Package 'bases'

May 7, 2026

Title Basis Expansions for Regression Modeling

Version 0.2.0

Description Provides various basis expansions for flexible regression modeling, including random Fourier features (Rahimi & Recht, 2007) <https://proceedings.neurips.cc/paper_files/paper/2007/file/013a006f03dbc5392effeb8f18fda755-Paper.pdf>, exact kernel / Gaussian process feature maps, prior features for Bayesian Additive Regression Trees (BART) (Chipman et al., 2010) <doi:10.1214/09-AOAS285>, and a helpful interface for n-way interactions. The provided functions may be used within any modeling formula, allowing the use of kernel methods and other basis expansions in modeling functions that do not otherwise support them. Along with the basis expansions, a number of kernel functions are also provided, which support kernel arithmetic to form new kernels. Basic ridge regression functionality is included as well.

Depends R (>= 3.6.0)

Imports rlang, stats, methods

Suggests mgcv, recipes, tibble, adj (>= 0.1.0), RSpectra, igraph, testthat (>= 3.0.0), knitr, rmarkdown

LinkingTo cpp11

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.3

VignetteBuilder knitr

URL https://corymccartan.com/bases/, https://github.com/CoryMcCartan/bases/

BugReports https://github.com/CoryMcCartan/bases/issues

Config/testthat/edition 3

Config/build/compilation-database true

NeedsCompilation yes

Author Cory McCartan [aut, cre, cph] (ORCID: <https://orcid.org/0000-0002-6251-669X>)

Maintainer Cory McCartan <mccartan@psu.edu>

Repository CRAN

Date/Publication 2026-02-27 15:42:10 UTC

Contents

`bases_mgcv` `mgcv` *integration*

Description

Provides methods so that `bases` expansions can be used as user-defined smooth classes in `mgcv::s()`. The `k` argument to `s()` maps to the main dimension parameter of each basis. Other arguments should be passed via the `xt` argument to `s()`, and will be forwarded to the basis function.

Examples

```
if (requireNamespace("mgcv", quietly = TRUE)) {
    x = 1:150
    z = c(1:50, rep(1, 100))
    y = as.numeric(BJsales)
    m = mgcv::gam(y ~ s(x, bs = "b_bart", k=10) + s(z, bs = "b_bart", k=20))
    summary(m)
    plot(x, y)
    lines(x, fitted(m), type="s", col="blue")
}
```

b_bart *Bayesian Additive Regression Tree (BART) features*

Description

Generates random features from a BART prior on symmetric trees. Equivalently, the features are the interaction of a small number of indicator functions. The number of interacted indicators is the depth of the symmetric tree, and is drawn from a prior on the tree depth which is calibrated to match the traditional BART prior of Chipman et al. (2010). The variable at each tree node is selected uniformly, and thresholds are selected uniformly from the range of each variable.

Usage

```
b_bart(
  ...,
  trees = 100,
  depths = bart_depth_prior()(trees),
  vars = NULL,
  thresh = NULL,
  drop = NULL,
  min_drop = 0L,
  ranges = NULL
)

bart_depth_prior(mean_depth = 1.25)
```

Arguments

`...`	The variable(s) to build features for. A single data frame or matrix may be provided as well. Missing values are not allowed.
`trees`	The number of trees to sample.
`depths`	The depths of each tree. By default, these are drawn from a Poisson distribution calibrated to produce trees with around 2.5 leaves, on average, matching the traditional BART prior.
`vars`	Integer indices of the variables to use for each tree. If provided, overrides those generated automatically by sampling uniformly from the available input features. Provided in flat form, so should have length equal to `sum(depths)`.
`thresh`	The thresholds for each variable. If provided, overrides those generated automatically by sampling uniformly from `ranges`, which defaults to the range of each input feature. Provided in flat form, so should have length equal to `sum(depths)`.
`drop`	Columns in the calculated indicator matrix to drop. By default, any leaves which match zero input rows are dropped. If provided, overrides this default.
`min_drop`	Controls the default dropping of columns. Leaves which match `min_drop` or fewer input rows are dropped. Defaults to 0, so only empty leaves are dropped.

ranges	The range of the input features, provided as a matrix with two rows and a column for each input feature. The first row is the minimum and the second row is the maximum.
mean_depth	The mean prior depth of each tree, where a single node has depth zero and a two-leaf tree has depth 1. This value minus one becomes the rate parameter of a Poisson distribution, whose samples are then shifted up by one. In this way, no zero-depth trees (which produce trivial features) are sampled.

Value

A matrix of indicator variables encoding the random features.

Functions

- `bart_depth_prior()`: Poisson depth prior for random trees, parametrized in terms of mean tree depth. Returns a function which generates samples from the prior with argument giving the number of samples. The default prior closely matches the average number of leaves in the original (asymmetric) BART prior.

References

Hugh A. Chipman. Edward I. George. Robert E. McCulloch. "BART: Bayesian additive regression trees." Ann. Appl. Stat. 4 (1) 266 - 298, March 2010. https://doi.org/10.1214/09-AOAS285

Examples

```
X = with(mtcars, b_bart(cyl, disp, hp, drat, wt, trees = 50))
all(colSums(X) > 0) # TRUE; empty leaves are pruned away
# each row belongs to 1 leaf node per tree; some trees pruned away
all(rowSums(X) == rowSums(X)[1]) # TRUE
all(rowSums(X) <= 50) # TRUE

x = 1:150
y = as.numeric(BJsales)
m = ridge(y ~ b_bart(x, trees=25))
plot(x, y)
lines(x, fitted(m), type="s", col="blue")
```

---

b_conv *Random convolutional features*

Description

Generates random convolutional features from a list of images. Convolutional kernels are generated randomly (either from a Gaussian distribution or as patches extracted from the training images), applied to each image via efficient matrix multiplication, and then pooled to produce a fixed-size feature vector per image.

Usage

```
b_conv(
  x,
  p = 100,
  size = 3,
  stride = 1,
  kernel_gen = c("rnorm", "patch"),
  activation = max,
  stdize = c("scale", "box", "symbox", "none"),
  kernels = NULL,
  shift = NULL,
  scale = NULL
)
```

Arguments

x	A list of images, where each image is a matrix (for grayscale) or a 3D array with dimensions (height, width, channels) for color images. Images may have different dimensions, but must be large enough to accommodate the convolution kernel size. Missing values are not allowed.
p	The number of random convolutional kernels to generate.
size	The size of the square convolutional kernel (e.g., 3 means a 3x3 kernel).
stride	The stride for the convolution operation, i.e., how many pixels to skip between kernel applications. Default is 1.
kernel_gen	Method for generating convolutional kernels. Either "rnorm" to generate kernels with entries drawn i.i.d. from a standard Normal distribution, or "patch" to extract random patches from the input images.
activation	A function to pool the convolution outputs for each kernel. Defaults to max(). The function should accept a numeric vector and return a scalar or vector of pooled values. Common choices include max(), mean(), functions like the proportion of positive values (PPV), which can be implemented with function(x) mean(x > 0). Multivariate pooling functions are also supported.
stdize	How to standardize the predictors, if at all. The default "scale" applies scale() to the input so that the features have mean zero and unit variance, "box" scales the data along each dimension to lie in the unit hypercube, and "symbox" scales the data along each dimension to lie in $[-0.5, 0.5]^d$.
kernels	Optional matrix of pre-specified convolutional kernels, where each column is a kernel in column-major format. If provided, overrides p, size, and kernel_gen.
shift	Vector of shifts, or single shift value, to use. If provided, overrides those calculated according to stdize.
scale	Vector of scales, or single scale value, to use. If provided, overrides those calculated according to stdize.

Value

A matrix of random convolutional features with one row per image in x and one column per kernel (or more columns if activation is multivariate).

Examples

```
x = outer(1:28, 1:28, function(x, y) {
    d = sqrt(4*(x - 14)^2 + (y - 14)^2)
    dnorm(d, mean = 10, sd = 0.8)
})
pal = gray.colors(256, 1, 0)
image(x, col = pal)

# one random kernel (no activation)
m = b_conv(list(x), p=1, activation=function(x) x)
image(matrix(m, nrow = 26), col = pal)

# many kernels (realistic use case)
m = b_conv(list(x), p = 100, size = 3)
str(m)
```

---

b_gff *Graph Fourier Feature basis*

Description

Generates features as the low-magnitude eigenvectors of a graph Laplacian, which can be thought of as a generalization of the Fourier basis to graph-structured data.

Usage

```
b_gff(x, p = min(length(x) - 1L, 50), symmetric = FALSE)
```

Arguments

x	An integer-indexed adjacency list, igraph::igraph, or adj::adj object.
p	The number of features to generate.
symmetric	Whether x is assumed symmetric.

Value

A matrix of graph Fourier features.

Examples

```
if (requireNamespace("adj", quietly = TRUE) && requireNamespace("RSpectra", quietly = TRUE)) {
    pal = hcl.colors(256)
    a = adj::adj(
       c(6, 2), c(1, 7, 3), c(2, 8, 4), c(3, 9, 5), c(4, 10), c(1, 11, 7),  c(6, 12, 2, 8),
         c(7, 13, 3, 9), c(8, 14, 4, 10), c(9, 5, 15), c(6, 12, 16), c(11, 7, 13, 17),
       c(12, 8, 18, 14), c(13, 9, 19, 15), c(14, 20, 10), c(11, 21, 17), c(12, 16, 22, 18),
         c(13, 17, 23, 19), c(18, 24, 14, 20), c(19, 15, 25), c(16, 22), c(21, 17, 23),
         c(22, 18, 24), c(23, 19, 25), c(24, 20)
```

```
    )

    m = b_gff(a, p = 3, symmetric = TRUE)
    image(matrix(m[, 1], 5, 5), col = pal)
    image(matrix(m[, 3], 5, 5), col = pal)

    if (requireNamespace("igraph", quietly = TRUE)) {
        a = igraph::make_lattice(c(100, 100))
        xy = igraph::layout_on_grid(a)
        m = b_gff(a, p = 25, symmetric = TRUE)
        eig_25 = m[, 25] # 25th Fourier feature
        image(matrix(eig_25, 100, 100), col=pal)
    }
}
```

`b_inter` *N-way interaction basis*

Description

Generates a design matrix that contains all possible interactions of the input variables up to a specified maximum depth. The default `"symbox"` standardization, which maps inputs to $[-0.5, 0.5]^d$, is strongly recommended, as it means that the interaction terms will have smaller variance and thus be penalized more by methods like the Lasso or ridge regression (see Gelman et al., 2008).

Usage

```
b_inter(
  ...,
  depth = 2,
  stdize = c("symbox", "box", "scale", "none"),
  shift = NULL,
  scale = NULL
)
```

Arguments

`...`	The variable(s) to build features for. A single data frame or matrix may be provided as well. Missing values are not allowed.
`depth`	The maximum interaction depth. The default is 2, which means that all pairwise interactions are included.
`stdize`	How to standardize the predictors, if at all. The default `"scale"` applies `scale()` to the input so that the features have mean zero and unit variance, `"box"` scales the data along each dimension to lie in the unit hypercube, and `"symbox"` scales the data along each dimension to lie in $[-0.5, 0.5]^d$.
`shift`	Vector of shifts, or single shift value, to use. If provided, overrides those calculated according to `stdize`.
`scale`	Vector of scales, or single scale value, to use. If provided, overrides those calculated according to `stdize`.

Value

A matrix with the rescaled and interacted features.

References

Gelman, A., Jakulin, A., Pittau, M. G., & Su, Y. S. (2008). A weakly informative default prior distribution for logistic and other regression models.

Examples

```
# default: all pairwise interactions
lm(mpg ~ b_inter(cyl, hp, wt), mtcars)

# how number of features depends on interaction depth
for (d in 1:6) {
    X = with(mtcars, b_inter(cyl, disp, hp, drat, wt, depth=d))
    print(ncol(X))
}
```

---

b_ker *Exact kernel feature basis*

Description

Generates a design matrix that exactly represents a provided kernel, so that the Gram matrix is equal to the kernel matrix. The feature map is

$$\phi(x') = K_{x,x}^{-1/2} k_{x,x'},$$

where $K_{x,x}$ is the kernel matrix for the data points x and $k_{x,x'}$ is the vector of kernel function evaluations at the data points and the new value. While exact, this function is not particularly computationally efficient. Both fitting and prediction require backsolving the Cholesky decomposition of the kernel matrix for the original data points.

Usage

```
b_ker(
  ...,
  kernel = k_rbf(),
  stdize = c("scale", "box", "symbox", "none"),
  x = NULL,
  shift = NULL,
  scale = NULL,
  L_inv = NULL
)
```

Arguments

...	The variable(s) to build features for. A single data frame or matrix may be provided as well. Missing values are not allowed.
kernel	A kernel function. If one of the recognized kernel functions such as k_rbf() is provided, then the computations will be exact. Otherwise, the fast Fourier transform of the provided kernel function is used to generate the random features. The kernel should be shift-invariant and decay to zero at positive and negative infinity.
stdize	How to standardize the predictors, if at all. The default "scale" applies scale() to the input so that the features have mean zero and unit variance, "box" scales the data along each dimension to lie in the unit hypercube, and "symbox" scales the data along each dimension to lie in $[-0.5, 0.5]^d$.
x	The (training) data points at which to evaluate the kernel. If provided, overrides ... .
shift	Vector of shifts, or single shift value, to use. If provided, overrides those calculated according to stdize.
scale	Vector of scales, or single scale value, to use. If provided, overrides those calculated according to stdize.
L_inv	The inverse of the Cholesky factor of the kernel matrix at the training points. Will be automatically computed if not provided, but in order to avoid recomputing it for new predictions, pass L_inv = TRUE, which will save and re-use this matrix for future calls.

Value

A matrix of kernel features.

Examples

```
data(quakes)

# exact kernel ridge regression
k = k_rbf(0.1)
m = ridge(depth ~ b_ker(lat, long, kernel = k), quakes)
cor(fitted(m), quakes$depth)^2

# Forecasting example involving combined kernels
data(AirPassengers)
x = seq(1949, 1961 - 1/12, 1/12)
y = as.numeric(AirPassengers)
x_pred = seq(1961 - 1/2, 1965, 1/12)

k = k_per(scale = 0.2, period = 1) * k_rbf(scale = 4)
m = ridge(y ~ b_ker(x, kernel = k, stdize="none"))
plot(x, y, type='l', xlab="Year", ylab="Passengers (thousands)",
    xlim=c(1949, 1965), ylim=c(100, 800))
lines(x_pred, predict(m, newdata = list(x = x_pred)), lty="dashed")
```

b_nn *Neural network basis*

Description

Generates random features from a one-layer neural network, i.e., a random linear transformation followed by a nonlinear activation function:

$$\phi(x) = g(w^\top x + b),$$

where w and b are randomly sampled weights and bias, and g is the chosen activation function.

Usage

```
b_nn(
  ...,
  p = 100,
  activation = function(x) (x + abs(x))/2,
  stdize = c("scale", "box", "symbox", "none"),
  weights = NULL,
  biases = NULL,
  shift = NULL,
  scale = NULL
)
```

Arguments

`...`	The variable(s) to build features for. A single data frame or matrix may be provided as well. Missing values are not allowed.
`p`	The number of random features.
`activation`	The activation function, which should take in a numeric vector and return another of the same length. The default is the rectified linear unit, i.e. `pmax(0, x)`; the implementation here is faster.
`stdize`	How to standardize the predictors, if at all. The default `"scale"` applies `scale()` to the input so that the features have mean zero and unit variance, `"box"` scales the data along each dimension to lie in the unit hypercube, and `"symbox"` scales the data along each dimension to lie in $[-0.5, 0.5]^d$.
`weights`	Matrix of weights; `nrow(weights)` must match the number of predictors. If provided, overrides those calculated automatically, thus ignoring `p`.
`biases`	Vector of biases to use. If provided, overrides those calculated automatically, thus ignoring `p`.
`shift`	Vector of shifts, or single shift value, to use. If provided, overrides those calculated according to `stdize`.
`scale`	Vector of scales, or single scale value, to use. If provided, overrides those calculated according to `stdize`.

Details

As with b_rff(), to reduce the variance, a moment-matching transformation is applied to ensure the sampled weights and biases have mean zero and the sampled weights have unit covariance.

Value

A matrix of random neural network features.

Examples

```
data(quakes)

m = ridge(depth ~ b_nn(lat, long, p = 100, activation = tanh), quakes)
plot(fitted(m), quakes$depth)

# In 1-D with ReLU (default), equivalent to piecewise
# linear regression with random knots
x = 1:150
y = as.numeric(BJsales)
m = lm(y ~ b_nn(x, p = 8))
plot(x, y)
lines(x, fitted(m), col="blue")
```

---

b_rff *Random Fourier feature basis*

Description

Generates a random Fourier feature basis matrix for a provided kernel, optionally rescaling the data to lie in the unit hypercube. A good review of random features is the Liu et al. (2021) review paper cited below. Random features here are of the form

$$\phi(x) = \cos(\omega^T x + b),$$

where ω is a vector of frequencies sampled from the Fourier transform of the kernel, and $b \sim \text{Unif}[-\pi, \pi]$ is a random phase shift. The input data `x` may be shifted and rescaled before the feature mapping is applied, according to the `stdize` argument.

Usage

```
b_rff(
  ...,
  p = 100,
  kernel = k_rbf(),
  stdize = c("scale", "box", "symbox", "none"),
  n_approx = nextn(4 * p),
  freqs = NULL,
  phases = NULL,
```

```
  shift = NULL,
  scale = NULL
)
```

Arguments

`...`	The variable(s) to build features for. A single data frame or matrix may be provided as well. Missing values are not allowed.
`p`	The number of random features.
`kernel`	A kernel function. If one of the recognized kernel functions such as `k_rbf()` is provided, then the computations will be exact. Otherwise, the fast Fourier transform of the provided kernel function is used to generate the random features. The kernel should be shift-invariant and decay to zero at positive and negative infinity.
`stdize`	How to standardize the predictors, if at all. The default `"scale"` applies `scale()` to the input so that the features have mean zero and unit variance, `"box"` scales the data along each dimension to lie in the unit hypercube, and `"symbox"` scales the data along each dimension to lie in $[-0.5, 0.5]^d$.
`n_approx`	The number of discrete frequencies to use in calculating the Fourier transform of the provided kernel. Not used for certain kernels for which an analytic Fourier transform is available; see above.
`freqs`	Matrix of frequencies to use; `nrow(freqs)` must match the number of predictors. If provided, overrides those calculated automatically, thus ignoring `p` and `kernel`.
`phases`	Vector of phase shifts to use. If provided, overrides those calculated automatically, thus ignoring `p` and `kernel`.
`shift`	Vector of shifts, or single shift value, to use. If provided, overrides those calculated according to `stdize`.
`scale`	Vector of scales, or single scale value, to use. If provided, overrides those calculated according to `stdize`.

Details

To reduce the variance of the approximation, a moment-matching transformation is applied to ensure the sampled frequencies have mean zero, per Shen et al. (2017). For the Gaussian/RBF kernel, second moment-matching is also applied to ensure the analytical and empirical frequency covariance matrices agree.

Value

A matrix of random Fourier features.

References

Rahimi, A., & Recht, B. (2007). *Random features for large-scale kernel machines.* Advances in Neural Information Processing Systems, 20.

Liu, F., Huang, X., Chen, Y., & Suykens, J. A. (2021). Random features for kernel approximation: A survey on algorithms, theory, and beyond. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(10), 7128-7148.

Shen, W., Yang, Z., & Wang, J. (2017, February). Random features for shift-invariant kernels with moment matching. In *Proceedings of the AAAI Conference on Artificial Intelligence* (Vol. 31, No. 1).

Examples

```
data(quakes)

m = ridge(depth ~ b_rff(lat, long), quakes)
plot(fitted(m), quakes$depth)

# more random featues means a higher ridge penalty
m500 = ridge(depth ~ b_rff(lat, long, p = 500), quakes)
c(default = m$penalty, p500 = m500$penalty)

# A shorter length scale fits the data better (R^2)
m_025 = ridge(depth ~ b_rff(lat, long, kernel = k_rbf(scale = 0.25)), quakes)
c(
  len_1 = cor(quakes$depth, fitted(m))^2,
  len_025 = cor(quakes$depth, fitted(m_025))^2
)
```

---

b_tpsob *Tensor-product Sobolev space basis*

Description

Generates features from a tensor-product Sobolev space basis for estimating functions in a Sobolev space with dominating mixed derivatives. Basis functions are of the form

$$\psi_{\mathbf{j}}(\mathbf{x}) = \prod_{k=1}^{d} \psi_{j_k}(x_k),$$

where

$$\phi_1(x) = 1 \quad \text{and} \quad \phi_j(x) = \sqrt{2}\cos(\pi(j-1)x).$$

The multi-indices $\mathbf{j}$ are generated in a specific order to maximize statistical efficiency. All inputs are standardized to lie in the unit hypercube $[0, 1]^d$.

Usage

```
b_tpsob(..., p = 100, shift = NULL, scale = NULL)
```

Arguments

...	The variable(s) to build features for. A single data frame or matrix may be provided as well. Missing values are not allowed.
p	The number of basis functions to generate.
shift	Vector of shifts, or single shift value, to use. If provided, overrides those calculated according to stdize.
scale	Vector of scales, or single scale value, to use. If provided, overrides those calculated according to stdize.

Value

A matrix of tensor-product Sobolev space basis features.

References

Zhang, T., & Simon, N. (2023). Regression in tensor product spaces by the method of sieves. *Electronic journal of statistics*, 17(2), 3660.

Examples

```
data(quakes)

m = ridge(depth ~ b_tpsob(lat, long, p = 100), quakes)
plot(fitted(m), quakes$depth)

x = 1:150
y = as.numeric(BJsales)
m = lm(y ~ b_tpsob(x, p = 10))
plot(x, y)
lines(x, fitted(m), col="blue")
```

---

kernel-arith *Kernel arithmetic*

Description

Kernel functions (see ?kernels) may be multiplied by constants, multiplied by each other, or added together.

Usage

```
## S3 method for class 'kernel'
x * k2

## S3 method for class 'kernel'
k1 + k2
```

Arguments

x	a numeric or a kernel function
k2	a kernel function
k1	a kernel function

Value

A new kernel function, with class c("kernel", "function").

Examples

```
x = seq(-1, 1, 0.5)
k = k_rbf()
k2 = k_per(scale=0.2, period=0.3)

k_add = k2 + 0.5*k
print(k_add)
image(k_add(x, x))
```

---

kernels *Kernel functions*

Description

These functions return vectorized kernel functions that can be used to calculate kernel matrices, or provided directly to other basis functions. These functions are designed to take a maximum value of one when identical inputs are provided. Kernels can be combined with arithmetic expressions; see ?kernel-arith.

Usage

```
k_rbf(scale = 1)

k_lapl(scale = 1)

k_rq(scale = 1, alpha = 2)

k_matern(scale = 1, nu = 1.5)

k_per(scale = 1, period = 1)
```

Arguments

scale	The kernel length scale.
alpha	The shape/df parameter. $\alpha = 1$ is the Cauchy kernel.
nu	The smoothness parameter. $\nu = 0.5$ is the Ornstein–Uhlenbeck kernel.
period	The period, in the same units as scale.

Value

A function which calculates a kernel matrix for vector arguments `x` and `y`. The function has class `c("kernel", "function")`.

Functions

- `k_rbf()`: Radial basis function kernel
- `k_lapl()`: Laplace kernel
- `k_rq()`: Rational quadratic kernel.
- `k_matern()`: Matérn kernel.
- `k_per()`: Periodic (exp-sine-squared) kernel.

Examples

```
k = k_rbf()
x = seq(-1, 1, 0.5)
k(0, 0)
k(0, x)
k(x, x)

k = k_per(scale=0.2, period=0.3)
round(k(x, x))
```

---

ridge *Ridge regression*

Description

Lightweight routine for ridge regression, fitted via a singular value decomposition. The penalty may be automatically determined by leave-one-out cross validation. The intercept term is unpenalized.

Usage

```
ridge(formula, data, penalty = "auto", ...)

## S3 method for class 'ridge'
fitted(object, ...)

## S3 method for class 'ridge'
coef(object, ...)

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

Arguments

formula	A model formula; see formula. The intercept term is unpenalized; to fit a penalized intercept, remove the intercept and add your own to the design matrix.
data	An optional data frame or object in which to interpret the variables occurring in formula.
penalty	The ridge penalty. Must be a single numeric or the string "auto", in which case the penalty will be determined via leave-one-out cross validation to minimize the mean squared error.
...	Further arguments, passed on to `model.frame()` and `model.matrix()`. These must be provided to `predict.ridge()` as well, if used.
object	A fitted `ridge()` model.
newdata	A data frame containing the new data to predict

Value

An object of class `ridge` with components including:

- `coef`, a vector of coefficients.
- `fitted`, a vector of fitted values.
- `penalty`, the penalty value.

Methods (by generic)

- `fitted(ridge)`: Fitted values
- `coef(ridge)`: Coefficients
- `predict(ridge)`: Predicted values

Examples

```
m_lm = lm(mpg ~ ., mtcars)
m_ridge = ridge(mpg ~ ., mtcars, penalty=1e3)
plot(fitted(m_lm), fitted(m_ridge), ylim=c(10, 30))
abline(a=0, b=1, col="red")
```

`step_basis` *Recipe step for basis expansions*

Description

`step_basis()` is a single function that creates a *specification* of a recipe step that will create new columns that are basis expansions, using any of the basis expansion functions in this package.

Usage

```
step_basis(
  recipe,
  ...,
  role = NA,
  trained = FALSE,
  fn = NULL,
  options = list(),
  object = NULL,
  prefix = deparse(substitute(fn)),
  skip = FALSE,
  id = recipes::rand_id("basis")
)
```

Arguments

recipe	A recipe object.
...	One or more selector functions to choose variables for this step. See recipes::selections() for more details.
role	For model terms created by this step, what analysis role should they be assigned? By default, the new columns created by this step from the original variables will be used as predictors in a model.
trained	A logical to indicate if the quantities for preprocessing have been estimated.
fn	The basis function to use, e.g., b_rff().
options	A list of options for the basis function fn.
object	The basis object created once the step has been trained.
prefix	The prefix to use for the new column names. Numbers will be appended, per recipes::names0(), to create column names.
skip	A logical. Should the step be skipped when the recipe is baked by recipes::bake()?
id	A character string that is unique to this step to identify it.

Value

An updated version of recipe with the new step added to the sequence of any existing operations.

Tuning Parameters

There are no tuning parameters made available to the tunable interface.

Case Weights

The underlying operation does not use case weights.

Examples

```
rec = recipes::recipe(depth ~ lat + long + mag, quakes)
rec_rff = step_basis(rec, lat, long, fn = b_rff,
                     options = list(p = 5, kernel = k_rbf(2), stdize="none"))
recipes::bake(recipes::prep(rec_rff), new_data=NULL)
```

Index