Package 'spluti'

February 3, 2021

Version 1.0

Date 2021-02-03

Title Utility Functions for Post-Processing Univariate Interpolation Splines, Smoothing Splines and Regression Splines

Author Zheyuan Li [aut, cre] (<https://orcid.org/0000-0002-7434-5947>)

Maintainer Zheyuan Li <zheyuan.li@bath.edu>

Depends R (>= 4.0.0), splines

Imports stats, graphics, utils

Suggests nlme, lme4

Description Univariate splines in regression and smoothing models often adopt basis representation, which is neither intuitive to interpret, nor easy to post-process (like to solve an inverse problem where x-values such that the spline or an order of its derivative equals to a given value is backsolved). This package offers functions that transform regression splines to piecewise polynomials and piecewise polynomial methods for standard generic functions like summary(), plot(), predict() and solve(). Users can then easily export piecewise polynomial equations as formatted strings for pretty print, plot a spline or its derivatives (optionally with knots shown), predict a spline or its derivatives at new x-values, and solve an inverse problem. Extra tools for piecewise polynomials like adding two piecewise polynomials are provided as well. While the package focuses on regression splines, it also transforms interpolation splines and smoothing splines to piecewise polynomials, hence all methods and tools for piecewise polynomials are applicable.

License GPL-3

LazyData true

NeedsCompilation yes

R topics documented:

`AddInterceptPP` *Add regression intercept to a "PP" or "PPA" object*

Description

A regression B-spline constructed by `splines::bs` and `splines::ns` does not contain intercept by default, so that the spline is pinned to 0 at its left boundary knot. This is a reasonable setup as otherwise the intercept in the regression model will be unidentifiable. However, for plotting and prediction purpose or to solve an inverse problem, we may want to incorporate the estimated model intercept to the fitted spline, and `AddInterceptPP` is a handy function for doing this.

Usage

```
AddInterceptPP(PPObj, intercept)
```

Arguments

`PPObj`	An object that inherits "PP" class, typically a "PP" or "PPA" object.
`intercept`	Regression intercept, must be provided by users. It can be either a single value or a vector of values, see **Details**.

Details

Once a regression spline is transformed to a "PP" or "PPA" object, `AddInterceptPP` can be used to add intercept to the piecewise polynomial. This function spares users the need to manually update polynomial coefficients, but does not automatically extract intercept value(s) from a fitted regression model. Therefore, users are required to provide intercept value(s) themselves, which usually means that they need to examine carefully the returned coefficients of a model object, and correctly pick out intercept value(s). The extraction process depends on model class and model structure and thus

varies a lot. The **Examples** section below offers a good list of examples but is far from complete. Users are encouraged to contribute new examples by contacting the maintainer.

`AddInterceptPP` is clever about its input `PPObj` and `intercept`:

- "PP" + single intercept value = "PP";
- "PP" + vector of intercept values = "PPA", i.e., each intercept value is added to the "PP" object in turn;
- "PPA" + single intercept value = "PPA", i.e., a common intercept value is added to all replicates in the "PPA" object;
- "PPA" + vector of intercept values = "PPA", i.e., in one-to-one matching, an intercept is added to a replicate. It is required that there are as many values in `intercept` as replicates in `PPObj`.

Value

A "PP" or "PPA" object.

Note

In other context than regression, `AddInterceptPP` can be used, for example, to add a constant to a "PP" or "PPA" object.

Author(s)

Zheyuan Li <zheyuan.li@bath.edu>

Examples

```
## list of regression examples
## 1. a single spline, via 'stats::lm'
## 2. 4 replicates of a spline, via 'stats::lm'
## 3. a spline for each factor level, via 'stats::lm'
## 4. a fixed-effect spline + random intercept, via 'nlme::lme'
## 5. a fixed-effect spline + a random-effect spline, via 'nlme::lme'

require(spluti)

########################################
## 1. a single spline, via 'stats::lm' ##
########################################

## fit 'y1' as a spline of 'x'
model <- lm(y1 ~ bs(x, df = 5), data = wtoyreg)

## convert the fitted bs() term to a "PP" and plot it
## the fitted bs() term is away from data by an evident vertical shift
spl <- RegBSpl2PP(model, "bs(x, df = 5)")
plot(spl, ylim.hint = range(wtoyreg$y1))
with(wtoyreg, points(x, y1, col = 8))

## after including intercept, the fitted spline now follows data
```

```
intercept <- model$coefficients[1]
spl <- AddInterceptPP(spl, intercept)
plot(spl, ylim.hint = range(wtoyreg$y1))
with(wtoyreg, points(x, y1, col = 8))

################################################
## 2. 4 replicates of a spline, via 'stats::lm' ##
################################################

## fit 'y1', 'y2', 'y3' and 'y4' as a spline of 'x'
model <- lm(cbind(y1, y2, y3, y4) ~ bs(x, df = 5), data = wtoyreg)

## convert the fitted bs() term to a "PPA"
spl <- RegBSpl2PP(model, "bs(x, df = 5)")

## add model intercept (a vector of 4 values) to "PPA" (4 replicates of a spline)
## now the fitted spline follows data
intercept <- model$coefficients[1, ]
spl <- AddInterceptPP(spl, intercept)
plot(spl, ylim.hint = with(wtoyreg, range(y1, y2, y3, y4)))
with(wtoyreg, matpoints(x, cbind(y1, y2, y3, y4), pch = 1, col = 8))

#########################################################
## 3. a spline for each factor level, via 'stats::lm' ##
#########################################################

## fit a spline for each factor level
model <- lm(y ~ bs(x, df = 5) + id + bs(x, df = 5):id, data = ltoyreg)

## baseline-level spline
spl_base <- RegBSpl2PP(model, "bs(x, df = 5)")
intercept_base <- model$coefficients[1]
spl_base <- AddInterceptPP(spl_base, intercept_base)

## factor-level spline
spl_fctr_coef <- model$coefficients[10:24]
spl_fctr_coef <- matrix(spl_fctr_coef, ncol = 3)
spl_fctr <- BSpl2PP(ltoyreg$x, fun = "bs", df = 5, coef = spl_fctr_coef)
intercept_fctr <- model$coefficients[7:9]
spl_fctr <- AddInterceptPP(spl_fctr, intercept_fctr)

#####################################################################
## 4. a fixed-effect spline + random intercept, via 'nlme::lme' ##
#####################################################################

require(nlme)
model <- lme(y ~ ns(age, df = 7), random = ~ 1 | id, data = BMC)

## fixed-effect spline
fx_spl <- RegBSpl2PP(model, "ns(age, df = 7)")  ## a "PP"
fx_intercept <- model$coefficients$fixed[1]  ## single value
fx_spl <- AddInterceptPP(fx_spl, fx_intercept)  ## still a "PP"
```

```
## random intercept
re_intercept <- model$coefficients$random$id[, 1]  ## a vector of 20 values

## add two parts together
spl <- AddInterceptPP(fx_spl, re_intercept)  ## now a "PPA"

## plot fitted spline and data
plot(spl, ylim.hint = range(BMC$y))
with(BMC, points(age, y, col = 8))

#########################################################################
## 5. a fixed-effect spline + a random-effect spline, via 'nlme::lme' ##
#########################################################################

## fitting a linear mixed model
require(nlme)
model <- lme(y ~ ns(age, df = 7), random = ~ ns(age, df = 3) | id, data = BMC)

## fixed-effect spline
fx_spl <- RegBSpl2PP(model, "ns(age, df = 7)")  ## a "PP"
fx_intercept <- model$coefficients$fixed[1]  ## a single value
fx_spl <- AddInterceptPP(fx_spl, fx_intercept)  ## still a "PP"

## random-effect spline
re_spl_coef <- model$coefficients$random$id[, -1, drop = FALSE]
re_spl <- BSpl2PP(BMC$age, fun = "ns", df = 3, coef = t(re_spl_coef))  ## a "PPA"
re_intercept <- model$coefficients$random$id[, 1]  ## a vector of 20 values
re_spl <- AddInterceptPP(re_spl, re_intercept)  ## still a "PPA"

## add two splines together
spl <- AddPP(fx_spl, re_spl)  ## "PP" + "PPA" = "PPA"

## plot fitted spline and data
plot(spl, ylim.hint = range(BMC$y))
with(BMC, points(age, y, col = 8))
```

AddPP *Add two objects that inherit "PP" class*

Description

If $f(x)$ is a piecewise polynomial with knots set A and $g(x)$ is a piecewise polynomial with knots set B, their sum $h(x) = f(x) + g(x)$ is still a piecewise polynomial but has knots set $C = A \cup B$. AddPP can add two piecewise polynomial objects, with knots automatically relocated. The function is versatile, able to add two "PP" objects, a "PP" object with a "PPA" object, and two "PPA" objects (with the same number of replicates).

Usage

```
AddPP(a, b)
```

Arguments

a, b	An object that inherits "PP" class, typically a "PP" or "PPA" object.

Details

When adding a "PP" object and another, the result is still a "PP" object.

When adding a "PP" object and a "PPA" object, the piecewise polynomial in the "PP" object is added to each replicate of the "PPA" object, resulting in a "PPA" object.

When adding two "PPA" objects, replicates from two objects are added in one-two-one matching and the result is still a "PPA" object. It is required that the two objects have the same number of replicates.

Value

A "PP" or "PPA" object.

Note

If $f(x)$ is a spline of polynomial degree d_1 and $g(x)$ a spline of degree d_2, their sum $h(x)$ is not a spline unless $d_1 = d_2$. To see this, let's say $d_1 > d_2$, then $f^{(d_2)}(x)$ is continuous but $g^{(d_2)}(x)$ is not, hence $h^{(d_2)}(x) = f^{(d_2)}(x) + g^{(d_2)}(x)$ is not continuous. See **Examples**.

Author(s)

Zheyuan Li <zheyuan.li@bath.edu>

Examples

```
## list of examples:
## 1. "PP" + "PP", where a spline + a spline is still a spline
## 2. "PP" + "PP", where a spline + a spline is NOT a spline
## 3. "PP" + "PPA"
## 4. "PPA" + "PPA"

require(spluti)

####################
## 1. "PP" + "PP" ##
####################

## a random cubic B-spline with 5 degree of freedom
spl1 <- BSpl2PP(1:10, df = 5, intercept = TRUE, coef = rnorm(5))

## another random cubic B-spline with 8 degree of freedom
spl2 <- BSpl2PP(1:10, df = 8, intercept = TRUE, coef = rnorm(8))

## they have different knots
spl1$knots
spl2$knots
```

```
## but they can be added with knots relocated
spl <- AddPP(spl1, spl2)
spl$knots

## the result is still a spline because both splines have the same polynomial degree
par(mfrow = c(2, 2), mar = c(4, 4, 1, 1))
plot(spl)  ## piecewise cubic
plot(spl, deriv = 1)  ## piecewise quadratic
plot(spl, deriv = 2)  ## piecewise linear
plot(spl, deriv = 3)  ## piecewise constant

############################
## 2. another "PP" + "PP" ##
############################

## a random cubic B-spline with 5 degree of freedom
spl1 <- BSpl2PP(1:10, df = 5, intercept = TRUE, coef = rnorm(5))

## a random quadratic B-spline with 8 degree of freedom
spl2 <- BSpl2PP(1:10, df = 8, degree = 2, intercept = TRUE, coef = rnorm(8))

## they can be added
PP <- AddPP(spl1, spl2)

## but the resulting piecewise polynomial is not a spline
## because its 2nd derivative is not continuous
par(mfrow = c(2, 2), mar = c(4, 4, 1, 1))
plot(PP)  ## piecewise cubic
plot(PP, deriv = 1)  ## piecewise quadratic
plot(PP, deriv = 2)  ## not continuous!!
plot(PP, deriv = 3)  ## piecewise constant

######################
## 3. "PP" + "PPA" ##
######################

## a random cubic B-spline with 5 degree of freedom
PP <- BSpl2PP(1:10, df = 5, intercept = TRUE, coef = rnorm(5))

## 4 replicates of a random cubic B-spline with 8 degree of freedom
PPA <- BSpl2PP(1:10, df = 8, intercept = TRUE, coef = matrix(rnorm(8 * 4), nrow = 8))

## the sum is a "PPA" object, and the order in which objects are given does not matter
PPA1 <- AddPP(PP, PPA)
PPA2 <- AddPP(PPA, PP)
identical(PPA1, PPA2)  ## TRUE

######################
## 4. "PPA" + "PPA" ##
######################

## 4 replicates of a random cubic B-spline with 5 degree of freedom
PPA1 <- BSpl2PP(1:10, df = 5, intercept = TRUE, coef = matrix(rnorm(5 * 4), nrow = 5))
```

```
## 4 replicates of a random cubic B-spline with 8 degree of freedom
PPA2 <- BSpl2PP(1:10, df = 8, intercept = TRUE, coef = matrix(rnorm(8 * 4), nrow = 8))

## the sum is still a "PPA" object
AddPP(PPA1, PPA2)
```

BMC *Bone Mineral Content dataset for demonstration purpose*

Description

A small longitudinal dataset containing 20 subjects' Bone Mineral Content (BMC) measured at growing ages.

Usage

```
BMC
```

Format

A dataframe of 271 observations with three variables:

- age, the age of the subject when the measurement is taken;
- y, the measured BMC;
- id, a 20-level factor (with levels "1", "2", . . . , "20") for subjects.

Author(s)

Zheyuan Li <zheyuan.li@bath.edu>

Examples

```
require(spluti)

## colour for each subject
col <- 5 - as.integer(BMC$id) %% 5

## pch for each subject
pch <- ceiling(as.integer(BMC$id) / 5)

## differnet subject has different colour AND pch
with(BMC, plot(age, y, col = col, pch = pch))
```

`BSpl2PP` *Transform a B-spline or many of its replicates to a "PP" or "PPA" object*

Description

Construct B-spline basis, then with user-provided basis coefficients, transform the corresponding B-spline(s) to piecewise polynomial(s).

Usage

```
BSpl2PP(x, fun = "bs", df = NULL, degree = 3, intercept = FALSE,
        knots = NULL, Boundary.knots = range(x), coef)
```

Arguments

`fun`	Character name of the function used to construct B-spline basis. Currently only "bs" and "ns" from package **splines** are supported. However, it is planned that additional functions "pbs" and "pns" be added in **spluti** to provide basis construction of periodic B-spline and periodic natural cubic spline.
`x`	The predictor variable. Missing values are allowed but omitted. This argument is passed to `fun`.
`df`	Desired degree of freedom (number of B-spline basis) of the spline. This argument is passed to `fun`.
`degree`	Polynomial degree of the spline. This argument is passed to `fun`. Note that the degree and the order of a B-spline are not the same thing. The order is (degree + 1).
`intercept`	Default is FALSE, implying that the spline is pinned to 0 at its left boundary knot. If TRUE, an intercept is included in the spline so that the spline can move away from 0 at this boundary. This argument is passed to `fun`.
`knots`	Interior knots for the spline. This argument is passed to `fun`. If provided, they must lie within range of non-NA x-values. If not provided, they will be automatically placed at quantiles of non-NA x-values by `fun`.
`Boundary.knots`	Clamped boundary knots for the spline, default to the range of the non-NA x-values. If set to other values, they must lie beyond range of non-NA x-values. This argument is passed to `fun`.
`coef`	Coefficients for the constructed B-spline basis. If a vector, a single spline is produced as a "PP" object; if a matrix (where each column gives a replicate of basis coefficients), multiple replicates of a spline are produced as a "PPA" object.

Details

How degree of freedom is calculated in a B-spline:

If a B-spline has k interior knots, then counting in the two boundary knots there are $(k+2)$ knots and hence $(k+1)$ polynomial pieces. If the polynomial is of degree d, then each piece has $(d+1)$ coefficients, resulting in a total of $(k+1)(d+1)$ coefficients. However, the 0 to $(d-1)$-th derivatives of a spline is continuous at interior knots, thus polynomial coefficients are subject to kd constraints, leaving $(k+1)(d+1) - kd = (k+d+1)$ effective number of coefficients, known as the *degree of freedom*. It is also identical to the number of B-spline basis for this spline. Removing intercept from the spline further decreases degree of freedom by 1. In addition, restricting the B-spline to be periodic reduces degree of freedom by an extra 1. Generally, there is

$$df = (k+d+1) - (1 - \text{intercept}) - \text{periodic} = k + d + \text{intercept} - \text{periodic},$$

where intercept and periodic are TRUE/FALSE or 1/0.

Natural cubic spline has $d = 3$, and there are two extra natural boundary conditions, thus there is

$$df = k + 3 + \text{intercept} - \text{periodic} - 2 = k + 1 + \text{intercept} - \text{periodic},$$

where intercept and periodic are TRUE/FALSE or 1/0.

Two usual ways to specify B-spline construction:

To construct B-spline basis with `fun`, we can either specify `df` or `knots`. With either one set, the other can be implied. We rarely given both at the same time, to avoid inconsistency due to misspecification.

Note that when only specifying `df`, there is a minimum degree of freedom at $k = 0$. As a result, it must be satisfied that $df \geq d + \text{intercept} - \text{periodic}$ for B-spline and $df \geq 1 + \text{intercept} - \text{periodic}$ for natural cubic spline.

Difference between *BSpl2PP* and *RegBSpl2PP*:

See `RegBSpl2PP` for details.

Value

A "PP" or "PPA" object.

Note

`BSpl2PP` forwards to `fun` all arguments it needs for B-spline basis construction, but `BSpl2PP` does not allow users to specify x-values beyond the range of boundary knots, which is valid in using `fun` itself.

Author(s)

Zheyuan Li <zheyuan.li@bath.edu>

Examples

```
require(spluti)

######################
## 1. a "PP" object ##
######################

## construct a random natural cubic spline and represent it as a "PP" object
spl <- BSpl2PP(x = rnorm(10), fun = "ns", df = 7, intercept = TRUE, coef = runif(7))
spl  ## print the "PP" object

######################
## 2. a "PPA" object ##
######################

## construct 10 random natural cubic splines and represent them as a "PPA" object
## note that all splines are pinned at 0 at the left boundary, when intercept = FALSE
spls1 <- BSpl2PP(x = rnorm(10), fun = "ns", df = 7, intercept = FALSE,
                 coef = matrix(runif(70), 7))
spls1  ## print the "PPA" object
plot(spls1)  ## plot the "PPA" object

## by contrast when intercept = TRUE, all splines have a "free" left boundary value
spls2 <- BSpl2PP(x = rnorm(10), fun = "ns", df = 7, intercept = TRUE,
                 coef = matrix(runif(70), 7))
spls2  ## print the "PPA" object
plot(spls2)  ## plot the "PPA" object
```

`DerivPP` *Differentiate a "PP" or "PPA" object*

Description

Differentiate all polynomial pieces simultaneously in a "PP" or "PPA" object to obtain derivatives of the object.

Usage

```
DerivPP(PPObj, deriv = 0)
```

Arguments

`PPObj`	An object that inherits "PP" class, typically a "PP" or "PPA" object.
`deriv`	Number of derivatives to take. Not allowed to exceed the polynomial degree of `PPObj`, otherwise the differentiated piecewise polynomial becomes 0.

Value

A "PP" or "PPA" object.

Author(s)

Zheyuan Li <zheyuan.li@bath.edu>

Examples

```
require(spluti)

###############
## Example 1 ##
###############

## construct a random B-spline and represent it as a "PP" object
## by default, the polynomial degree is 3
spl <- BSpl2PP(x = rnorm(10), df = 7, intercept = TRUE, coef = runif(7))
spl

## take 1st derivative, resulting in piecewise quadratic polynomials
d1_spl <- DerivPP(spl, deriv = 1)

## take 2nd derivative, resulting in piecewise linear polynomials
d2_spl <- DerivPP(spl, deriv = 2)

## take 3rd derivative, resulting in piecewise constants
d3_spl <- DerivPP(spl, deriv = 3)

###############
## Example 2 ##
###############

## construct 10 random B-splines and represent them as a "PPA" object
## by default, the polynomial degree is 3
spls <- BSpl2PP(x = rnorm(10), df = 7, intercept = TRUE, coef = matrix(runif(70), 7))
spls

## take 1st derivative, resulting in an array of piecewise quadratic polynomials
d1_spls <- DerivPP(spls, deriv = 1)
d1_spls

## take 2nd derivative, resulting in an array of piecewise linear polynomials
d2_spls <- DerivPP(spls, deriv = 2)
d2_spls

## take 3rd derivative, resulting in an array of piecewise constants
d3_spls <- DerivPP(spls, deriv = 3)
d3_spls
```

ICSpl2PP *Construct and transform an interpolation cubic spline to a "PP" object*

Description

stats::spline and stats::splinefun can interpolate (x, y) data using various types of cubic splines, and evaluate the spline or its derivatives at new x-coordinate values. However, there is no built-in method to solve an inverse problem. ICSpl2PP restructures an interpolation spline as a "PP" object, so that an inverse problem can be solved using the "PP" method for solve.

Usage

```
ICSpl2PP(x, y, method)
```

Arguments

x, y	Vectors giving the coordinates of the points to be interpolated.
method	Methods for cubic spline interpolation, including "fmm", "natural", "periodic" and "hyman" as used by stats::spline and stats::splinefun. Note that "monoH.FC" method is not supported.

Details

ICSpl2PP is a wrapper of stats::splinefun. The latter routine readily constructs the interpolation spline as piecewise cubic polynomials but stores construction details like knots and piecewise polynomial coefficients in an environment. ICSpl2PP simply restructures those data as a "PP" object.

Value

A "PP" object.

Author(s)

Zheyuan Li <zheyuan.li@bath.edu>

Examples

```
require(spluti)

## perform interpolation
fmm <- ICSpl2PP(toyinterp$x, toyinterp$y1, "fmm")
natural <- ICSpl2PP(toyinterp$x, toyinterp$y1, "natural")
periodic <- ICSpl2PP(toyinterp$x, toyinterp$y2, "periodic")
hyman <- ICSpl2PP(toyinterp$x, toyinterp$y3, "hyman")

## generic function print()
fmm
natural
periodic
hyman

## export all polynomial equations as formatted strings
summary(fmm)
```

```
summary(natural)
summary(periodic)
summary(hyman)

## plot all splines
par(mfrow = c(2, 2), mar = c(4, 4, 1, 1))
plot(fmm, show.knots = TRUE); points(toyinterp$x, toyinterp$y1)
plot(natural, show.knots = TRUE); points(toyinterp$x, toyinterp$y1)
plot(periodic, show.knots = TRUE); points(toyinterp$x, toyinterp$y2)
plot(hyman, show.knots = TRUE); points(toyinterp$x, toyinterp$y3)

## plot 1st derivatives of all splines
par(mfrow = c(2, 2), mar = c(4, 4, 1, 1))
plot(fmm, deriv = 1)
plot(natural, deriv = 1)
plot(periodic, deriv = 1)
plot(hyman, deriv = 1)

## backsolve the spline given y = 2.85
xr1 <- solve(fmm, b = 2.85)
xr2 <- solve(natural, b = 2.85)
xr3 <- solve(periodic, b = 2.85)
xr4 <- solve(hyman, b = 2.85)
par(mfrow = c(2, 2), mar = c(4, 4, 1, 1))
plot(fmm); abline(h = 2.85, lty = 2); points(xr1, rep.int(2.85, length(xr1)))
plot(natural); abline(h = 2.85, lty = 2); points(xr2, rep.int(2.85, length(xr2)))
plot(periodic); abline(h = 2.85, lty = 2); points(xr3, rep.int(2.85, length(xr3)))
plot(hyman); abline(h = 2.85, lty = 2); points(xr4, rep.int(2.85, length(xr4)))

## find all extrema
## "hyman" excluded as it is monotonic spline
xs1 <- solve(fmm, deriv = 1)
xs2 <- solve(natural, deriv = 1)
xs3 <- solve(periodic, deriv = 1)
par(mfrow = c(2, 2), mar = c(4, 4, 1, 1))
ys1 <- predict(fmm, xs1)
ys2 <- predict(natural, xs2)
ys3 <- predict(periodic, xs3)
plot(fmm); points(xs1, ys1, pch = 19)
plot(natural); points(xs2, ys2, pch = 19)
plot(periodic); points(xs3, ys3, pch = 19)
```

`plot.PP` *"PP" and "PPA" methods for generic function plot()*

Description

Plot a "PP" or "PPA" object, or its derivatives.

Usage

```
## S3 method for class 'PP'
plot(x, spread = 3, deriv = 0, show.knots = FALSE, ylim.hint = NULL, ...)
```

Arguments

x	An object that inherits "PP" class, typically a "PP" or "PPA" object.
spread	A graphical parameter determining how dense the evaluation grid is set up for plotting. Note that this is a multiplier factor. For a spline of polynomial degree d, there will be `spread * (d + 1)` evenly spaced evaluation points between two adjacent knots.
deriv	Number of derivatives to take. Not allowed to exceed polynomial degree, otherwise the differentiated piecewise polynomial becomes 0.
show.knots	If `TRUE`, grey vertical dotted lines will be drawn at knots of the spline.
ylim.hint	A hint on the y-axis range of the plot. This does not enforce the y-axis range; rather, it suggests that y-axis of the plot should at least cover this range. The actual y-axis range will be jointly determined by the evaluated spline values used for plotting. The argument proves useful when users want to reserver enough space in the plotting region so that they can later draw extra data on the plot. See **Examples**.
...	Not used by this method. See **Note** below.

Value

The function invisibly returns data for plotting in a list of two variables:

- `x`, a vector of x-coordinate values where splines are evaluated;
- `y`, evaluated spline values. either a vector if a "PP" object is plotted, or a matrix (a column per replicate) if a "PPA" object is plotted.

Note

While the method contains `...` in its arguments, it is unused so do not expect to customize the graphical display of the plot, by passing in general graphical parameters like `xlab`, `ylab`, etc. However, users can take the invisibly returned `x`, `y` values by the method, and produce a fancy plot with for example **ggplot2** package.

Author(s)

Zheyuan Li <zheyuan.li@bath.edu>

Examples

```
require(spluti)

## fit a smoothing spline
sm <- smooth.spline(wtoyreg$x, wtoyreg$y3)
```

```
## coerce the "smooth.spline" object to a "PP" object
spl <- SmSpl2PP(sm)

## plot the fitted smoothing spline against data
par(mfrow = c(1, 2))
## some data are out of the plotting region hence not displayed
plot(spl)
points(wtoyreg$x, wtoyreg$y3, col = 8)
## reserve space so that all data can be displayed in the plotting region
plot(spl, ylim.hint = range(wtoyreg$y3))
points(wtoyreg$x, wtoyreg$y3, col = 8)
```

`Poly2PP` *Convert a polynomial or many of its replicates to a "PP" or "PPA" object*

Description

Sometimes we want to add a polynomial $p(x)$ defined on $(-\infty, \infty)$ to a spline. Since a spline is always handled as a piecewise polynomial in shifted form by **spluti**, the addition is straightforward once $p(x)$ is expressed in the same piecewise fashion. Taking coefficients of $p(x)$ and knots of the spline, `Poly2PP` can convert $p(x)$ to such required form. The resulting "PP" or "PPA" object is then ready for addition with another "PP" or "PPA" object using `AddPP`.

Usage

```
Poly2PP(pc, knots)
```

Arguments

`pc`	Polynomial coefficients in ascending order of power. It can either be a vector if there is only a single polynomial to transform, or a matrix if there are multiple replicates of a polynomial to transform. In the latter case, each column of the matrix should give coefficients of one replicate.
`knots`	Knots of the resulting piecewise polynomial.

Value

A "PP" or "PPA" object.

Note

In regression models, polynomial coefficients may be estimated by a polynomial term constructed from `stats::poly`. Note that this function by default constructs an orthogonal polynomial rather than an ordinary (or raw) polynomial, and the corresponding regression coefficients can not be interpreted as polynomial coefficients.

Author(s)

Zheyuan Li <zheyuan.li@bath.edu>

Examples

```
## list of examples:
## 1. express a single polynomial in piecewise fashion
## 2. express multiple replicates of a polynomial in piecewise fashion
## 3. a fixed-effect spline and + random linear polynomials
## 4. a fixed-effect spline and + random quadratic polynomials

require(spluti)

############################################################
## 1. express a single polynomial in piecewise fashion ##
############################################################

## a linear polynomial "-1 + x"
PP <- Poly2PP(pc = c(-1, 1), knots = (-4):4)

## print the object
PP

## display new piecewise polynomial coefficients
PP$coef

## by express the piecewise polynomial in unshift form
## we will see that all columns are identical to 'pc'
UnshiftPP(PP)$coef

###################################
## 2. convert a single polynomial ##
###################################

## 2 replicates of a linear polynomial: "-1 + x" and "2 - 3x"
PPA <- Poly2PP(pc = cbind(c(-1, 1), c(2, -3)), knots = (-4):4)

## print the object
PPA

## display new piecewise polynomial coefficients
PPA$coef

#################################################################
## 3. a fixed-effect spline and + random linear polynomials ##
#################################################################

require(nlme)
model <- lme(y ~ ns(age, df = 7), random = ~ age | id, data = BMC)

## express the fixed-effect spline as a "PP" object
fx_spl <- RegBSpl2PP(model, "ns(age, df = 7)")
```

```
fx_intercept <- model$coefficients$fixed[1]
fx_spl <- AddInterceptPP(fx_spl, fx_intercept)

## express random linear polynomials as a "PPA" object
re_coef <- model$coefficients$random$id
re_poly <- Poly2PP(t(re_coef), fx_spl$knots)

## add "PP" and "PPA" to get estimated spline for each subject
spl <- AddPP(fx_spl, re_poly)

## plot those splines and overlay observations
plot(spl, ylim.hint = range(BMC$y))
with(BMC, points(age, y, col = 8))

#####################################################################
## 4. a fixed-effect spline and + random quadratic polynomials ##
#####################################################################
```

PP *The "PP" and "PPA" classes*

Description

Define the "PP" and "PPA" classes, explain their stored contents, and how to extract a "PP" object from a "PPA" object.

Usage

```
PPAEntry(PPAObj, i)
```

Arguments

PPAObj	A "PPA" object.
i	Position index informing which (single) replicate to extract.

Details

One single spline:

Consider a spline $f(x)$ of degree d (or order $(d+1)$) and k knots $x_1, x_2, \ldots, x_k$. In its piecewise polynomial representation, the i-th polynomial piece $f_i(x)$ on interval $[x_i, x_{i+1}]$ is parametrized in a *shifted* form:

$$f_i(x) = a_{0,i} + a_{1,i}(x - x_i) + a_{2,i}(x - x_i)^2 + \ldots + a_{d,i}(x - x_i)^d.$$

Coefficents for all pieces can then be stored as a $(d+1) \times (k-1)$ matrix:

$$\begin{matrix} a_{0,1} & a_{0,1} & \ldots & a_{0,k-1} \\ a_{1,1} & a_{1,1} & \ldots & a_{1,k-1} \\ \vdots & \vdots & & \vdots \\ a_{d-1,1} & a_{d-1,1} & \ldots & a_{d-1,k-1} \end{matrix}$$

If function UnshiftPP is used to transform the "PP" object from shifted form to unshifted form:

$$f_i(x) = b_{0,i} + b_{1,i}x + b_{2,i}x^2 + \ldots + b_{d,i}x^d,$$

then the coefficient matrix will instead be:

$$\begin{matrix} b_{0,1} & b_{0,1} & \ldots & b_{0,k-1} \\ b_{1,1} & b_{1,1} & \ldots & b_{1,k-1} \\ \vdots & \vdots & & \vdots \\ b_{d-1,1} & b_{d-1,1} & \ldots & b_{d-1,k-1} \end{matrix}$$

Several splines:

To motivate this, consider a dataset with variables `x`, `y1`, `y2` and `y3`. In practice, `y1`, `y2`, `y3` and `y4` can be:

- different variables;
- repeated measurements of the same variable on the same subject;
- the same variable measured on different subjects;
- simulated data in bootstrapping.

Now a multi-response linear regression via:

```
lm(cbind(y1, y2, y3, y4) ~ bs(x, df = 10), data = dat)
```

will produce 4 spline curves with the same knots and degree but different coefficients. Here we call them 4 *replicates* of a spline. Since piecewise polynomial coefficient matrix for each replicate has the same size, they can be combined into a 3-dimensional array `A`, where `A[,,i]` gives the coefficient matrix associated with the i-th replicate.

Value

Piecewise polynomials can be stored in a list with 3 components:

`coef`	A $(d+1) \times (k-1)$ matrix for a single spline, or a $(d+1) \times (k-1) \times n$ 3-dimensional array for n replicates of a spline.
`knots`	A vector of knots.
`shift`	A TRUE/FALSE logical value indicating whether the piecewise polynomial is in shifted form.

If `coef` is a matrix, the list has a single "PP" class and is called a "PP" object.

If `coef` is a 3-dimensional array, the list has classes "PPA" (primary) and "PP" (secondary). Primary class identifies an object so it is called a "PPA" object, but it *inherits* the "PP" class.

If users want to extract a replicate from a "PPA" object for a "PP" object, they can use the auxiliary function `PPAEntry`. See **Examples**.

Note

Unless otherwise stated, all piecewise polynomials in **spluti** are in shifted form. Transformation routines transform splines to "PP" or "PPA" objects in shifted form; all methods and tools for piecewise polynomials take and return "PP" or "PPA" objects in shifted form. The only exception is `UnshiftPP`, which transforms "PP" or "PPA" objects from shifted to unshifted form. But **spluti** has minimal support for unshifted form. See UnshiftPP for more.

Author(s)

Zheyuan Li <zheyuan.li@bath.edu>

Examples

```
require(spluti)

## Here I simply use Poly2PP() to create example "PP" and "PPA" objects

######################
## 1. a "PP" object ##
######################

## express a random cubic polynomial in piecewise fashion
PP <- Poly2PP(pc = runif(4), knots = 1:7)

## view piecewise polynomial coefficients (a matrix)
PP$coef

## convert the object from shifted to unshifted form and note the coefficient change
PP_unshifted <- UnshiftPP(PP)
PP_unshifted$coef

######################
## 2. a "PPA" object ##
######################

## express two replicates of a random cubic polynomial in piecewise fashion
PPA <- Poly2PP(pc = matrix(runif(4 * 2), nrow = 4), knots = 1:7)

## view piecewise polynomial coefficients (now a 3-dimensional array)
PPA$coef

## extract the 1st replicate as a "PP" object
PP1 <- PPAEntry(PPA, 1)
PP1$coef  ## a matrix

## convert the object from shifted to unshifted form and note the coefficient change
PPA_unshifted <- UnshiftPP(PPA)
PPA_unshifted$coef

## extract the 2nd replicate as a "PP" object
PP2_unshifted <- PPAEntry(PPA_unshifted, 2)
PP2_unshifted$coef  ## a matrix
```

`predict.PP` *"PP" and "PPA" methods for generic function predict()*

Description

Evaluate a "PP" or "PPA" object, or its derivatives, at a vector of x-coordinate values.

Usage

```
## S3 method for class 'PP'
predict(object, newx, deriv = 0, ...)
```

Arguments

object	An object that inherits "PP" class, typically a "PP" or "PPA" object.
newx	A vector of x-coordinate values where the piecewise polynomial or its derivatives is evaluated. All values must be within the range of knots in object; out-of-boundary prediction is not allowed.
deriv	Number of derivatives to take. Not allowed to exceed polynomial degree, otherwise the differentiated piecewise polynomial becomes 0.
...	Not used by the method.

Value

Evaluated y-coordinate values, either a vector if a "PP" object is predicted, or a matrix (a column per replicate) if a "PPA" object is predicted.

Author(s)

Zheyuan Li <zheyuan.li@bath.edu>

---

print.PP *"PP" and "PPA" methods for generic function print()*

Description

Nicely print or display a "PP" or "PPA" object.

Usage

```
## S3 method for class 'PP'
print(x, ...)

## S3 method for class 'PPA'
print(x, ...)
```

Arguments

x	An object that inherits "PP" class, typically a "PP" or "PPA" object.
...	Not used by the method.

Value

These methods have no returned values.

Author(s)

Zheyuan Li <zheyuan.li@bath.edu>

`RegBSpl2PP` *Transform a B-spline or many of its replicates in a fitted regression model to a "PP" or "PPA" object*

Description

When a regression model contains a B-spline term constructed by `splines::bs` or `splines::ns`, `RegBSpl2PP` may be used to transform this spline to a piecewise polynomial. There are conditions for this to work; see **Details**.

Usage

```
RegBSpl2PP(RegModel, BSplTerm)

## S3 method for class 'lm'
RegBSpl2PP(RegModel, BSplTerm)

## S3 method for class 'mlm'
RegBSpl2PP(RegModel, BSplTerm)

## S3 method for class 'lme'
RegBSpl2PP(RegModel, BSplTerm)
```

Arguments

`RegModel`	A fitted regression model with one or more spline term specified by `splines::bs` or `splines:ns`.
`BSplTerm`	A character string giving the name of the spline term to be transformed. The name must be correctly provided, otherwise the function throws a "BSplTerm not found" error. However, the error message also lists names of all terms, making it easy for users to do copy-and-paste for a fix.

Details

Conditions for *RegBSpl2PP* to work successfully:

`RegBSpl2PP` is a convenient generic function for users working with fitted regression models, but it only works successfully in two conditions:

1. the model class can be recognized by `RegBSpl2PP`;
2. the basis construction parameters for the B-spline term requested by users can be found in the fitted model object.

When these are met, RegBSpl2PP can extract those construction parameters and automatically find basis coefficients from regression coefficients to proceed to the transformation.

Condition 1 may be relaxed as more methods are written for RegBSpl2PP, but there is a bottom line: the fitted model must be produced by a regression routine that has a formula interface where users can specify their model structure using formula(e). For example, stats::lm, stats::glm, nlme::lme have such formula interface hence "lm", "mlm" and "lme" methods have been provided. (There is no need for a "glm" method, as a "glm" object inherits "lm" class and the "lm" method suffices.) "lme4" methods corresponding to lme4::lmer and lme4::glmer is also planned. But there is no way to write a method for glmnet::glmnet for example.

Condition 2 should generally be satisfied if the regression routine has been properly written, as for *Safe Prediction* the predict call (containing all basis construction parameters) for splines::bs and splines::ns terms should be available in the fitted regression model. However, we have found that such a predict call does not exist in a "lme" object returned by nlme::lme, if a B-spline term is specified in the formula for the random effect.

What to do when conditions are unmet:

Consider using BSpl2PP instead. It is a more primitive way to transform B-spline(s) to piecewise polynomial(s). To use it in this settings, users need to

1. ensure that parameters, particularly x.
2. basis coefficients are correctly extracted from regresssion coefficients.

Value

A "PP" or "PPA" object.

Note

It is incorrect to think values of a spline term as fitted values of the regression model, even if this spline is the only term in the RHS of the model formula. See AddInterceptPP.

Author(s)

Zheyuan Li <zheyuan.li@bath.edu>

Examples

```
## list of regression examples
## 1. a single spline, via 'stats::lm'
## 2. 4 replicates of a spline, via 'stats::lm'
## 3. a spline for each factor level, via 'stats::lm'
## 4. a fixed-effect spline + random intercept, via 'nlme::lme'
## 5. a fixed-effect spline + a random-effect spline, via 'nlme::lme'

require(spluti)

########################################
## 1. a single spline, via 'stats::lm' ##
########################################

## fit 'y1' as a spline of 'x'
```

```
model <- lm(y1 ~ bs(x, df = 5), data = wtoyreg)

## convert the fitted bs() term to a "PP" and plot it
## the fitted bs() term is away from data by an evident vertical shift
spl <- RegBSpl2PP(model, "bs(x, df = 5)")
plot(spl, ylim.hint = range(wtoyreg$y1))
with(wtoyreg, points(x, y1, col = 8))

## after including intercept, the fitted spline now follows data
intercept <- model$coefficients[1]
spl <- AddInterceptPP(spl, intercept)
plot(spl, ylim.hint = range(wtoyreg$y1))
with(wtoyreg, points(x, y1, col = 8))

## get more about this example under ?AddInterceptPP

#################################################
## 2. 4 replicates of a spline, via 'stats::lm' ##
#################################################

## fit 'y1', 'y2', 'y3' and 'y4' as a spline of 'x'
model <- lm(cbind(y1, y2, y3, y4) ~ bs(x, df = 5), data = wtoyreg)

## convert the fitted bs() term to a "PPA"
spl <- RegBSpl2PP(model, "bs(x, df = 5)")

## add model intercept (a vector of 4 values) to "PPA" (4 replicates of a spline)
## now the fitted spline follows data
intercept <- model$coefficients[1, ]
spl <- AddInterceptPP(spl, intercept)
plot(spl, ylim.hint = with(wtoyreg, range(y1, y2, y3, y4)))
with(wtoyreg, matpoints(x, cbind(y1, y2, y3, y4), pch = 1, col = 8))

#########################################################
## 3. a spline for each factor level, via 'stats::lm' ##
#########################################################

## fit a spline for each factor level
model <- lm(y ~ bs(x, df = 5) + id + bs(x, df = 5):id, data = ltoyreg)

## baseline-level spline
spl_base <- RegBSpl2PP(model, "bs(x, df = 5)")
intercept_base <- model$coefficients[1]
spl_base <- AddInterceptPP(spl_base, intercept_base)

## factor-level spline
spl_fctr_coef <- model$coefficients[10:24]
spl_fctr_coef <- matrix(spl_fctr_coef, ncol = 3)
spl_fctr <- BSpl2PP(ltoyreg$x, fun = "bs", df = 5, coef = spl_fctr_coef)
intercept_fctr <- model$coefficients[7:9]
spl_fctr <- AddInterceptPP(spl_fctr, intercept_fctr)

#####################################################################
```

```
## 4. a fixed-effect spline + random intercept, via 'nlme::lme' ##
#################################################################

require(nlme)
model <- lme(y ~ ns(age, df = 7), random = ~ 1 | id, data = BMC)

## fixed-effect spline
fx_spl <- RegBSpl2PP(model, "ns(age, df = 7)")  ## a "PP"
fx_intercept <- model$coefficients$fixed[1]  ## single value
fx_spl <- AddInterceptPP(fx_spl, fx_intercept)  ## still a "PP"

## random intercept
re_intercept <- model$coefficients$random$id[, 1]  ## a vector of 20 values

## add two parts together
spl <- AddInterceptPP(fx_spl, re_intercept)  ## now a "PPA"

## plot fitted spline and data
plot(spl, ylim.hint = range(BMC$y))
with(BMC, points(age, y, col = 8))

#############################################################################
## 5. a fixed-effect spline + a random-effect spline, via 'nlme::lme' ##
#############################################################################

## fitting a linear mixed model
require(nlme)
model <- lme(y ~ ns(age, df = 7), random = ~ ns(age, df = 3) | id, data = BMC)

## fixed-effect spline
fx_spl <- RegBSpl2PP(model, "ns(age, df = 7)")  ## a "PP"
fx_intercept <- model$coefficients$fixed[1]  ## a single value
fx_spl <- AddInterceptPP(fx_spl, fx_intercept)  ## still a "PP"

## random-effect spline
re_spl_coef <- model$coefficients$random$id[, -1, drop = FALSE]
re_spl <- BSpl2PP(BMC$age, fun = "ns", df = 3, coef = t(re_spl_coef))  ## a "PPA"
re_intercept <- model$coefficients$random$id[, 1]  ## a vector of 20 values
re_spl <- AddInterceptPP(re_spl, re_intercept)  ## still a "PPA"

## add two splines together
spl <- AddPP(fx_spl, re_spl)  ## "PP" + "PPA" = "PPA"

## plot fitted spline and data
plot(spl, ylim.hint = range(BMC$y))
with(BMC, points(age, y, col = 8))
```

SmSpl2PP *Reparametrize a fitted smoothing spline to a "PP" object*

Description

`stats::smooth.spline` can smooth (x, y) data using a natural cubic spline, and evaluate the spline or its derivatives at new x-coordinate values. However, there is no built-in method to solve an inverse problem. `SmSpl2PP` transforms a smoothing spline to a "PP" object, so that an inverse problem can be solved using the "PP" method for `solve`.

Usage

```
SmSpl2PP(SmSpl)
```

Arguments

`SmSpl`	A smoothing spline, typically a fitted model returned by `stats::smooth.spline`.

Details

The function deals with a fitted natural cubic smoothing spline model created by `stats::smooth.spline`. It extracts knots from the fitted spline, evaluates the spline at knots, then calls `ICSpl2PP` to construct a natural cubic interpolation spline.

Value

A "PP" object.

Author(s)

Zheyuan Li <zheyuan.li@bath.edu>

Examples

```
require(spluti)

## fit a smoothing spline
sm <- smooth.spline(wtoyreg$x, wtoyreg$y1)

## coerce the "smooth.spline" object to a "PP" object
spl <- SmSpl2PP(sm)

## print the "PP"
spl

## plot the "PP"
plot(spl)

## find all the roots
xr <- solve(spl)
points(xr, rep.int(0, length(xr)))
abline(h = 0, lty = 2)

## find all stationary / saddle points
xs <- solve(spl, deriv = 1)
```

```
## predict the "PP" at stationary / saddle points
ys <- predict(spl, xs)
points(xs, ys, pch = 19)
```

`solve.PP` *"PP" and "PPA" methods for generic function solve()*

Description

Solve an inverse problem $f^{(m)}(x) = b$, where $f(x)$ is a "PP" or "PPA" object.

Usage

```
## S3 method for class 'PP'
solve(a, b = 0, deriv = 0, ...)

## S3 method for class 'PPA'
solve(a, b = 0, deriv = 0, ...)
```

Arguments

`a`	An object that inherits "PP" class, typically a "PP" or "PPA" object.
`b`	The b value in an inverse problem.
`deriv`	The m value, i.e., the number of derivatives to take in an inverse problem. Must be strictly smaller than the polynomial degree.
`...`	Not used by the method.

Value

x-coordinate values that solve the given inverse problem, either a vector if a "PP" is solved, or a list (an entry per replicate) of vectors if a "PPA" is solved. Note that an empty vector `numeric(0)` implies no solution.

Author(s)

Zheyuan Li <zheyuan.li@bath.edu>

`spluti` *An introduction to spluti*

Description

Piecewise polynomial definition of a spline:

A spline $f(x)$ of degree d (or order $(d+1)$) and k knots $x_1, x_2, \ldots, x_k$, is by definition a set of $(k-1)$ piecewise polynomial of degree d that are $(d-1)$ times continuously differentiable at the knots. The i-th polynomial piece $f_i(x)$ on interval $[x_i, x_{i+1}]$ is often parametrized in a *shifted* form:

$$f_i(x) = a_{0,i} + a_{1,i}(x - x_i) + a_{2,i}(x - x_i)^2 + \ldots + a_{d,i}(x - x_i)^d.$$

This form is appealing because polynomial coefficients are directly associated with derivatives at knots:

$$f^{(m)}(x_i) = f_i^{(m)}(x_i) = m! \times a_{mi}.$$

It also make it easy to integrate the spline.

Representation of a spline in applications:

Piecewise polynomial representation of a spline is conventionally used in interpolation problems, but not in regression problems and smoothing problems. For example, R routines `stats::spline` and `stats::splinefun` for interpolation cubic spline readily saves piecewise polynomial information in its returned environment, but routine `stats::smooth.spline` for fitting a smoothing spline uses B-spline representation for the resulting natural cubic spline. In addition, splines constructed by `splines::bs` and `splines::ns`, later passed to regression routines like `stats::lm`, `stats::glm`, `nlme::lme`, `lme4::lmer`, `lme4::glmer`, also represent splines with B-spline basis.

Why piecewise polynomial representation is useful:

Coefficients for B-spline basis are difficult to interpret. It might thus be helpful to reparametrize a fitted regression spline or smoothing spline into piecewise polynomial for appreciation of broder audience.

In fact, piecewise polynomial representation makes it easier to post-process a fitted spline. For example, evaluation of a spline and its derivatives is straightforward. Moreover, an inverse problem:

$$f^{(m)}(x) = b$$

with $0 \le m \le d-1$ can be easily solved, by simply finding real roots of the polynomial $f^{(m)}(x) - b$, piece by piece, which can be done by for example, R routine `base::polyroot`. Many important practical questions can be stated as such inverse problem, for example, finding extrema and extreme values of a spline is identical to the $m = 0$, $b = 0$ case.

What spluti offers:

Package **spluti** offers two sets of routines:

1. routines that reparametrize splines constructed in applications to piecewise polynomial;
2. routines that process piecewise polynomial objects.

The first set is best summarized by the table below, where links to relevant routines and abbreviations used in naming are listed.

PP	Piecewise Polynomial Object Class	PP
PPA	Piecewise Polynomial Array Object class	PPA
Poly	Polynomial defined on $(-\infty, \infty)$	Poly2PP
ICSpl	Interpolation Cubic Spline	ICSpl2PP
SmSpl	Smoothing Spline	SmSpl2PP
BSpl	B-Spline	BSpl2PP
RegBSpl	Regression B-Spline	RegBSpl2PP

The second set contains on one hand "PP" method (and "PPA" method only if a different handling from "PP" method is necessary) for some standard generics, and on the other hand some extra utility routines proved to be practically useful. See summary tables below.

print	print "PP" and "PPA" objects in a pretty way	print.PP, print.PPA
summary	format piecewise polynomial equations as strings	summary.PP, summary.PPA
plot	plot piecewise polynomial or its derivatives	plot.PP
predict	predict piecewise polynomial or its derivatives	predict.PP
solve	solve an inverse problem	solve.PP, solve.PPA

add regression intercept to a regression spline	AddInterceptPP
differentiate piecewise polynomial	DerivPP
add two piecewise polynomial	AddPP
extract one "PP" object from a "PPA" object	PPAEntry

Author(s)

Zheyuan Li <zheyuan.li@bath.edu>

---

summary.PP *"PP" and "PPA" methods for generic function summary()*

Description

Print polynomial equations as formatted strings for all polynomial pieces in a "PP" or "PPA" object.

Usage

```
## S3 method for class 'PP'
summary(object, nhead = NULL, ...)

## S3 method for class 'PPA'
summary(object, nhead = NULL, ...)
```

Arguments

object	An object that inherits "PP" class, typically a "PP" or "PPA" object.
nhead	Number of piecewise polynomial to export. Normally not used by users (hence left as NULL and later set as the number of piecewise polynomial). A small value of 6 is internally used by print.PP for a preview.
...	Not used by the method.

Value

Formatted polynomial equations, either a character vector if a "PP" object is summarized, or a list (an entry per replicate) of character vectors if a "PPA" object is summarized.

Note

The polynomial equations always use "x" as variable.

Author(s)

Zheyuan Li <zheyuan.li@bath.edu>

Examples

```
require(spluti)

## Here I simply use Poly2PP() to create example "PP" and "PPA" objects

##################
## "PP" object ##
##################

PP <- Poly2PP(pc = runif(4), knots = 1:7)

## view piecewise polynomial coefficients (a matrix)
PP$coef

## summarize the object
summary(PP)

###################
## "PPA" object ##
###################

## 2 replicates
PPA <- Poly2PP(pc = matrix(runif(4 * 2), nrow = 4), knots = 1:7)

## view piecewise polynomial coefficients (now a 3-dimensional array)
PPA$coef

## summarize the object
summary(PPA)
```

ToyInterp *A toy dataset for interpolation*

Description

A toy dataset for making reproducible examples on how to use **spluti** for interpolation splines.

Usage

```
toyinterp
```

Format

A data frame with 4 variables, where

- x contains 10 unevenly spaced data in ascending order of value;
- y1 contains 10 random numbers from N(3, 1) distribution;
- y2 modifies y1 by replacing its last value by its first value;
- y3 sorts y1 into ascending order of value.

(x, y1), (x, y2) and (x, y3) data are intended for general, periodic and monotonic interpolation splines, respectively.

Author(s)

Zheyuan Li <zheyuan.li@bath.edu>

ToyReg *Two toy datasets for regression*

Description

Two toy datasets for making reproducible examples on how to use **spluti** for regression splines.

Usage

```
wtoyreg
ltoyreg
```

Format

`wtoyreg` (wide-form toy dataset) is a data frame of 101 observations, with:

- 1 independent variable `x`, taking evenly spaced values on [0, 10];
- 4 response variables `y1` to `y4`, observed with i.i.d. Gaussian noise from 4 random replicates of a cubic B-spline on [0, 10] with 6 degree of freedom (including intercept), respectively, at above `x` values.

This dataset is useful for demonstrating single-response or multi-response linear regression model.

`ltoyreg` (long-form toy dataset) is adapted from `wtoyreg`, by independently and randomly selecting 50 observations from each of (`x`, `y1`) to (`x`, `y4`) pair, and stacking those subsamples to long format. It is a data frame of 200 observations, with three variables:

- `x`, concatenating subsampled `x` values for each replicate;
- `y`, concatenating subsampled `y1`, `y2`, `y3` and `y4` values;
- `id`, a 4-level factor variable (with levels "1", "2", "3", "4") indicating which replicate the (`x`, `y`) pair comes from.

This dataset is useful for demonstrating single-response linear regression model fitting a spline per factor level.

Note

There are not many factor levels in `ltoyreg` to fit a reasonable linear mixed model with a random effect per factor level. For demonstration on linear mixed models, consider the `BMC` longitudinal dataset.

Author(s)

Zheyuan Li <zheyuan.li@bath.edu>

Examples

```
require(spluti)

## scatter plot of 'wtoyreg'
with(wtoyreg, matplot(x, cbind(y1, y2, y3, y4), type = "p", pch = 1, ylab = "y"))

## scatter plot of 'ltoyreg'
with(ltoyreg, plot(x, y, pch = 1, col = as.integer(id)))
```

UnshiftPP *Change a "PP" or "PPA" object from shifted to unshifted form*

Description

By default, a polynomial piece in a "PP" or "PPA" object has shifted form that looks like $f_i(x) = a_{0,i} + a_{1,i}(x - x_i) + a_{2,i}(x - x_i)^2 + \ldots + a_{d,i}(x - x_i)^d$, where x_i is the i-th knot. `UnshiftPP` can do some extra calculations and express the polynomial piece in unshifted form as $f_i(x) = b_{0,i} + b_{1,i}x + b_{2,i}x^2 + \ldots + b_{d,i}x^d$.

Usage

```
UnshiftPP(PPObj)
```

Arguments

`PPObj`	An object that inherits "PP" class, typically a "PP" or "PPA" object.

Value

A "PP" or "PPA" object in unshifted form.

Note

Unshifted form is numerically less stable to `predict`, `plot` and `solve`, thus **spluti** discourages users from using it. When given a "PP" or "PPA" object in unshifted form, most routines will stop and report that such form is not supported, except for `print`, `summary` and the auxiliary routine `PPAEntry`. The idea is that unshifted form should only be used for display purpose, not for any serious computations.

Author(s)

Zheyuan Li <zheyuan.li@bath.edu>

Examples

```
require(spluti)

######################
## 1. a "PP" object ##
######################

## construct a random cubic B-spline and represent it as a "PP" object
spl <- BSpl2PP(x = rnorm(10), df = 7, intercept = TRUE, coef = runif(7))

## the object is in shifted form
spl$shift  ## TRUE

## print equations of the first 2 polynomial pieces
```

```
summary(spl, nhead = 2)

## now convert the object to unshifted form
spl_unshifted <- UnshiftPP(spl)
spl_unshifted$shift  ## FALSE

## print equations of the first 2 polynomial pieces
summary(spl_unshifted, nhead = 2)

########################
## 2. a "PPA" object ##
########################

## construct 2 random cubic B-splines and represent them as a "PPA" object
spls <- BSpl2PP(x = rnorm(10), df = 7, intercept = TRUE, coef = matrix(runif(14), 7))

## print equations of the first 2 polynomial pieces for each replicate
summary(spls, nhead = 2)

## now convert the object to unshifted form and print equations again
spls_unshifted <- UnshiftPP(spls)
summary(spls_unshifted, nhead = 2)
```

Index