# Package 'basefun'

October 5, 2025

**Title** Infrastructure for Computing with Basis Functions

**Version** 1.2-5

**Date** 2025-10-02

**Description** Some very simple infrastructure for basis functions.

**Depends** variables (>= 1.1-0), R (>= 3.2.0)

**Imports** stats, polynom, Matrix, orthopolynom, methods

**Suggests** quadprog, numDeriv

**URL** http://ctm.R-forge.R-project.org

**License** GPL-2

**NeedsCompilation** yes

**Author** Torsten Hothorn [aut, cre] (ORCID:
<https://orcid.org/0000-0001-8301-0471>)

**Maintainer** Torsten Hothorn <Torsten.Hothorn@R-project.org>

**Repository** CRAN

**Date/Publication** 2025-10-05 05:10:17 UTC

## Contents

basefun-package *General Information on the* **basefun** *Package*

## Description

The **basefun** package offers a small collection of objects for handling basis functions and corresponding methods.

The package was written to support the **mlt** package and will be of limited use outside this package.

## Author(s)

This package is authored by Torsten Hothorn <Torsten.Hothorn@R-project.org>.

## References

Torsten Hothorn (2018), Most Likely Transformations: The mlt Package, *Journal of Statistical Software*, forthcoming. URL: https://cran.r-project.org/package=mlt.docreg

as.basis *Convert Formula or Factor to Basis Function*

## Description

Convert a formula or factor to basis functions

## Usage

```
as.basis(object, ...)
## S3 method for class 'formula'
as.basis(object, data = NULL, remove_intercept = FALSE,
         ui = NULL, ci = NULL, negative = FALSE, scale = FALSE,
         Matrix = FALSE, prefix = "", drop.unused.levels = TRUE, ...)
## S3 method for class 'factor_var'
as.basis(object, ...)
## S3 method for class 'ordered_var'
as.basis(object, ...)
## S3 method for class 'factor'
as.basis(object, ...)
## S3 method for class 'ordered'
as.basis(object, ...)
```

## Arguments

| | |
|---|---|
| object | a formula or an object of class factor, factor_var, ordered or ordered_var |
| data | either a vars object or a data.frame |
| remove_intercept | a logical indicating if any intercept term shall be removed |
| ui | a matrix defining constraints |
| ci | a vector defining constraints |
| negative | a logical indicating negative basis functions |
| scale | a logical indicating a scaling of each column of the model matrix to the unit interval (based on observations in data) |
| Matrix | a logical requesting a sparse model matrix, that is, a Matrix object. |
| prefix | character prefix for model matrix column names (allows disambiguation of parameter names). |
| drop.unused.levels | logical, should factors have unused levels dropped? |
| ... | additional arguments to model.matrix, for example contrasts |

## Details

as.basis returns a function for the evaluation of the basis functions with corresponding model.matrix and predict methods.

Unordered factors (classes factor and factor_var) use a dummy coding and ordered factor (classes ordered or ordered_var) lead to a treatment contrast to the last level and removal of the intercept term with monotonicity constraint. Additional arguments (...) are ignored for ordered factors.

Linear constraints on parameters parm are defined by ui %*% parm >= ci.

## Examples

```
## define variables and basis functions
v <- c(numeric_var("x"), factor_var("y", levels = LETTERS[1:3]))
fb <- as.basis(~ x + y, data = v, remove_intercept = TRUE, negative = TRUE,
               contrasts.arg = list(y = "contr.sum"))

## evaluate basis functions
model.matrix(fb, data = as.data.frame(v, n = 10))
## basically the same as (but wo intercept and times -1)
model.matrix(~ x + y, data = as.data.frame(v, n = 10))

### factor
xf <- gl(3, 1)
model.matrix(as.basis(xf), data = data.frame(xf = xf))

### ordered
xf <- gl(3, 1, ordered = TRUE)
model.matrix(as.basis(xf), data = data.frame(xf = unique(xf)))
```

b *Box Product of Basis Functions*

## Description

Box product of two basis functions

## Usage

```
b(..., sumconstr = FALSE)
```

## Arguments

| | |
|---|---|
| ... | named objects of class basis |
| sumconstr | a logical indicating if sum constraints shall be applied |

## Details

b() joins the corresponding design matrices by the row-wise Kronecker (or box) product.

## Examples

```
### set-up a Bernstein polynomial
xv <- numeric_var("x", support = c(1, pi))
bb <- Bernstein_basis(xv, order = 3, ui = "increasing")
## and treatment contrasts for a factor at three levels
fb <- as.basis(~ g, data = factor_var("g", levels = LETTERS[1:3]))

### join them: we get one intercept and two deviation _functions_
bfb <- b(bern = bb, f = fb)

### generate data + coefficients
x <- expand.grid(mkgrid(bfb, n = 10))
cf <- c(1, 2, 2.5, 2.6)
cf <- c(cf, cf + 1, cf + 2)

### evaluate bases
model.matrix(bfb, data = x)

### plot functions
plot(x$x, predict(bfb, newdata = x, coef = cf), type = "p",
     pch = (1:3)[x$g])
legend("bottomright", pch = 1:3,
       legend = colnames(model.matrix(fb, data = x)))
```

Bernstein_basis *Bernstein Basis Functions*

## Description

Basis functions defining a polynomial in Bernstein form

## Usage

```
Bernstein_basis(var, order = 2, ui = c("none", "increasing", "decreasing",
                                       "cyclic", "zerointegral", "positive",
                                       "negative", "concave", "convex"),
                extrapolate = FALSE, log_first = FALSE, fun = NULL)
```

## Arguments

| | |
|---|---|
| var | a numeric_var object |
| order | the order of the polynomial, one defines a linear function |
| ui | a character describing possible constraints |
| extrapolate | logical; if TRUE, the polynomial is extrapolated linearily outside support(var). In particular, the second derivative of the polynomial at support(var) is constrained to zero. |
| log_first | logical; the polynomial in Bernstein form is defined on the log-scale if TRUE. It makes sense to define the support as c(1, q)$, ie putting the first basis function of the polynomial on log(1). |
| fun | a list of functions for transforming the input data, typically of length three: the monotone transformation and its first two derivatives. |

## Details

Bernstein_basis returns a function for the evaluation of the basis functions with corresponding model.matrix and predict methods.

## References

Rida T. Farouki (2012), The Bernstein Polynomial Basis: A Centennial Retrospective, *Computer Aided Geometric Design*, **29**(6), 379–419, doi:10.1016/j.cagd.2012.03.001.

## Examples

```
### set-up basis
bb <- Bernstein_basis(numeric_var("x", support = c(0, pi)),
                      order = 3, ui = "increasing")

### generate data + coefficients
x <- as.data.frame(mkgrid(bb, n = 100))
cf <- c(1, 2, 2.5, 2.6)
```

```
### evaluate basis (in two equivalent ways)
bb(x[1:10,,drop = FALSE])
model.matrix(bb, data = x[1:10, ,drop = FALSE])

### check constraints
cnstr <- attr(bb(x[1:10,,drop = FALSE]), "constraint")
all(cnstr$ui %*% cf > cnstr$ci)

### evaluate and plot Bernstein polynomial defined by
### basis and coefficients
plot(x$x, predict(bb, newdata = x, coef = cf), type = "l")

### evaluate and plot first derivative of
### Bernstein polynomial defined by basis and coefficients
plot(x$x, predict(bb, newdata = x, coef = cf, deriv = c(x = 1)),
     type = "l")

### illustrate constrainted estimation by toy example
N <- 100
order <- 10
x <- seq(from = 0, to = pi, length.out = N)
y <- rnorm(N, mean = -sin(x) + .5, sd = .5)

if (require("quadprog")) {
  prnt_est <- function(ui) {
    xv <- numeric_var("x", support = c(0, pi))
    xb <- Bernstein_basis(xv, order = 10, ui = ui)
    X <- model.matrix(xb, data = data.frame(x = x))
    uiM <- as(attr(X, "constraint")$ui, "matrix")
    ci <- attr(X, "constraint")$ci
    if (all(is.finite(ci)))
      parm <- solve.QP(crossprod(X), crossprod(X, y),
                       t(uiM), ci)$solution
    else
      parm <- coef(lm(y ~ 0 + X))
    plot(x, y, main = ui)
    lines(x, X %*% parm, col = col[ui], lwd = 2)
  }
  ui <- eval(formals(Bernstein_basis)$ui)
  col <- 1:length(ui)
  names(col) <- ui
  layout(matrix(1:length(ui),
                ncol = ceiling(sqrt(length(ui)))))
  tmp <- sapply(ui, function(x) try(prnt_est(x)))
}
```

---

c.basis *Join Basis Functions*

---

## Description

Concatenate basis functions column-wise

## Usage

```
## S3 method for class 'basis'
c(..., recursive = FALSE)
```

## Arguments

| | |
|---|---|
| `...` | named objects of class `basis` |
| `recursive` | always `FALSE` |

## Details

`c()` joins the corresponding design matrices column-wise, ie, the two functions defined by the two bases are added.

## Examples

```
### set-up Bernstein and log basis functions
xv <- numeric_var("x", support = c(1, pi))
bb <- Bernstein_basis(xv, order = 3, ui = "increasing")
lb <- log_basis(xv, remove_intercept = TRUE)

### join them
blb <- c(bern = bb, log = lb)

### generate data + coefficients
x <- as.data.frame(mkgrid(blb, n = 100))
cf <- c(1, 2, 2.5, 2.6, 2)

### evaluate bases
model.matrix(blb, data = x[1:10, ,drop = FALSE])

### evaluate and plot function defined by
### bases and coefficients
plot(x$x, predict(blb, newdata = x, coef = cf), type = "l")

### evaluate and plot first derivative of function
### defined by bases and coefficients
plot(x$x, predict(blb, newdata = x, coef = cf, deriv = c(x = 1)),
     type = "l")
```

`cyclic_basis` *Cyclic Basis Function*

## Description

The cyclic basis function for modelling seasonal effects

## Usage

```
cyclic_basis(var, order = 3, frequency)
```

## Arguments

| | |
|---|---|
| `var` | a `numeric_var` object |
| `order` | the order of the basis function |
| `frequency` | frequency |

## Details

`cyclic_basis` returns a set of sin and cosine functions for modelling seasonal effects, see Held and Paul (2012), Section 2.2.

For any choice of coefficients, the function returns the same value when evaluated at multiples of `frequency`.

## References

Leonhard Held and Michaela Paul (2012), Modeling Seasonality in Space-time Infectious Disease Surveillance Data, *Biometrical Journal*, **54**(6), 824–843, doi:10.1002/bimj.201200037

## Examples

```
### set-up basis
cb <- cyclic_basis(numeric_var("x"), order = 3, frequency = 2 * pi)

### generate data + coefficients
x <- data.frame(x = c(0, pi, 2 * pi))

### f(0) = f(2 * pi)
cb(x)
```

## intercept_basis *Intercept-Only Basis Function*

### Description

A simple intercept as basis function

### Usage

```
intercept_basis(ui = c("none", "increasing", "decreasing"), negative = FALSE)
```

### Arguments

| | |
|---|---|
| ui | a character describing possible constraints |
| negative | a logical indicating negative basis functions |

### Details

`intercept_basis` returns a function for the evaluation of the basis functions with corresponding `model.matrix` and `predict` methods.

### Examples

```
### set-up basis
ib <- intercept_basis()

### generate data + coefficients
x <- as.data.frame(mkgrid(ib))

### 2 * 1
predict(ib, newdata = x, coef = 2)
```

## Legendre_basis *Legendre Basis Functions*

### Description

Basis functions defining a Legendre polynomial

### Usage

```
Legendre_basis(var, order = 2, ui = c("none", "increasing", "decreasing",
                                      "cyclic", "positive", "negative"), ...)
```

### Arguments

| | |
|---|---|
| var | a numeric_var object |
| order | the order of the polynomial, one defines a linear function |
| ui | a character describing possible constraints |
| ... | additional arguments passed to legendre.polynomials |

### Details

Legendre_basis returns a function for the evaluation of the basis functions with corresponding model.matrix and predict methods.

### References

Rida T. Farouki (2012), The Bernstein Polynomial Basis: A Centennial Retrospective, *Computer Aided Geometric Design*, **29**(6), 379–419, doi:10.1016/j.cagd.2012.03.001.

### Examples

```
### set-up basis
lb <- Legendre_basis(numeric_var("x", support = c(0, pi)),
                     order = 3)

### generate data + coefficients
x <- as.data.frame(mkgrid(lb, n = 100))
cf <- c(1, 2, 2.5, 1.75)

### evaluate basis (in two equivalent ways)
lb(x[1:10,,drop = FALSE])
model.matrix(lb, data = x[1:10, ,drop = FALSE])

### evaluate and plot Legendre polynomial defined by
### basis and coefficients
plot(x$x, predict(lb, newdata = x, coef = cf), type = "l")
```

---

## log_basis *Logarithmic Basis Function*

---

### Description

The logarithmic basis function

### Usage

```
log_basis(var, ui = c("none", "increasing", "decreasing"),
          remove_intercept = FALSE)
```

### Arguments

| | |
|---|---|
| var | a numeric_var object |
| ui | a character describing possible constraints |
| remove_intercept | a logical indicating if the intercept term shall be removed |

### Details

log_basis returns a function for the evaluation of the basis functions with corresponding model.matrix and predict methods.

### Examples

```
### set-up basis
lb <- log_basis(numeric_var("x", support = c(0.1, pi)))

### generate data + coefficients
x <- as.data.frame(mkgrid(lb, n = 100))

### 1 + 2 * log(x)
max(abs(predict(lb, newdata = x, coef = c(1, 2)) - (1 + 2 * log(x$x))))
```

---

## polynomial_basis *Polynomial Basis Functions*

---

### Description

Basis functions defining a polynomial

### Usage

```
polynomial_basis(var, coef, ui = NULL, ci = NULL)
```

### Arguments

| | |
|---|---|
| var | a numeric_var object |
| coef | a logical defining the order of the polynomial |
| ui | a matrix defining constraints |
| ci | a vector defining constraints |

### Details

polynomial_basis returns a function for the evaluation of the basis functions with corresponding model.matrix and predict methods.

## Examples

```
### set-up basis of order 3 ommiting the quadratic term
pb <- polynomial_basis(numeric_var("x", support = c(0, pi)),
                       coef = c(TRUE, TRUE, FALSE, TRUE))

### generate data + coefficients
x <- as.data.frame(mkgrid(pb, n = 100))
cf <- c(1, 2, 0, 1.75)

### evaluate basis (in two equivalent ways)
pb(x[1:10,,drop = FALSE])
model.matrix(pb, data = x[1:10, ,drop = FALSE])

### evaluate and plot polynomial defined by
### basis and coefficients
plot(x$x, predict(pb, newdata = x, coef = cf), type = "l")
```

---

`predict.basis` *Evaluate Basis Functions*

---

## Description

Evaluate basis functions and compute the function defined by the corresponding basis

## Usage

```
## S3 method for class 'basis'
predict(object, newdata, coef, dim = !is.data.frame(newdata), ...)
## S3 method for class 'cbind_bases'
predict(object, newdata, coef, dim = !is.data.frame(newdata),
         terms = names(object), ...)
## S3 method for class 'box_bases'
predict(object, newdata, coef, dim = !is.data.frame(newdata), ...)
```

## Arguments

| | |
|---|---|
| `object` | a `basis` or `bases` object |
| `newdata` | a `list` or `data.frame` |
| `coef` | a vector of coefficients |
| `dim` | either a logical indicating that the dimensions shall be obtained from the `bases` object or an integer vector with the corresponding dimensions (the latter option being very experimental |
| `terms` | a character vector defining the elements of a `cbind_bases` object to be evaluated |
| `...` | additional arguments |

## Details

`predict` evaluates the basis functions and multiplies them with `coef`. There is no need to expand multiple variables as `predict` uses array models (Currie et al, 2006) to compute the corresponding predictions efficiently.

## References

Ian D. Currie, Maria Durban, Paul H. C. Eilers, P. H. C. (2006), Generalized Linear Array Models with Applications to Multidimensional Smoothing, *Journal of the Royal Statistical Society, Series B: Methodology*, **68**(2), 259–280.

## Examples

```
### set-up a Bernstein polynomial
xv <- numeric_var("x", support = c(1, pi))
bb <- Bernstein_basis(xv, order = 3, ui = "increasing")
## and treatment contrasts for a factor at three levels
fb <- as.basis(~ g, data = factor_var("g", levels = LETTERS[1:3]))

### join them: we get one intercept and two deviation _functions_
bfb <- b(bern = bb, f = fb)

### generate data + coefficients
x <- mkgrid(bfb, n = 10)
cf <- c(1, 2, 2.5, 2.6)
cf <- c(cf, cf + 1, cf + 2)

### evaluate predictions for all combinations in x (a list!)
predict(bfb, newdata = x, coef = cf)

## same but slower
matrix(predict(bfb, newdata = expand.grid(x), coef = cf), ncol = 3)
```

# Index