

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

Value

A vector of the same length as \texttt{eval.pts} giving the CV scores.

Author(s)

Philip Reiss \textless{}phil.reiss@nyumc.org\textgreater{}

References


See Also

\texttt{fosr,lofocv}

Examples

```r
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)

## S3 method for class 'pffr'
qq(
    object,
    rep = 0,
    level = 0.9,
    s.rep = 10,
    type = c("deviance", "pearson", "response"),
    pch = ".",
    rl.col = 2,
    rep.col = "gray80",
    ...
)

Arguments

object a fitted pffr-object
rep How many replicate datasets to generate to simulate quantiles of the residual distribution. 0 results in an efficient simulation free method for direct calculation, if this is possible for the object family.
level If simulation is used for the quantiles, then reference intervals can be provided for the QQ-plot, this specifies the level. 0 or less for no intervals, 1 or more to simply plot the QQ plot for each replicate generated.
s.rep how many times to randomize uniform quantiles to data under direct computation.
type what sort of residuals should be plotted? See residuals.gam.
pch plot character to use. 19 is good.
rl.col color for the reference line on the plot.
**quadWeights**

*Compute quadrature weights*

**Description**

Utility function for numerical integration.

**Usage**

```r
quadWeights(argvals, method = "trapezoidal")
```

**Arguments**

- `argvals`: function arguments.
- `method`: quadrature method. Can be either trapezoidal or midpoint.

**Value**

A vector of quadrature weights for the points supplied in `argvals`.

**Author(s)**

Clara Happ, with modifications by Philip Reiss

---

**re**

*Random effects constructor for fgam*

**Description**

Sets up a random effect for the levels of `x`. Use the by-argument to request random slopes.

**Usage**

```r
re(x, ...)
```

**Arguments**

- `x`: a grouping variable: must be a factor
- `...`: further arguments handed over to `s`, see `random.effects`

**Details**

See `random.effects` in mgcv.
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 and fitted values for a pffr models

Description

See predict.pffr for alternative options to extract estimated values from a pffr object. "Fitted values" here refers to the estimated additive predictor values, these will not be on the scale of the response for models with link functions.

Usage

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

## 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 (regular grid data) or, respectively, in the shape of the originally supplied ydata argument (sparse/irregular data), or, if FALSE, simply as a long vector as returned by resid.gam()?

... other arguments, passed to residuals.gam.

Value

A matrix or ydata-like data.frame or a vector of residuals / fitted values (see reformat-argument)

Author(s)

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

**Description**

NOTE: this function is designed to work with `pfr_old()` rather than `pfr()`. 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**

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

**Arguments**

- `pfr.obj`: an object returned by `pfr_old()`  
- `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 <bswhart@jhsp.edu>

References

See Also
pfr, predict.pfr, package RLRsim

Examples

```r
## 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

```r
visit.1.rest <- matrix(as.numeric(Y > 1), ncol=1)
covar.in <- visit.1.rest
```

## note there is missingness in the functional predictors

```r
apply(is.na(W1), 2, mean)
apply(is.na(W2), 2, mean)
```

## fit two univariate models, then one model with both functional predictors

```r
pfr.obj.t1 <- pfr_old(Y = O, covariates=covar.in, funcs = list(W1), subj = id, kz = 10, kb = 50)
pfr.obj.t2 <- pfr_old(Y = O, covariates=covar.in, funcs = list(W2), subj = id, kz = 10, kb = 50)
pfr.obj.t3 <- pfr_old(Y = O, covariates=covar.in, funcs = list(W1, W2), subj = id, kz = 10, kb = 50)
```

## plot the coefficient function and bounds

```r
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

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

```r
t1$test.stat
```

```r
t1$p.val
```

```r
t2$test.stat
```

```r
t2$p.val
```

```r
t3$test.stat
```

```r
t3$p.val
```

## do some testing with rlrt.pfr(); same as above but subj = NULL

```r
pfr.obj.t1 <- pfr(Y = O, covariates=covar.in, funcs = list(W1), subj = NULL, kz = 10, kb = 50)
```
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

vector of indices of evaluations of $X_i(s)$, i.e. $(s_1, \ldots, s_S)$

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

da 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.dt.smooth.spec

Domain Transformation basis constructor

Description

The dt basis allows for any of the standard mgcv (or user-defined) bases to be applied to a transformed version of the original terms. Smooths may be of any number of terms. Transformations are specified by supplying a function of any or all of the original terms. "by" variables are not transformed.

Usage

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

**object** a smooth specification object, generated by `s()`, `te()`, `ti()`, or `t2()`, with `bs="dt"`

**data** a list containing just the data (including any by variable) required by this term, with names corresponding to `object$term` (and `object$by`). The by variable is the last element.

**knots** a list containing any knots supplied for basis setup - in same order and with same names as data. Can be NULL.

Details

object should be created with an `xt` argument. For non-tensor-product smooths, this will be a list with the following elements:

1. **tf** (required): a function or character string (or list of functions and/or character strings) defining the coordinate transformations; see further details below.
2. **bs** (optional): character string indicating the bs for the basis applied to the transformed coordinates; if empty, the appropriate defaults are used.
3. **basistype** (optional): character string indicating type of bivariate basis used. Options include "s" (the default), "te", "ti", and "t2", which correspond to `s`, `te`, `ti`, and `t2`.
4. **...** (optional): for tensor product smooths, additional arguments to the function specified by `basistype` that are not available in `s()` can be included here, e.g. `d`, `np`, etc.

For tensor product smooths, we recommend using `s()` to set up the basis, and specifying the tensor product using `xt$basistype` as described above. If the basis is set up using `te()`, then the variables in `object$term` will be split up, meaning all transformation functions would have to be univariate.

Value

An object of class "dt.smooth". This will contain all the elements associated with the smooth.construct object from the inner smooth (defined by `xt$bs`), in addition to an `xt` element used by the `Predict.matrix` method.

Transformation Functions

Let `nterms = length(object$term)`. The tf element can take one of the following forms:

1. a function of `nargs` arguments, where `nargs <= nterms`. If `nterms > 1`, it is assumed that this function will be applied to the first term of `object$term`. If all argument names of the function are term names, then those arguments will correspond to those terms; otherwise, they will correspond to the first `nargs` terms in `object$term`.
2. a character string corresponding to one of the built-in transformations (listed below).
3. A list of length `ntfuncs`, where `ntfuncs<=nterms`, containing either the functions or character strings described above. If this list is named with term names, then the transformation functions will be applied to those terms; otherwise, they will be applied to the first `ntfuncs` terms in `object$term`.

The following character strings are recognized as built-in transformations:
• "log": log transformation (univariate)
• "ecdf": empirical cumulative distribution function (univariate)
• "linear01": linearly rescale from 0 to 1 (univariate)
• "s-t": first term ("s") minus the second term ("t") (bivariate)
• "s/t": first term ("s") divided by the second term ("t") (bivariate)
• "QTransform": performs a time-specific ecdf transformation for a bivariate smooth, where time is indicated by the first term, and x by the second term. Primarily for use with refund::af.

Some transformations rely on a fixed "pivot point" based on the data used to fit the model, e.g. quantiles (such as the min or max) of this data. When making predictions based on these transformations, the transformation function will need to know what the pivot points are, based on the original (not prediction) data. In order to accomplish this, we allow the user to specify that they want their transformation function to refer to the original data (as opposed to whatever the "current" data is). This is done by appending a zero ("0") to the argument name.

For example, suppose you want to scale the term linearly so that the data used to define the basis ranges from 0 to 1. The wrong way to define this transformation function: function(x) {(x -min(x))/(max(x) -min(x))}. This function will result in incorrect predictions if the range of data for which predictions are being made is not the same as the range of data that was used to define the basis. The proper way to define this function: function(x) {(x -min(x0))/(max(x0) -min(x0))}. By referring to x0 instead of x, you are indicating that you want to use the original data instead of the current data. This may seem strange to refer to a variable that is not one of the arguments, but the "dt" constructor explicitly places these variables in the environment of the transformation function to make them available.

Author(s)
Jonathan Gellar

See Also
smooth.construct
Arguments

- **object**: a `fpc.smooth.spec` object, usually generated by a term `s(x, bs="fpc")`; see Details.
- **data**: a list containing the data (including any by variable) required by this term, with names corresponding to `object$term` (and `object$by`). Only the first element of this list is used.
- **knots**: not used, but required by the generic `smooth.construct`.

Details

- **object**: must contain an `xt` element. This is a list that can contain the following elements:
  - **X**: (required) matrix of functional predictors
  - **method**: (required) the method of finding principal components; options include "svd" (unconstrained), "fpca.sc", "fpca.face", or "fpca.ssvd"
  - **npc**: (optional) the number of PC’s to retain
  - **pve**: (only needed if `npc` not supplied) the percent variance explained used to determine `npc`
  - **penalize**: (required) if FALSE, the smoothing parameter is set to 0
  - **bs**: the basis class used to pre-smooth `X`; default is "ps"

Any additional options for the pre-smoothing basis (e.g. `k`, `m`, etc.) can be supplied in the corresponding elements of `object`. See `s` for a full list of options.

Value

An object of class "fpc.smooth". In addition to the elements listed in `smooth.construct`, the object will contain

- **sm**: the smooth that is fit in order to generate the basis matrix over `object$term`
- **V.A**: the matrix of principal components

Author(s)

Jonathan Gellar <JGellar@mathematica-mpr.com>

References


See Also

- `fpcr`
Principal coordinate ridge regression

Description

Smooth constructor function for principal coordinate ridge regression fitted by `gam`. When the principal coordinates are defined by a relevant distance among functional predictors, this is a form of nonparametric scalar-on-function regression. Reiss et al. (2016) describe the approach and apply it to dynamic time warping distances among functional predictors.

Usage

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

Arguments

- `object`: a smooth specification object, usually generated by a term of the form `s(dummy, bs="pco", k, xt)`; see Details.
- `data`: a list containing just the data.
- `knots`: IGNORED!

Value

An object of class `pco.smooth`. The resulting object has an `xt` element which contains details of the multidimensional scaling, which may be interesting.

Details

The constructor is not normally called directly, but is rather used internally by `gam`.

In a `gam` term of the above form `s(dummy, bs="pco", k, xt),`

- `dummy` is an arbitrary vector (or name of a column in `data`) whose length is the number of observations. This is not actually used, but is required as part of the input to `s`. Note that if multiple `pco` terms are used in the model, there must be multiple unique term names (e.g., "dummy1", "dummy2", etc).
- `k` is the number of principal coordinates (e.g., `k=9` will give a 9-dimensional projection of the data).
- `xt` is a list supplying the distance information, in one of two ways. (i) A matrix `Dmat` of distances can be supplied directly via `xt=list(D=Dmat,...)`. (ii) Alternatively, one can use `xt=list(realdata=..., dist_fn=..., ...)` to specify a data matrix `realdata` and distance function `dist_fn`, whereupon a distance matrix `dist_fn(realdata)` is created.

The list `xt` also has the following optional elements:
• add: Passed to `cmdscale` when performing multidimensional scaling; for details, see the help for that function. (Default FALSE.)

• fastcmd: if TRUE, multidimensional scaling is performed by `cmdscale_lanczos`, which uses Lanczos iteration to eigendecompose the distance matrix; if FALSE, MDS is carried out by `cmdscale`. Default is FALSE, to use `cmdscale`.

Author(s)
David L Miller, based on code from Lan Huo and Phil Reiss

References

Examples
```r
## Not run:
# a simulated example
library(refund)
library(mgcv)
require(dtw)
## First generate the data
Xnl <- matrix(0, 30, 101)
set.seed(813)
 tt <- sort(sample(1:90, 30))
for(i in 1:30){
 Xnl[i, tt[i]: (tt[i]+4)] <- -1
 Xnl[i, (tt[i]+5): (tt[i]+9)] <- 1
}
X.toy <- Xnl + matrix(rnorm(30*101, 0, 0.05), 30)
y.toy <- tt + rnorm(30, 0, 0.05)
y.rainbow <- rainbow(30, end=0.9)[(y.toy-min(y.toy))/
  diff(range(y.toy))*29+1]
## Display the toy data
par(mfrow=c(2, 2))
matplot((0:100)/100, t(Xnl[c(4, 25), ]), type="l", xlab="t", ylab="",
  ylim=range(X.toy), main="Noiseless functions")
matplot((0:100)/100, t(X.toy[c(4, 25), ]), type="l", xlab="t", ylab="",
  ylim=range(X.toy), main="Observed functions")
matplot((0:100)/100, t(X.toy), type="l", lty=1, col=y.rainbow, xlab="t",
  ylab="", main="Rainbow plot")
## Obtain DTW distances
D.dtw <- dist(X.toy, method="dtw", window.type="sakoechiba", window.size=5)
## Compare PC vs. PCo ridge regression
```
# matrix to store results
GCVmat <- matrix(NA, 15, 2)
# dummy response variable
dummy <- rep(1,30)

# loop over possible projection dimensions
for (k. in 1:15){
  # fit PC (m1) and PCo (m2) ridge regression
  m1 <- gam(y.toy ~ s(dummy, bs="pco", k=k.,
          xt=list(realdata=X.toy, dist_fn=dist)), method="REML")
  m2 <- gam(y.toy ~ s(dummy, bs="pco", k=k., xt=list(D=D.dtw)), method="REML")
  # calculate and store GCV scores
  GCVmat[k., ] <- length(y.toy) * c(sum(m1$residuals^2)/m1$df.residual^2,
        sum(m2$residuals^2)/m2$df.residual^2)
}

## plot the GCV scores per dimension for each model
matplot(GCVmat, lty=1:2, col=1, pch=16:17, type="o", ylab="GCV",
        xlab="Number of principal components / coordinates",
        main="GCV score")
legend("right", c("PC ridge regression", "DTW-based PCoRR"), lty=1:2, pch=16:17)

## example of making a prediction
# fit a model to the toy data
m <- gam(y.toy ~ s(dummy, bs="pco", k=2, xt=list(D=D.dtw)), method="REML")

# first build the distance matrix
# in this case we just subsample the original matrix
# see ?pco_predict_preprocess for more information on formatting this data
dist_list <- list(dummy = as.matrix(D.dtw)[, c(1:5,10:15)])

# preprocess the prediction data
pred_data <- pco_predict_preprocess(m, newdata=NULL, dist_list)

# make the prediction
p <- predict(m, pred_data)

# check that these are the same as the corresponding fitted values
print(cbind(fitted(m)[ c(1:5,10:15)],p))

## End(Not run)
smooth.construct.peer.smooth.spec

Description

Sets up design matrix for functional random effects based on the PC scores of the covariance op-
erator 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.peer.smooth.spec

Basis constructor for PEER terms

Description

Smooth basis constructor to define structured penalties (Randolph et al., 2012) for smooth terms.

Usage

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

Arguments

object       a peer.smooth.spec object, usually generated by a term s(x, bs="peer"); see Details.
data         a list containing the data (including any by variable) required by this term, with
names corresponding to object$term (and object$by). Only the first element
of this list is used.
knots        not used, but required by the generic smooth.construct.
Details

The smooth specification object, defined using `s()`, should contain an `xt` element. `xt` will be a list that contains additional information needed to specify the penalty. The type of penalty is indicated by `xt$pentype`. There are four types of penalties available:

1. `xt$pentype`="RIDGE" for a ridge penalty, the default
2. `xt$pentype`="D" for a difference penalty. The order of the difference penalty is specified by the `m` argument of `s()`.
3. `xt$pentype`="DECOMP" for a decomposition-based penalty, $bP_Q + a(I - P_Q)$, where $P_Q = Q'(QQ')^{-1}Q$. The $Q$ matrix must be specified by `xt$Q`, and the scalar $a$ by `xt$phia`. The number of columns of $Q$ must be equal to the length of the data. Each row represents a basis function where the functional predictor is expected to lie, according to prior belief.
4. `xt$pentype`="USER" for a user-specified penalty matrix $L$, supplied by `xt$L`.

Value

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

Author(s)

Madan Gopal Kundu <mgkundu@iupui.edu> and Jonathan Gellar

References


See Also

`peer`
smooth.construct.pi.smooth.spec

Usage

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

Arguments

object       a smooth specification object, generated by, e.g., s(x,y,t,bs="pi",xt=list(g=list(g1,g2,g3))). For transformation functions g1, g2, and g3, see Details below.
data         a list containing the variables of the smooth (x, y, and t above), as well as any by variable.
knots       a list containing any knots supplied for basis setup - in same order and with same names as data. Can be NULL.

Details

All functions \( f_k() \) are defined using the same basis set. Accordingly, they are penalized using a single block-diagonal penalty matrix and one smoothing parameter. Future versions of this function may be able to relax this assumption.

object should be defined (using s()) with an xt argument. This argument is a list that could contain any of the following elements:

1. g: the functions \( g_k(t) \), specified as described below.
2. bs: the basis code used for the functions \( f_k() \); defaults to thin-plate regression splines, which is mgcv’s default. The same basis will be used for all \( k \).
3. idx: an integer index indicating which variable from object$term is to be parameterized, i.e., the t variable; defaults to length(object$term)
4. mp: flag to indicate whether multiple penalties should be estimated, one for each \( f_k() \). Defaults to TRUE. If FALSE, the penalties for each \( k \) are combined into a single block-diagonal penalty matrix (with one smoothing parameter).
5. . . .: further xt options to be passed onto the basis for \( f_k() \).

xt$g can be entered in one of the following forms:

1. a list of functions of length \( k \), where each function is of one argument (assumed to be t)
2. one of the following recognized character strings: code"linear", indicating a linear interaction, i.e. \( f(x,t) = f_1(x) + t \cdot f_2(x) \); "quadratic", indicating a quadratic interaction, i.e. \( f(x,t) = f_1(x) + t \cdot f_2(x) + t^2 \cdot f_3(x) \); or "none", indicating no interaction with t, i.e. \( f(x,t) = f_1(x) \).

The only one of the above elements that is required is xt. If default values for bs, idx, and mp are desired, xt may also be entered as the g element itself; i.e. xt=g, where g is either the list of functions or an acceptable character string.

Additional arguments for the lower-dimensional basis over \( f_k \) may be entered using the corresponding arguments of s(), e.g. k, m, sp, etc. For example, s(x,t,bs="pi",k=15,xt=list(g="linear",bs="ps")) will define a linear interaction with t of a univariate p-spline basis of dimension 15 over x.

Value

An object of class "pi.smooth". See smooth.construct for the elements it will contain.
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 specified 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 $\nu_m$ 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

Description

A dataset containing the SOFA scores (Vincent et al, 1996). for 520 patients, hospitalized in the intensive care unit (ICU) with Acute Lung Injury. Daily measurements are available for as long as each one remains in the ICU. This is an example of variable-domain functional data, as described by Gellar et al. (2014).

Usage

sofa

Format

A data frame with 520 rows (subjects) and 7 variables:

- death binary indicator that the subject died in the ICU
- SOFA 520 x 173 matrix in variable-domain format (a ragged array). Each column represents an ICU day. Each row contains the SOFA scores for a subject, one per day, for as long as the subject remained in the ICU. The remaining cells of each row are padded with NAs. SOFA scores range from 0 to 24, increasing with severity of organ failure. Missing values during one’s ICU stay have been imputed using LOCF.
- SOFA_raw Identical to the SOFA element, except that it contains some missing values during one’s hospitalization. These missing values arise when a subject leaves the ICU temporarily, only to be re-admitted. SOFA scores are not monitored outside the ICU.
- los ICU length of stay, i.e., the number of days the patient remained in the ICU prior to death or final discharge.
- age Patient age
- male Binary indicator for male gender
- Charlson Charlson co-morbidity index, a measure of baseline health status (before hospitalization and ALI).

Details

The data was collected as part of the Improving Care of ALI Patients (ICAP) study (Needham et al., 2006). If you use this dataset as an example in written work, please cite the study protocol.

References


---

**summary.pffr**  
Summary for a pffr fit

**Description**

Take a fitted pffr-object and produce summaries from it. See `summary.gam()` for details.

**Usage**

```r
## 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

---

**summary.pfr**  
Summary for a pfr fit

**Description**

Take a fitted pfr-object and produce summaries from it. See `summary.gam()` for details.

**Usage**

```r
## S3 method for class 'pfr'
summary(object, ...)
```
vb_cs_fpca

Arguments

object  a fitted pfr-object
...  see summary.gam() for options.

Details

This function currently simply strips the "pfr" class label and calls summary.gam.

Value

A list with summary information, see summary.gam()

Author(s)

Jonathan Gellar <JGellar@mathematica-mpr.com>, Fabian Scheipl

---

vb_cs_fpca  Cross-sectional FoSR using Variational Bayes and FPCA

Description

Fitting function for function-on-scalar regression for cross-sectional data. This function estimates model parameters using a VB and estimates the residual covariance surface using FPCA.

Usage

vb_cs_fpca(
  formula,
  data = NULL,
  verbose = TRUE,
  Kt = 5,
  Kp = 2,
  alpha = 0.1,
  Aw = NULL,
  Bw = NULL,
  Apsi = NULL,
  Bpsi = NULL,
  argvals = NULL
)

Arguments

formula  a formula indicating the structure of the proposed model.
data  an optional data frame, list or environment containing the variables in the model. If not found in data, the variables are taken from environment(formula), typically the environment from which the function is called.
vb_cs_wish

**verbose**  
logical defaulting to TRUE – should updates on progress be printed?

**Kt**  
number of spline basis functions used to estimate coefficient functions

**Kp**  
number of FPCA basis functions to be estimated

**alpha**  
tuning parameter balancing second-derivative penalty and zeroth-derivative penalty  
(alpha = 0 is all second-derivative penalty)

**Aw**  
hyperparameter for inverse gamma controlling variance of spline terms for population-level effects

**Bw**  
hyperparameter for inverse gamma controlling variance of spline terms for population-level effects

**Apsi**  
hyperparameter for inverse gamma controlling variance of spline terms for FPC effects

**Bpsi**  
hyperparameter for inverse gamma controlling variance of spline terms for FPC effects

**argvals**  
not currently implemented

**Author(s)**

Jeff Goldsmith <ajg2202@cumc.columbia.edu>

**References**


---

**vb_cs_wish**  
*Cross-sectional FoSR using Variational Bayes and Wishart prior*

**Description**

Fitting function for function-on-scalar regression for cross-sectional data. This function estimates model parameters using VB and estimates the residual covariance surface using a Wishart prior.

**Usage**

```r
vb_cs_wish(
  formula,
  data = NULL,
  verbose = TRUE,
  Kt = 5,
  alpha = 0.1,
  min.iter = 10,
  max.iter = 50,
  Aw = NULL,
  Bw = NULL,
  v = NULL
)
```
vb_mult_fpca

**Arguments**

- `formula`: a formula indicating the structure of the proposed model.
- `data`: an optional data frame, list or environment containing the variables in the model. If not found in data, the variables are taken from `environment(formula)`, typically the environment from which the function is called.
- `verbose`: logical defaulting to `TRUE` – should updates on progress be printed?
- `Kt`: number of spline basis functions used to estimate coefficient functions
- `alpha`: tuning parameter balancing second-derivative penalty and zeroth-derivative penalty (alpha = 0 is all second-derivative penalty)
- `min.iter`: minimum number of iterations of VB algorithm
- `max.iter`: maximum number of iterations of VB algorithm
- `Aw`: hyperparameter for inverse gamma controlling variance of spline terms for population-level effects; if `NULL`, defaults to `Kt/2`.
- `Bw`: hyperparameter for inverse gamma controlling variance of spline terms for population-level effects; if `NULL`, defaults to `1/2 tr(mu.q.beta)` of the model
- `v`: hyperparameter for inverse Wishart prior on residual covariance; if `NULL`, `Psi` defaults to an FPCA decomposition of the residual covariance in which residuals are estimated based on an OLS fit of the model (note the "nugget effect" on this covariance is assumed to be constant over the time domain).

**Author(s)**

Jeff Goldsmith <ajg2202@columbia.edu>

**References**


---

**vb_mult_fpca**  
*Multilevel FoSR using Variational Bayes and FPCA*

**Description**

Fitting function for function-on-scalar regression for multilevel data. This function estimates model parameters using a VB and estimates the residual covariance surface using FPCA.
Usage

```r
vb_mult_fpca(
  formula,
  data = NULL,
  verbose = TRUE,
  Kt = 5,
  Kp = 2,
  alpha = 0.1,
  argvals = NULL
)
```

Arguments

- `formula`: a formula indicating the structure of the proposed model.
- `data`: an optional data frame, list or environment containing the variables in the model. If not found in data, the variables are taken from `environment(formula)`, typically the environment from which the function is called.
- `verbose`: logical defaulting to `TRUE` – should updates on progress be printed?
- `Kt`: number of spline basis functions used to estimate coefficient functions
- `Kp`: number of FPCA basis functions to be estimated
- `alpha`: tuning parameter balancing second-derivative penalty and zeroth-derivative penalty (alpha = 0 is all second-derivative penalty)
- `argvals`: not currently implemented

Author(s)

Jeff Goldsmith <ajg2202@cumc.columbia.edu>

References


Description

Fitting function for function-on-scalar regression for cross-sectional data. This function estimates model parameters using VB and estimates the residual covariance surface using a Wishart prior. If prior hyperparameters are `NULL` they are estimated using the data.
Usage

vb_mult_wish(
    formula,
    data = NULL,
    verbose = TRUE,
    Kt = 5,
    alpha = 0.1,
    min.iter = 10,
    max.iter = 50,
    Az = NULL,
    Bz = NULL,
    Aw = NULL,
    Bw = NULL,
    v = NULL
)

Arguments

formula a formula indicating the structure of the proposed model.
data an optional data frame, list or environment containing the variables in the model. If not found in data, the variables are taken from environment(formula), typically the environment from which the function is called.
verbose logical defaulting to TRUE – should updates on progress be printed?
Kt number of spline basis functions used to estimate coefficient functions
alpha tuning parameter balancing second-derivative penalty and zeroth-derivative penalty (alpha = 0 is all second-derivative penalty)
min.iter minimum number of iterations of VB algorithm
max.iter maximum number of iterations of VB algorithm
Az hyperparameter for inverse gamma controlling variance of spline terms for subject-level effects
Bz hyperparameter for inverse gamma controlling variance of spline terms for subject-level effects
Aw hyperparameter for inverse gamma controlling variance of spline terms for population-level effects
Bw hyperparameter for inverse gamma controlling variance of spline terms for population-level effects
v hyperparameter for inverse Wishart prior on residual covariance

Author(s)

Jeff Goldsmith <ajg2202@cumc.columbia.edu>
References


---

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

```r
vis.fgam(
  object,  # an fgam object, produced by fgam
  af.term, # character; 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 = NULL,  # 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 = NULL,  # 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
  deriv2 = FALSE,  # logical; 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 = 50,  # numeric; viewing angle; see persp
  plot.type = "persp",  # one of "contour" (to use levelplot) or "persp" (to use persp). Ignored if either xval or tval is specified
  ticktype = "detailed",  # how to draw the tick marks if plot.type="persp". Defaults to "detailed"
  ...  # other options to be passed to persp, levelplot, or plot
)
```

Arguments

- **object**: an fgam object, produced by `fgam`
- **af.term**: character; 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
- **deriv2**: logical; 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**: numeric; viewing angle; see `persp`
- **plot.type**: one of "contour" (to use `levelplot`) or "persp" (to use `persp`). Ignored if either `xval` or `tval` is specified
- **ticktype**: how to draw the tick marks if `plot.type="persp"`. Defaults to "detailed"
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" intervals 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 <mathew.w.mclean@gmail.com>

References


See Also

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

Examples

```
############################ 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
callusom 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)

---

vis.pfr

**Visualization of PFR objects**

**Description**

Produces perspective or contour plot views of an estimated surface corresponding smooths over two or more dimensions. Alternatively plots “slices” of the estimated surface or estimated second derivative surface with one of its arguments fixed. Corresponding twice-standard error “Bayesian” confidence bands are constructed using the method in Marra and Wood (2012). See the details.

**Usage**

vis.pfr(
  object,
  select = 1,
  xval = NULL,
  tval = NULL,
  deriv2 = FALSE,
  theta = 50,
  plot.type = "persp",
  ticktype = "detailed",
  ...
)

**Arguments**

- **object**
  - an pfr object, produced by pfr

- **select**
  - index for the smooth term to be plotted, according to its position in the model formula (and in object$smooth). Not needed if only one multivariate term is present.

- **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  logical; 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  numeric; viewing angle; see persp
plot.type  one of "contour" (to use levelplot) or "persp" (to use 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" intervals 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 <mathew.w.mclean@gmail.com>

References


See Also

vis.gam, plot.gam, pfr, persp, levelplot

Examples

################################ DTI Example ################################
data(DTI)

## only consider first visit and cases (since no PASAT scores for controls),
## and remove missing data
DTI <- DTI[DTI$visit==1 & DTI$case==1 & complete.cases(DTI$cca),]

## Fit the PFR 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 <- pfr(pasat ~ af(cca, basistype="te", k=c(8,8), m=list(c(2,3),c(2,3)), bs="ps"),
          method="GCV.Ct", gamma=1.2, data=DTI)

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

## similar to Figure 5 from McLean et al.
## Bands seem too conservative in some cases
xval <- runif(1, min(fit$pfr$ft[[1]]$Xrange), max(fit$pfr$ft[[1]]$Xrange))
tval <- runif(1, min(fit$pfr$ft[[1]]$xind), max(fit$pfr$ft[[1]]$xind))
par(mfrow=c(2, 2))
vis.pfr(fit, deriv2=FALSE, xval=xval)
vis.pfr(fit, deriv2=FALSE, tval=tval)
vis.pfr(fit, deriv2=TRUE, xval=xval)
vis.pfr(fit, deriv2=TRUE, tval=tval)

---

Xt_siginv_X

**Internal computation function**

**Description**

Internal function used compute the products \((X \otimes \Theta)^t (I \otimes \Sigma^{-1}) (X \otimes \Theta)\) and \((X \otimes \Theta)^t (I \otimes \Sigma^{-1}) (y)\) in cross-sectional VB algorithm and Gibbs sampler

**Usage**

Xt_siginv_X(tx, siginv, y = NULL)

**Arguments**

- **tx**
  - transpose of the X design matrix
- **siginv**
  - inverse variance matrix
- **y**
  - outcome matrix. if NULL, function computes first product; if not, function computes second product.

**Author(s)**

Jeff Goldsmith <ajg2202@cumc.columbia.edu>
Index

* datasets
  cd4, 14
  DTI2, 23
  gasoline, 73
  sofa, 163

* package
  refund-package, 5

af, 5, 33, 81, 86, 115, 116, 129, 170
af_old, 8
amc, 37
bam, 5, 8, 33, 45, 80, 82, 85, 107, 108, 112, 113, 115
basisfd, 44
bayes_fosr, 10, 133, 134
boot, 18
boot.ci, 18
call, 24
ccb.fpc, 12
cd4, 14
checkError (refund-internal), 147
cmdscale, 15, 157
cmdscale_lanczos, 15, 157
coeff.pfr, 16, 18, 19, 108
coeff.pfr (coefficients.pfr), 19
coeffboot.pfr, 18
coefficients.pfr, 19
create.prep.func, 6, 7, 21, 80, 81
data.frame, 98
decorrelate (refund-internal), 147
dist, 15
DTI, 22
DTI2, 23
ecdf, 10
expand.call, 24
f_sum, 70
f_sum2, 71
f_sum4, 72
f_trace, 72
family.mgcv, 109
fbps, 24, 130
fd, 6, 10, 35, 37, 39, 44, 46, 67, 80, 86, 101
ff, 27, 30, 107, 108, 112, 114, 162
fpc, 28, 29, 107, 108
fpcplot, 31
fgam, 8, 10, 32, 85, 86, 132, 133, 170, 171
first.last_test (refund-internal), 147
fitted.pfr, 108
fitted.pffr (residuals.pffr), 147
formula, 42
fosr, 5, 34, 39–41, 44, 45, 123, 144
fosr.perm, 38
fosr.vs, 41, 123, 124, 135
fosr2s, 5, 43
fpc, 45, 115, 116
fpc.aface, 22, 48, 63, 65
fpc.1fda, 51
fpc.sc, 22, 29, 30, 37, 50, 58, 63, 65, 113
fpc.ssvd, 22, 50, 61, 65
fpc2s, 50, 63, 64
fpca, 5, 47, 66, 73, 125, 126, 155
fpca2s, 50, 63, 64
fpcc.feature, 16, 69
gam, 32, 33, 35, 36, 49, 58, 68, 82, 85, 92, 98, 107, 108, 112–115, 118, 156
gam.check, 111
gamm, 5, 8, 33, 45, 80, 82, 85, 107, 112, 115
gamm4, 33, 58, 85, 107, 108, 112, 115
gamObject, 16, 69
gasoline, 73
getNPC.DonohoGavish (refund-internal), 147
getRsq (refund-internal), 147
getShrtlbls (refund-internal), 147
getShrunckDist (refund-internal), 147
gibbs.cs_fpc, 73
gibbs_cs_wish, 75
gibbs_mult_fpca, 76
gibbs_mult_wish, 77
gls_cs, 79
grpreg, 42

imwd_test (refund-internal), 147
irreg2mat (refund-internal), 147

jagam, 107, 108

levelplot, 170, 171, 173
lf, 7, 10, 33, 46, 47, 80, 83, 84, 101, 115, 116
lf.vd, 82, 115, 116
lf.old, 85
linear.functional.terms, 7, 10, 29, 81, 84, 86, 152
list2df (refund-internal), 147
lme, 88, 104
lofocv, 36, 144
lpeer, 5, 87, 126
lpfr, 91
lw.test (refund-internal), 147

match.call, 24
mfPCA.sc, 94
model.matrix.pffr, 96, 108
mrf, 108

nearPD, 113

ols_cs, 97
Omegas (refund-internal), 147
optim, 49
optimize, 62
osplinepen2d (refund-internal), 147

par, 40, 123
parse.predict.pfr (refund-internal), 147
pca.fd, 37
pco, 98
pco (smooth.construct.pco.smooth.spec), 156
pco.predict.preprocess, 98
pcre, 99, 107, 159
peer, 5, 100, 115, 116, 127, 160
PEER.Sim, 103
peer.old, 102, 103
persp, 32, 170, 171, 173

pffr, 5, 18, 27, 29, 30, 44, 45, 106, 111–113, 139, 145, 151
pffr.check, 108, 111
pffrGLS, 112
pffrSIM, 113
pfr, 5, 7, 20, 22, 45, 80–82, 84, 101, 102, 114, 129, 141, 142, 149, 172, 173
pfr.old, 117
plot, 126, 127, 170, 173
plot.fosr, 37, 122
plot.fosr.perm (fosr.perm), 38
plot.fosr.vs, 123
plot.fpcr, 125
plot.gam, 17, 128, 129, 171, 173
plot.lpeer, 126
plot.ppeer, 127
plot.pfr, 108, 128
plot.pfr, 129

porridge
(smooth.construct.pco.smooth.spec), 156

postprocess.pfr (refund-internal), 147
predict.fbps, 130
predict.fgam, 33, 132
predict.fosr, 133
predict.fosr.vs, 135
predict.gam, 17, 96, 98, 132, 133, 139–142
Predict.matrix.dt.smooth, 136
Predict.matrix.fpc.smooth, 137
Predict.matrix.pco.smooth
(smooth.construct.pco.smooth.spec), 156

Predict.matrix.pcre.random.effect, 137
Predict.matrix.peer.smooth, 138
Predict.matrix.pi.smooth, 139
Predict.matrix.pss.smooth
(refund-internal), 147

predict.pffr, 108, 139, 147
predict.pfr, 21, 119, 141, 149
preprocess.pfr (refund-internal), 147
print.summary.gam, 142, 143
print.summary.pffr, 142
pspline.perm (refund-internal), 147
pwcv, 143

Q (PEER.Sim), 103
qq.gam, 111, 145
qq.pffr, 108, 145
quadWeights, 146
random.effects, 146, 147
re, 10, 33, 115, 116, 146
refund (refund-package), 5
refund-internal, 147
refund-package, 5
residuals.gam, 111, 145, 147
residuals.pffr, 108, 147
rlrt.pfr, 119, 148
s, 6, 9, 28, 30, 47, 80–83, 86, 115, 146, 152, 153, 155, 156
safeDeparse (refund-internal), 147
scale, 36, 143
sff, 107, 108, 151
slanczos, 15
smooth.basisPar, 9, 10, 86
smooth.construct, 136–139, 153–155, 159–162
smooth.construct.dt.smooth.spec, 84, 136, 152
smooth.construct.fpc.smooth.spec, 47, 137, 154
smooth.construct.pco.smooth.spec, 15, 98, 99, 156
smooth.construct.pcre.smooth.spec, 158
smooth.construct.peer.smooth.spec, 102, 138, 159
smooth.construct.pi.smooth.spec, 84, 139, 160
smooth.construct.pss.smooth.spec, 162
smooth.construct.re.smooth.spec, 159
smooth.terms, 5, 80, 81, 110, 162
sofa, 163
summary.gam, 164, 165
summary.pffr, 108, 143, 164
summary.pfr, 164
svd, 47
t2, 6, 9, 28, 33, 107, 115, 152, 153
te, 6, 9, 28, 33, 82, 83, 86, 115, 152, 153
ti, 107, 109, 153
vb_cs_fpca, 165
vb_cs_wish, 166
vb_mult_fpca, 167
vb_mult_wish, 168
vis.fgam, 33, 170
vis.gam, 171, 173
vis.pfr, 172
Xt_simginv_X, 174