Package ‘cpr’

February 15, 2024

Title Control Polygon Reduction

Version 0.4.0

Description Implementation of the Control Polygon Reduction and Control Net Reduction methods for finding parsimonious B-spline regression models.

Depends R (>= 3.5.0)

License GPL (>= 2)

Encoding UTF-8

URL https://github.com/dewittpe/cpr/, http://www.peteredewitt.com/cpr/

BugReports https://github.com/dewittpe/cpr/issues

Language en-us

LazyData true

Imports ggplot2 (>= 3.0.0), lme4 (>= 1.1.35.1), plot3D, Rcpp (>= 1.0.11), rgl, scales

LinkingTo Rcpp, RcppArmadillo

Suggests Matrix (>= 1.6-4), geepack, ggpubr, knitr, qwraps2 (>= 0.6.0)

RoxygenNote 7.3.1

VignetteBuilder knitr

NeedsCompilation yes

Author Peter DeWitt [aut, cre] (<https://orcid.org/0000-0002-6391-0795>),
Samantha MaWhinney [ths],
Nichole Carlson [ths]

Maintainer Peter DeWitt <dewittpe@gmail.com>

Repository CRAN

Date/Publication 2024-02-15 15:40:02 UTC

R topics documented:

bsplineD — *B-spline Derivatives*

Description

Generate the first and second derivatives of a B-spline Basis.

Usage

```
bsplineD(
  x,
  iknots = NULL,
  df = NULL,
  bknots = range(x),
  order = 4L,
  derivative = 1L
)
```

Arguments

`x`	a numeric vector
`iknots`	internal knots
`df`	degrees of freedom: sum of the order and internal knots. Ignored if `iknots` is specified.
`bknots`	boundary knot locations, defaults to `range(x)`.
`order`	order of the piecewise polynomials, defaults to 4L.
`derivative,`	(integer) first or second derivative

Value

a numeric matrix

References

C. de Boor, "A practical guide to splines. Revised Edition," Springer, 2001.

H. Prautzsch, W. Boehm, M. Paluszny, "Bezier and B-spline Techniques," Springer, 2002.

See Also

`bsplines` for bspline basis. `get_spline` will give you the spline or the derivative thereof for a control polygon.

Examples

```
###############################################################################
# Example 1 - pefectly fitting a cubic function
f <- function(x) {
  x^3 - 2 * x^2 - 5 * x + 6
}

fprime <- function(x) { # first derivatives of f(x)
  3 * x^2 - 4 * x - 5
}

fdoubleprime <- function(x) { # second derivatives of f(x)
  6 * x - 4
}

# Build a spline to fit
bknots = c(-3, 5)

x     <- seq(-3, 4.999, length.out = 200)
bmat  <- bsplines(x, bknots = bknots)
theta <- matrix(coef(lm(f(x) ~ bmat + 0)), ncol = 1)

bmatD1 <- bsplineD(x, bknots = bknots, derivative = 1L)
bmatD2 <- bsplineD(x, bknots = bknots, derivative = 2L)

# Verify that we have perfectly fitted splines to the function and its
# derivatives.
# check that the function f(x) is recovered
all.equal(f(x), as.numeric(bmat %*% theta))
all.equal(fprime(x), as.numeric(bmatD1 %*% theta))
all.equal(fdoubleprime(x), as.numeric(bmatD2 %*% theta))

# Plot the results
old_par <- par()
par(mfrow = c(1, 3))
plot(x, f(x), type = "l", main = bquote(f(x)), ylab = "", xlab = "")
points(x, bmat %*% theta, col = 'blue')
grid()

plot(
    x
  , fprime(x)
  , type = "l"
  , main = bquote(frac(d,dx)~f(x))
  , ylab = ""
  , xlab = ""
)
points(x, bmatD1 %*% theta, col = 'blue')
grid()

plot(
    x
```

```
  , fdoubleprime(x)
  , type = "l"
  , main = bquote(frac(d^2,dx^2)~f(x))
  , ylab = ""
  , xlab = ""
)
points(x, bmatD2 %*% theta, col = 'blue')
grid()

par(old_par)

################################################################################
# Example 2
set.seed(42)

xvec <- seq(0.1, 9.9, length = 1000)
iknots <- sort(runif(rpois(1, 3), 1, 9))
bknots <- c(0, 10)

# basis matrix and the first and second derivatives thereof, for cubic
# (order = 4) b-splines
bmat  <- bsplines(xvec, iknots, bknots = bknots)
bmat1 <- bsplineD(xvec, iknots, bknots = bknots, derivative = 1)
bmat2 <- bsplineD(xvec, iknots, bknots = bknots, derivative = 2)

# control polygon ordinates
theta <- runif(length(iknots) + 4L, -5, 5)

# plot data
plot_data <-
  data.frame(
      Spline            = as.numeric(bmat %*% theta)
    , First_Derivative  = as.numeric(bmat1 %*% theta)
    , Second_Derivative = as.numeric(bmat2 %*% theta)
  )
plot_data <- stack(plot_data)
plot_data <- cbind(plot_data, data.frame(x = xvec))

ggplot2::ggplot(plot_data) +
ggplot2::theme_bw() +
ggplot2::aes(x = x, y = values, color = ind) +
ggplot2::geom_line() +
ggplot2::geom_hline(yintercept = 0) +
ggplot2::geom_vline(xintercept = iknots, linetype = 3)
```

bsplines *B-Splines*

Description

An implementation of Carl de Boor's recursive algorithm for building B-splines.

Usage

```
bsplines(x, iknots = NULL, df = NULL, bknots = range(x), order = 4L)
```

Arguments

`x`	a numeric vector
`iknots`	internal knots
`df`	degrees of freedom: sum of the order and internal knots. Ignored if `iknots` is specified.
`bknots`	boundary knot locations, defaults to `range(x)`.
`order`	order of the piecewise polynomials, defaults to 4L.

Details

There are several differences between this function and `bs`.

The most important difference is how the two methods treat the right-hand end of the support. `bs` uses a pivot method to allow for extrapolation and thus returns a basis matrix where non-zero values exist on the `max(Boundary.knots)` (`bs` version of `bsplines`'s `bknots`). `bsplines` use a strict definition of the splines where the support is open on the right hand side, that is, `bsplines` return right-continuous functions.

Additionally, the attributes of the object returned by `bsplines` are different from the attributes of the object returned by `bs`. See the `vignette(topic = "cpr", package = "cpr")` for a detailed comparison between the `bsplines` and `bs` calls and notes about B-splines in general.

References

C. de Boor, "A practical guide to splines. Revised Edition," Springer, 2001.

H. Prautzsch, W. Boehm, M. Paluszny, "Bezier and B-spline Techniques," Springer, 2002.

See Also

`plot.cpr_bs` for plotting the basis, `bsplineD` for building the basis matrices for the first and second derivative of a B-spline.

See `update_bsplines` for info on a tool for updating a `cpr_bs` object. This is a similar method to the `update` function from the `stats` package.

`vignette(topic = "cpr", package = "cpr")` for details on B-splines and the control polygon reduction method.

Examples

```
# build a vector of values to transform
xvec <- seq(-3, 4.9999, length = 100)

# cubic b-spline
bmat <- bsplines(xvec, iknots = c(-2, 0, 1.2, 1.2, 3.0), bknots = c(-3, 5))
bmat
```

```
# plot the splines
plot(bmat)                # each spline will be colored by default
plot(bmat, color = FALSE) # black and white plot
plot(bmat, color = FALSE) + ggplot2::aes(linetype = spline) # add a linetype

# Axes
# The x-axis, by default, show the knot locations.  Other options are numeric
# values, and/or to use a second x-axis

plot(bmat, show_xi = TRUE,  show_x = FALSE) # default, knot, symbols, on lower
                                            # axis

plot(bmat, show_xi = FALSE, show_x = TRUE)  # Numeric value for the knot
                                            # locations

plot(bmat, show_xi = TRUE,  show_x = TRUE)  # symbols on bottom, numbers on top

# quadratic splines
bmat <- bsplines(xvec, iknots = c(-2, 0, 1.2, 1.2, 3.0), order = 3L)
bmat
plot(bmat) + ggplot2::ggtitle("Quadratic B-splines")
```

---

btensor *btensor*

Description

Tensor products of B-splines.

Usage

```
btensor(x, df = NULL, iknots = NULL, bknots, order)
```

Arguments

x	a list of variables to build B-spline transforms of. The tensor product of these B-splines will be returned.
df	degrees of freedom. A list of the degrees of freedom for each marginal.
iknots	a list of internal knots for each x. If omitted, the default is to place no internal knots for all x. If specified, the list needs to contain the internal knots for all x. If `df` and `iknots` are both given, the `df` will take precedence.
bknots	a list of boundary knots for each x. As with the iknots, if omitted the default will be to use the range of each x. If specified, the use must specify the bknots for each x.
order	a list of the order for each x; defaults to 4L for all x.

Details

The return form this function is the tensor product of the B-splines transformations for the given variables. Say we have variables X, Y, and Z to build the tensor product of. The columns of the returned matrix correspond to the column products of the three B-splines:

x1y1z1 x2y1z1 x3y1z1 x4y1z1 x1y2z1 x2y2z1 ... x4y4z4

for three fourth order B-splines with no internal knots. The columns of X cycle the quickest, followed by Y, and then Z. This would be the same result as `model.matrix( ~ bsplines(X) : bsplines(Y) : bsplines(Z) + 0)`.

See `vignette(topic = "cnr", package = "cpr")` for more details.

Value

A matrix with a class `cpr_bt`

See Also

`bsplines`, `vignette(topic = "cnr", package = "cpr")`

Examples

```
tp <- with(mtcars,
           btensor(x = list(d = disp, h = hp, m = mpg),
                   iknots = list(numeric(0), c(100, 150), numeric(0)))
           )
tp
```

`build_tensor` *Build Tensor*

Description

Tensor products of Matrices.

Usage

```
build_tensor(x = NULL, y = NULL, ...)
```

Arguments

`x`	a matrix
`y`	a matrix
`...`	additional numeric matrices to build the tensor product

Value

a matrix

A matrix

See Also

`vignette("cnr", package = "cpr")` for details on tensor products.

Examples

```
A <- matrix(1:4, nrow = 10, ncol = 20)
B <- matrix(1:6, nrow = 10, ncol = 6)

# Two ways of building the same tensor product
tensor1 <- build_tensor(A, B)
tensor2 <- do.call(build_tensor, list(A, B))
all.equal(tensor1, tensor2)

# a three matrix tensor product
tensor3 <- build_tensor(A, B, B)
str(tensor3)
```

---

cn — *Control Nets*

Description

Generate the control net for a uni-variable B-spline

Usage

```
cn(x, ...)

## S3 method for class 'cpr_bt'
cn(x, theta, ...)

## S3 method for class 'formula'
cn(
  formula,
  data,
  method = stats::lm,
  method.args = list(),
  keep_fit = TRUE,
  check_rank = TRUE,
  ...
)
```

Arguments

x	a cpr_bt object
...	pass through
theta	a vector of (regression) coefficients, the ordinates of the control net.
formula	a formula that is appropriate for regression method being used.
data	a required data.frame
method	the regression method such as lm, glm, lmer, etc.
method.args	a list of additional arguments to pass to the regression method.
keep_fit	(logical, defaults to FALSE). If TRUE the regression model fit is retained and returned in the the fit element. If FALSE the regression model is not saved and the fit element will be NA.
check_rank	(logical, defaults to TRUE) if TRUE check that the design matrix is full rank.

Details

cn generates the control net for the given B-spline function. There are several methods for building a control net.

Value

a cpr_cn object. This is a list with the following elements. Some of the elements are omitted when the using the cn.cpr_bt method.

cn the control net, data.frame with each row defining a vertex of the control net

bspline_list A list of the marginal B-splines

call the call

keep_fit logical, indicates if the regression models was retained

fit if isTRUE(keep_fit) then the regression model is here, else NA.

coefficients regression coefficients, only the fixed effects if a mixed effects model was used.

vcov The variance-covariance matrix for the coefficients

loglik The log-likelihood for the regression model

rse the residual standard error for the regression models

See Also

summary.cpr_cn, cnr, plot.cpr_cn for plotting control nets

Examples

```
acn <- cn(log10(pdg) ~
            btensor(   x      = list(day, age)
                    , df     = list(30, 4)
                    , bknots = list(c(-1, 1), c(44, 53))
            )
```

```
           , data = spdg)
str(acn, max.level = 1)
```

---

cnr *Control Net Reduction*

Description

Run the Control Net Reduction Algorithm.

Usage

```
cnr(x, margin, n_polycoef = 20L, progress = c("cnr", "influence", "none"), ...)
```

Arguments

x	a cnr_cn object
margin	the margins to apply the CNR algorithm to. Passed to influence_weights.
n_polycoef	the number of polynomial coefficients to use when assessing the influence of each internal knot.
progress	controls the level of progress messaging.
...	not currently used

Details

cnr runs the control net reduction algorithm.

keep will keep the regression fit as part of the cnr_cp object for models with up to and including keep fits. For example, if keep = 10 then the resulting cnr_cnr object will have the regression fit stored in the first keep + 1 (zero internal knots, one internal knot, . . . , keep internal knots) cnr_cp objects in the list. The limit on the number of stored regression fits is to keep memory usage down.

Value

A cpr_cnr object. This is a list of cpr_cn objects.

See Also

cn for defining a control net, influence_weights for finding the influence of the internal knots, cpr for the uni-variable version, Control Polygon Reduction.

vignette(topic = "cnr", package = "cpr")

Examples

```
acn <- cn(log10(pdg) ~ btensor(list(day, age)
                              , df = list(10, 8)
                              , bknots = list(c(-1, 1), c(44, 53)))
         , data = spdg)
cnr0 <- cnr(acn)
cnr0
summary(cnr0)
plot(cnr0)
```

`coef_vcov` *Extract Regression Coefficients for B-Splines and Tensor Products of B-splines*

Description

An S3 method for extracting the regression coefficients of the `bsplines` and `btensor` terms. By Default this uses `stats::coef` to extract all the regression coefficients. A specific method for `lmerMod` objects has been provided. If you are using a regression method which `stats::coef` will not return the regression coefficients, you'll need to define an S3 method for `stats::coef` to do so.

Usage

```
coef_vcov(fit, theta_idx)
```

Arguments

`fit`	a regression model fit
`theta_idx`	numeric index for the theta related coefficients

Details

These functions are called in the `cp` and `cn` calls.

Value

A list with four elements

theta theta regression coefficients

coef all regression coefficients

vcov_theta subsection of variance-covariance matrix pertaining to the theta values

vcov full variance-covariance matrix

See Also

`coef` `cp` `cn`

Examples

```
cp0 <- cp(log10(pdg) ~ bsplines(day, df = 6, bknots = c(-1, 1)) + age + ttm, data = spdg)
cv <- cpr:::coef_vcov(cp0$fit)

summary(cv)
```

---

cp — *Control Polygons*

Description

Generate the control polygon for a uni-variable B-spline

Usage

```
cp(x, ...)

## S3 method for class 'cpr_bs'
cp(x, theta, ...)

## S3 method for class 'formula'
cp(
  formula,
  data,
  method = stats::lm,
  method.args = list(),
  keep_fit = TRUE,
  check_rank = TRUE,
  ...
)
```

Arguments

`x`	a `cpr_bs` object
`...`	pass through
`theta`	a vector of (regression) coefficients, the ordinates of the control polygon.
`formula`	a formula that is appropriate for regression method being used.
`data`	a required `data.frame`
`method`	the regression method such as `lm`, `glm`, `lmer`, etc.
`method.args`	a list of additional arguments to pass to the regression method.
`keep_fit`	(logical, default value is `TRUE`). If `TRUE` the regression model fit is retained and returned in as the `fit` element. If `FALSE` the `fit` element with be `NA`.
`check_rank`	(logical, defaults to `TRUE`) if `TRUE` check that the design matrix is full rank.

Details

cp generates the control polygon for the given B-spline function.

Value

a cpr_cp object, this is a list with the element cp, a data.frame reporting the x and y coordinates of the control polygon. Additional elements include the knot sequence, polynomial order, and other meta data regarding the construction of the control polygon.

Examples

```
# Support
xvec <- runif(n = 500, min = 0, max = 6)
bknots <- c(0, 6)

# Define the basis matrix
bmat1 <- bsplines(x = xvec, iknots = c(1, 1.5, 2.3, 4, 4.5), bknots = bknots)
bmat2 <- bsplines(x = xvec, bknots = bknots)

# Define the control vertices ordinates
theta1 <- c(1, 0, 3.5, 4.2, 3.7, -0.5, -0.7, 2, 1.5)
theta2 <- c(1, 3.4, -2, 1.7)

# build the two control polygons
cp1 <- cp(bmat1, theta1)
cp2 <- cp(bmat2, theta2)

# black and white plot
plot(cp1)
plot(cp1, show_spline = TRUE)

# multiple control polygons
plot(cp1, cp2, show_spline = TRUE)
plot(cp1, cp2, color = TRUE)
plot(cp1, cp2, show_spline = TRUE, color = TRUE)

# via formula
DF  <- data.frame(x = xvec, y = sin((xvec - 2)/pi) + 1.4 * cos(xvec/pi))
cp3 <- cp(y ~ bsplines(x, bknots = bknots), data = DF)

# plot the spline and target data.
plot(cp3, show_cp = FALSE, show_spline = TRUE) +
  ggplot2::geom_line(mapping = ggplot2::aes(x = x, y = y, color = "Target"),
                     data = DF, linetype = 2)
```

`cpr` *Control Polygon Reduction*

Description

Run the Control Polygon Reduction Algorithm.

Usage

```
cpr(x, progress = c("cpr", "influence", "none"), ...)
```

Arguments

`x`	a cpr_cp object
`progress`	controls the level of progress messaging. See Details.
`...`	not currently used

Details

`cpr` runs the control polygon reduction algorithm.

The algorithm is generally speaking fast, but can take a long time to run if the number of interior knots of initial control polygon is high. To help track the progress of the execution you can have `progress = "cpr"` which will show a progress bar incremented for each iteration of the CPR algorithm. `progress = "influence"` will use a combination of messages and progress bars to report on each step in assessing the influence of all the internal knots for each iteration of the CPR algorithm. See `influence_of_iknots` for more details.

Value

a list of cpr_cp objects

See Also

`influence_of_iknots`

Examples

```
###############################################################################
# Example 1: find a model for log10(pdg) = f(day) from the spdg data set

# need the lme4 package to fit a mixed effect model
require(lme4)

# construct the initial control polygon.  Forth order spline with fifty
# internal knots.  Remember degrees of freedom equal the polynomial order
# plus number of internal knots.
init_cp <- cp(log10(pdg) ~ bsplines(day, df = 24, bknots = c(-1, 1)) + (1|id),
              data = spdg, method = lme4::lmer)
```

```
cpr_run <- cpr(init_cp)
plot(cpr_run, color = TRUE)

s <- summary(cpr_run)
s
plot(s, type = "rse")

# preferable model is in index 5 by eye
preferable_cp <- cpr_run[["cps"]][[5]]


##############################################################################
# Example 2: logistic regression
# simulate a binary response Pr(y = 1 | x) = p(x)
p <- function(x) { 0.65 * sin(x * 0.70) + 0.3 * cos(x * 4.2) }

set.seed(42)
x <- runif(2500, 0.00, 4.5)
sim_data <- data.frame(x = x, y = rbinom(2500, 1, p(x)))

# Define the initial control polygon
init_cp <- cp(formula = y ~ bsplines(x, df = 24, bknots = c(0, 4.5)),
              data    = sim_data,
              method  = glm,
              method.args = list(family  = binomial())
              )

# run CPR
cpr_run <- cpr(init_cp)

# preferable model is in index 6
s <- summary(cpr_run)
plot(s, color = TRUE, type = "rse")

plot(
    cpr_run
  , color = TRUE
  , from = 5
  , to = 7
  , show_spline = TRUE
  , show_cp = FALSE
  )


# plot the fitted spline and the true p(x)
sim_data$pred_select_p <- plogis(predict(cpr_run[[7]], newdata = sim_data))
ggplot2::ggplot(sim_data) +
ggplot2::theme_bw() +
ggplot2::aes(x = x) +
ggplot2::geom_point(mapping = ggplot2::aes(y = y), alpha = 0.1) +
ggplot2::geom_line(
    mapping = ggplot2::aes(y = pred_select_p, color = "pred_select_p")
  ) +
```

```
ggplot2::stat_function(fun = p, mapping = ggplot2::aes(color = 'p(x)'))

# compare to gam and a binned average
sim_data$x2 <- round(sim_data$x, digits = 1)
bin_average <-
  lapply(split(sim_data, sim_data$x2), function(x) {
           data.frame(x = x$x2[1], y = mean(x$y))
         })
bin_average <- do.call(rbind, bin_average)

ggplot2::ggplot(sim_data) +
ggplot2::theme_bw() +
ggplot2::aes(x = x) +
ggplot2::stat_function(fun = p, mapping = ggplot2::aes(color = 'p(x)')) +
ggplot2::geom_line(
    mapping = ggplot2::aes(y = pred_select_p, color = "pred_select_p")
  ) +
ggplot2::stat_smooth(mapping = ggplot2::aes(y = y, color = "gam"),
                     method = "gam",
                     formula = y ~ s(x, bs = "cs"),
                     se = FALSE,
                     n = 1000) +
ggplot2::geom_line(data = bin_average
                   , mapping = ggplot2::aes(y = y, color = "bin_average"))
```

---

cpr-defunct *Defunct Functions*

Description

A major refactor of the package between v0.3.0 and v.0.4.0 took place and many functions were made defunct. The refactor was so extensive that moving the functions to deprecated was not a viable option.

Usage

```
refine_ordinate(...)

coarsen_ordinate(...)

hat_ordinate(...)

insertion_matrix(...)

wiegh_iknots(...)

influence_of(...)
```

```
influence_weights(...)
```

Arguments

...	pass through

---

cp_diff *Vertical Difference between two Control Polygons*

Description

Vertical Difference between two Control Polygons

Usage

```
cp_diff(cp1, cp2)
```

Arguments

cp1	a cpr_cp object
cp2	a cpr_cp object

Value

the vertical distance between the control vertices of cp1 to the control polygon cp2.

See Also

cp, cp_value

Examples

```
xvec <- runif(n = 500, min = 0, max = 6)

# Define the basis matrix
bmat1 <- bsplines(x = xvec, iknots = c(1, 1.5, 2.3, 4, 4.5), bknots = c(0, 6))
bmat2 <- bsplines(x = xvec, bknots = c(0, 6))

# Define the control vertices ordinates
theta1 <- c(1, 0, 3.5, 4.2, 3.7, -0.5, -0.7, 2, 1.5)
theta2 <- c(1, 3.4, -2, 1.7)

# build the two control polygons
cp1 <- cp(bmat1, theta1)
cp2 <- cp(bmat2, theta2)

cp_diff(cp1, cp2)
```

```
df <- data.frame(x = cp1$cp$xi_star,
                 y = cp1$cp$theta,
                 yend = cp1$cp$theta + cp_diff(cp1, cp2))


plot(cp1, cp2) +
ggplot2::geom_segment(data = df
  , mapping = ggplot2::aes(x = x, xend = x, y = y, yend = yend)
  , color = "red"
  , inherit.aes = FALSE)
```

---

cp_value *Control Polygon Value*

Description

Find the y value of a Control Polygon for a given x

Usage

```
cp_value(obj, x)
```

Arguments

obj	a cpr_cp object or `data.frame` where the first column is the abscissa and the second column is the ordinate for the control polygon vertices.
x	abscissa at which to determine the ordinate on control polygon cp

Value

`cp_value` returns the ordinate on the control polygon line segment for the abscissa `x` given. `x` could be a control vertex or on a line segment defined by two control vertices of the control polygon provided.

`cp_diff` returns the vertical distance between the control vertices of cp1 to the control polygon cp2.

See Also

`cp`, `cp_diff`

Examples

```
xvec <- seq(0, 6, length = 500)

# Define the basis matrix
bmat1 <- bsplines(x = xvec, iknots = c(1, 1.5, 2.3, 4, 4.5))
bmat2 <- bsplines(x = xvec)

# Define the control vertices ordinates
```

```
theta1 <- c(1, 0, 3.5, 4.2, 3.7, -0.5, -0.7, 2, 1.5)
theta2 <- c(1, 3.4, -2, 1.7)

# build the two control polygons
cp1 <- cp(bmat1, theta1)
cp2 <- cp(bmat2, theta2)

x <- c(0.2, 0.8, 1.3, 1.73, 2.15, 3.14, 4.22, 4.88, 5.3, 5.9)
cp_value(cp1, x = x)

df <- data.frame(x = x, y = cp_value(cp1, x = x))

plot(cp1, show_x = TRUE, show_spline = TRUE) +
ggplot2::geom_point(data = df
  , mapping = ggplot2::aes(x = x, y = y)
  , color = "red"
  , shape = 4
  , size = 3
  , inherit.aes = FALSE)
```

`generate_cp_formula_data` *Generate Control Polygon Formula and Data*

Description

Construct a `data.frame` and `formula` to be passed to the regression modeling tool to generate a control polygon.

Usage

```
generate_cp_formula_data(f, data, formula_only = FALSE, envir = parent.frame())
```

Arguments

`f`	a formula
`data`	the data set containing the variables in the formula
`formula_only`	if TRUE then only generate the formula, when FALSE, then generate and assign the data set too.
`envir`	the environment the generated formula and data set will be assigned too.

Details

This function is expected to be called from within the `cp` function and is not expected to be called by the end user directly.

`generate_cp_data` exists because of the need to build what could be considered a varying means model. `y ~ bsplines(x1) + x2` will generate a rank deficient model matrix—the rows of the bspline basis matrix sum to one with is perfectly collinear with the implicit intercept term. Specifying a formula `y ~ bsplines(x1) + x2 - 1` would work if `x2` is a continuous variable. If, however, `x2` is a factor, or coerced to a factor, then the model matrix will again be rank deficient as a column for all levels of the factor will be generated. We need to replace the intercept column of the model matrix with the bspline. This also needs to be done for a variety of possible model calls, `lm`, `lmer`, etc.

By returning an explicit `formula` and `data.frame` for use in the fit, we hope to reduce memory use and increase the speed of the cpr method.

We need to know the `method` and `method.args` to build the data set. For example, for a `geeglm` the `id` variable is needed in the data set and is part of the `method.args` not the `formula`.

Value

TRUE, invisibly. The return isn't needed as the assignment happens within the call.

Examples

```
data <-
  data.frame(
               x1 = runif(20)
             , x2 = runif(20)
             , x3 = runif(20)
             , xf = factor(rep(c("l1","l2","l3","l4"), each = 5))
             , xc = rep(c("c1","c2","c3","c4", "c5"), each = 4)
             , pid = gl(n = 2, k = 10)
             , pid2 = rep(1:2, each = 10)
  )

f <- ~ bsplines(x1, bknots = c(0,1)) + x2 + xf + xc + (x3 | pid2)

cpr:::generate_cp_formula_data(f, data)

stopifnot(isTRUE(
  all.equal(
            f_for_use
            ,
            . ~ bsplines(x1, bknots = c(0, 1)) + x2 + (x3 | pid2) + xfl2 +
                xfl3 + xfl4 + xcc2 + xcc3 + xcc4 + xcc5 - 1
            )
))

stopifnot(isTRUE(identical(
  names(data_for_use)
  ,
  c("x1", "x2", "x3", "pid", "pid2", "xfl2", "xfl3", "xfl4"
    , "xcc2" , "xcc3", "xcc4", "xcc5")
)))
```

get_spline — *Get the Control Polygon and the Spline Function*

Description

Generate `data.frames` for interpolating and plotting a spline function, given a `cpr_cp` or `cpr_cn` object.

Usage

```
get_spline(x, margin = 1, at, n = 100, se = FALSE, derivative = 0)
```

Arguments

`x`	a `cpr_cp` or `cpr_cn` object.
`margin`	an integer identifying the marginal of the control net to slice along. Only used when working `x` is a `cpr_cn` object.
`at`	point value for marginals not defined in the `margin`. Only used when `x` is a `cpr_cn` object. Expected input is a list of length `length(attr(x, "bspline_list"))`. Entries for elements `marginal` are ignored. If omitted, the midpoint between the boundary knots for each marginal is used.
`n`	the length of sequence to use for interpolating the spline function.
`se`	if `TRUE` return the estimated standard error for the spline or the derivative.
`derivative`	A value of 0 (default) returns the spline, 1 the first derivative, 2 the second derivative.

Details

A control polygon, `cpr\_cp` object, has a spline function f(x). `get_spline` returns a list of two `data.frame`. The `cp` element is a `data.frame` with the (x, y) coordinates control points and the `spline` element is a `data.frame` with `n` rows for interpolating f(x).

For a control net, `cpr\_cn` object, the return is the same as for a `cpr\_cp` object, but conceptually different. Where a `cpr\_cp` objects have a uni-variable spline function, `cpr\_cn` have multi-variable spline surfaces. `get_spline` returns a "slice" of the higher dimensional object. For example, consider a three-dimensional control net defined on the unit cube with marginals `x1`, `x2`, and `x3`. The implied spline surface is the function f(x1, x2, x3). `get_spline(x, margin = 2, at = list(0.2, NA, 0.5))` would return the control polygon and spline surface for f(0.2, x, 0.5).

See `get_surface` for taking a two-dimensional slice of a three-plus dimensional control net, or, for generating a useful data set for plotting the surface of a two-dimensional control net.

Value

a `data.frame` `n` rows and two columns `x` and `y`, the values for the spline. A third column with the standard error is returned if requested.

See Also

get_surface

Examples

```
data(spdg, package = "cpr")

## Extract the control polygon and spline for plotting.  We'll use base R
## graphics for this example.
a_cp <- cp(pdg ~ bsplines(day, df = 10, bknots = c(-1, 1)), data = spdg)

spline <- get_spline(a_cp)
plot(spline$x, spline$y, type = "l")

# compare to the plot.cpr_cp method
plot(a_cp, show_spline = TRUE)

# derivatives
f0 <- function(x) {
  #(x + 2) * (x - 1) * (x - 3)
  x^3 - 2 * x^2 - 5 * x + 6
}
f1 <- function(x) {
  3 * x^2 - 4 * x - 5
}
f2 <- function(x) {
  6 * x - 4
}

x <- sort(runif(n = 100, min = -3, max = 5))
bknots = c(-3, 5)
bmat <- bsplines(x, bknots = bknots)
theta <- coef(lm(f0(x) ~ bsplines(x, bknots = bknots) + 0) )

cp0 <- cp(bmat, theta)
spline0 <- get_spline(cp0, derivative = 0)
spline1 <- get_spline(cp0, derivative = 1)
spline2 <- get_spline(cp0, derivative = 2)

old_par <- par()

par(mfrow = c(1, 3))
plot(x, f0(x), type = "l", main = "spline")
points(spline0$x, spline0$y, pch = 2, col = 'blue')

plot(x, f1(x), type = "l", main = "first derivative")
points(spline1$x, spline1$y, pch = 2, col = 'blue')

plot(x, f2(x), type = "l", main = "second derivative")
points(spline2$x, spline2$y, pch = 2, col = 'blue')

par(old_par)
```

get_surface *Get Surface*

Description

Get Two-Dimensional Control Net and Surface from n-dimensional Control Nets

Usage

```
get_surface(x, margin = 1:2, at, n = 100)
```

Arguments

x	a cpr_cn object
margin	an integer identifying the marginal of the control net to slice along. Only used when working x is a cpr_cn object.
at	point value for marginals not defined in the margin. Only used when x is a cpr_cn object. Expected input is a list of length length(attr(x, "bspline_list")). Entries for elements marginal are ignored. If omitted, the midpoint between the boundary knots for each marginal is used.
n	the length of sequence to use for interpolating the spline function.

Value

a list with two elements

cn the control net

surface a data.frame with three columns to define the surface

See Also

get_spline

Examples

```
## Extract the control net and surface from a cpr_cn object.
a_cn <- cn(log10(pdg) ~ btensor(list(day, age, ttm)
           , df = list(15, 3, 5)
           , bknots = list(c(-1, 1), c(45, 53), c(-9, -1))
           , order = list(3, 2, 3))
           , data = spdg)

cn_and_surface <- get_surface(a_cn, n = 50)
str(cn_and_surface, max.level = 2)
```

```
old_par <- par()
par(mfrow = c(1, 2))
with(cn_and_surface$cn,
     plot3D::persp3D(unique(Var1),
                     unique(Var2),
                     matrix(z,
                            nrow = length(unique(Var1)),
                            ncol = length(unique(Var2))),
                     main = "Control Net")
     )
with(cn_and_surface$surface,
     plot3D::persp3D(unique(Var1),
                     unique(Var2),
                     matrix(z,
                            nrow = length(unique(Var1)),
                            ncol = length(unique(Var2))),
                     main = "Surface")
     )

par(old_par)
```

`iknots_or_df` *Internal Knots or Degrees of Freedom*

Description

Check order, degrees of freedom (df) and iknots

Usage

```
iknots_or_df(x, iknots, df, order)
```

Arguments

`x`	the support - a numeric vector
`iknots`	internal knots - a numeric vector
`df`	degrees of freedom - a numeric value of length 1
`order`	polynomial order

Details

This is an internal function, not to be exported, and used in the calls for `bsplines` and `bsplineD`.

Use `iknots` preferentially. If iknots are not provided then return the `trimmed_quantile` for the appropriate `df` and `order`

Value

a numeric vector to use as the internal knots defining a B-spline.

See Also

bsplines, bsplineD, trimmed_quantile

Examples

```
xvec <- runif(600, min = 0, max = 3)

# return the iknots
cpr:::iknots_or_df(x = xvec, iknots = 1:2, df = NULL, order = NULL)

# return the iknots even when the df and order are provided
cpr:::iknots_or_df(x = xvec, iknots = 1:2, df = 56, order = 12)

# return numeric(0) when df <= order (df < order will also give a warning)
cpr:::iknots_or_df(x = xvec, iknots = NULL, df = 6, order = 6)

# return trimmed_quantile when df > order
# probs = (df - order) / (df - order + 1)
cpr:::iknots_or_df(x = xvec, iknots = NULL, df = 10, order = 4)
cpr::trimmed_quantile(xvec, probs = 1:6 / 7)
```

---

influence_of_iknots *Determine the influence of the internal knots of a control polygon*

Description

Determine the influence of the internal knots of a control polygon

Usage

```
influence_of_iknots(x, verbose = FALSE, ...)

## S3 method for class 'cpr_cn'
influence_of_iknots(
  x,
  verbose = FALSE,
  margin = seq_along(x$bspline_list),
  n_polycoef = 20L,
  ...
)
```

Arguments

x	cpr_cp or cpr_cn object
verbose	print status messages
...	pass through

margin	which margin(s) to consider the influence of iknots
n_polycoef	number of polynomial coefficients to use when assessing the influence of a iknot

Value

a cpr_influence_of_iknots object. A list of six elements:

original_cp

coarsened_cps

restored_cps

d

influence

chisq

Examples

```
x <- seq(0 + 1/5000, 6 - 1/5000, length.out = 5000)
bmat <- bsplines(x, iknots = c(1, 1.5, 2.3, 4, 4.5), bknots = c(0, 6))
theta <- matrix(c(1, 0, 3.5, 4.2, 3.7, -0.5, -0.7, 2, 1.5), ncol = 1)
cp0 <- cp(bmat, theta)

icp0 <- influence_of_iknots(cp0)

plot(cp0, icp0$coarsened_cps[[1]], icp0$restored_cps[[1]], color = TRUE, show_spline = TRUE)
plot(cp0, icp0$restored_cps[[1]], color = TRUE, show_spline = TRUE)

plot(cp0, icp0$coarsened_cps[[2]], icp0$restored_cps[[2]], color = TRUE, show_spline = TRUE)
plot(cp0, icp0$restored_cps[[2]], color = TRUE, show_spline = TRUE)

plot(cp0, icp0$coarsened_cps[[3]], icp0$restored_cps[[3]], color = TRUE, show_spline = TRUE)
plot(cp0, icp0$restored_cps[[3]], color = TRUE, show_spline = TRUE)

plot(cp0, icp0$coarsened_cps[[4]], icp0$restored_cps[[4]], color = TRUE, show_spline = TRUE)
plot(cp0, icp0$restored_cps[[4]], color = TRUE, show_spline = TRUE)

plot(cp0, icp0$coarsened_cps[[5]], icp0$restored_cps[[5]], color = TRUE, show_spline = TRUE)
plot(cp0, icp0$restored_cps[[5]], color = TRUE, show_spline = TRUE)

# When the cp was defined by regression
df <- data.frame(x = x, y = as.numeric(bmat %*% theta) + rnorm(5000, sd = 0.2))
cp1 <- cp(y ~ bsplines(x, iknots = c(1, 1.5, 2.3, 3, 4, 4.5), bknots = c(0, 6)), data = df)
icp1 <- influence_of_iknots(cp1)
icp1
```

insert_a_knot *Insert a Knot into a Control Polygon*

Description

Insert a knot into a control polygon without changing the spline

Usage

```
insert_a_knot(x, xi_prime, ...)
```

Arguments

x	a cpr_cp object
xi_prime	the value of the knot to insert
...	not currently used

Value

a cpr_cp object

Examples

```
x <- seq(1e-5, 5.99999, length.out = 100)
bmat <- bsplines(x, iknots = c(1, 1.5, 2.3, 4, 4.5), bknots = c(0, 6))
theta <- matrix(c(1, 0, 3.5, 4.2, 3.7, -0.5, -0.7, 2, 1.5), ncol = 1)
cp0 <- cp(bmat, theta)
cp1 <- insert_a_knot(x = cp0, xi_prime = 3)
plot(cp0, cp1, color = TRUE, show_spline = TRUE)
```

knot_expr *Knot Expressions*

Description

Non-exported function used to build expressions for the knot sequences to be labeled well on a plot.

Usage

```
knot_expr(x, digits)
```

Arguments

x	a cpr_cp or cpr_bs object
digits	digits to the right of the decimal point to report

Value

a list

Examples

```
bmat <- bsplines(mtcars$hp, df = 8, bknots = c(50, 350))
ke <- cpr:::knot_expr(bmat, digits = 1)
summary(ke)

plot(x = ke$breaks, y = rep(1, length(ke$breaks)), type = "n")
text(
       x = ke$breaks
     , y = rep(1, length(ke$breaks))
     , labels = parse(text = ke$xi_expr)
)
```

---

loglikelihood — *Determine the (quasi) Log Likelihood for a regression object.*

Description

Return, via logLik or a custom S3 method, the (quasi) log likelihood of a regression object.

Usage

```
loglikelihood(x, ...)
```

Arguments

x	a regression fit object
...	passed through to logLik

Details

This function is used by cpr and cnr to determine the (quasi) log likelihood returned in the cpr_cpr and cpr_cnr objects.

Generally this function defaults to logLik. Therefore, if an S3 method for determining the (quasi) log likelihood exists in the workspace everything should work. If an S3 method does not exist you should define one.

See methods(loglikelihood) for a list of the provided methods. The default method uses logLik.

Value

the numeric value of the (quasi) log likelihood.

See Also

cpr cnr logLik

Examples

```
fit <- lm(mpg ~ wt, data = mtcars)
stats::logLik(fit)
cpr:::loglikelihood(fit)
```

---

matrix_rank *Rank of a Matrix*

Description

Determine the rank (number of linearly independent columns) of a matrix.

Usage

```
matrix_rank(x)
```

Arguments

x	a numeric matrix

Details

Implementation via the Armadillo C++ linear algebra library. The function returns the rank of the matrix `x`. The computation is based on the singular value decomposition of the matrix; a std::runtime_error exception will be thrown if the decomposition fails. Any singular values less than the tolerance are treated as zeros. The tolerance is `max(m, n) * max_sv * arma::datum::eps`, where `m` is the number of rows of `x`, `n` is the number of columns of `x`, `max_sv` is the maximal singular value of `x`, and `arma::datum::eps` is the difference between 1 and the least value greater than 1 that is representable.

Value

the rank of the matrix as a numeric value.

References

Conrad Sanderson and Ryan Curtin. Armadillo: a template-based C++ library for linear algebra. Journal of Open Source Software, Vol. 1, pp. 26, 2016.

Examples

```
# Check the rank of a matrix
set.seed(42)
mat <- matrix(rnorm(25000 * 120), nrow = 25000)
matrix_rank(mat) == ncol(mat)
matrix_rank(mat) == 120L

# A full rank B-spline basis
bmat <- bsplines(seq(0, 1, length = 100), df = 15)
matrix_rank(bmat) == 15L

# A rank deficient B-spline basis
bmat <- bsplines(seq(0, 1, length = 100), iknots = c(0.001, 0.002))
ncol(bmat) == 6L
matrix_rank(bmat) == 5L
```

---

newknots *New Knots for CPs and CNs in CPR and CNR*

Description

Non-exported function, `newknots` are used in the `cpr` and `cnr` calls. Used to create a new control polygon or control net from with different internal knots.

Usage

```
newknots(form, nk)
```

Arguments

`form`	a `formula`
`nk`	numeric vector, or a list of numeric vectors, to be used in a `bsplines` or `btensor` call, respectively.

Details

Think of this function as an analogue to the `stats{update}` calls. Where `stats{update}` will modify a `call`, the `newknots` will update just the `iknots` argument of a `bsplines` or `btensor` call within the `formula` argument of a `cp` or `cn` call.

Value

Expected use is within the `cpr` and `cnr` calls. The return object a formula to define a control polygon/net with different knots than then ones found within `form`.

See Also

`update_bsplines` for a more generic tool for the end user.

Examples

```
cp0 <- cp(log(pdg) ~ bsplines(day, iknots = c(-.25, 0, 0.25), bknots = c(-1, 1)), data = spdg)

new_knots <- c(-0.85, 0, 0.25, 0.3)
f <- cpr:::newknots(cp0$call$formula, nk = new_knots)
f
cp(f, data = spdg)
```

---

order_statistics *Distribution of Order Statistics*

Description

Density of distribution function for the jth order statistics from a sample of size n from a known distribution function.

Usage

```
d_order_statistic(x, n, j, distribution, ...)

p_order_statistic(q, n, j, distribution, ...)
```

Arguments

`x, q`	vector or quantiles
`n`	sample size
`j`	jth order statistics
`distribution`	character string defining the distribution. See Details.
`...`	additional arguments passed to the density and distribution function

Details

For a known distribution with defined density and distribution functions, e.g., normal (`dnorm`, `pnorm`), or chisq (`dchisq`, `pchisq`), we define the density function of of the jth order statistic, from a sample of size n, to be

$$\frac{n!}{(j-1)!(n-j)!} f(x) F(x)^{j-1} (1 - F(x))^{n-j}$$

.

and the distribution function to be

$$\sum_{k=j}^{n} \binom{n}{k} \left[F(x)\right]^{k} \left[1 - F(x)\right]^{n-k}$$

.

Value

a numeric vector

References

George Casella and Roger L. Berger (2002). Statistical Inference. 2nd edition. Duxbury Thomson Learning.

Examples

```
# Example 1
# Find the distribution of the minimum from a sample of size 54 from a
# standard normal distribution

simulated_data <- matrix(rnorm(n = 54 * 5000), ncol = 54)

# find all the minimums for each of the simulated samples of size 54
mins <- apply(simulated_data, 1, min)

# get the density values
x <- seq(-5, 0, length.out = 100)
d <- d_order_statistic(x, n = 54, j = 1, distribution = "norm")

# plot the histogram and density
hist(mins, freq = FALSE)
points(x, d, type = "l", col = "red")

# plot the distribution function
plot(ecdf(mins))
points(x, p_order_statistic(q = x,  n = 54, j = 1, distribution = "norm"), col = "red")

# Example 2
# Find the density and distrubition of the fourth order statistic from a
# sample of size 12 from a chisq distribution with 3 degrees of freedom

simulated_data <- matrix(rchisq(n = 12 * 5000, df = 3), ncol = 12)

os4 <- apply(simulated_data, 1, function(x) sort(x)[4])

x <- seq(min(os4), max(os4), length.out = 100)
d <- d_order_statistic(x, n = 12, j = 4, distribution = "chisq", df = 3)
p <- p_order_statistic(x, n = 12, j = 4, distribution = "chisq", df = 3)

hist(os4, freq = FALSE); points(x, d, type = "l", col = "red")
plot(ecdf(os4)); points(x, p, col = "red")

# Example 3
# For a set of j observations, find the values for each of the j order
# statistics
simulated_data <- matrix(rnorm(n = 6 * 5000), ncol = 6)
simulated_data <- apply(simulated_data, 1, sort)
```

```
xs <- apply(simulated_data, 1, range)
xs <- apply(xs, 2, function(x) {seq(x[1], x[2], length.out = 100)})
ds <- apply(xs, 1, d_order_statistic, n = 6, j = 1:6, distribution = "norm")
ps <- apply(xs, 1, p_order_statistic, n = 6, j = 1:6, distribution = "norm")

old_par <- par() # save current settings

par(mfrow = c(2, 3))
for (i in 1:6) {
  hist(simulated_data[i, ]
       , freq = FALSE
       , main = substitute(Density~of~X[(ii)], list(ii = i))
       , xlab = ""
  )
  points(xs[, i], ds[i, ], type = "l", col = "red")
}

for (i in 1:6) {
  plot(ecdf(simulated_data[i, ])
       , main = substitute(CDF~of~X[(ii)], list(ii = i))
       , ylab = ""
       , xlab = ""
       )
  points(xs[, i], ps[i, ], type = "p", col = "red")
}

par(mfrow = c(1, 1))
plot(xs[, 1], ps[1, ], type = "l", col = 1, xlim = range(xs), ylab = "", xlab = "")
for(i in 2:6) {
  points(xs[, i], ps[i, ], type = "l", col = i)
}
legend("topleft", col = 1:6, lty = 1, legend =
       c(
         expression(CDF~of~X[(1)]),
         expression(CDF~of~X[(2)]),
         expression(CDF~of~X[(3)]),
         expression(CDF~of~X[(4)]),
         expression(CDF~of~X[(5)]),
         expression(CDF~of~X[(5)])
         ))

par(old_par) # reset par to setting prior to running this example
```

`plot.cpr_bs` *Plot B-spline Basis*

Description

Wrapper around several ggplot2 calls to plot a B-spline basis

Usage

```
## S3 method for class 'cpr_bs'
plot(x, ..., show_xi = TRUE, show_x = FALSE, color = TRUE, digits = 2, n = 100)
```

Arguments

x	a cpr_bs object
show_xi	logical, show the knot locations, using the Greek letter xi, on the x-axis
show_x	logical, show the x values of the knots on the x-axis
color	logical, if TRUE (default) the splines are plotted in color. If FALSE all splines are black lines.
digits	number of digits to the right of the decimal place to report for the value of each knot.
n	number of values to use to plot the splines, defaults to 100
...	not currently used

Value

a ggplot

See Also

bsplines

Examples

```
bmat <- bsplines(seq(-3, 2, length = 1000), iknots = c(-2, 0, 0.2))
plot(bmat, show_xi = TRUE,  show_x = TRUE)
plot(bmat, show_xi = FALSE, show_x = TRUE)
plot(bmat, show_xi = TRUE,  show_x = FALSE)  ## Default
plot(bmat, show_xi = FALSE, show_x = FALSE)
plot(bmat, show_xi = FALSE, show_x = FALSE)
plot(bmat, show_xi = FALSE, show_x = FALSE, color = FALSE)
```

---

plot.cpr_cn *Plotting Control Nets*

Description

Three-dimensional plots of control nets and/or surfaces

Usage

```
## S3 method for class 'cpr_cn'
plot(
  x,
  ...,
  xlab = "",
  ylab = "",
  zlab = "",
  show_net = TRUE,
  show_surface = FALSE,
  get_surface_args,
  net_args,
  surface_args,
  rgl = TRUE
)
```

Arguments

`x`	a `cpr_cn` object
`...`	common arguments which would be used for both the plot of the control net and the surface, e.g., xlim, ylim, zlim.
`xlab, ylab, zlab`	labels for the axes.
`show_net`	logical, show the control net
`show_surface`	logical, show the tensor product surface
`get_surface_args`	a list of arguments passed to the `get_surface` call. This call generates the needed data sets used in the plotting.
`net_args`	arguments to be used explicitly for the control net. Ignored if `show_net = FALSE`.
`surface_args`	arguments to be used explicitly for the surface. Ignored if `show_surface = FALSE`.
`rgl`	If `TRUE`, the default, generate use `rgl::persp3d` to generate the graphics. If `FALSE`, use `plot3D::persp3D` to generate the graphics.

Details

This plotting method generates three-dimensional plots of the control net, surface, or both, for a `cpr_cn` objects. The three-dimensional plots are generated by either `persp3D` form the `plot3D` package or `persp3d` from the `rgl` package. `rgl` graphics may or may not work on your system depending on support for OpenGL.

Building complex and customized graphics might be easier for you if you use `get_surface` to generate the needed data for plotting. See `vignette(topic = "cnr", package = "cpr")` for examples of building different plots.

For rgl graphics, the surface_args and net_args are lists of rgl.material and other arguments passed to persp3d. Defaults are col = "black", front = "lines", back = "lines" for the net_args and col = "grey20", front = "fill", back = "lines" for the surface_args.

For plot3D graphics there are no defaults values for the net_args and surface_args.

Value

the plotting data needed to generate the plot is returned invisibly.

See Also

plot.cpr_cp for plotting control polygons and splines, persp3d and rgl.material for generating and controlling rgl graphics. persp3D for building plot3D graphics. get_surface for generating the data sets needed for the plotting methods.

vignette(topic = "cnr", package = "cpr")

Examples

```
 acn <- cn(log10(pdg) ~ btensor(  x = list(day, age)
                                , df = list(30, 4)
                                , bknots = list(c(-1, 1), c(44, 53)))
           , data = spdg)

# plot3D
plot(acn, rgl = FALSE)

# rgl
if (require(rgl)) {
  plot(acn, rgl = TRUE)
}
```

---

plot.cpr_cnr *Control Net Reduction Plots*

Description

A collection of function for the inspection and evaluation of the control polygon reduction.

Usage

```
## S3 method for class 'cpr_cnr'
plot(x, type = "rse", from = 1, to, ...)
```

Arguments

x	a cpr_cnr object
type	type of diagnostic plot. "loglik" for the log likelihood by degrees of freedom, "rse" for residual standard error by model index
from	the first index of x to plot
to	the last index of x to plot
...	pass through

Value

a ggplot

Examples

```
initial_cn <- cn(log10(pdg) ~ btensor(list(day, age)
                        , df = list(10, 8)
                        , bknots = list(c(-1, 1), c(44, 53))
                        )
          , data = spdg)

cnr0 <- cnr(initial_cn)

plot(cnr0)
```

---

plot.cpr_cp *Plotting Control Polygons*

Description

Plotting control polygon(s) and/or the associated spline(s) via ggplot2

Usage

```
## S3 method for class 'cpr_cp'
plot(
  x,
  ...,
  comparative,
  show_cp = TRUE,
  show_spline = FALSE,
  show_xi = TRUE,
  color = FALSE,
  n = 100,
  show_x = FALSE,
  digits = 2
)
```

Arguments

x	a cpr_cp object
...	additional cpr_cp objects
comparative	when TRUE use color to distinguish one spline from another, when FALSE color to highlight the control polygon and spline with different colors, and plot the knots the way plot.cpr_bs does. When missing, the default if TRUE if more than one cpr_cp object is passed in, and FALSE is only one cpr_cp object is passed.
show_cp	logical (default TRUE), show the control polygon(s)?
show_spline	logical (default FALSE) to plot the spline function?
show_xi	logical (default TRUE) use geom_rug to show the location of the knots in the respective control polygons.
color	Boolean (default FALSE) if more than one cpr_cp object is to be plotted, set this value to TRUE to have the graphic in color (line types will be used regardless of the color setting).
n	the number of data points to use for plotting the spline
show_x	boolean, so x-values
digits	number of digits to the right of the decimal place to report for the value of each knot. Only used when plotting on control polygon with comparative = FALSE.

Value

a ggplot object

Examples

```
x <- runif(n = 500, 0, 6)
bmat <- bsplines(x, iknots = c(1, 1.5, 2.3, 4, 4.5), bknots = c(0, 6))
theta1 <- matrix(c(1, 0, 3.5, 4.2, 3.7, -0.5, -0.7, 2, 1.5), ncol = 1)
theta2 <- theta1 + c(-0.15, -1.01, 0.37, 0.19, -0.53, -0.84, -0.19, 1.15, 0.17)
cp1 <- cp(bmat, theta1)
cp2 <- cp(bmat, theta2)

# compare two control polygons on one plot
plot(cp1, cp2)
plot(cp1, cp2, color = TRUE)
plot(cp1, cp2, color = TRUE, show_spline = TRUE)
plot(cp1, cp2, color = TRUE, show_cp = FALSE, show_spline = TRUE)

# Show one control polygon with knots on the axis instead of the rug and
# color/linetype for the control polygon and spline, instead of different
# control polygons
plot(cp1, comparative = FALSE)
plot(cp1, comparative = FALSE, show_spline = TRUE)
plot(cp1, comparative = FALSE, show_spline = TRUE, show_x = TRUE)
plot(cp2, comparative = FALSE, show_spline = TRUE, show_x = TRUE)
```

plot.cpr_cpr *Control Polygon Reduction Plots*

Description

A wrapper around several ggplot2 calls to help evaluate results of a CPR run.

Usage

```
## S3 method for class 'cpr_cpr'
plot(x, from = 1, to, ...)
```

Arguments

x	a cpr_cpr object
from	the first index of x to plot
to	the last index of x to plot
...	arguments passed to plot.cpr_cp

Value

a ggplot object

See Also

plot.cpr_cp, cpr, cp

Examples

```
set.seed(42)
x <- seq(0 + 1/5000, 6 - 1/5000, length.out = 100)
bmat <- bsplines(x, iknots = c(1, 1.5, 2.3, 4, 4.5), bknots = c(0, 6))
theta <- matrix(c(1, 0, 3.5, 4.2, 3.7, -0.5, -0.7, 2, 1.5), ncol = 1)
DF <- data.frame(x = x, truth = as.numeric(bmat %*% theta))
DF$y <- as.numeric(bmat %*% theta + rnorm(nrow(bmat), sd = 0.3))

initial_cp0 <-
  cp(y ~ bsplines(x, iknots = c(1, 1.5, 2.3, 3.0, 4, 4.5), bknots = c(0, 6))
     , data = DF
     , keep_fit = TRUE # default is FALSE
  )
cpr0 <- cpr(initial_cp0)

plot(cpr0)
plot(cpr0, show_spline = TRUE, show_cp = FALSE, color = TRUE, from = 2, to = 4)
```

plot.cpr_summary_cpr_cpr
Plotting Summaries of Control Polygon Reductions

Description

Plotting Summaries of Control Polygon Reductions

Usage

```
## S3 method for class 'cpr_summary_cpr_cpr'
plot(
  x,
  type = c("rse", "rss", "loglik", "wiggle", "fdsc", "Pr(>w_(1))"),
  from = 1,
  to,
  ...
)
```

Arguments

x	a cpr_summary_cpr_cpr object
type	response to plot by index
from	the first index of x to plot
to	the last index of x to plot
...	pass through

Value

a ggplot object

See Also

plot.cpr_cpr, cpr summary.cpr_cpr

Examples

```
set.seed(42)
x <- seq(0 + 1/5000, 6 - 1/5000, length.out = 100)
bmat <- bsplines(x, iknots = c(1, 1.5, 2.3, 4, 4.5), bknots = c(0, 6))
theta <- matrix(c(1, 0, 3.5, 4.2, 3.7, -0.5, -0.7, 2, 1.5), ncol = 1)
DF <- data.frame(x = x, truth = as.numeric(bmat %*% theta))
DF$y <- as.numeric(bmat %*% theta + rnorm(nrow(bmat), sd = 0.3))

initial_cp0 <-
  cp(y ~ bsplines(x, iknots = c(1, 1.5, 2.3, 3.0, 4, 4.5), bknots = c(0, 6))
     , data = DF
```

```
      , keep_fit = TRUE # default is FALSE
  )
cpr0 <- cpr(initial_cp0)
s0   <- summary(cpr0)

plot(s0, type = "rse")
plot(s0, type = "rss")
plot(s0, type = "loglik")
plot(s0, type = "wiggle")
plot(s0, type = "fdsc")
plot(s0, type = "Pr(>w_(1))")
```

---

predict.cpr_cp *Model Prediction*

Description

Model prediction for cpr_cp and cpr_cn objects.

Usage

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

Arguments

object	a cpr_cp or cpr_cn object
...	passed to predict

Value

the same as you would get from calling predict on the object$fit.

Examples

```
acp <- cp(log10(pdg) ~ bsplines(age, df = 12, bknots = c(45, 53))
           , data = spdg
          , keep_fit = TRUE)
acp_pred0 <- predict(acp$fit, se.fit = TRUE)
acp_pred <- predict(acp, se.fit = TRUE)
all.equal(acp_pred0, acp_pred)
```

print.cpr_bs *Print bsplines*

Description

Print bsplines

Usage

```
## S3 method for class 'cpr_bs'
print(x, n = 6L, ...)
```

Arguments

x	a cpr_bs object.
n,	number of rows of the B-spline basis matrix to display, defaults to 6L.
...	not currently used.

Value

the object x is returned invisibly

sign_changes *Sign Changes*

Description

Count the number of times the first, or second, derivative of a spline changes sign.

Usage

```
sign_changes(
  object,
  lower = min(object$bknots),
  upper = max(object$bknots),
  n = 1000,
  derivative = 1L,
  ...
)
```

Arguments

object	a cpr_cp object
lower	the lower limit of the integral
upper	the upper limit of the integral
n	number of values to assess the derivative between lower and upper.
derivative	integer value denoted first or second derivative
...	pass through

Value

the number of times the sign of the first or second derivative changes within the specified interval.

See Also

wiggle

Examples

```
xvec <- seq(0, 6, length = 500)

# Define the basis matrix
bmat1 <- bsplines(x = xvec, iknots = c(1, 1.5, 2.3, 4, 4.5))
bmat2 <- bsplines(x = xvec)

# Define the control vertices ordinates
theta1 <- c(1, 0, 3.5, 4.2, 3.7, -0.5, -0.7, 2, 1.5)
theta2 <- c(1, 3.4, -2, 1.7)

# build the two control polygons
cp1 <- cp(bmat1, theta1)
cp2 <- cp(bmat2, theta2)
plot(cp1, cp2, show_cp = FALSE, show_spline = TRUE)

sign_changes(cp1)
sign_changes(cp2)
```

---

spdg — *Simulated Pregnanediol glucuronide (PDG) Data*

Description

A Simulated data set based on the Study of Women's Health Across the Nation (SWAN) Daily Hormone Study (DHS).

Usage

```
spdg
```

Format

a `data.frame`. Variables in the data set:

id Subject ID

age Age, in years of the subject

ttm Time-to-menopause, in years

ethnicity Ethnicity, a factor with five levels: Caucasian, Black, Chinese, Hispanic, and Japanese

bmi Body Mass Index

day_from_dlt A integer value for the number of days from Day of Luteal Transition (DLT). The DLT is `day_from_dlt == 0`. Negative values indicate the follicular phase, positive values for the luteal phase.

day_of_cycle the day of cycle

day A scaled day-of-cycle between [-1, 1] with 0 for the DLT. See Details

pdg A simulated PDG value

Details

Pregnanediol glucuronide (PDG) is the urine metabolite of progesterone. This data set was simulated to have similar characteristics to a subset of the SWAN DHS data. The SWAN DHS data was the motivating data set for the method development that lead to the `cpr` package. The DHS data cannot be made public, so this simulated data set has been provided for use in examples and instructions for use of the `cpr` package.

Source

This is simulated data. To see the script that generated the data set please visit https://github.com/dewittpe/cpr and look at the scripts in the data-raw directory.

References

Santoro, Nanette, et al. "Body size and ethnicity are associated with menstrual cycle alterations in women in the early menopausal transition: The Study of Women's Health across the Nation (SWAN) Daily Hormone Study." The Journal of Clinical Endocrinology & Metabolism 89.6 (2004): 2622-2631.

summary.cpr_cn *Summary of Control Net*

Description

Generate a summary of control net object

Usage

```
## S3 method for class 'cpr_cn'
summary(object, ...)
```

Arguments

`object`	a `cpr_cn` object
`...`	pass through

Value

a `data.frame`

Examples

```

acn <- cn(log10(pdg) ~ btensor(list(day, age)
                               , df = list(10, 8)
                               , bknots = list(c(-1, 1), c(44, 53)))
         , data = spdg)

summary(acn)
```

summary.cpr_cnr *Summarize Control Net Reduction Objects*

Description

Summarize Control Net Reduction Objects

Usage

```
## S3 method for class 'cpr_cnr'
summary(object, ...)
```

Arguments

object	a cpr_cnr object
...	pass through

Value

a cpr_summary_cpr_cnr object, that is just a data.frame

Examples

```
acn <- cn(log10(pdg) ~ btensor(list(day, age)
                               , df = list(10, 8)
                               , bknots = list(c(-1, 1), c(44, 53)))
         , data = spdg)
cnr0 <- cnr(acn)
cnr0
summary(cnr0)
```

---

summary.cpr_cp *Summarize a Control Polygon Object*

Description

Summarize a Control Polygon Object

Usage

```
## S3 method for class 'cpr_cp'
summary(object, wiggle = TRUE, integrate.args = list(), ...)
```

Arguments

object	a cpr_cp object
wiggle	logical, if TRUE then the integral of the squared second derivative of the spline function will be calculated via integrate.
integrate.args	a list of arguments passed to wiggle and ultimately integrate.
...	pass through

Value

a cpr_summary_cpr_cp object, that is just a data.frame

Examples

```
set.seed(42)
x <- seq(0 + 1/5000, 6 - 1/5000, length.out = 100)
bmat <- bsplines(x, iknots = c(1, 1.5, 2.3, 4, 4.5), bknots = c(0, 6))
theta <- matrix(c(1, 0, 3.5, 4.2, 3.7, -0.5, -0.7, 2, 1.5), ncol = 1)
DF <- data.frame(x = x, truth = as.numeric(bmat %*% theta))
DF$y <- as.numeric(bmat %*% theta + rnorm(nrow(bmat), sd = 0.3))

initial_cp <-
  cp(y ~ bsplines(x, iknots = c(1, 1.5, 2.3, 3.0, 4, 4.5), bknots = c(0, 6))
     , data = DF
     , keep_fit = TRUE # default is FALSE
  )

summary(initial_cp)
```

---

summary.cpr_cpr *Summarize a Control Polygon Reduction Object*

Description

Summarize a Control Polygon Reduction Object

Usage

```
## S3 method for class 'cpr_cpr'
summary(object, ...)
```

Arguments

`object`	a `cpr_cpr` object
`...`	pass through

Value

a `data.frame` with the attribute `elbow` which is a programmatic attempt to identify a useful trade-off between degrees of freedom and fit statistic.

Examples

```
set.seed(42)
x <- seq(0 + 1/5000, 6 - 1/5000, length.out = 100)
bmat <- bsplines(x, iknots = c(1, 1.5, 2.3, 4, 4.5), bknots = c(0, 6))
theta <- matrix(c(1, 0, 3.5, 4.2, 3.7, -0.5, -0.7, 2, 1.5), ncol = 1)
DF <- data.frame(x = x, truth = as.numeric(bmat %*% theta))
DF$y <- as.numeric(bmat %*% theta + rnorm(nrow(bmat), sd = 0.3))
```

```
initial_cp <-
  cp(y ~ bsplines(x, iknots = c(1, 1.5, 2.3, 3.0, 4, 4.5), bknots = c(0, 6))
     , data = DF
     , keep_fit = TRUE # default is FALSE
  )

cpr0 <- cpr(initial_cp)
s <- summary(cpr0)
s
plot(s, type = "rse")
```

---

trimmed_quantile *Trimmed Quantiles*

Description

For data $X = x_1, x_2, \ldots, x_n$, with order statistics $x_{(1)}, x_{(2)}, \ldots, x_{(r)}$ return the quantiles for a trimmed data set, e.g., $\boldsymbol{X} \backslash \{x_{(1)}, x_{(r)}\}$ (trim = 1), or $\boldsymbol{X} \backslash \{x_{(1)}, x_{(2)}, x_{(r-1)}, x_{(r)}\}$ (trim = 2).

Usage

```
trimmed_quantile(x, trim = 1L, use_unique = TRUE, ...)
```

Arguments

x	a numeric vector
trim	defaults to 1, omitting the min and the max
use_unique	logical, if true (defaults), base the quantiles on unique values, if false, base the quantiles on all data, after trimming.
...	other arguments to pass to stats::quantile

Value

a numeric vector, the return from quantile

See Also

quantile

Examples

```
trimmed_quantile(1:100, prob = 1:23 / 24, name = FALSE)

# Warning
# trimmed_quantile(1:100, trim = .3, prob = 1:23 / 24, name = FALSE)

# no warning
trimmed_quantile(1:100, trim = 3, prob = 1:23 / 24, name = FALSE)
```

update_bsplines *Update bsplines or btensor calls*

Description

Update cpr_bs and cpr_bt objects alone or within cpr_cp and cpr_cn objects.

Usage

```
update_bsplines(object, ..., evaluate = TRUE)

update_btensor(object, ..., evaluate = TRUE)
```

Arguments

object	an object to update.
...	arguments to update, expected to be iknots, df, bknots, or order.
evaluate	whether or not to evaluate the updated call.

Value

If evaluate = TRUE then a cpr_bs or cpr_bt object is returned, else, an unevaluated call is returned.

See Also

update, bsplines, btensor

Examples

```
################################################################################
##                          Updating a cpr_bs object                          ##
# construct a B-spline basis
bmat <- bsplines(runif(10, 1, 10), df = 5, order = 3, bknots = c(1, 10))

# look at the structure of the basis
str(bmat)

# change the order
str(update_bsplines(bmat, order = 4))

# change the order and the degrees of freedom
str(update_bsplines(bmat, df = 12, order = 4))

################################################################################
##                          Updating a cpr_bt object                          ##
# construct a tensor product
tpmat <- btensor(list(x1 = seq(0, 1, length = 10), x2 = seq(0, 1, length = 10)),
                 df = list(4, 5))
tpmat
```

```
# update the degrees of freedom
update_btensor(tpmat, df = list(6, 7))

################################################################################
##      Updating bsplines or btensor on the right and side of a formula       ##

f1 <- y ~ bsplines(x, df = 14) + var1 + var2
f2 <- y ~ btensor(x = list(x1, x2), df = list(50, 31), order = list(3, 5))  + var1 + var2

update_bsplines(f1, df = 13, order = 5)
update_btensor(f2, df = list(13, 24), order = list(3, 8))

################################################################################
##                          Updating a cpr_cp object                          ##
data(spdg, package = "cpr")
init_cp <- cp(pdg ~ bsplines(day, df = 30) + age + ttm, data = spdg)
updt_cp <- update_bsplines(init_cp, df = 5)

################################################################################
##                          Updating a cpr_cn object                          ##
init_cn <- cn(pdg ~ btensor(list(day, age), df = list(30, 4)) + ttm, data = spdg)
updt_cn <- update_btensor(init_cn, df = list(30, 2), order = list(3, 2))
```

`us_covid_cases` *United States Laboratory Confirmed COVID-19 Cases*

Description

Number of laboratory-confirmed COVID-19 cases in the United States, as reported by the Centers for Disease Control, between January 1 2020 and May 11, 2023, the end of the public health emergency declaration.

Usage

```
us_covid_cases
```

Format

a `data.frame` with two columns

date year, month, day

cases number of reported laboratory-confirmed COVID-19 cases

Source

Download original data from <https://data.cdc.gov/Case-Surveillance/COVID-19-Case-Surveillance-Public-Use-Data/vbim-akqf> on December 5, 2023. The reported data set was last updated on November 3, 2023.

`wiggle` *Wiggliness of a Spline function*

Description

Calculate the integral of the squared second derivative of the spline function.

Usage

```
wiggle(object, lower, upper, stop.on.error = FALSE, ...)
```

Arguments

`object`	a `cpr_cp` object
`lower`	the lower limit of the integral
`upper`	the upper limit of the integral
`stop.on.error`	default to `FALSE`, see `integrate`.
`...`	additional arguments passed to `integrate`

Details

The wiggliness of the spline function is defined as

$$\int \left(\frac{\mathrm{d}^2}{\mathrm{d}x^2} f\left(x\right) \right)^2 \mathrm{d}x.$$

Value

Same as `integrate`.

See Also

`cp`, `integrate`, `sign_changes`

Examples

```
xvec <- seq(0, 6, length = 500)

# Define the basis matrix
bmat1 <- bsplines(x = xvec, iknots = c(1, 1.5, 2.3, 4, 4.5))
bmat2 <- bsplines(x = xvec)

# Define the control vertices ordinates
theta1 <- c(1, 0, 3.5, 4.2, 3.7, -0.5, -0.7, 2, 1.5)
theta2 <- c(1, 3.4, -2, 1.7)

# build the two control polygons
```

```
cp1 <- cp(bmat1, theta1)
cp2 <- cp(bmat2, theta2)
plot(cp1, cp2, show_cp = FALSE, show_spline = TRUE)

wiggle(cp1)
wiggle(cp2)
```

Index