Package ‘DensityRegression’

March 1, 2025

Type Package

Title Conditional density regression for individual-level data

Version 0.1.0

Maintainer Eva-Maria Maier <eva-maria.maier@hu-berlin.de>

Description Structured additive regression models for mixed (discrete/continuous) densities as response with scalar covariates, fitted via Poisson regression models based on 'mgcv', where the continuous part of the density is approximated via histograms. Implementation for Maier et al. (2025b) <doi:.../arXiv...>.

License GPL-2

Depends mgcv

Imports cowplot,
data.table,
dplyr,
ggplot2,
manipulate,
MASS,
Matrix,
png,
reshape2,
Rdpack,
splines,
stats,
stringr,
tidyr,
utils,
weights

Suggests testthat

Encoding UTF-8

LazyData true

RoxygenNote 7.3.2

R topics documented:

DensityRegression-package 2
clr 3

data2counts 4
densreg 8
Predict.matrix.mdspline.smooth 16
smooth.construct.md.smooth.spec 18

Index **23**

`DensityRegression-package`

DensityRegression: Conditional density regression for individual-level data

---

Description

Structured additive regression models for mixed (discrete/continuous) densities as response with scalar covariates, fitted via Poisson regression models based on `gam`, where the continuous part of the density is approximated via histograms.

Details

This package implements the approach of Maier et al. (2025b) for fitting regression models with densities in mixed (continuous/discrete) Bayes Hilbert spaces (including continuous and discrete ones as special cases) as response given scalar covariates, based on observed samples of the conditional distributions via a (penalized) maximum likelihood approach. The main function (which estimates the models) is `densreg`. The (penalized) log-likelihood function is approximated via the (penalized) log-likelihood of an appropriate Poisson regression model, which - after constructing the count data appropriately via `data2counts` - is then fitted using `mgcv`'s `gam` with a new smooth term `bs="md"` (see `smooth.construct.md.smooth.spec`) for mixed densities.

For details see Maier et al. (2025b).

Author(s)

Eva-Maria Maier, Lea Runge, and Alexander Fottner

References

Maier, E.-M., Fottner, A., Stoecker, A., Okhrin, Y., & Greven, S. (2025b): Conditional density regression for individual-level data. arXiv preprint arXiv:XXXX.XXXXX.

See Also

`densreg` for the main fitting function.

clr — *Clr and inverse clr transformation*

Description

`clr` computes the (inverse) clr transformation of a vector `f` of (clr transformed) density evaluations with respect to integration weights `w` corresponding to a Bayes Hilbert space $B^2(\mu) = B^2(\mathcal{Y}, \mathcal{A}, \mu)$. This is a copy of `FDboost::clr` (with slightly adjusted help page).

Usage

```
clr(f, w = 1, inverse = FALSE)
```

Arguments

`f` A vector containing the function values (evaluated on a grid) of the function f to transform. If `inverse = FALSE`, `f` must be a density, i.e., all entries must be positive and usually `f` integrates to one. If `inverse = TRUE`, `f` should integrate to zero, see Details. For `densreg` objects `dr`, one column of `dr$f_hat` or `dr$f_hat_clr` (for inverse clr) are appropriate choices for `f` (or use `apply(dr$f_hat,MARGIN = 2,FUN = clr,w = dr$domain_data$Delta)` to clr transform all columns of `dr$f_hat` at once).

`w` A vector of length one or of the same length as `f` containing positive integration weights. If `w` has length one, this weight is used for all function values. The integral of f is approximated via $\int_{\mathcal{Y}} f \, \mathrm{d}\mu \approx \sum_{j=1}^{m} \mathtt{w}_j \, \mathtt{f}_j$, where m equals the length of `f`. For `densreg` objects `dr`, the vector `dr$domain_data$Delta` corresponds the appropriate weights to be used for `w`, when applying `clr` to columns of `dr$f_hat` or `dr$f_hat_clr` (for inverse clr).

`inverse` Logical, indicating if clr (`inverse = FALSE`; default) or inverse clr transformation (`inverse = TRUE`) shall be computed.

Details

The clr transformation maps a density f from $B^2(\mu)$ to $L^2_0(\mu) := \{f \in L^2(\mu) \mid \int_{\mathcal{Y}} f \, \mathrm{d}\mu = 0\}$ via

$$\mathrm{clr}(f) := \log f - \frac{1}{\mu(\mathcal{Y})} \int_{\mathcal{Y}} \log f \, \mathrm{d}\mu.$$

The inverse clr transformation maps a function f from $L^2_0(\mu)$ to $B^2(\mu)$ via

$$\mathrm{clr}^{-1}(f) := \frac{\exp f}{\int_{\mathcal{Y}} \exp f \, \mathrm{d}\mu}.$$

Note that in contrast to Maier et al. (2025b), this definition of the inverse clr transformation includes normalization, yielding the respective probability density function (representative of the equivalence class of proportional functions in $B^2(\mu)$).

The (inverse) clr transformation depends not only on f, but also on the underlying measure space $(\mathcal{Y}, \mathcal{A}, \mu)$, which determines the integral. In `clr` this is specified via the integration weights `w`. E.g., for a discrete set $\mathcal{Y}$ with $\mathcal{A} = \mathcal{P}(\mathcal{Y})$ the power set of $\mathcal{Y}$ and $\mu = \sum_{t \in Y} \delta_t$ the sum of dirac measures at $t \in \mathcal{Y}$, the default `w = 1` is the correct choice. In this case, integrals are indeed computed exactly, not only approximately. For an interval $\mathcal{Y} = [a, b]$ with $\mathcal{A} = \mathcal{B}$ the Borel σ-algebra restricted to $\mathcal{Y}$

and $\mu = \lambda$ the Lebesgue measure, the choice of w depends on the grid on which the function was evaluated: w_j must correspond to the length of the subinterval of $[a, b]$, which f_j represents. E.g., for a grid with equidistant distance d, where the boundary grid values are $a + \frac{d}{2}$ and $b - \frac{d}{2}$ (i.e., the grid points are centers of intervals of size d), equal weights d should be chosen for w.

Value

A vector of the same length as f containing the (inverse) clr transformation of f.

Author(s)

Eva-Maria Maier

References

FDboost::clr

Maier, E.-M., Fottner, A., Stoecker, A., Okhrin, Y., & Greven, S. (2025b): Conditional density regression for individual-level data. arXiv preprint arXiv:XXXX.XXXXX.

Examples

```
### Continuous case (Y = [0, 1] with Lebesgue measure):
# evaluate density of a Beta distribution on an equidistant grid
g <- seq(from = 0.005, to = 0.995, by = 0.01)
f <- dbeta(g, 2, 5)
# compute clr transformation with distance of two grid points as integration weight
f_clr <- clr(f, w = 0.01)
# visualize result
plot(g, f_clr , type = "l")
abline(h = 0, col = "grey")
# compute inverse clr transformation (w as above)
f_clr_inv <- clr(f_clr, w = 0.01, inverse = TRUE)
# visualize result
plot(g, f, type = "l")
lines(g, f_clr_inv, lty = 2, col = "red")
```

---

data2counts *From observations to a vector of observed (histogram) counts*

Description

data2counts prepares data containing the individual response observations y_i appropriately to be used in gam Poisson models by combining all observations of the same conditional distribution (i.e., all observations sharing identical values in all covariates) into a vector of counts via a histogram on $I \setminus D$ and counts on D where I is the interval of the continuous domain and D the set of discrete values. the discrete component of the underlying mixed (continuous/discrete) Bayes Hilbert space $B^2(\mu) = B^2(\mathcal{Y}, \mathcal{A}, \mu)$. We briefly summarize the approach below in the details. Please see Section 2.3 in Maier et al. (2025b) for comprehensive description.

Usage

```
data2counts(
  data,
  y = NULL,
  var_vec,
  sample_weights = NULL,
  counts = NULL,
  weighted_counts = NULL,
  bin_width = NULL,
  bin_number = NULL,
  values_discrete = c(0, 1),
  weights_discrete = 1,
  domain_continuous = c(0, 1)
)
```

Arguments

`data` Data set of type `data.frame` or `data.table` containing the observations (y_i, x_i) of response and covariates as well as optional sample weights (compare `sample_weights`) for each observation in the rows ($i = 1, ..., N$).

`y` Variable in `data` containing the response observations y_i. Either the variable name can be given as string or the column position of the variable in `data` as integer. If missing (`NULL`), if there is a unique column of `data` not specified in `var_vec`, `sample_weights`, `counts`, and `weighted_counts` (see below), this unique column is used.

`var_vec` Vector of variables of `data` on which the covariate combinations are based. The vector can either contain the variable names as strings or the column positions of the respective variables in `data`.

`sample_weights` (Optional) variable in `data` which contains a sample weight for each observation. Either the variable name can be given as string or the column position of the variable in `data` as integer. If missing (`NULL`), no sample weights are included per default (i.e., all observations have the same weight 1).

`counts` (Optional) variable in `data` which contains a count for each observation (for cases, where the available data contains counts instead of individual observations, which is common in particular for discrete data). Either the variable name can be given as string or the column position of the variable in `data` as integer. If `bin_number` and `bin_width` are both `NULL`, midpoints of unique observations y_i are used as boundaries of histogram bins (if I is not empty, i.e., if there is a continuous component), which are then used to compute the bin widths (which are later required es offset in the regression model). If argument `counts` is missing (`NULL`; default), counts are constructed from individual observations (which is equivalent to each observation counted once). Note that if `counts` and `sample_weights` are both not `NULL`, the latter is ignored (also indicated via a warning message). Please use `weighted_counts` (additionally to `counts`) to include possible weighted data.

`weighted_counts`

(Optional) variable in `data` which contains a weighted count for each observation (compare Appendix D of Maier et al. (2025b)). In this case, also absolute counts have to be specified via `counts`. Otherwise, `weighted_counts` is ignored. Either the variable name can be given as string or the column position of the variable in data as integer. If missing (NULL), counts are constructed from individual observations (which is equivalent to all observations counted once).

bin_width
: Width of histogram bins partitioning I. Can be one scalar value (specifying an equidistant bin width), or a vector containing the width of each bin. The combined length of the specified bins must match the length of the continuous part of the domain. Alternative to bin_number. If bin_number and bin_width are both given and the two values are not compatible, an error is returned.

bin_number
: Number of equidistant histogram bins partitioning I. Alternative to bin_width. If neither parameter is specified, bin_number = 100 is used as default. If bin_number and bin_width are both given and the two values are not compatible, an error is returned.

values_discrete
: Vector of values in $\mathcal{D}$ (the subset of the domain corresponding to the discrete part of the densities). Defaults to missing (NULL) in which case it is set to c(0,1). If set to FALSE, the discrete component is considered to be empty, i.e., the Lebesgue measure is used as reference measure (continuous special case).

weights_discrete
: Vector of weights for the Dirac measures corresponding to values_discrete. If missing (NULL) it is set to 1 in all components as default. Can be a scalar for equal weights for all discrete values or a vector with specific weights for each corresponding discrete value.

domain_continuous
: An interval (i.e., a vector of length 2) specifying I (the subset of the domain corresponding to the continuous part of the densities). If missing (NULL) it is set to c(0,1) as default. If set to FALSE, the continuous component is considered to be empty, i.e., a weighted sum of dirac measures is used as reference measure (discrete special case).

Value

The function returns an object of the class count_data, which is a data.table with columns:

- counts - For each by var_vec defined covariate combination the observed (histogram) counts in the first column.
- weighted_counts - If sample_weights is not NULL: Weighted (histogram) counts for the respective bin/discrete value incorporating the sample weights given by sample_weights.
- Name of variable given to y - Marks the mid of the respective histogram bin (for values in $I \setminus D$) or the discrete value. If a mid corresponds to a discrete value, the mid is shifted to the right by 0.0001 times the minimal distance to the next interval limit OR discrete value so that no mid is exactly corresponding to a discrete value. A warning message is generated in this case.
- Names of all variable columns which where specified by var_vec - These columns contain the values of the respective variables.
- group_id - ID of each covariate combination.
- gam_weights - Vector to be passed to argument weights in gam when fitting the Poisson Model, if data contains sample weights, see Appendix C of Maier et al. (2025b).
- gam_offset - Negative logarithm of gam_weights to be used as offset to the predictor of the Poisson Model, if data contains sample weights, see Appendix C of Maier et al. (2025b).
- Delta - Width of the histogram bin or weight of the Dirac measure for a discrete value defined by weights_discrete. The Poisson model uses offset(log(Delta)) to add the necessary additive term in the predictor that includes binwidths/dirac weights into the estimation.

- discrete - Logical value indicating whether the respective y is a discrete value in D.

Note that a plot-method for objects of class count_data is available via DensityRegression:::count_data, however, it is not exported, since it is not tested/documented appropriately, yet.

Author(s)

Lea Runge, Eva-Maria Maier

References

Maier, E.-M., Fottner, A., Stoecker, A., Okhrin, Y., & Greven, S. (2025b): Conditional density regression for individual-level data. arXiv preprint arXiv:XXXX.XXXXX.

See Also

densreg

Examples

```
set.seed(101)

# create data where 0 and 1 are the discrete observations, values
# equal 2 are replaced below by drawing from a beta distribution

data <- data.frame(obs_density = sample(0:2, 100, replace = TRUE,
                  prob = c(0.15, 0.1, 0.75)),
                  covariate1 = sample(c("a", "b"), 100, replace = TRUE),
                  covariate2 = sample(c("c", "d"), 100, replace = TRUE),
                  sample_weights = runif (100, 0, 2))
data[which(data$obs_density == 2), ]$obs_density <- rbeta(length(which(data$obs_density == 2)),
                                                          shape1 = 3, shape2 = 3)

# Create count dataset for data using 10 equidistant
# bins and default values for continuous domain, discrete
# values and discrete weights while considering a mixed case
# of continuous and discrete domains. The following function calls are
# equivalent:

data2counts(data, var_vec = c("covariate1", "covariate2"), y = "obs_density",
           sample_weights = "sample_weights", bin_number = 10)
data2counts(data, var_vec = c(2, 3), y = 1, sample_weights = 4, bin_width = 0.1)

# Use the vector bin_width to define non-equidistant bins and
# specify with values_discrete and weights_discrete discrete values
# and weights besides the default (0,1) and weight 1:

data2counts(data, var_vec = c(2, 3), y = 1, sample_weights = 4,
           bin_width = c(0.1, 0.5, 0.4), values_discrete = c(0, 1),
           weights_discrete = c(0.5, 2))

# The use of "values_discrete=FALSE" refers to a purely continuous setting,
# i.e., a usual histogram over the whole domain. The observations at 0 and 1
# are now counted towards the outer bins.

data2counts(data, var_vec = c(2, 3), y = 1, sample_weights = 4, bin_width = 0.1,
           values_discrete = FALSE)
```

```
# filter data set for only observations valued in discrete domain

dta_discrete <- data[which(data$obs_density %in% c(0, 1)), ]

# The use of "domain_continuous=FALSE" refers to a purely discrete setting:

data2counts(dta_discrete, var_vec = c(2, 3), y = 1, sample_weights = 4,
          bin_width = 0.1, values_discrete = c(0, 1),
          weights_discrete = c(0.5, 2), domain_continuous  = FALSE)

# use the optional argument counts to "data2counts" already preprocessed
# data and select only one of two variables for the grouping
dta_discrete <- unique(dta_discrete[, 1:3])
dta_discrete$counts <- sample(0:10, nrow(dta_discrete), replace = TRUE)

data2counts(dta_discrete, var_vec = "covariate1", y = "obs_density",
          counts = "counts", bin_width = 0.1, values_discrete = c(0, 1),
          weights_discrete = c(0.5, 2), domain_continuous  = FALSE)
```

---

densreg *Conditional density regression for individual-level data*

Description

This function implements the approach by Maier et al. (2025b) for fitting structured additive regression models with densities in a mixed (continuous/ discrete) Bayes Hilbert space $B^2(\mu) = B^2(\mathcal{Y}, \mathcal{A}, \mu)$ as response given scalar covariates x_i, based on observations y_i from the conditional distributions given x_i, via a (penalized) maximum likelihood approach. The (penalized) log-likelihood function is approximated via the (penalized) log-likelihood of an appropriate poisson regression model, which - after constructing the count data appropriately via `data2counts` - is fitted using mgcv's `gam` with a new smooth term bs="md" (see `smooth.construct.md.smooth.spec`) for mixed densities in $L^2_0(\mu)$. We briefly summarize the approach below in the details. Please see Maier et al. (2025b) for comprehensive description.

Usage

```
densreg(
  data,
  y = NULL,
  var_vec = NULL,
  sample_weights = NULL,
  counts = NULL,
  weighted_counts = NULL,
  values_discrete = c(0, 1),
  weights_discrete = 1,
  domain_continuous = c(0, 1),
  bin_number = NULL,
  bin_width = NULL,
  m_continuous = c(2, 2),
  k_continuous = 10,
  sp_y = NULL,
```

```
    method = "REML",
    penalty_discrete = NULL,
    group_specific_intercepts = NULL,
    linear_effects = NULL,
    smooth_effects = NULL,
    varying_coefficients = NULL,
    smooth_interactions = NULL,
    ...
)
```

Arguments

`data` Data set of type `data.frame` or `data.table` containing the observations (y_i, x_i) of response and covariates as well as optional sample weights (compare `sample_weights`) for each observation in the rows ($i = 1, ..., N$).

`y` Variable in `data` containing the response observations y_i. Either the variable name can be given as string or the column position of the variable in `data` as integer. If missing (`NULL`), if there is a unique column of `data` not specified in `sample_weights`, `counts`, and `weighted_counts` and not used as covariate during effect specification (see below), this unique column is used.

`var_vec` (Optional) vector of covariates of `data` as passed to `data2counts`, specifying the order of the covariate combinations. Must match the covariates used for effect specification (see below). The vector can either contain the variable names as strings or the column positions of the respective variables in `dta`. If missing (`NULL`) order of effects as appearing in `group_specific_intercepts`, `linear_effects`, `smooth_effects`, `varying_coefficients`, `smooth_interactions` is used.

`sample_weights` (Optional) variable in `data` which contains a sample weight for each observation. Either the variable name can be given as string or the column position of the variable in `data` as integer. If missing (`NULL`), no sample weights are included per default (i.e., all observations have the same weight 1).

`counts` (Optional) variable in `data` which contains a count for each observation (for cases, where the available data contains counts instead of individual observations, which is common in particular for discrete data). Either the variable name can be given as string or the column position of the variable in `data` as integer. If `bin_number` and `bin_width` are both `NULL`, midpoints of unique observations y_i are used as boundaries of histogram bins (if I is not empty, i.e., if there is a continuous component), which are then used to compute the bin widths (which are later required es offset in the regression model). If argument `counts` is missing (`NULL`; default), counts are constructed from individual observations (which is equivalent to each observation counted once). Note that if `counts` and `sample_weights` are both not `NULL`, the latter is ignored (also indicated via a warning message). Please use `weighted_counts` (additionally to `counts`) to include possible weighted data.

`weighted_counts` (Optional) variable in `data` which contains a weighted count for each observation (compare Appendix D of Maier et al. (2025b)). In this case, also absolute counts have to be specified via `counts`. Otherwise, `weighted_counts` is ignored. Either the variable name can be given as string or the column position of the variable in `data` as integer. If missing (`NULL`), counts are constructed from individual observations (which is equivalent to all observations counted once).

values_discrete
: Vector of values in $\mathcal{D}$ (the subset of the domain corresponding to the discrete part of the densities). Defaults to missing (NULL) in which case it is set to c(0,1). If set to FALSE, the discrete component is considered to be empty, i.e., the Lebesgue measure is used as reference measure (continuous special case).

weights_discrete
: Vector of weights for the Dirac measures corresponding to values_discrete. If missing (NULL) it is set to 1 in all components as default. Can be a scalar for equal weights for all discrete values or a vector with specific weights for each corresponding discrete value.

domain_continuous
: An interval (i.e., a vector of length 2) specifying I (the subset of the domain corresponding to the continuous part of the densities). If missing (NULL) it is set to c(0,1) as default. If set to FALSE, the continuous component is considered to be empty, i.e., a weighted sum of dirac measures is used as reference measure (discrete special case).

bin_number
: Number of equidistant histogram bins partitioning I. Alternative to bin_width. If neither parameter is specified, bin_number = 100 is used as default. If bin_number and bin_width are both given and the two values are not compatible, an error is returned.

bin_width
: Width of histogram bins partitioning I. Can be one scalar value (specifying an equidistant bin width), or a vector containing the width of each bin. The combined length of the specified bins must match the length of the continuous part of the domain. Alternative to bin_number. If bin_number and bin_width are both given and the two values are not compatible, an error is returned.

m_continuous
: Vector of two integers specifying the order of the B-spline basis over I and the order of the difference penalty (like the argument m for P-Spline smooth terms bs = "ps" in ti, etc.) for basis functions in $L^2(\lambda)$, before transformation to $L_0^2(\lambda)$ (compare details). If missing it is set to cubic splines with second order difference penalty, i.e., m_continuous = c(2,2), as default.

k_continuous
: Integer specifying the number of B-spline basis functions in $L^2(\lambda)$ (like the argument k for P-Spline smooth terms bs = "ps" in ti, etc.), before transformation to $L_0^2(\lambda)$ (compare details). Note that the transformation reduces the basis number by 1. The basis $b_{\mathcal{Y}} b_Y$ in the mixed Bayes Hilbert space $B^2(\mu)$ will thus have k_continuous -1 + length(values_discrete) elements. See also choose.k for more information on choosing this parameter. If missing (NULL) it is set to 10.

sp_y
: Integer or vector specifying the smoothing parameter for the marginal penalty matrix for $b_{\mathcal{Y}} b_Y$ (anisotropic penalty; like the argument sp in ti, etc.). If a vector is submitted, its length must be the number of partial effects (including the intercept), with the j-th entry specifying the smoothing parameter for the j-th partial effect. The order of parameters within this vector corresponds to the intercept followed by the partial effects in the order as specified below (group_specific_intercepts, linear_effects, smooth_effects, varying_coefficients, and smooth_interactions). Upenalized estimation is accomplished by setting sp_y to zero. If missing (NULL) or a negative value is supplied, the parameter will be estimated by gam via the estimation method specified in method (with REML-estimation per default). Any positive or zero value is treated as fixed (see gam).

method
: String characterizing the smoothing parameter estimation method as in gam. Defaults to "REML" (REML-estimation).

`penalty_discrete`
: Integer or `NULL` giving the order of differences to be used for the penalty of the discrete component, with 0 corresponding to the identity matrix as penalty matrix (analogously to m[2] for the continuous component); Note that the order of differences must be smaller than the number of values in the discrete component, i.e., the length of `values_discrete`.

  If missing (`NULL`), the discrete component is estimated unpenalized (in the mixed case, by setting the discrete component of the penalty matrix to zero - with the continuous component non-zero -, in the discrete case by setting the smoothing parameter to zero (with a diagonal penalty matrix) since due to technical reasons it is not possible to set the whole penalty matrix to zero).

`group_specific_intercepts`
: Vector of the form `c("x_a","x_b",...)` of names of categorical covariates, i.e., `factor` variables contained in `data` for which group-specific intercepts $\beta_{x_a}, \beta_{x_b}, \ldots$ (one per category of the respective covariate) shall be estimated. For `ordered` factors, the first level is used as reference category (see `gam.models` for more details). If missing (`NULL`), no group-specific intercept is included.

`linear_effects`
: Vector of the form `c("x_a","x_b",...)` of names of numeric covariates contained in `data` for which linear effects $x_a \odot \beta_{x_a}, x_b \odot \beta_{x_b}, \ldots$ shall be estimated. If missing (`NULL`), no linear effect is included.

`smooth_effects`
: List of (named) lists of the form `list(list(cov = "x_a",bs = bs_a,m = m_a,k = k_a,mc = mc_a,by = "x_b"),...)`. If the lists are unnamed, the names are assigned in the order of the given elements. The list is filled with `NULL` if it contains less than 6 elements. Each list is adding one (group-specific) smooth effect of the form $g_{x_b}(x_a)$ to the model with:

  - `cov`: Name of a numeric covariate contained in `data`.
  - `bs`: Character string specifying the type for the marginal basis in covariate direction. See `smooth.terms` for details and full list. If not specified or `NULL`, a P-spline basis `"ps"` is used.
  - `m`: Vector of two integers or `NULL` giving the order of the marginal spline basis b_j in covariate direction and the order of its penalty (as in `ti`). If not specified or `NULL`, `c(2,2)` is used.
  - `k`: Integer or `NULL` giving the dimension of the marginal basis b_j (as in `ti`). See `choose.k` for more information. If not specified or `NULL`, `10` is used.
  - `mc`: Logical indicating if the marginal in covariate direction should have centering constraints applied. By default all marginals are constrained, i.e., `mc = TRUE`.
  - `by`: Optional, name of a categorical covariate, i.e., `factor` variable contained in `data` specifying whether the smooth effect should be modeled specifically for each level of the by-covariate (group-specific). For `ordered` factors, the first level is used as reference category (see `gam.models` for more details). If missing or `NULL`, the smooth effect is not depending on the level of an additional covariate.

  If missing (`NULL`), no (group-specific) smooth effect is included.

`varying_coefficients`
: List of lists of the form `list(list(cov = "x_a",by = "x_b",bs = bs_a,m = m_a,k = k_a,mc = mc_a),...)`. If the lists are unnamed, the names are assigned in the order of the given elements. The list is filled with `NULL` if it contains less than 6 elements. Each list is adding one varying coefficient of the form $x_b \odot g(x_a)$ to the model with:

- `cov`: Name of a numeric covariate contained in `data`.
- `by`: Name of a numeric covariate contained in `data`.
- `bs`: Character string specifying the type for the marginal basis in covariate direction. See `smooth.terms` for details and full list. If not specified or `NULL`, a P-spline basis `"ps"` is used.
- `m`: Vector of two integers or `NULL` giving the order of the marginal spline basis b_j in covariate direction and the order of its penalty (as in `ti`). If not specified or `NULL`, `c(2,2)` is used
- `k`: Integer or `NULL` giving the dimension of the marginal basis b_j (as in `ti`). See `choose.k` for more information. If not specified or `NULL`, `10` is used.
- `mc`: Logical indicating if the marginal in the direction of the first covariate should have centering constraints applied. By default all marginals are constrained, i.e., `mc = TRUE`.

If missing (`NULL`), no varying coeffecient is included.

`smooth_interactions`
: List of lists of the form `list(list(covs = c("x_a","x_b",...),bs = c(bs_a,bs_b,...),m = list(m_a,m_b,...),k = c(k_a,k_b,...),mc = c(mc_a,mc_b,...),by = "x_by"),list(...),` If the lists are unnamed, the names are assigned in the order of the given elements. The list is filled with vectors/lists of `NULL` if it contains less than 6 elements. Each list is specifying a (group-specific) smooth interaction effect between at least two continuous covariates of the form $g_{x_b y}(x_a, x_b, ...)$ in the model with:

  - `covs`: Vector of names of numeric covariates contained in `data`.
  - `bs`: Vector of character strings specifying the types for each marginal basis of each covariate. See `smooth.terms` for details and full list. If not specified or `NULL`, a P-spline basis `"ps"` is used.
  - `m`: List containing vectors of two integers or `NULL` giving the order of the marginal spline basis in direction of the respective covariate in `covs` for the smooth effect and the order of its penalty (as in `ti`). If not specified or `NULL`, a list of `c(2,2)` is used.
  - `k`: Vector of integers or `NULL` giving the dimension of the marginal basis in direction of the respective covariate in `covs` for the smooth effect. See `choose.k` for more information. If not specified or `NULL`, a vector specifying each parameter as `10` is used.
  - `mc`: Logical vector indicating if the marginals in covariate direction should have centering constraints applied. By default all marginals are constrained, i.e., `TRUE`.
  - `by`: Optional, name of a categorical covariate, i.e., `factor` variable contained in `data` specifying whether the smooth interaction effect should be modeled specifically for each level of the by-covariate (group-specific). For `ordered` factors, the first level is used as reference category (see `gam.models` for more details). If missing or `NULL`, the smooth interaction effect is not depending on the level of an additional covariate.

  If missing (`NULL`), no (group-specific) smooth interaction is included.

`...`
: further arguments for passing on to `gam`.

Details

The function `densreg` estimates the densities f_{x_i} of conditional distributions of random variables $Y_i \mid x_i$ from independent observations (y_i, x_i), where y_i are realizations of $Y_i \mid x_i$, and x_i is a vector of covariate observations, $i = 1, ..., N$. The densities are considered as elements of a mixed Bayes Hilbert space $B^2(\mu) = B^2(\mathcal{Y}, \mathcal{A}, \mu)$ (including continuous and discrete ones as special cases). The subset of the domain $\mathcal{Y}$ corresponding to the continuous part of the densities is denoted with I, the one corresponding to the discrete part with $\mathcal{D}$. The densities are modeled via a structured additive regression model

$$f_{x_i} = \bigoplus_{j=1}^{J} h_j(x_i)$$

with partial effects $h_j(x_i) \in B^2(\mu)$ depending on no, one, or several covariates x_i. Each partial effect is represented using a tensor product basis, consisting of an appropriate vector of basis functions $b_{\mathcal{Y}} b_Y$ in $B^2(\mu)$ over the domain of $B^2(\mu)$, and a vector of basis functions $b_{\mathcal{X},j}$ over the respective covariates. To obtain basis functions $b_{\mathcal{Y}}$ consider the orthogonal decomposition of the mixed Bayes Hilbert space $B^2(\mu)$ into a discrete Bayes Hilbert space $B^2(\delta^\bullet)$ and a continuous Bayes Hilbert space $B^2(\lambda)$ developed in Section 3.4 of Maier et al. (2025a). Note that for the domain of the discrete Bayes Hilbert space, the discrete part $\mathcal{D}$ has to be extended by an additional arbitrary discrete value (for the discrete summary of the continuous part). We construct basis functions in both these spaces by transforming appropriate basis functions in the corresponding (unconstrained) L^2 spaces (more precisely, indicator functions with optional difference penalty for the discrete and a P-spline basis for the continuous Bayes Hilbert space), to the respective L^2_0 spaces, i.e., the subspaces of L^2 containing only functions that integrate to zero.

See appendix D of Maier et al. (2025a) for details on this transformation. Applying the inverse centered log-ratio (clr) transformation (which is an isometric isomorphism, allowing to consider densities in some B^2 space equivalently in the respective L^2_0 space) yields one basis in $B^2(\delta^\bullet)$ and one in $B^2(\lambda)$. The basis in the actually considered mixed Bayes Hilbert space $B^2(\mu)$ is then obtained by applying the respective embeddings to these two bases (compare Section 3.4 of Maier et al. (2025a) and Section 2.2 of Maier et al. (2025b)). The choice of $b_{\mathcal{X},j}$ determines the type of the partial effect, e.g., linear or smooth for a continuous covariate. For smooth effects, the same marginal bases as in `gam` can be used. In particular, penalization is also possible. The resulting log-likelihood is then approximated by the log-likelihood of a corresponding multinomial model, which can equivalently be estimated by a Poisson model. The data for these models is obtained from the original observations $y_1, ..., y_N$ by combining all observations of the same conditional distribution (i.e., all observations sharing identical values in all covariates) into a vector of counts via a histogram on the continuous part of the domain of the densities and counts on the discrete part of the domain. For details, see Maier et al. (2025b). `densreg` calls `data2counts` to constructs the respective count data from the individual observations and estimates the corresponding Poisson model via `gam`. Furthermore, the resulting (clr transformed) densities as well as the estimated (clr transformed) partial effects are calculated (using `clr`).

Value

The function returns an object of the class `densreg`, which is a `list` with elements:

- `f_hat`: Matrix containing the estimated conditional densities $\hat{f}$ (evaluated at the bin midpoints and discrete values respectively, compare row names and `domain_data`) for the different covariate combinations as columns, ordered according to `group_id`, compare column names and `covariate_data`. Computed by applying `clr` to `f_hat_clr` column-wise.
- `f_hat_clr`: Matrix containing the estimated conditional clr transformed densities $clr[\hat{f}]$ (evaluated at the bin midpoints and discrete values respectively, compare `obs_density` in `count_data`) for the different covariate combinations as columns, ordered according to `group_id`, compare column names and `covariate_data`.
- `effects`: List with elements `estimated_effects_clr` and `estimated_effects` containing the (clr transformed) estimated partial effects.

- `count_data`: `count_data`-object containing the count data obtained by applying `data2counts` to `data`.
- `covariate_data`: `data.table` giving an overview over the assignment of the unique covariate combinations to the group IDs.
- `domain_data`: `data.table` giving an overview over the binning of the domain `Ycal` of the density for the construction of `count_data`.
- `model_matrix`: Model matrix.
- `theta_hat`: Estimated coeffecient vector $\hat{\theta}$.
- `params`: List of `domain_continuous`,`values_discrete` and `bin_number` as given to the function.
- `specified_effects`: List of lists (`group_specific_intercepts`,`smooth effects`,`linear_effects`,`varying_` collecting the specification of all partial effects as given to the function in the respective parameters.
- `model`: `gam`-object corresponding to the estimated Poisson model.

Note that `plot`- and `predict`-methods for objects of class `densreg` are available via `DensityRegression:::plot.dens` and `DensityRegression:::predict.densreg()`, however, they are not exported, since some functionality (like plots on effect level) are not working properly, and they are not tested/documented appropriately, yet.

Author(s)

Lea Runge, Eva-Maria Maier

References

Maier, E.-M., Fottner, A., Stoecker, A., Okhrin, Y., & Greven, S. (2025b): Conditional density regression for individual-level data. arXiv preprint arXiv:XXXX.XXXXX.

Maier, E.-M., Stoecker, A., Fitzenberger, B., Greven, S. (2025a): Additive Density-on-Scalar Regression in Bayes Hilbert Spaces with an Application to Gender Economics. Annals of Applied Statistics, 19(1), ???-???.

Examples

```
### Note that the following simulated data are only to illustrate
### function usage and do not possess significant covariate effects

# for further information on the parameters of the preprocessing step see ?data2counts

# create data for the mixed case
set.seed(101)

data <- data.frame(obs_density = sample(0:2, 150, replace = TRUE, prob = c(0.15, 0.1, 0.75)),
                   covariate1 = sample(c("a", "b", "c"), 150, replace = TRUE),
                   covariate2 = sample(c("c", "d"), 150, replace = TRUE),
                   covariate3 = rep(rnorm(n = 15), 10),
                   covariate4 = rep(rnorm(n = 10), 15),
                  covariate5 = rep(rnorm(n = 10), 15), sample_weights = runif (150, 0, 2))
data[which(data$obs_density == 2), ]$obs_density <- rbeta(length(which(data$obs_density== 2)),
                                                          shape1 = 3, shape2 = 3)
data$covariate1 <- ordered(data$covariate1)
data$covariate2 <- ordered(data$covariate2)
```

```
# create discrete data

dta_dis <- data.frame(obs_density = sample(0:2, 150, replace = TRUE, prob = c(0.25, 0.45, 0.3)),
                      covariate1 = sample(c("a", "b", "c"), 150, replace = TRUE),
                      covariate2 = sample(c("c", "d"), 150, replace = TRUE),
                      covariate3 = rep(rnorm(n = 15), 10),
                      covariate4 = rep(rnorm(n = 10), 15),
                      covariate5 = rep(rnorm(n = 10), 15),
                      sample_weights = runif (150, 0, 2))
dta_dis$covariate1 <- ordered(dta_dis$covariate1)
dta_dis$covariate2 <- ordered(dta_dis$covariate2)

# examples for different partial effects

## group specific intercepts
group_specific_intercepts <- c("covariate1", "covariate2")
## linear effects
linear_effects <- c("covariate4")
## smooth effects
smooth_effects <- list(list(cov = "covariate3", bs = "ps", m = c(2, 2), k = 4),
                       list(cov = "covariate3", bs = "ps", m = c(2, 2), k = 4,
                            mc = FALSE, by = "covariate1"))
## varying coefficient
varying_coef <- list(list(cov = "covariate3", by = "covariate4", bs = "ps", m = c(2, 2) , k = 4))
## smooth interaction
smooth_inter <- list(list(covs = c("covariate3", "covariate4", "covariate5"),
                          bs = c("ps", "ps", "ps"),
                          m = list( c(2, 2), c(2, 2), c(2, 2)), k = c( 4, 4, 5),
                          mc = c(TRUE, FALSE, TRUE), by = NULL))

# fit models (warning: calculation may take a few minutes)

## fit model for the mixed case with group specific intercepts and linear effects
### use fixed smoothing parameters in density direction

m_mixed <- densreg(data = data, y = 1, m_continuous = c(2, 2),
  k_continuous = 4, group_specific_intercepts = group_specific_intercepts,
  linear_effects = linear_effects, sp_y = c(1, 3, 5, 0.5))

## fit model for the discrete case with smooth effects and smooth interaction

m_dis <- densreg(
  data = dta_dis, y = 1, values_discrete = c(0, 1, 2),
  weights_discrete = c(1, 1, 1), domain_continuous = FALSE, m_continuous = c(2, 2),
  k_continuous = 4, group_specific_intercepts = group_specific_intercepts,
  smooth_effects = smooth_effects, smooth_interactions = smooth_inter)

# fit model for the continuous case with a functional varying coeffecient

m_cont <- densreg(data = data[which(!(data$obs_density %in% c(0, 1))), ],
  y = 1, values_discrete = FALSE, m_continuous = c(2, 2),
  k_continuous = 12, varying_coefficients = varying_coef)
```

Predict.matrix.mdspline.smooth

Predict matrix method function for mixed density smooth

Description

Predict.matrix method function for smooth class mdspline.smooth to enable prediction from a model fitted with mgcv's gam.

Usage

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

Arguments

object	a smooth specification object, usually generated by a term ti(x,bs="md",...)
data	A data frame containing the values of the (named) covariates at which the smooth term is to be evaluated. Exact requirements are as for smooth.construct and smooth.construct2

Details

The Predict matrix function is not normally called directly, but is rather used internally by mgcv's predict.gam etc. to predict from a fitted gam model. See Predict.matrix for more details, or smooth.construct.md.smooth.spec for details on the mixed density smooth "md".

Value

A matrix mapping the coefficients for the smooth term to its values at the supplied data values.

Author(s)

Eva-Maria Maier, Alexander Fottner

References

Maier, E.-M., Fottner, A., Stoecker, A., Okhrin, Y., & Greven, S. (2025b): Conditional density regression for individual-level data. arXiv preprint arXiv:XXXX.XXXXX.

Examples

```
### create data
set.seed(101)

N <- 100
dta <- data.frame(covariate1 = sample(c("a", "b", "c"), N, replace = TRUE),
                  covariate2 = rnorm(n = N))
dta$obs_density <- sapply(seq_len(N),
                          function(i) {
                            a_0 <- ifelse(dta$covariate1[i] == "a", 0.1,
                                          ifelse(dta$covariate1[i] == "b", 0.2, 0.3))
```

```
                                p_0 <- a_0 * sin(dta$covariate2[i]) + a_0 + 0.05
                                a_1 <- ifelse(dta$covariate1[i] == "a", 0.25,
                                              ifelse(dta$covariate1[i] == "b", 0.15, 0.05))
                                p_1 <- a_1 * cos(dta$covariate2[i]) + a_1 + 0.1
                                sample(0:2, 1, prob = c(p_0, p_1, 1 - p_0 - p_1))
                              })

dta$covariate1 <- ordered(dta$covariate1)

dta_mixed <- dta
ind_cont <- which(dta_mixed$obs_density == 2)
dta_mixed[ind_cont, ]$obs_density <-
  sapply(seq_along(ind_cont), function(i)
    rbeta(1, shape1 = 1 + exp(dta_mixed$covariate2[i]),
          shape2 = 1 + as.numeric(dta_mixed$covariate1[i])))
n_bins <- 20
dta_mixed <- data2counts(data = dta_mixed, var_vec = c("covariate1", "covariate2"),
                        bin_number = n_bins, values_discrete = c(0, 1),
                        domain_continuous = c(0, 1))

### fit model
m_mixed <- gam(counts ~
                 # no scalar global intercept (we add a density-intercept instead)
                 - 1 +
                 # intercept (corresponding to reference covariate1 = "a")
                 ti(obs_density, bs = "md", m = list(c(2, 2)), k = 8,
                    mc = FALSE, np = FALSE) +
                 # group specific intercept for leveles "b" and "c" of covariate2
                 ti(obs_density, bs = "md", m = list(c(2, 2)), k = 8,
                    mc = FALSE, np = FALSE, by = covariate1) +
                 # smooth effect of covariate2 (modeled via P-splines)
                 ti(covariate2, obs_density, bs = c("ps", "md"),
                    m = list(c(2, 2), c(2, 2)), k = c(8, 8), mc = c(TRUE, FALSE),
                    np = FALSE) +
                 # intercepts per covariate combination (modeling absolute
                 # counts for poisson regression)
                 as.factor(group_id) +
                 # offsets: log(Delta) accounting for bin width, gam_offsets
                 # for weighted observations
                 offset(log(Delta)),
               family = poisson(), method = "REML", data = dta_mixed)

### Functions using S3 Predict.matrix-method for class 'mdspline.smooth' are, e.g.,
### model.matrix and predict

# Using model.matrix (and internally Predict.matrix.mdspline.smooth) to extract
# design matrix and construct estimated densities thereof
X <- model.matrix(m_mixed)
# The design matrix includes intercepts per covariate combination modeling the
# absolute counts for the Poisson model, which are not of interest on density-level.
# Thus, we remove them.
intercepts <- which(grepl("group_id", colnames(X)))
X <- X[, -intercepts]
theta_hat <- m_mixed$coefficients[-intercepts]
# compute estimated conditional clr-transformed densities
f_hat_clr <- matrix(c(X %*% theta_hat), nrow = length(unique(dta_mixed$obs_density)))
f_hat <- apply(f_hat_clr, 2,
```

```
                clr, w = c(1, rep(1/n_bins, n_bins), 1), inverse = TRUE)
t <- unique(dta_mixed$obs_density)
matplot(t[2:(length(t) - 1)], f_hat[2:(length(t) - 1), ], type = "l",
        col = rainbow(ncol(f_hat)), lty = rep(1:5, ceiling(ncol(f_hat) / 5)),
        ylim = range(f_hat))
matpoints(t[c(1, length(t))], f_hat[c(1, length(t)), ],
          col = rainbow(ncol(f_hat)), pch = 1)

# Using predict (and internally Predict.matrix.mdspline.smooth) to extract
# estimated partial effects
pred_terms <- predict(m_mixed, type = "terms")
```

---

smooth.construct.md.smooth.spec

P-Splines with integrate-to-zero constraint for density regression

---

Description

Specifying bs = "md" in the tensor product smooth ti (setting mc = FALSE) constructs a smoother to be used in mgcv's gam with family = poisson() for the case that the response variable of interest is a density in a mixed Bayes Hilbert space $B^2(\mu) = B^2(\mathcal{Y}, \mathcal{A}, \mu)$ (including continuous or discrete ones as special cases) as in Maier et al. (2025b). The subset of the domain $\mathcal{Y}$ corresponding to the continuous part of the densities is denoted with I, the one corresponding to the discrete part with $\mathcal{D}$. Corresponding covariate observations are the discrete values for the discrete component and the midpoints of the bins underlying the histograms approximating the densities for the continuous component. Response observations are the counts of the discrete observations and the histograms (the count data can easily be constructed from individual data with data2counts). Specification of basis and penalization details for the continuous component is as for bs = "ps", see mgcv's smooth.construct.ps.smooth.spec, which our implementation is oriented on. Specification of the domain and for the discrete component is possible via the argument xt in ti (see below).

Note that the implementation includes the integrate-to-zero constraint of L^2_0, i.e., further centering is not reasonable! This means, that the constructor should not be used with s or te, which always include centering (sum-to-zero) constraints for all marginals. Instead, it is supposed to be used with ti, setting mc = FALSE for the corresponding component.

We recommend to use link{densreg} to specify density regression models (based on smooth.construct.md.smooth.s instead of via gam directly, since the is quite cumbersome and specification has to be done with extreme care to obtain a reasonable model.

Usage

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

Arguments

object	a smooth specification object, usually generated by a term ti(x,bs="md",...)
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`. See details for further information.

Details

The basis and penalty are constructed from a P-spline basis (continuous component) respectively indicator functions with optional difference penalty (discrete component) transformed to L_0^2 as described in Appendix D of Maier et al. (2025a) and embedded to the mixed Bayes Hilbert space as described in Section 2.2 of Maier et al. (2025b). The argument `xt` in `ti` is used for further specification regarding the underlying Bayes Hilbert space. `xt` has to be a list of the same length as the vector `bs` specifying the marginal bases. For all `"md"` type marginal bases, the corresponding `xt`-list element is again a list with the following elements:

- `values_discrete`: Vector of values in $\mathcal{D}$ (the subset of the domain corresponding to the discrete part of the densities). Defaults to missing (`NULL`) in which case it is set to `c(0,1)`. If set to `FALSE`, the discrete component is considered to be empty, i.e., the Lebesgue measure is used as reference measure (continuous special case).
- `weights_discrete`: Vector of weights for the Dirac measures corresponding to `values_discrete`. If missing (`NULL`) it is set to 1 in all components as default. Can be a scalar for equal weights for all discrete values or a vector with specific weights for each corresponding discrete value.
- `domain_continuous`: An interval (i.e., a vector of length 2) specifying I. If missing (`NULL`) it is set to `c(0,1)` as default. If set to `FALSE`, the continuous component is considered to be empty, i.e., a weighted sum of dirac measures is used as reference measure (discrete special case).
- `penalty_discrete`: integer (or `NULL`) giving the order of differences to be used for the penalty of the discrete component, with `0` corresponding to the identity matrix as penalty matrix (analogously to `m[2]` for the continuous component); Note that the order of differences must be smaller than the number of values in the discrete component, i.e., the length of `values_discrete`; if missing (`NULL`), in the mixed case, it is set to a zero matrix (corresponding to no penalization), while in the discrete case, it is set to a diagonal matrix (as in a ridge penalty). In the discrete case, please set the corresponding argument of `sp` in `ti` to `0` to estimate an unpenalized model.

Value

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

Author(s)

Eva-Maria Maier, Alexander Fottner

References

Maier, E.-M., Fottner, A., Stoecker, A., Okhrin, Y., & Greven, S. (2025b): Conditional density regression for individual-level data. arXiv preprint arXiv:XXXX.XXXXX.

Maier, E.-M., Stoecker, A., Fitzenberger, B., Greven, S. (2025a): Additive Density-on-Scalar Regression in Bayes Hilbert Spaces with an Application to Gender Economics. Annals of Applied Statistics, 19(1), ???-???.

Examples

```
### create data
set.seed(101)

N <- 100
dta <- data.frame(covariate1 = sample(c("a", "b", "c"), N, replace = TRUE),
                  covariate2 = rnorm(n = N))
dta$obs_density <- sapply(seq_len(N),
                          function(i) {
                            a_0 <- ifelse(dta$covariate1[i] == "a", 0.1,
                                          ifelse(dta$covariate1[i] == "b", 0.2, 0.3))
                            p_0 <- a_0 * sin(dta$covariate2[i]) + a_0 + 0.05
                            a_1 <- ifelse(dta$covariate1[i] == "a", 0.25,
                                          ifelse(dta$covariate1[i] == "b", 0.15, 0.05))
                            p_1 <- a_1 * cos(dta$covariate2[i]) + a_1 + 0.1
                            sample(0:2, 1, prob = c(p_0, p_1, 1 - p_0 - p_1))
                          })

dta$covariate1 <- ordered(dta$covariate1)

# data for the mixed case
dta_mixed <- dta
ind_cont <- which(dta_mixed$obs_density == 2)
dta_mixed[ind_cont, ]$obs_density <-
  sapply(seq_along(ind_cont), function(i)
    rbeta(1, shape1 = 1 + exp(dta_mixed$covariate2[i]),
          shape2 = 1 + as.numeric(dta_mixed$covariate1[i])))
n_bins <- 20
dta_mixed <- data2counts(data = dta_mixed, var_vec = c("covariate1", "covariate2"),
                        bin_number = n_bins, values_discrete = c(0, 1),
                        domain_continuous = c(0, 1))

# data for the continuous case
dta_cont <- dta_mixed[which(!dta_mixed$discrete), ]

# data for the discrete case
dta_dis <- data2counts(data = dta, c("covariate1", "covariate2"),
                      values_discrete = c(0, 1, 2), domain_continuous = FALSE)

### fit model in the mixed case
# All marginals in density-direction are specified via the mixed density basis "md",
# which per default uses a mixed Bayes Hilbert space with continuous domain (0, 1)
# and discrete values at 0 and 1. Since this is exactly our scenario, we don't
# actually need to specify xt here.

m_mixed <- gam(counts ~
                 # no scalar global intercept (we add a density-intercept instead)
                 - 1 +
                 # intercept (corresponding to reference covariate1 = "a")
                 ti(obs_density, bs = "md", m = list(c(2, 2)), k = 8,
                    mc = FALSE, np = FALSE) +
                 # group specific intercept for leveles "b" and "c" of covariate2
                 ti(obs_density, bs = "md", m = list(c(2, 2)), k = 8,
                    mc = FALSE, np = FALSE, by = covariate1) +
                 # smooth effect of covariate2 (modeled via P-splines)
                 ti(covariate2, obs_density, bs = c("ps", "md"),
```

```
                 m = list(c(2, 2), c(2, 2)), k = c(8, 8), mc = c(TRUE, FALSE),
                 np = FALSE) +
              # intercepts per covariate combination (modeling absolute
              # counts for poisson regression)
              as.factor(group_id) +
              # offsets: log(Delta) accounting for bin width, gam_offsets
              # for weighted observations
              offset(log(Delta)),
            family = poisson(), method = "REML", data = dta_mixed)

### fit model in the continuous case
# Continuous domain (0, 1) without discrete values
xt_c <- list(values_discrete = FALSE, domain_continuous = c(0, 1))

m_cont <- gam(counts ~
                # no scalar global intercept (we add a density-intercept instead)
                - 1 +
                # intercept (corresponding to reference covariate1 = "a")
                ti(obs_density, bs = "md", m = list(c(2, 2)), k = 8,
                   mc = FALSE, np = FALSE, xt = list(xt_c)) +
                # group specific intercept for leveles "b" and "c" of covariate2
                ti(obs_density, bs = "md", m = list(c(2, 2)), k = 8,
                   mc = FALSE, np = FALSE, xt = list(xt_c), by = covariate1) +
                # smooth effect of covariate2 (modeled via P-splines)
               ti(covariate2, obs_density, bs = c("ps", "md"), m = list(c(2,2), c(2,2)),
                   k = c(8, 8), mc = c(TRUE, FALSE), np = FALSE,
                   xt = list(NULL, xt_c)) +
                # intercepts per covariate combination (modeling absolute
                # counts for poisson regression)
                as.factor(group_id) +
                # offsets: log(Delta) accounting for bin width, gam_offsets
                # for weighted observations
                offset(log(Delta)),
              family = poisson(), method = "REML", data = dta_cont)

### fit model in the discrete case
# Continuous domain empty, discrete values 0, 1, 2
xt_d <- list(domain_continuous = FALSE, values_discrete = c(0, 1, 2))

m_dis <- gam(counts ~
               # no scalar global intercept (we add a density-intercept instead)
               - 1 +
               # intercept (corresponding to reference covariate1 = "a")
               ti(obs_density, bs = "md", m = list(c(2, 2)), k = 8,
                  mc = FALSE, np = FALSE, xt = list(xt_d), sp = 0) +
               # group specific intercept for leveles "b" and "c" of covariate2
               ti(obs_density, bs = "md", m = list(c(2, 2)), k = 8,
                  mc = FALSE, np = FALSE, xt = list(xt_d), sp = 0, by = covariate1) +
               # smooth effect of covariate2 (modeled via P-splines)
               ti(covariate2, obs_density, bs = c("ps", "md"), m = list(c(2,2), c(2,2)),
                  k = c(8, 8), mc = c(TRUE, FALSE), np = FALSE,
                  xt = list(NULL, xt_d), sp = c(-1, 0)) +
               # intercepts per covariate combination (modeling absolute
               # counts for poisson regression)
               as.factor(group_id) +
               # offsets: log(Delta) accounting for bin width, gam_offsets
               # for weighted observations
```

```
  offset(log(Delta)),
family = poisson(), method = "REML", data = dta_dis)
```

Index

choose.k, *10–12*
clr, 3, *13*

data.frame, *5*, *9*
data.table, *5, 6*, *9*
data2counts, *2*, 4, *8, 9*, *13, 14*, *18*
DensityRegression
(DensityRegression-package), 2
DensityRegression-package, 2
DensityRegression_package
(DensityRegression-package), 2
densreg, *2, 3*, *7*, 8

factor, *11, 12*

gam, *2*, *4*, *6*, *8*, *10*, *12–14*, *16*, *18*
gam.models, *11, 12*

list, *13*

ordered, *11, 12*

package-DensityRegression
(DensityRegression-package), 2
predict.gam, *16*
Predict.matrix, *16*
Predict.matrix.mdspline.smooth, 16

s, *18*
smooth.construct, *16*, *19*
smooth.construct.md.smooth.spec, *2*, *8*,
16, 18
smooth.construct.ps.smooth.spec, *18*
smooth.terms, *11, 12*

te, *18*
ti, *10–12*, *18, 19*