Package ‘sparseFLMM’

June 19, 2021

Type Package

Title Functional Linear Mixed Models for Irregularly or Sparsely Sampled Data

Version 0.4.1

Maintainer Jona Cederbaum <Jona.Cederbaum@gmail.com>

Description Estimation of functional linear mixed models for irregularly or sparsely sampled data based on functional principal component analysis.

License GPL-2

LazyData TRUE

Depends R (>= 3.3), mgcv (>= 1.8-12), refund (>= 0.1-22)

Imports methods, parallel, MASS, Matrix, data.table

Suggests

Collate 'call_all_functions.R' 'cov_estimation_tri.R' 'cov_estimation_tri_constr.R' 'cov_estimation_whole.R' 'cov_estimation_tri_constr_nest.R' 'fpc_estimation.R' 'fpc_famm_estimation.R' 'get_cross_products.R' 'mean_estimation.R' 'tri_constraint_constructor.R' 'useful_functions.R' 'zzz.R'

RoxygenNote 7.1.1

Encoding UTF-8

NeedsCompilation no

Repository CRAN

Date/Publication 2021-06-19 09:10:02 UTC

Author Jona Cederbaum [aut, cre],
Alexander Volkmann [aut],
Almond Stöcker [aut]

R topics documented:

`acoustic` *Phonetics acoustic data (complete)*

Description

The data are part of a large study on consonant assimilation, which is the phenomenon that the articulation of two consonants becomes phonetically more alike when they appear subsequently in fluent speech. The data set contains the audio signals of nine different speakers which repeated the same sixteen German target words each five times. The target words are bisyllabic noun-noun compound words which contained the two abutting consonants of interest, s and sh, in either order. Consonant assimilation is accompanied by a complex interplay of language-specific, perceptual and articulatory factors. The aim in the study was to investigate the assimilation of the two consonants as a function of their order (either first s, then sh or vice-versa), syllable stress (stressed or unstressed) and vowel context, i.e. which vowels are immediately adjacent to the target consonants of interest. The vowels are either of the form ia or ai. For more details, see references below.

Format

A data frame with 24830 rows and 11 variables

Details

The variables are as follows:

- `subject_long`: unique identification number for each speaker.
- `word_long`: unique identification number for each target word.
- `combi_long`: number of the repetition of the combination of the corresponding speaker and target word.
- `y_vec`: the response values for each observation point
- `n_long`: unique identification number for each curve.
- `t`: the observations point locations.
- `covariate.1`: (order of the consonants, reference category first /s/ then /sh/).
- `covariate.2`: (stress of the final syllable of the first compound, reference category 'stressed').
- `covariate.3`: (stress of the initial syllable of the second compound, reference category 'stressed').
- `covariate.4`: (vowel context, reference category ia).
- `word_names_long`: names of the target words.

References

Pouplier, Marianne and Hoole, Philip (2016): Articulatory and Acoustic Characteristics of German Fricative Clusters, Phonetica, 73(1), 52–78.

Cederbaum, Pouplier, Hoole, Greven (2016): Functional Linear Mixed Models for Irregularly or Sparsely Sampled Data. Statistical Modelling, 16(1), 67-88.

`acoustic_subset` *Phonetics acoustic data (subset)*

Description

A small subset of the phonetics acoustic data set `acoustic` with observations from two speakers and two items only. This will not produce meaningful results but can be used as a toy data set when testing the code. The variables are as in the full data set, see `acoustic`.

Format

A data frame with 656 rows and 11 variables

References

Pouplier, Marianne and Hoole, Philip (2016): Articulatory and Acoustic Characteristics of German Fricative Clusters, Phonetica, 73(1), 52–78.

Cederbaum, Pouplier, Hoole, Greven (2016): Functional Linear Mixed Models for Irregularly or Sparsely Sampled Data. Statistical Modelling, 16(1), 67-88.

`make_summation_matrix` *Construct symmetry constraint matrix for bivariate symmetric smoothing.*

Description

This function can be used to construct a symmetry constraint matrix that imposes a (skew-)symmetry constraint on (cyclic) spline coefficients in symmetric bivariate smoothing problems and is especially designed for constructing objects of the class "symm.smooth", see `smooth.construct.symm.smooth.spec`.

Usage

```
make_summation_matrix(F, skew = FALSE, cyclic.degree = 0)
```

Arguments

F	number of marginal basis functions.
skew	logical, should the basis be constraint to skew-symmetry instead of symmetry.
cyclic.degree	integer, specifying the number of basis functions identified with each other at the boundaries in order to implement periodicity. Should be specified to match the degree of the utilized B-spline basis.

Details

Imposing a symmetry constraint to the spline coefficients in order to obtain a reduced coefficient vector is equivalent to right multiplication of the bivariate design matrix with the symmetry constraint matrix obtained with function `make_summation_matrix`. The penalty matrix of the bivariate smooth needs to be adjusted to the reduced coefficient vector by left and right multiplication with the symmetry constraint matrix. This function is used in the constructor function `smooth.construct.symm.smooth.spec`.

Value

A basis transformation matrix of dimension $F^2 \times G$ with $G < F^2$ depending on the specified constraint.

Author(s)

Jona Cederbaum, Almond Stoecker

References

Cederbaum, Scheipl, Greven (2016): Fast symmetric additive covariance smoothing. Submitted on arXiv.

See Also

`smooth.construct` and `smoothCon` for details on constructors

`Predict.matrix.symm.smooth` *Predict matrix method for (skew-)symmetric bivariate smooths.*

Description

Predict matrix method for (skew-)symmetric bivariate smooths.

Usage

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

Arguments

`object`	is a symm.smooth object created by `smooth.construct.symm.smooth.spec`, see `smooth.construct`.
`data`	see `smooth.construct`.

Author(s)

Jona Cederbaum, Almond Stoecker

See Also

`Predict.matrix` and `smoothCon` for details on constructors.

`smooth.construct.symm.smooth.spec`
Symmetric bivariate smooths constructor

Description

The `symm` class is a smooth class that is appropriate for symmetric bivariate smooths, e.g. of covariance functions, using tensor-product smooths in a `gam` formula. A constraint matrix is constructed (see `make_summation_matrix`) to impose a (skew-)symmetry constraint on the (cyclic) spline coefficients, which considerably reduces the number of coefficients that have to be estimated.

Usage

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

Arguments

`object`	is a smooth specification object or a smooth object.
`data`	a data frame, model frame or list containing the values of the (named) covariates at which the smooth term is to be evaluated.
`knots`	an optional data frame supplying any knot locations to be supplied for basis construction.

Details

By default a symmetric bivariate B-spline smooth g is specified, in the sense that $g(s,t) = g(t,s)$. By setting `s(...,bs = "symm",xt = list(skew = TRUE))`, a skew-symmetric (or anti-smmetric) smooth with $g(s,t) = -g(t,s)$ can be specified instead. In both cases, the smooth can also be constraint to be cyclic with the property $g(s,t) = g(s+c,t) = g(s,t+c)$ for some fixed constant c via specifying `xt = list(cyclic = TRUE)`. Note that this does not correspond to specifying a tensor-product smooth from cyclic marginal B-splines as given by the `cp`-smooth. In the cyclic case, it is recommended to explicitly specify the range of the domain of the smooth via the knots argument, as this determines the period and often deviates from the observed range.

The underlying procedure is the following: First, the marginal spline design matrices and the corresponding marginal difference penalties are built. Second, the tensor product of the marginal design matrices is built and the bivariate penalty matrix is set up. Third, the constraint matrix is applied to the tensor product design matrix and to the penalty matrix.

Value

An object of class "symm.smooth". See smooth.construct for the elements it will contain.

Author(s)

Jona Cederbaum, Almond Stoecker

References

Cederbaum, Scheipl, Greven (2016): Fast symmetric additive covariance smoothing. Submitted on arXiv.

See Also

smooth.construct and smoothCon for details on constructors

Examples

```
require(sparseFLMM)

# (skew-)symmetric smooths ---------------------------------------

# generate random surface
dat1 <- data.frame(arg1 = 1:50)
dat2 <- expand.grid(arg1 = 1:50, arg2 = 1:50)

Bskew <- Predict.matrix(
  smooth.construct(
    s(arg1, arg2, bs = "symm", xt = list(skew = TRUE)),
    data = dat2, knots = NULL ),
  data = dat2 )
Bsymm <- Predict.matrix(
  smooth.construct(
    s(arg1, arg2, bs = "symm", xt = list(skew = FALSE)),
    data = dat2, knots = NULL ),
  data = dat2 )

set.seed(934811)
dat2$yskew <- c(Bskew %*% rnorm(ncol(Bskew)))
dat2$ysymm <- c(Bsymm %*% rnorm(ncol(Bsymm)))

# fit sum of skew-symmetric and symmetric parts with corresponding smooths
modpa <- gam( I(yskew + ysymm) ~ s(arg1, arg2, bs = "symm", xt = list(skew = TRUE)) +
```

```
                s(arg1, arg2, bs = "symm", xt = list(skew = FALSE)), data = dat2)
# predict surfaces
preds <- predict(modpa, type = "terms")
dat1 <- as.list(dat1)
dat1$arg2 <- dat1$arg1
dat1$predskew <- matrix(preds[,1], nrow = length(dat1$arg1))
dat1$predsymm <- matrix(preds[,2], nrow = length(dat1$arg1))

cols <- hcl.colors(12, "RdBu")
opar <- par(mfcol = c(2,2))
# symm part (intercept missing)
with(dat1, image(arg1, arg2, predsymm, asp = 1,
                 main = "Symmetric part of y",
                 col = cols))
with(dat1, image(arg1, arg2, asp = 1,
                 main = "Fit via symm.smooth",
                 matrix(dat2$ysymm, nrow = length(arg1)),
                                   col = cols))
# skew-symm part
with(dat1, image(arg1, arg2, predskew, asp = 1,
                 main = "Skew-symmetric part of y",
                 col = cols))
with(dat1, image(arg1, arg2, asp = 1,
                 main = "Fit via symm.smooth",
                 matrix(dat2$yskew, nrow = length(arg1)),
                 col = cols))
par(opar)

stopifnot(all.equal(dat1$predskew, - t(dat1$predskew)))
stopifnot(all.equal(dat1$predsymm, t(dat1$predsymm)))




# cyclic (skew-)symmetric splines ---------------------------------------

# fit the above example with cyclic smooths
modpac <- gam( I(yskew + ysymm) ~ s(arg1, arg2, bs = "symm",
                                  xt = list(skew = TRUE, cyclic = TRUE)) +
                s(arg1, arg2, bs = "symm", xt = list(skew = FALSE, cyclic = TRUE)),
              knots = list(arg1 = c(1, 50), arg2 = c(1,50)),
              # specify arg range to specify 'wavelength'!
              data = dat2)
plot(modpac, asp = 1, se = FALSE, pages = 1)

predsc <- predict(modpac, type = "terms")
dat1$predskewc <- matrix(predsc[,1], nrow = length(dat1$arg1))
dat1$predsymmc <- matrix(predsc[,2], nrow = length(dat1$arg1))

# check cyclic margins
opar <- par(mfrow = c(1,2))
with(dat1, matplot(arg1, predsymmc[, c(1,10, 40)], t = "l",
                   main = "symmetric smooth"))
```

```
abline(h = dat1$predsymmc[1, c(1,10, 40)], col = "darkgrey")
abline(v = c(1,50), col = "darkgrey")

with(dat1, matplot(arg1, predskewc[, c(1,10, 40)], t = "l",
                   main = "skew-symmetric smooth"))
abline(h = dat1$predskewc[1, c(1,10, 40)], col = "darkgrey")
abline(v = c(1,50), col = "darkgrey")
par(opar)



# 1D point symmetric B-splines -------------------------------------------

# generate toy data
dat <- data.frame( x = 1:100 )
ps_obj <- with(dat, s(x, bs = "ps"))
B <- Predict.matrix(smooth.construct(ps_obj, dat, NULL), dat)
set.seed(3904)
dat$y <- B %*% rnorm(ncol(B))
plot(dat, t = "l")

# fit skew-symmetric spline
mod0 <- gam( y ~ s(x, bs = "symm", xt = list(skew = TRUE)),
             knots = list(x = c(0,100)), # specify x range to determine inversion point
             dat = dat )
lines(dat$x, predict(mod0), col = "cornflowerblue", lty = "dashed")

# or a symmetric spline to first part only
mod1 <- gam( y ~ s(x, bs = "symm"),
             knots = list(x=c(0,50)),
                          dat = dat[1:50, ])
lines(dat[1:50, ]$x, predict(mod1), col = "darkred", lty = "dashed")
```

`sparseFLMM` *Functional Linear Mixed Models for Irregularly or Sparsely Sampled Data*

Description

Estimation of functional linear mixed models (FLMMs) for irregularly or sparsely sampled data based on functional principal component analysis (FPCA). The implemented models are special cases of the general FLMM

$$Y_i(t_{ij}) = \mu(t_{ij}, x_i) + z_i^T U(t_{ij}) + \epsilon_i(t_{ij}), i = 1, ..., n, j = 1, ..., D_i,$$

with $Y_i(t_{ij})$ the value of the response of curve i at observation point t_{ij}, $\mu(t_{ij}, x_i)$ is a mean function, which may depend on covariates $x_i = (x_{i1}, \ldots, x_{ip})^T$. z_i is a covariate vector, which is multiplied with the vector of functional random effects $U(t_{ij})$. $\epsilon_i(t_{ij})$ is independent and identically distributed white noise measurement error with homoscedastic, constant variance. For more details, see references below.

The current implementation can be used to fit four special cases of the above general FLMM:

- a model for independent functional data (e.g. longitudinal data), for which $z_i^T U(t_{ij})$ only consists of a smooth curve-specific deviation (smooth error curve)
- a model for correlated functional data with one functional random intercept (fRI) for one grouping variable in addition to a smooth curve-specific error
- a model for correlated functional data with two crossed fRIs for two grouping variables in addition to a smooth curve-specific error
- a model for correlated functional data with two nested fRIs for two grouping variables in addition to a smooth curve-specific error.

Usage

```
sparseFLMM(
  curve_info,
  use_RI = FALSE,
  use_simple = FALSE,
  method = "fREML",
  use_bam = TRUE,
  bs = "ps",
  d_grid = 100,
  min_grid = 0,
  max_grid = 1,
  my_grid = NULL,
  bf_mean = 8,
  bf_covariates = 8,
  m_mean = c(2, 3),
  covariate = FALSE,
  num_covariates,
  covariate_form,
  interaction,
  which_interaction = matrix(NA),
  save_model_mean = FALSE,
  para_estim_mean = FALSE,
  para_estim_mean_nc = 0,
  bf_covs,
  m_covs,
  use_whole = FALSE,
  use_tri = FALSE,
  use_tri_constr = TRUE,
  use_tri_constr_weights = FALSE,
  np = TRUE,
  mp = TRUE,
  use_discrete_cov = FALSE,
  para_estim_cov = FALSE,
  para_estim_cov_nc = 0,
  var_level = 0.95,
```

```
  N_B = NA,
  N_C = NA,
  N_E = NA,
  use_famm = FALSE,
  use_bam_famm = TRUE,
  bs_int_famm = list(bs = "ps", k = 8, m = c(2, 3)),
  bs_y_famm = list(bs = "ps", k = 8, m = c(2, 3)),
  save_model_famm = FALSE,
  use_discrete_famm = FALSE,
  para_estim_famm = FALSE,
  para_estim_famm_nc = 0,
  nested = FALSE
)
```

Arguments

`curve_info` data table in which each row represents a single observation point. `curve_info` needs to contain the following columns:

- `y_vec` (numeric): the response values for each observation point
- `t` (numeric): the observations point locations, i.e. t_{ij}
- `n_long` (integer): unique identification number for each curve
- `subject_long` (integer): unique identification number for each level of the first grouping variable (e.g. speakers for the phonetics data in the example below). In the case of independent functions, `subject_long` should be set equal to `n_long`.

For models with two crossed functional random intercepts, the data table additionally needs to have columns:

- `word_long` (integer): unique identification number for each level of the second grouping variable (e.g. words for the phonetics data in the example below)
- `combi_long` (integer): number of the repetition of the combination of the corresponding level of the first and of the second grouping variable.

For models with two nested functional random intercepts, the data table additionally needs to have columns: #'

- `word_long` (integer): unique identification number for each level of the second grouping variable (e.g. phases of a randomized controled trial). Note that the nested model is only implemented for two levels in the second grouping variable.
- `combi_long` (integer): number of the repetition of the combination of the corresponding level of the first and of the second grouping variable.

For models with covariates as part of the mean function $\mu(t_{ij}, x_i)$, the covariate values (numeric) need to be in separate columns with names: `covariate.1`, `covariate.2`, etc.

`use_RI` TRUE to specify a model with one functional random intercept for the first grouping variable (`subject_long`) and a smooth random error curve. Defaults to `FALSE`, which specifies a model with crossed functional random intercepts for the first and second grouping variable and a smooth error curve.

use_simple
: TRUE to specify a model with only a smooth random error function, use_RI should then also be set to TRUE. Defaults to FALSE.

method
: estimation method for gam or bam, see mgcv for more details. Defaults to "fREML".

use_bam
: TRUE to use function bam instead of function gam (syntax is the same, bam is faster for large data sets). bam is recommended and set as default.

bs
: spline basis function type for the estimation of the mean function and the auto-covariance, see s and te for more details. Defaults to penalized B-splines, i.e. bs = "ps". This choice is recommended as others have not been tested yet.

d_grid
: pre-specified grid length for equidistant grid on which the mean, the auto-covariance surfaces, the eigenfunctions and the functional random effects are evaluated. NOTE: the length of the grid can be important for computation time (approx. quadratic influence). Defaults to d_grid = 100.

min_grid
: minimum value of equidistant grid (should approx. correspond to minimum value of time interval). Defaults to min_grid = 0.

max_grid
: maximum value of equidistant grid (should approx. correspond to maximum value of time interval). Defaults to max_grid = 1.

my_grid
: optional evaluation grid, which can be specified and used instead of d_grid, min_grid, max_grid. NOTE: the grid should be equidistant.

bf_mean
: basis dimension (number of basis functions) used for the functional intercept $f_0(t_{ij})$ in the mean estimation via bam/gam. Defaults to bf_mean = 8.

bf_covariates
: basis dimension (number of basis functions) used for the functional effects of covariates in the mean estimation via bam/gam. Defaults to bf_covariates = 8. NOTE: in the current implementation, the same basis dimension for all covariates is used.

m_mean
: order of the penalty for this term in bam/gam of mean estimation, for bs = "ps" spline and penalty order, defaults to m_mean = c(2,3), i.e., cubic B-splines with third order difference penalty, see s for details.

covariate
: TRUE to estimate covariate effects (as part of the mean function).

num_covariates
: number of covariates that are included in the model. NOTE: not number of effects in case interactions of covariates are specified.

covariate_form
: vector with entries for each covariate that specify the form in which the respective covariate enters the mean function. Possible forms are "by" for varying-coefficient ($f(t_{ij}) * covariate$), which is possible for dummy coded covariates and metric covariates and "smooth" for smooth effect in t and in covariate ($f(t_{ij}, covariate)$), which is only possible for metric covariates! NOTE: metric covariates should be centered such that the global functional intercept $f_0(t_{ij})$ can be interpreted as global mean function and the effect can be interpreted as difference from the global mean.

interaction
: TRUE to estimate interaction effects of covariates, which interactions, see which_interaction (below). Interactions are possible for dummy-coded covariates that act as varying coefficients.

which_interaction
: symmetric matrix that specifies which interactions should be considered in case covariate = TRUE and interaction = TRUE. Entry which_interaction[k, l]

specifies that the interaction between covariate.k and covariate.l is modeled (example below). NOTE: entries are redundant, which_interaction[l, k] should be set to the same as which_interaction[k, l] (symmetric). Defaults to which_interaction = matrix(NA) which should be specified when interaction = FALSE.

save_model_mean
: TRUE to give out gam/bam object (attention: can be large!), defaults to FALSE.

para_estim_mean
: TRUE to parallelize mean estimation (only possible using bam), defaults to FALSE.

para_estim_mean_nc
: number of cores for parallelization of mean estimation (only possible using bam, only active if para_estim_mean = TRUE). Defaults to 0.

bf_covs
: vector of marginal basis dimensions (number of basis functions) used for covariance estimation via bam/gam for each functional random effect (including the smooth error curve). In the case of multiple grouping variables, the first entry corresponds to the first grouping variable, the second vector entry corresponds to the second grouping variable, and the third to the smooth error curve.

m_covs
: list of marginal orders of the penalty for bam/gam for covariance estimation, for bs = "ps" marginal spline and penalty order. As only symmetric surfaces are considered: same for both directions.
For crossed fRIs: list of three vectors, e.g. m_covs = list(c(2,3),c(2,3),c(2,3)), where first and second entry correspond to first and second grouping variable, respectively and third entry corresponds to smooth error. For one fRI: list of two vectors, e.g. m_covs = list(c(2,3),c(2,3)), where first entry corresponds to (first) grouping variable and second entry corresponds to smooth error. For independent curves: list of one vector, e.g. m_covs = list(c(2,3)) corresponding to smooth error.

use_whole
: TRUE to estimate the whole auto-covariance surfaces without symmetry constraint. Defaults to FALSE as is much slower than use_tri_constr and use_tri_constr_weights. For more details, see references below.

use_tri
: TRUE to estimate only the upper triangle of the auto-covariance surfaces without symmetry constraint. Defaults to FALSE and not recommended. For more details, see references below.

use_tri_constr
: TRUE to estimate only the upper triangle of the auto-covariance surfaces with symmetry constraint using the smooth class 'symm'. Defaults to TRUE. For more details, see references below.

use_tri_constr_weights
: TRUE to estimate only the upper triangle of the auto-covariances with symmetry constraint, using the smooth class 'symm' and weights of 0.5 on the diagonal to use the same weights as for estimating the whole auto-covariance surfaces. Defaults to FALSE. For more details, see references below.

np
: TRUE to use 'normal parameterization' for a tensor product smooth, see te for more details. Defaults to TRUE.

mp
: FALSE to use Kronecker product penalty instead of Kronecker sum penalty with only one smoothing parameter (use_whole = TRUE and use_tri = TRUE), for details see te. For use_tri_constr = TRUE and use_tri_constr_weights = TRUE, only one smoothing parameter is estimated anyway. Defaults to TRUE.

use_discrete_cov
: TRUE to further speed up the auto-covariance computation by discretization of covariates for storage and efficiency reasons, includes parallelization controlled by para_estim_cov_nc (below), see bam for more details. Defaults to FALSE.

para_estim_cov
: TRUE to parallelize auto-covariance estimation (only possible using bam), defaults to FALSE.

para_estim_cov_nc
: number of cores (if use_discrete_cov = FALSE) or number of threads (if use_discrete_cov = TRUE) for parallelization of auto-covariance estimation (only possible using bam, only active if para_estim_cov = TRUE). Defaults to 0.

var_level
: pre-specified level of explained variance used for the choice of the number of the functional principal components (FPCs). Alternatively, a specific number of FPCs can be specified (see below). Defaults to var_level = 0.95.

N_B
: number of components for B (fRI for first grouping variable) to keep, overrides var_level if not NA.

N_C
: number of components for C (fRI for second grouping variable) to keep, overrides var_level if not NA.

N_E
: number of components for E (smooth error) to keep, overrides var_level if not NA.

use_famm
: TRUE to embed the model into the framework of functional additive mixed models (FAMMs) using re-estimation of the mean function together with the prediction of the FPC weights (scores). This allows for point-wise confidence bands for the covariate effects. Defaults to FALSE.

use_bam_famm
: TRUE to use function bam instead of function gam in FAMM estimation (reduces computation time for large data sets), highly recommended. Defaults to TRUE.

bs_int_famm
: specification of the estimation of the functional intercept $f_0(t_{ij})$ (as part of the mean function), see pffr for details. Defaults to bs_int = list(bs = "ps",k = 8,m = c(2,3)), where bs: type of basis functions, k: number of basis functions, m: order of the spline and order of the penalty.

bs_y_famm
: specification of the estimation of the covariates effects (as part of the mean function), see pffr for details. Defaults to bs_y_famm = list(bs = "ps",k = 8,m = c(2,3)), where bs: type of basis functions, k: number of basis functions, m: order of the spline and order of the penalty.

save_model_famm
: TRUE to give out the FAMM model object (attention: can be very large!). Defaults to FALSE.

use_discrete_famm
: TRUE to further speed up the fpc-famm computation by discretization of # covariates for storage and efficiency reasons, includes parallelization controlled by para_estim_famm_nc (below), see bam for more details. Defaults to FALSE.

para_estim_famm
: TRUE to parallelize FAMM estimation. Defaults to FALSE.

para_estim_famm_nc
: number of cores (if use_discrete_famm = FALSE) or number of threads (if use_discrete_famm = TRUE) for parallelization of FAMM estimation (only possible using bam, only active if para_estim_famm = TRUE). Defaults to 0.

nested — `TRUE` to specify a model with nested functional random intercepts for the first and second grouping variable and a smooth error curve. Defaults to `FALSE`.

Details

The code can handle irregularly and possibly sparsely sampled data. Of course, it can also be used to analyze regular grid data, but as it is especially designed for the irregular case and there may be a more efficient way to analyze regular grid data.

The mean function is of the form

$$\mu(t_{ij}, x_i) = f_0(t_{ij}) + \sum_{k=1}^{r} f_k(t_{ij}, x_{ik}),$$

where $f_0(t_{ij})$ is a functional intercept. Currently implemented are effects of dummy-coded and metric covariates which act as varying-coefficients of the form $f_k(t_{ij}) * x_{ik}$ and smooth effects of metric covariates (smooth in t and in the covariate) of the form $f(t_{ij}, x_{ik})$. NOTE: metric covariates should be centered such that the global functional intercept can be interpreted as global mean function and the effect can be interpreted as difference from the global mean. Interaction effects of dummy-coded covariates acting as varying coefficients are possible.

The estimation consists of four main steps:

1. estimation of the smooth mean function (including covariate effects) under independence assumption using splines.
2. estimation of the smooth auto-covariances of the functional random effects. A fast bivariate symmetric smoother implemented in the smooth class 'symm' can be used to speed up estimation (see below).
3. eigen decomposition of the estimated auto-covariances, which are evaluated on a pre-specified equidistant grid. This yields estimated eigenvalues and eigenfunctions, which are rescaled to ensure orthonormality with respect to the L2-scalar product.
4. prediction of the functional principal component weights (scores) yielding predictions for the functional random effects.

The estimation of the mean function and auto-covariance functions is based on package **mgcv**.
The functional principal component weights (scores) are predicted as best (linear) unbiased predictors. In addition, this implementation allows to embed the model in the general framework of functional additive mixed models (FAMM) based on package **refund**, which allows for the construction of point-wise confidence bands for covariate effects (in the mean function) conditional on the FPCA. Note that the estimation as FAMM may be computationally expensive as the model is re-estimated in a mixed model framework.

The four special cases of the general FLMM (two nested fRIs, two crossed fRIs, one fRI, independent curves) are implemented as follows:

- In the special case with two nested fRIs, three random processes B, C, and E are considered, where B is the fRI for the first grouping variable (e. g. patient in a random controlled trial), C denotes the fRI for the second grouping variable (e.g. individual specific effect in the follow-up) and E denotes the smooth error. For this special case, arguments `use_RI` and `use_simple` are both set to `FALSE` and argument `nested` is set to `TRUE`. Note that this implementation only allows for a simple before/after study design.

- In the special case with two crossed fRIs, three random processes B, C, and E are considered, where B is the fRI for the first grouping variable (e.g. speakers in the phonetics example below), C denotes the fRI for the second grouping variable (e.g. target words in the phonetics example below) and E denotes the smooth error. For this special case, arguments `use_RI` and `use_simple` are both set to `FALSE`.
- In the special case with only one fRI, only B and E are considered and the number of levels for the second grouping variable is to zero. For this special case, argument `use_RI` is set to `TRUE` and argument `use_simple` is set to `FALSE`.
- The special case with independent curves is internally seen as a special case of the model with one fRI for the first grouping variable, with the number of levels for this grouping variable corresponding to the number of curves. Thus, for each level of the first grouping variable there is one curve. Therefore, for the special case of independent curves, the estimation returns an estimate for the auto-covariance of B (instead of E) and all corresponding results are indicated with `'_B'`, although they correspond to the smooth error. For this special case, arguments `use_RI` and `use_simple` are both set to `TRUE`.

Value

The function returns a list of two elements: `time_all` and `results`.
`time_all` contains the total system.time() for calling function `sparseFLMM()`.
`results` is a list of all estimates, including:

- `mean_hat`: includes the components of the estimated mean function.
  - `mean_pred` contains effects of dummy covariates or metric covariates with a linear effect (varying coefficients).
  - `mean_pred_smooth` contains effects of metric covariates with a smooth effect.
  - `intercept` is the estimated intercept, which is part of $f_0(t_{ij})$.

For each auto-covariance smoothing alternative `X` (`use_whole`, `use_tri`, `use_tri_constr`, `use_tri_constr_weights`):

- `cov_hat_X`: includes
  - `sigmasq`: the estimated error variance
  - `sigmasq_int`: the integral of the estimated error variance over the domain
  - `grid_mat_B/C/E`: the estimated auto-covariance(s) evaluated on the pre-specified grid
  - `sp`: the smoothing parameter(s) for smoothing the auto-covariance(s)
  - `time_cov_estim`: the time for the smoothing the auto-covariance(s) only
  - `time_cov_pred_grid`: the time for evaluating the estimated auto-covariance(s) on the pre-specified grid.
- `time_cov_X`: the total system.time() for the auto-covariance estimation
- `fpc_hat_X`: including
  - `phi_B/C/E_hat_grid`: the estimated rescaled eigenfunctions evaluated on the pre-specified grid
  - `nu_B/C/E_hat`: the estimated rescaled eigenvalues
  - `N_B/C/E`: the estimated truncation numbers, i.e., number of FPCs which are chosen
  - `total_var`: the estimated total variance
  - `var_explained`: the estimated explained variance

  - xi_B/C/E_hat: the predicted FPC weights (scores).
- time_fpc_X: the total system.time() for the eigen decompositions and prediction on the FPC weights (scores) If use_famm = TRUE, the list results additionally contains:
  - fpc_famm_hat_X: including
    * intercept: the estimated intercept, which is part of $f_0(t_{ij})$
    * residuals: the residuals of the FAMM estimation
    * xi_B/C/E_hat_famm: the predicted basis weights
    * famm_predict_B/C/E: the predicted functional processes evaluated on the pre-specified grid
    * famm_cb_mean: the re-estimated functional intercept $f_0(t_{ij})$
    * famm_cb_covariate.1, famm_cb_covariate.1, etc: possible re-estimated covariate effects
    * famm_cb_inter_1_2, famm_cb_inter_1_3, etc: possible interaction effects
    * time_fpc_famm_X: the total system.time() for the FAMM estimation.

The unique identification numbers for the levels of the grouping variables and curves are renumbered for convenience during estimation from 1 in ascending order. The original identification numbers are returned in the list results:

  - n_orig: curve levels as they entered the estimation
  - subject_orig: levels of the first grouping variable as they entered the estimation
  - word_orig: levels of the second grouping variable (if existent) as they entered the estimation
  - my_grid: pre-specified grid.

Author(s)

Jona Cederbaum

References

Cederbaum, Pouplier, Hoole, Greven (2016): Functional Linear Mixed Models for Irregularly or Sparsely Sampled Data. Statistical Modelling, 16(1), 67-88.

Cederbaum, Scheipl, Greven (2016): Fast symmetric additive covariance smoothing. Submitted on arXiv.

Scheipl, F., Staicu, A.-M. and Greven, S. (2015): Functional Additive Mixed Models, Journal of Computational and Graphical Statistics, 24(2), 477-501.

See Also

Note that sparseFLMM calls bam or gam directly.

For functional linear mixed models with complex correlation structures for data sampled on equal grids based on functional principal component analysis, see function denseFLMM in package denseFLMM.

Examples

```
## Not run:
# subset of acoustic data (very small subset, no meaningful results can be expected and
# FAMM estimation does not work for this subset example. For FAMM estimation, see below.)
data("acoustic_subset")

acoustic_results <- sparseFLMM(curve_info = acoustic_subset, use_RI = FALSE, use_simple = FALSE,
              method = "fREML", use_bam = TRUE, bs = "ps", d_grid = 100, min_grid = 0,
            max_grid = 1, my_grid = NULL, bf_mean = 8, bf_covariates = 8, m_mean = c(2,3),
              covariate = TRUE, num_covariates = 4, covariate_form = rep("by", 4),
              interaction = TRUE,
              which_interaction = matrix(c(FALSE, TRUE, TRUE, TRUE, TRUE,
              FALSE, FALSE, FALSE, TRUE, FALSE, FALSE, FALSE, TRUE, FALSE,
              FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE),
              byrow = TRUE, nrow = 4, ncol = 4),
              save_model_mean = FALSE, para_estim_mean = FALSE, para_estim_mean_nc = 0,
              bf_covs = c(5, 5, 5), m_covs = list(c(2, 3), c(2, 3), c(2, 3)),
              use_whole = FALSE, use_tri = FALSE, use_tri_constr = TRUE,
              use_tri_constr_weights = FALSE, np = TRUE, mp = TRUE,
              use_discrete_cov = FALSE,
              para_estim_cov = FALSE, para_estim_cov_nc = 5,
              var_level = 0.95, N_B = NA, N_C = NA, N_E = NA,
              use_famm = FALSE, use_bam_famm = TRUE,
              bs_int_famm = list(bs = "ps", k = 8, m = c(2, 3)),
              bs_y_famm = list(bs = "ps", k = 8, m = c(2, 3)),
              save_model_famm = FALSE, use_discrete_famm = FALSE,
              para_estim_famm = FALSE, para_estim_famm_nc = 0)
## End(Not run)

## Not run:
# whole data set with estimation in the FAMM framework

data("acoustic")
acoustic_results <- sparseFLMM(curve_info = acoustic, use_RI = FALSE, use_simple = FALSE,
              method = "fREML", use_bam = TRUE, bs = "ps", d_grid = 100, min_grid = 0,
            max_grid = 1, my_grid = NULL, bf_mean = 8, bf_covariates = 8, m_mean = c(2,3),
              covariate = TRUE, num_covariates = 4, covariate_form = rep("by", 4),
              interaction = TRUE,
              which_interaction = matrix(c(FALSE, TRUE, TRUE, TRUE, TRUE,
              FALSE, FALSE, FALSE, TRUE, FALSE, FALSE, FALSE, TRUE, FALSE,
              FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE),
              byrow = TRUE, nrow = 4, ncol = 4),
              save_model_mean = FALSE, para_estim_mean = FALSE, para_estim_mean_nc = 0,
              bf_covs = c(5, 5, 5), m_covs = list(c(2, 3), c(2, 3), c(2, 3)),
              use_whole = FALSE, use_tri = FALSE, use_tri_constr = TRUE,
              use_tri_constr_weights = FALSE, np = TRUE, mp = TRUE,
              use_discrete_cov = FALSE,
              para_estim_cov = TRUE, para_estim_cov_nc = 5,
              var_level = 0.95, N_B = NA, N_C = NA, N_E = NA,
              use_famm = TRUE, use_bam_famm = TRUE,
              bs_int_famm = list(bs = "ps", k = 8, m = c(2, 3)),
              bs_y_famm = list(bs = "ps", k = 8, m = c(2, 3)),
```

```
              save_model_famm = FALSE, use_discrete_famm = FALSE,
              para_estim_famm = TRUE, para_estim_famm_nc = 5)
## End(Not run)
```

Index