Package: gps (via r-universe)

August 28, 2024

Version 1.2

Date 2023-05-21

Title General P-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)

Imports stats, splines, Matrix, methods, graphics, grDevices

Description General P-splines are non-uniform B-splines penalized by a general difference penalty, proposed by Li and Cao (2022) <arXiv:2201.06808>. Constructible on arbitrary knots, they extend the standard P-splines of Eilers and Marx (1996) <doi:10.1214/ss/1038425655>. They are also related to the O-splines of O'Sullivan (1986) <doi:10.1214/ss/1177013525> via a sandwich formula that links a general difference penalty to a derivative penalty. The package includes routines for setting up and handling difference and derivative penalties. It also fits P-splines and O-splines to (x, y) data (optionally weighted) for a grid of smoothing parameter values in the automatic search intervals of Li and Cao (2023) <doi:10.1007/s11222-022-10178-z>. It aims to facilitate other packages to implement P-splines or O-splines as a smoothing tool in their model estimation framework.

License GPL-3

NeedsCompilation yes

URL https://github.com/ZheyuanLi/gps

Repository https://zheyuanli.r-universe.dev

RemoteUrl https://github.com/zheyuanli/gps

RemoteRef HEAD

RemoteSha ed5ea12192d369927e8b4e0ae29dc58399daef19

Contents

`DemoBS` *Demonstrate the construction of ordinary B-splines*

Description

Demonstrate the construction of 4 ordinary cubic B-splines on 8 knots.

Usage

```
DemoBS(uniform = TRUE, clamped = FALSE)
```

Arguments

`uniform`	if TRUE, place uniform knots; if FALSE, place non-uniform knots.
`clamped`	if TRUE, place clamped boundary knots when `uniform = FALSE`. For aesthetic reason, only boundary knots on the left end are clamped. This parameter is ignored when `uniform = TRUE`.

Value

This function has no returned values.

Author(s)

Zheyuan Li <zheyuan.li@bath.edu>

Examples

```
require(gps)

## uniform B-splines
DemoBS(uniform = TRUE)

## non-uniform B-splines
DemoBS(uniform = FALSE, clamped = FALSE)

## non-uniform & clamped B-splines
DemoBS(uniform = FALSE, clamped = TRUE)
```

DemoKnots *Demonstrate ordinary cubic B-splines on three types of knots*

Description

Demonstrate ordinary cubic B-splines on three types of knots: (a) uniform knots; (b) non-uniform knots; (c) non-uniform knots with clamped boundary knots. The same interior knots are positioned in cases (b) and (c).

Usage

```
DemoKnots(aligned = TRUE)
```

Arguments

aligned	if TRUE, interior knots in cases (b) and (c) are aligned for a better display.

Value

This function has no returned values.

Author(s)

Zheyuan Li <zheyuan.li@bath.edu>

Examples

```
require(gps)

DemoKnots(aligned = TRUE)
```

DemoNull *Demonstrate the null space of P-splines*

Description

Cubic P-splines set up with non-uniform B-splines and a 2nd order standard or general difference penalty are fitted to observations simulated from $y = x$. Should the resulting standard or general P-splines have the correct null space, the limiting fit at $\lambda = +\infty$ will be a straight line regardless of knot locations. In this demo, non-uniform knots from different distributions (primarily Beta distributions with varying shape parameters) are attempted. Results show that standard P-splines have an incorrect and unpredictable limiting behavior that is sensitive to knot locations, whereas general P-splines have a correct and consistent limiting behavior.

Usage

```
DemoNull(n, k, gps = FALSE)
```

Arguments

n	number of simulated observations from $y = x$.
k	number of interior knots to place.
gps	if TRUE, fit general P-splines; if FALSE, fit standard P-splines.

Value

This function has no returned values.

Author(s)

Zheyuan Li <zheyuan.li@bath.edu>

Examples

```
require(gps)

## standard P-splines
DemoNull(n = 100, k = 10, gps = FALSE)

## general P-splines
DemoNull(n = 100, k = 10, gps = TRUE)
```

DemoPBS *Demonstrate the construction of periodic B-splines*

Description

Demonstrate the construction of 6 periodic cubic B-splines on 7 domain knots.

Usage

```
DemoPBS(uniform = TRUE)
```

Arguments

uniform	if TRUE, place uniform knots; if FALSE, place non-uniform knots.

Value

This function has no returned values.

Author(s)

Zheyuan Li <zheyuan.li@bath.edu>

Examples

```
require(gps)

## uniform periodic cubic B-splines
DemoPBS(uniform = TRUE)

## non-uniform periodic cubic B-splines
DemoPBS(uniform = FALSE)
```

DemoSpl *Demonstrate a polynomial spline and its B-spline representation*

Description

Demonstrate a cubic spline and its B-spline representation.

Usage

```
DemoSpl(uniform = TRUE)
```

Arguments

uniform	if TRUE, place uniform knots; if FALSE, place non-uniform knots.

Value

A list giving the domain knots, B-spline coefficients and piecewise polynomial coefficients of the illustrated cubic spline.

Author(s)

Zheyuan Li <zheyuan.li@bath.edu>

Examples

```
require(gps)

## a cubic spline with uniform knots
DemoSpl(uniform = TRUE)

## a cubic spline with non-uniform knots
DemoSpl(uniform = FALSE)
```

---

gps2GS *Penalized B-splines estimation with automatic grid search of their smoothing parameter*

Description

Fit penalized B-splines (including standard or general P-splines and O-splines) to `(x, y, w)` for a grid of smoothing parameter values in the automatic search intervals of Li and Cao (2023). The GCV score and effective degree of freedom of each fit are also returned.

Usage

```
gps2GS(x, y, w = NULL, xt, d = 4, m = 2, gps = TRUE, periodic = FALSE,
       ng = 20, scalePen = TRUE)

DemoRhoLim(fit, plot = TRUE)
```

Arguments

`x, y, w`	a vector of x-values, y-values and weights.
`xt`	full knot sequence for ordinary B-splines (`length(xt) >= 2 * d`).
`d`	B-spline order ($d \geq 2$).
`m`	penalty order ($1 \leq m \leq d-1$).
`gps`	if TRUE, use a difference penalty; if FALSE, use a derivative penalty.
`periodic`	if TRUE, periodic boundary conditions are applied to B-splines and their penalty, so that periodic P-splines are estimated.
`ng`	number of grid points in the grid search of ρ; can be set to 0 to set up the grid search only, without actual P-splines estimation.

scalePen	if TRUE, scale the penalty matrix $\boldsymbol{S}$ (as **mgcv** does).
fit	fitted P-splines returned by gps2GS.
plot	if TRUE, produce summary plots.

Details

We smooth y_i using $f(x_i) = \boldsymbol{B_i}\boldsymbol{\beta}$, where $\boldsymbol{B_i}$ is i-th row of the B-spline design matrix $\boldsymbol{B}$ and $\boldsymbol{\beta}$ is a vector of B-spline coefficients. These coefficients are estimated by minimizing:

$$\|\boldsymbol{y} - \boldsymbol{B\beta}\|^2 + \exp(\rho) \cdot \|\boldsymbol{D\beta}\|^2,$$

where the L_2 penalty $\|\boldsymbol{D\beta}\|^2$ is some wiggliness measure for $f(x)$ and $\rho \in (-\infty, +\infty)$ is a smoothing parameter.

Value

gps2GS returns a large list with the following components:

- eqn
- eigen
- rho.lim
- E
- pwls

Author(s)

Zheyuan Li <zheyuan.li@bath.edu>

References

Zheyuan Li and Jiguo Cao (2023). Automatic search intervals for the smoothing parameter in penalized splines, *Statistics and Computing*, doi:10.1007/s1122202210178z

Examples

```
require(gps)

x <- rnorm(100)
xt <- PlaceKnots(x, d = 4, k = 10)

## set ng = 0 to set up grid search only
## here the y-values does not matter; we simply use the x-values
setup <- gps2GS(x, x, xt = xt, d = 4, m = 2, ng = 0)

## compute exact eigenvalues
DemoResult <- DemoRhoLim(setup)

## simulate 100 (x, y) data from g(x) = sin(2 * pi * x) on [0, 1]
## x-values are not equidistant but at quantiles of Beta(2, 2)
## note that g(x) is a periodic function
```

```
x <- qbeta(seq.int(0, 1, length.out = 100), 2, 2)
gx <- sin(2 * pi * x)
y <- rnorm(length(x), gx, sd = 0.1)

## place quantile knots with clamped boundary knots
xt <- PlaceKnots(x, d = 4, k = 10)

## fit a general P-spline with different boundary constraints
ordinary <- gps2GS(x, y, xt = xt, d = 4, m = 2)
periodic <- gps2GS(x, y, xt = xt, d = 4, m = 2, periodic = TRUE)

## identify the optimal fit minimizing GCV score
opt.ordinary <- which.min(ordinary$pwls$gcv)
opt.periodic <- which.min(periodic$pwls$gcv)

## inspect grid search result
## column 1: ordinary cubic spline
## column 2: periodic cubic spline
op <- par(mfcol = c(2, 2), mar = c(2, 2, 1.5, 0.5))
## ordinary spline
with(ordinary$pwls, plot(rho, edf, ann = FALSE))
title("edf v.s. log(lambda)")
with(ordinary$pwls, plot(rho, gcv, ann = FALSE))
with(ordinary$pwls, points(rho[opt.ordinary], gcv[opt.ordinary], pch = 19))
title("GCV v.s. log(lambda)")
## periodic spline
with(periodic$pwls, plot(rho, edf, ann = FALSE))
title("edf v.s. log(lambda)")
with(periodic$pwls, plot(rho, gcv, ann = FALSE))
with(periodic$pwls, points(rho[opt.periodic], gcv[opt.periodic], pch = 19))
title("GCV v.s. log(lambda)")
par(op)

## inspect fitted splines
yhat.ordinary <- with(ordinary, eqn$B %*% pwls$beta)
yhat.periodic <- with(periodic, eqn$B %*% pwls$beta)
op <- par(mfrow = c(1, 2), mar = c(2, 2, 1.5, 0.5))
## ordinary spline
matplot(x, yhat.ordinary, type = "l", lty = 1, ann = FALSE)
title("ordinary")
## periodic spline
matplot(x, yhat.periodic, type = "l", lty = 1, ann = FALSE)
title("periodic")
par(op)

## pick and plot the optimal fit minimizing GCV score
best.ordinary <- yhat.ordinary[, opt.ordinary]
best.periodic <- yhat.periodic[, opt.periodic]
op <- par(mfrow = c(1, 2), mar = c(2, 2, 1.5, 0.5))
## ordinary spline
plot(x, y, ann = FALSE)
lines(x, gx, lwd = 2, col = 2)
lines(x, best.ordinary, lwd = 2)
```

```
title("ordinary")
## periodic spline
plot(x, y, ann = FALSE)
lines(x, gx, lwd = 2, col = 2)
lines(x, best.periodic, lwd = 2)
title("periodic")
par(op)
```

GramBS *Gram matrix of B-splines*

Description

Compute the Gram matrix $\boldsymbol{G}$, i.e., the matrix of inner products between B-splines $b_1(x), b_2(x), \ldots$. Precisely, its element is $G_{uv} = \int b_u(x) b_v(x) \mathrm{d}x$. Such matrix is useful for estimating functional linear models.

The Gram matrix of differentiated B-splines gives the derivative penalty matrix $\boldsymbol{S}$ for O-splines. Precisely, its element is $S_{uv} = \int b_u^{(m)}(x) b_v^{(m)}(x) \mathrm{d}x$. Such matrix is straightforward to compute using the results of SparseD; see Examples.

Usage

```
GramBS(xt, d)
```

Arguments

xt	full knot sequence for ordinary B-splines (length(xt) >= 2 * d).
d	B-spline order ($d \geq 2$).

Value

A sparse matrix of "dsCMatrix" class.

Author(s)

Zheyuan Li <zheyuan.li@bath.edu>

Examples

```
require(gps)
require(Matrix)

## 11 domain knots at equal quantiles of Beta(3, 3) distribution
xd <- qbeta(seq.int(0, 1, by = 0.1), 3, 3)
## full knots (with clamped boundary knots) for constructing cubic B-splines
xt <- c(0, 0, 0, xd, 1, 1, 1)
## compute Gram matrix of B-splines
G <- GramBS(xt, d = 4)
```

```
round(G, digits = 3)

## Gram matrix of differentiated B-splines, i.e., a derivative penalty matrix
## compute derivative penalty matrices of all orders (m = NULL in SparseD)
D <- SparseD(xt, d = 4, gps = FALSE)
S <- lapply(D, crossprod)
lapply(S, round, digits = 1)
```

MakeGrid *Make a grid of x-values between domain knots*

Description

Place equidistant grid points on each knot span to produce a grid of x-values between domain knots, suitable for evaluating B-splines.

Usage

```
MakeGrid(xd, n, rm.dup = FALSE)
```

Arguments

xd	domain knot sequence.
n	number of equidistant grid points on each knot span.
rm.dup	if FALSE, interior knots will appear twice on the grid; if TRUE, they will appear only once.

Details

Denote the domain knot sequence by $s_0, s_1, s_2, \ldots, s_k, s_{k+1}$, where $(s_j)_1^k$ are interior knots and $s_0 = a$, $s_{k+1} = b$ are domain endpoints. A knot span is the interval between two successive knots, i.e., $[s_j, s_{j+1}]$.

To make a suitable grid on $[a, b]$ for evaluating B-splines, we can place n equidistant grid points on each knot span, ending up with $n(k + 1)$ grid points in total. Interior knots will show up twice in this grid. To keep only one instance, set `rm.dup = TRUE`.

Value

A vector of grid points.

Author(s)

Zheyuan Li <zheyuan.li@bath.edu>

Examples

```
require(gps)

## 4 domain knots: two interior knots 0.5 and 1.5 in domain [0, 3]
xd <- c(0, 0.5, 1.5, 3)

## interior knots will appear twice
MakeGrid(xd, 5, rm.dup = FALSE)

## interior knots only appear once
MakeGrid(xd, 5, rm.dup = TRUE)
```

`penalty` *Wiggliness penalties for penalized B-splines*

Description

For penalized B-splines (including standard or general P-splines and O-splines), (1) construct matrix $\boldsymbol{D}$ in the wiggliness penalty $\|\boldsymbol{D\beta}\|^2$; (2) sample B-spline coefficients from their prior distribution $\mathrm{N}(\mathbf{0},\ (\boldsymbol{D'D})^-)$; (3) compute the Moore-Penrose generalized inverse matrix $(\boldsymbol{D'D})^-$.

Usage

```
SparseD(xt, d, m = NULL, gps = TRUE)

PriorCoef(n, D)

MPinv(D, only.diag = FALSE)
```

Arguments

`xt`	full knot sequence for ordinary B-splines (`length(xt) >= 2 * d`).
`d`	B-spline order ($d \geq 2$).
`m`	penalty order ($1 \leq m \leq d-1$). Can be a vector of multiple values for `SparseD`.
`gps`	if TRUE, return $\boldsymbol{D}_{\text{gps}}$; if FALSE, return $\boldsymbol{D}_{\text{os}}$.
`n`	number of samples to draw from the prior distribution.
`D`	matrix $\boldsymbol{D}_{\text{gps}}$ or $\boldsymbol{D}_{\text{os}}$.
`only.diag`	if TURE, only diagonal elements are computed.

Details

General Difference Penalty for General P-Splines:

A general P-spline is characterized by an order-m general difference matrix $\boldsymbol{D}_{\text{gps}}$, which can be computed by `SparseD(..., gps = TRUE)`. For interpretation, the differenced coefficients $\boldsymbol{D}_{\text{gps}}\boldsymbol{\beta}$ are in fact $f^{(m)}(x)$'s B-spline coefficients, so the penalty is their squared L_2 norm.

Derivative Penalty for O-Splines:

An O-spline is characterized by $\boldsymbol{D}_{\text{os}}$ such that $\|\boldsymbol{D}_{\text{os}}\boldsymbol{\beta}\|^2 = \int_a^b f^{(m)}(x)^2\mathrm{d}x$. Since $f^{(m)}(x)$ has B-spline coefficients $\boldsymbol{D}_{\text{gps}}\boldsymbol{\beta}$, the integral can be shown to be $\boldsymbol{\beta}'\boldsymbol{D}'_{\text{gps}}\bar{\boldsymbol{S}}\boldsymbol{D}_{\text{gps}}\boldsymbol{\beta}$, where $\bar{\boldsymbol{S}}$ is the Gram matrix of those B-splines representing $f^{(m)}(x)$. Following the Cholesky factorization $\bar{\boldsymbol{S}} = \boldsymbol{U}'\boldsymbol{U}$, the quadratic form becomes $\|\boldsymbol{U}\boldsymbol{D}_{\text{gps}}\boldsymbol{\beta}\|^2$, so that $\boldsymbol{D}_{\text{os}} = \boldsymbol{U}\boldsymbol{D}_{\text{gps}}$. This matrix can be computed by `SparseD(..., gps = FALSE)`, with $\bar{\boldsymbol{S}}$ and $\boldsymbol{D}_{\text{gps}}$ also returned in a "sandwich" attribute.

Penalty Matrix:

We can express the L_2 penalty $\|\boldsymbol{D}\boldsymbol{\beta}\|^2$ as quadratic form $\boldsymbol{\beta}'\boldsymbol{S}\boldsymbol{\beta}$, where $\boldsymbol{S} = \boldsymbol{D}'\boldsymbol{D}$ is called a penalty matrix. It is trivial to compute $\boldsymbol{S}$ (using function `crossprod`) once $\boldsymbol{D}$ is available, so we don't feel the need to provide a function for this. Note that the link between $\boldsymbol{D}_{\text{os}}$ and $\boldsymbol{D}_{\text{gps}}$ implies a sandwich formula $\boldsymbol{S}_{\text{os}} = \boldsymbol{D}'_{\text{gps}}\bar{\boldsymbol{S}}\boldsymbol{D}_{\text{gps}}$, wherease $\boldsymbol{S}_{\text{gps}} = \boldsymbol{D}'_{\text{gps}}\boldsymbol{D}_{\text{gps}}$.

The Bayesian View:

In the Bayesian view, the penalty $\boldsymbol{\beta}'\boldsymbol{S}\boldsymbol{\beta}$ is a Gaussian prior for B-spline coefficients $\boldsymbol{\beta}$. But it is an improper one because $\boldsymbol{S}$ has a null space where an unpenalized order-m polynomial lies. Let's decompose $\boldsymbol{\beta} = \boldsymbol{\xi} + \boldsymbol{\theta}$, where $\boldsymbol{\xi}$ (the projection of $\boldsymbol{\beta}$ on this null space) is the coefficients of this order-m polynomial, and $\boldsymbol{\theta}$ (orthogonal to $\boldsymbol{\xi}$) is the component that can be shrunk to zero by the penalty. As a result, $\boldsymbol{\xi} \propto \mathbf{1}$ is not proper, but $\boldsymbol{\theta} \sim \mathrm{N}(\mathbf{0},\ \boldsymbol{S}^-)$ is. Function `PriorCoef` samples this distribution, and the resulting B-spline coefficients can be used to create random spline curves. The algorithm behind `PriorCoef` bypasses the Moore-Penrose generalized inverse and is very efficient. We don't recommend forming this inverse matrix because it, being completely dense, is expensive to compute and store. But if we need it anyway, it can be computed using function `MPinv`.

Value

`SparseD` returns a list of sparse matrices (of "dgCMatrix" class), giving $\boldsymbol{D}_{\text{gps}}$ or $\boldsymbol{D}_{\text{os}}$ of order `m[1]`, `m[2]`, ..., `m[length(m)]`. In the latter case, $\bar{\boldsymbol{S}}$ (sparse matrices of "dsCMatrix" or "ddiMatrix" class) and $\boldsymbol{D}_{\text{gps}}$ for computing $\boldsymbol{D}_{\text{os}}$ are also returned in a "sandwich" attribute.

`PriorCoef` returns a list of two components:

- `coef` gives a vector of B-spline coefficients when `n = 1`, or a matrix of `n` columns when `n > 1`, where each column is an independent sample;
- `sigma` is a vector, giving the marginal standard deviation for each B-spline coefficient.

`MPinv` returns the dense Moore-Penrose generalized inverse matrix $(\boldsymbol{D}'\boldsymbol{D})^-$ if `only.diag = FALSE`, and the diagonal entries of this matrix if `only.diag = TRUE`.

Author(s)

Zheyuan Li <zheyuan.li@bath.edu>

References

Zheyuan Li and Jiguo Cao (2022). General P-splines for non-uniform splines, doi:10.48550/arXiv.2201.06808

Examples

```
require(Matrix)
require(gps)

## 11 domain knots at equal quantiles of Beta(3, 3) distribution
xd <- qbeta(seq.int(0, 1, by = 0.1), 3, 3)
## full knots (with clamped boundary knots) for constructing cubic B-splines
xt <- c(0, 0, 0, xd, 1, 1, 1)

## compute D matrices of order 1 to 3 for O-splines
D.os <- SparseD(xt, d = 4, gps = FALSE)
D1.os <- D.os[[1]]; D2.os <- D.os[[2]]; D3.os <- D.os[[3]]

## get D matrices of order 1 to 3 for general P-splines
## we can of course compute them with D.gps <- SparseD(xt, d = 4, gps = TRUE)
## but they are readily stored in the "sandwich" attribute of 'D.os'
D.gps <- attr(D.os, "sandwich")$D
D1.gps <- D.gps[[1]]; D2.gps <- D.gps[[2]]; D3.gps <- D.gps[[3]]

## we can compute the penalty matrix S = D'D
S.gps <- lapply(D.gps, crossprod)
S1.gps <- S.gps[[1]]; S2.gps <- S.gps[[2]]; S3.gps <- S.gps[[3]]
S.os <- lapply(D.os, crossprod)
S1.os <- S.os[[1]]; S2.os <- S.os[[2]]; S3.os <- S.os[[3]]

## if we want to verify the sandwich formula for O-splines
## extract 'Sbar' matrices stored in the "sandwich" attribute
## and compute the relative error between S and t(D) %*% Sbar %*% D
Sbar <- attr(D.os, "sandwich")$Sbar
Sbar1 <- Sbar[[1]]; Sbar2 <- Sbar[[2]]; Sbar3 <- Sbar[[3]]
range(S1.os - t(D1.gps) %*% Sbar1 %*% D1.gps) / max(abs(S1.os))
range(S2.os - t(D2.gps) %*% Sbar2 %*% D2.gps) / max(abs(S2.os))
range(S3.os - t(D3.gps) %*% Sbar3 %*% D3.gps) / max(abs(S3.os))

## sample B-spline coefficients from their prior distribution
b.gps <- PriorCoef(n = 5, D2.gps)$coef
b.os <- PriorCoef(n = 5, D2.os)$coef
op <- par(mfrow = c(1, 2), mar = c(2, 2, 1.5, 0.5), oma = c(0, 0, 1, 0))
## prior B-spline coefficients with a general difference penalty
matplot(b.gps, type = "l", lty = 1, ann = FALSE)
title("general difference penalty")
## prior B-spline coefficients with a derivative penalty
matplot(b.os, type = "l", lty = 1, ann = FALSE)
title("derivative penalty")
title("random B-spline coefficients from their prior", outer = TRUE)
par(op)

## plot the corresponding cubic splines with these B-spline coefficients
x <- MakeGrid(xd, n = 11)
B <- splines::splineDesign(xt, x, ord = 4, sparse = TRUE)
y.gps <- B %*% b.gps
y.os <- B %*% b.os
```

```
op <- par(mfrow = c(1, 2), mar = c(2, 2, 1.5, 0.5), oma = c(0, 0, 1, 0))
matplot(x, y.gps, type = "l", lty = 1, ann = FALSE)
title("general difference penalty")
matplot(x, y.os, type = "l", lty = 1, ann = FALSE)
title("derivative penalty")
title("random cubic splines with prior B-spline coefficients", outer = TRUE)
par(op)
```

`penutils` *Utility functions for working with wiggliness penalties*

Description

Evaluate $\|\boldsymbol{D}\boldsymbol{\beta}\|^2$ without using $\boldsymbol{D}$.

Usage

```
DiffCoef(b, xt, d, m)

btSb(b, xt, d, m)
```

Arguments

`b`	a vector of B-spline coefficients (`length(b) == length(xt) - d`).
`xt`	full knot sequence for ordinary B-splines (`length(xt) >= 2 * d`).
`d`	B-spline order ($d \geq 2$).
`m`	penalty order ($1 \leq m \leq d-1$).

Details

Implicit Evaluation of the Penalty:

Sometimes we want to evaluate the penalty $\|\boldsymbol{D}\boldsymbol{\beta}\|^2$ for some $\boldsymbol{\beta}$. The obvious way is to do the matrix-vector multiplication $\boldsymbol{D}\boldsymbol{\beta}$ then compute its L_2 norm, however, implicit evaluation without using $\boldsymbol{D}$ is possible. For general P-splines, we can calculate $\boldsymbol{D}_{\text{gps}}\boldsymbol{\beta}$ by taking order-m general differences between elements of $\boldsymbol{\beta}$, and function `DiffCoef` does this. For O-splines, the evaluation can be more refined. Denote domain knots by $s_0,\ s_1,\ s_2,\ \ldots,\ s_k,\ s_{k+1}$, where $(s_j)_1^k$ are interior knots and $s_0 = a$, $s_{k+1} = b$ are domain endpoints. The derivative penalty adds up local wiggliness measure on each interval: $\int_a^b f^{(m)}(x)^2\mathrm{d}x = \sum_{j=0}^{k}\int_{s_j}^{s_{j+1}} f^{(m)}(x)^2\mathrm{d}x$. Function `btSb` calculates each integral in the summation and returns those additive components in a vector.

Value

`DiffCoef` (for general P-splines only) returns $\boldsymbol{D}_{\text{gps}}\boldsymbol{\beta}$ as a vector.

`btSb` (for O-splines only) returns a vector with element $\int_{s_j}^{s_{j+1}} f^{(m)}(x)^2\mathrm{d}x$.

Examples

```
require(Matrix)
require(gps)

## 11 domain knots at equal quantiles of Beta(3, 3) distribution
xd <- qbeta(seq.int(0, 1, by = 0.1), 3, 3)
## full knots (with clamped boundary knots) for constructing cubic B-splines
xt <- c(0, 0, 0, xd, 1, 1, 1)

## compute 2nd order D matrix for O-splines
D.os <- SparseD(xt, d = 4, m = 2, gps = FALSE)
D2.os <- D.os$order.2

## get 2nd order D matrix for general P-splines
## we can of course compute it with D.gps <- SparseD(xt, d = 4, m = 2, gps = TRUE)
## but it is readily stored in the "sandwich" attribute of 'D.os'
D.gps <- attr(D.os, "sandwich")$D
D2.gps <- D.gps$order.2

## random B-spline coefficients
b <- rnorm(ncol(D2.gps))

## two ways to evaluate a difference penalty
diff.b1 <- DiffCoef(b, xt, d = 4, m = 2)  ## implicit
diff.b2 <- as.numeric(D2.gps %*% b)       ## explicit
range(diff.b1 - diff.b2) / max(abs(diff.b1))

## several ways to evaluate a derivative penalty
sum(btSb(b, xt, d = 4, m = 2))  ## recommended
sum(as.numeric(D2.os %*% b) ^ 2)
S2.os <- crossprod(D2.os); sum(b * as.numeric(S2.os %*% b))
```

`periodic` *Design matrix and general difference matrices for periodic B-splines*

Description

For order-d periodic B-splines, `pbsDesign` evaluates B-splines or their derivatives at given x-values, and `SparsePD` computes general difference matrices of order 1 to $d - 1$.

Usage

```
pbsDesign(x, xd, d, nDeriv = 0, sparse = FALSE, wrap = TRUE)

SparsePD(xd, d, wrap = TRUE)
```

Arguments

x	x-values where periodic B-splines are to be evaluated.
xd	domain knot sequence for periodic B-splines (length(xd) >= d + 1).
d	B-spline order ($d \geq 2$).
nDeriv	derivative order.
sparse	if TRUE, create a sparse design matrix of "dgCMatrix" class.
wrap	if TRUE, the knots wrapping strategy is used; if FALSE, the linear constraint strategy is used.

Details

These functions perform type-2 construction, by transforming design matrix and general difference matrices for ordinary B-splines to satisfy periodic boundary constraints (see Details). By contrast, pbsDesign and SparsePD in **gps** perform type-1 construction by basis wrapping.

A spline $f(x)$ on domain $[a, b]$ can be constructed to satisfy periodic boundary constraints, that is, $f^{(q)}(a) = f^{(q)}(b)$, q = 0, 1, ..., degree - 1. These are actually linear equality constraints

Unlike ordinary B-splines, period B-splines do not require explicit auxiliary boundary knots for their construction. The magic is that auxiliary boundary knots will be automatically positioned by periodic extension of interior knots.

Denote the domain knot sequence by $s_0, s_1, s_2, \ldots, s_k, s_{k+1}$, where $(s_j)_1^k$ are interior knots and $s_0 = a$, $s_{k+1} = b$ are domain endpoints. For order-d B-splines, we replicate the first $d - 1$ interior knots (after adding $b - a$) to the right of $[a, b]$ for an augmented set of $K = k + d + 1$ knots, which spawns $p = K - d = k + 1$ ordinary B-splines. It turns out that periodic B-splines can be obtained by wrapping segments of those ordinary B-splines that stretch beyond $[a, b]$ to the start of the domain (a demo is offered by DemoPBS).

Note that we must have at least $d - 1$ interior knots to do such periodic extension. This means that $d + 1$ domain knots are required at a minimum for construction of periodic B-splines.

Value

pbsDesign returns a design matrix with length(x) rows and length(xd) - 1 columns. SparsePD returns a list of sparse matrices (of "dgCMatrix" class), giving general difference matrices of order 1 to $d - 1$.

Author(s)

Zheyuan Li <zheyuan.li@bath.edu>

Examples

```
require(gps)

## 5 domain knots: three interior knots 0.5, 1.5 and 1.8 in domain [0, 3]
xd <- c(0, 0.5, 1.5, 1.8, 3)

## make a grid
```

```
x <- MakeGrid(xd, n = 10)

## construct periodic cubic B-splines
PB1 <- pbsDesign(x, xd, d = 4, wrap = TRUE)
PB2 <- pbsDesign(x, xd, d = 4, wrap = FALSE)

## construct general difference matrices of order 1 to 3
SparsePD(xd, d = 4, wrap = TRUE)
SparsePD(xd, d = 4, wrap = FALSE)
```

`PlaceKnots` *Automatically place knots according to data*

Description

Place knots for ordinary B-splines or periodic B-splines using automatic strategies.

Usage

```
PlaceKnots(x, d, k, domain = NULL, uniform = FALSE, periodic = FALSE)
```

Arguments

`x`	observed x-values.
`d`	B-spline order.
`k`	number of interior knots.
`domain`	a vector of two values giving domain interval $[a, b]$. Will use `min(x)` and `max(x)` if not specified.
`uniform`	if TRUE, place equidistant knots; if FALSE, place quantile knots with clamped boundary knots.
`periodic`	if TRUE, return the domain knot sequence that is sufficient for constructing periodic B-splines (see `pbsDesign`); if FALSE, return the full knot sequence that is required for constructing ordinary B-splines.

Value

A vector of $K = k + 2d$ knots for ordinary B-splines, or $k + 2$ knots for periodic B-splines.

Author(s)

Zheyuan Li <zheyuan.li@bath.edu>

Examples

```
require(gps)

x <- rnorm(50)

## uniform knots for uniform cubic B-splines
xt1 <- PlaceKnots(x, d = 4, k = 5, uniform = TRUE)
B1 <- splines::splineDesign(xt1, x, ord = 4)

## clamped quantile knots for clamped non-uniform cubic B-splines
xt2 <- PlaceKnots(x, d = 4, k = 5, uniform = FALSE)
B2 <- splines::splineDesign(xt2, x, ord = 4)

## uniform knots for uniform periodic cubic B-splines
xd1 <- PlaceKnots(x, d = 4, k = 5, uniform = TRUE, periodic = TRUE)
PB1 <- pbsDesign(x, xd1, d = 4)

## quantile knots for non-uniform periodic cubic B-splines
xd2 <- PlaceKnots(x, d = 4, k = 5, uniform = FALSE, periodic = TRUE)
PB2 <- pbsDesign(x, xd2, d = 4)
```

---

rspl *Simulate random cubic splines*

Description

Simulate random cubic splines.

Usage

```
rspl(x, domain = NULL, n = 1)
```

Arguments

x	x-values where simulated cubic splines are evaluated.
domain	a vector of two values giving domain interval $[a, b]$. Will use `min(x)` and `max(x)` if not specified.
n	number of replicates to simulate.

Value

A list of four components:

- y is a vector of random cubic spline values evaluated at x when n = 1, or a matrix of n columns when n > 1, where each column is an independent replicate of random cubic splines;
- b is a vector of random B-spline coefficients when n = 1, or a matrix of n columns when n > 1, where each column is an independent replicate of random B-spline coefficients;
- xt is the full knot sequence for B-splines;
- domain gives the domain of the simulated spline(s).

Author(s)

Zheyuan Li <zheyuan.li@bath.edu>

Examples

```
require(gps)

x <- seq.int(0, 1, 0.01)

## a random cubic spline
y <- rspl(x, n = 1)$y
op <- par(mar = c(2, 2, 1.5, 0.5))
plot(x, y, type = "l", ann = FALSE)
title("a random cubic spline")
par(op)

## 5 random cubic splines
Y <- rspl(x, n = 5)$y
op <- par(mar = c(2, 2, 1.5, 0.5))
matplot(x, Y, type = "l", lty = 1, ylab = "y")
title("5 random cubic splines")
par(op)
```

Index