Package ‘basefun’

September 17, 2018

Title Infrastructure for Computing with Basis Functions
Version 1.0-2
Date 2018-09-17
Description Some very simple infrastructure for basis functions.
Depends variables (>= 0.0-30), R (>= 3.2.0)
Imports stats, polynom, Matrix, orthopolynom, methods
License GPL-2
NeedsCompilation yes
Author Torsten Hothorn [aut, cre] (<https://orcid.org/0000-0001-8301-0471>)
Maintainer Torsten Hothorn <Torsten.Hothorn@R-project.org>
Repository CRAN
Date/Publication 2018-09-17 15:40:05 UTC

R topics documented:

basefun-package .................................................. 2
as.basis ................................................................. 2
b .............................................................. 4
Bernstein_basis ..................................................... 5
c.basis ............................................................ 6
intercept_basis .................................................... 7
Legendre_basis .................................................... 8
log_basis ............................................................. 9
polynomial_basis .................................................. 9
predict.basis ....................................................... 10

Index 12
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


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, ...)
## 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)
...               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)))
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

```r
### 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_
bf <- b(bern = bb, f = fb)

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

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

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

Description

Basis functions defining a Bernstein polynomial

Usage

Bernstein_basis(var, order = 2, ui = c("none", "increasing", "decreasing", "cyclic", "zerointegral", "positive", "negative"), extrapolate = FALSE)

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 linearly outside support(var). In particular, the second derivative of the polynomial at support(var) is constrained to zero.

Details

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

References


Examples

```r
### 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")

---

c.basis  

**Join Basis Functions**

**Description**

Concatenate basis functions column-wise

**Usage**

```r
## 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**

```r
### 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))
```
intercept_basis

### 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")

---

### 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

```r
### 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 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

Aided Geometric Design*, 29(6), 379–419. http://dx.doi.org/10.1016/j.cagd.2012.03.001

Examples

```r
### 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))
coef <- 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 = coef), type = "l")
```
**log_basis**

---

**Logarithmic Basis Function**

**Description**

The logarithmic basis function

**Usage**

```r
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**

```r
### 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 + R * 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**

```r
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

```r
### set-up basis of order 3 omitting 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)
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

```r
## 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), ...)
```
**predict.basis**

**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**


**Examples**

```r
### 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_
bbfb <- 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

>Topic package
  basefun-package, 2
as.basis, 2
b, 4
basefun (basefun-package), 2
basefun-package, 2
Bernstein_basis, 5
c.basis, 6
intercept_basis, 7
legendre.polynomials, 8
Legendre_basis, 8
log_basis, 9
model.matrix, 3
numeric_var, 5, 8–10
polynomial_basis, 9
predict.basis, 10
predict.box_bases (predict.basis), 10
predict.cbind_bases (predict.basis), 10
vars, 3