Package: bspline (via r-universe)

July 3, 2024

Type Package

Title B-Spline Interpolation and Regression

Version 2.2.2

Author Serguei Sokol <sokol@insa-toulouse.fr>

Maintainer Serguei Sokol <sokol@insa-toulouse.fr>

Description Build and use B-splines for interpolation and regression. In case of regression, equality constraints as well as monotonicity and/or positivity of B-spline weights can be imposed. Moreover, knot positions (not only spline weights) can be part of optimized parameters too. For this end, 'bspline' is able to calculate Jacobian of basis vectors as function of knot positions. User is provided with functions calculating spline values at arbitrary points. These functions can be differentiated and integrated to obtain B-splines calculating derivatives/integrals at any point. B-splines of this package can simultaneously operate on a series of curves sharing the same set of knots. 'bspline' is written with concern about computing performance that's why the basis and Jacobian calculation is implemented in C++. The rest is implemented in R but without notable impact on computing speed.

URL https://github.com/MathsCell/bspline

BugReports https://github.com/MathsCell/bspline/issues

License GPL-2

Encoding UTF-8

Imports Rcpp (>= 1.0.7), nlsic (>= 1.0.2), arrApply

LinkingTo Rcpp, RcppArmadillo

RoxygenNote 7.3.1

Suggests RUnit

Copyright INRAE/INSA/CNRS

NeedsCompilation yes

Repository CRAN

Date/Publication 2024-07-02 15:00:02 UTC

Contents

`bcurve` *nD B-curve governed by (x,y,...) control points.*

Description

nD B-curve governed by (x,y,...) control points.

Usage

```
bcurve(xy, n = 3)
```

Arguments

`xy`	Real matrix of (x,y,...) coordinates, one control point per row.
`n`	Integer scalar, polynomial order of B-spline (3 by default)

Details

The curve will pass by the first and the last points in 'xy'. The tangents at the first and last points will coincide with the first and last segments of control points. Example of signature is inspired from this blog.

Value

Function of one argument calculating B-curve. The argument is supposed to be in [0, 1] interval.

Examples

```
  # simulate doctor's signature ;)
  set.seed(71);
  xy=matrix(rnorm(16), ncol=2)
  tp=seq(0,1,len=301)
  doc_signtr=bcurve(xy)
  plot(doc_signtr(tp), t="l", xaxt='n',  yaxt='n', ann=FALSE, frame.plot=FALSE,
      xlim=range(xy[,1]), ylim=range(xy[,2]))
  # see where control points are
  text(xy, labels=seq(nrow(xy)), col=rgb(0, 0, 0, 0.25))
  # join them by segments
  lines(bcurve(xy, n=1)(tp), col=rgb(0, 0, 1, 0.25))

  # randomly curved wire in 3D space
## Not run:
  if (requireNamespace("rgl", quietly=TRUE)) {
     xyz=matrix(rnorm(24),ncol=3)
     tp=seq(0,1,len=201)
     curv3d=bcurve(xyz)
     rgl::plot3d(curv3d(tp), t="l", decorate=FALSE)
  }

## End(Not run)
```

---

bsc *Basis matrix and knot Jacobian for B-spline of order 0 (step function) and higher*

Description

This function is analogous but not equivalent to `splines:bs()` and `splines2::bSpline()`. It is also several times faster.

Usage

```
bsc(x, xk, n = 3L, cjac = FALSE)
```

Arguments

`x`	Numeric vector, abscissa points
`xk`	Numeric vector, knots
`n`	Integer scalar, polynomial order (3 by default)
`cjac`	Logical scalar, if `TRUE` makes to calculate Jacobian of basis vectors as function of knot positions (FALSE by default)

Details

For n==0, step function is defined as constant on each interval `[xk[i]; xk[i+1][`, i.e. closed on the left and open on the right except for the last interval which is closed on the right too. The Jacobian for step function is considered 0 in every x point even if in points where x=xk, the derivative is not defined.
For n==1, Jacobian is discontinuous in such points so for these points we take the derivative from the right.

Value

Numeric matrix (for cjac=FALSE), each column correspond to a B-spline calculated on x; or List (for cjac=TRUE) with components

mat basis matrix of dimension `nx x nw`, where nx is the length of x and `nw=nk-n-1` is the number of basis vectors

jac array of dimension `nx x (n+2) x nw` where n+2 is the number of support knots for each basis vector

See Also

[splines::bs()], [splines2::bSpline()]

Examples

```
x=seq(0, 5, length.out=101)
# cubic basis matrix
n=3
m=bsc(x, xk=c(rep(0, n+1), 1:4, rep(5, n+1)), n=n)
matplot(x, m, t="l")
stopifnot(all.equal.numeric(c(m), c(splines::bs(x, knots = 1:4, degree = n, intercept = TRUE))))
```

`bsp` *Calculate B-spline values from their coefficients qw and knots xk*

Description

Calculate B-spline values from their coefficients qw and knots xk

Usage

```
bsp(x, xk, qw, n = 3L)
```

Arguments

`x`	Numeric vector, abscissa points at which B-splines should be calculated. They are supposed to be non decreasing.
`xk`	Numeric vector, knots of the B-splines. They are supposed to be non decreasing.
`qw`	Numeric vector or matrix, coefficients of B-splines. `NROW(qw)` must be equal to `length(xk)-n-1` where `n` is the next parameter
`n`	Integer scalar, polynomial order of B-splines, by default cubic splines are calculated.

Details

This function does nothing else than calculate a dot-product between a B-spline basis matrix calculated by `bsc()` and coefficients `qw`. If qw is a matrix, each column corresponds to a separate set of coefficients. For x values falling outside of xk range, the B-splines values are set to 0. To get a function calculating spline values at arbitrary points from `xk` and `qw`, cf. `par2bsp()`.

Value

Numeric matrix (column number depends on qw dimensions), B-spline values on x.

See Also

[bsc], [par2bsp]

`bspline` *bspline: build and use B-splines for interpolation and regression.*

Description

Build and use B-splines for interpolation and regression. In case of regression, equality constraints as well as monotonicity requirement can be imposed. Moreover, knot positions (not only spline coefficients) can be part of optimized parameters too. User is provided with functions calculating spline values at arbitrary points. This functions can be differentiated to obtain B-splines calculating derivatives at any point. B-splines of this package can simultaneously operate on a series of curves sharing the same set of knots. 'bspline' is written with concern about computing performance that's why the basis calculation is implemented in C++. The rest is implemented in R but without notable impact on computing speed.

bspline functions

`"bsc:"` basis matrix (implemented in C++)

`"bsp:"` values of B-spline from its coefficients

`"dbsp:"` derivative of B-spline

`"par2bsp:"` build B-spline function from parameters

`"bsppar:"` retrieve B-spline parameters from its function "smbsp:" build smoothing B-spline

"fitsmbsp:" build smoothing B-spline with optimized knot positions

"diffn:" finite differences

See Also

Useful links:

- https://github.com/MathsCell/bspline
- Report bugs at https://github.com/MathsCell/bspline/issues

---

bsppar *Retrieve parameters of B-splines*

Description

Retrieve parameters of B-splines

Usage

```
bsppar(f)
```

Arguments

f	Function, B-splines such that returned by par3bsp(), smbsp(), ...

Value

List having components: n - polynomial order, qw - coefficients, xk - knots

---

dbsp *Derivative of B-spline*

Description

Derivative of B-spline

Usage

```
dbsp(f, nderiv = 1L, same_xk = FALSE)
```

Arguments

f	Function, B-spline such as returned by `smbsp()` or `par2bsp()`
nderiv	Integer scalar >= 0, order of derivative to calculate (1 by default)
same_xk	Logical scalar, if TRUE, indicates to calculate derivative on the same knot grid as original function. In this case, coefficient number will be incremented by 2. Otherwise, extreme knots are removed on each side of the grid and coefficient number is maintained (FALSE by default).

Value

Function calculating requested derivative

Examples

```
x=seq(0., 1., length.out=11L)
y=sin(2*pi*x)
f=smbsp(x, y, nki=2L)
d_f=dbsp(f)
xf=seq(0., 1., length.out=101) # fine grid for plotting
plot(xf, d_f(xf)) # derivative estimated by B-splines
lines(xf, 2.*pi*cos(2*pi*xf), col="blue") # true derivative
xk=bsppar(d_f)$xk
points(xk, d_f(xk), pch="x", col="red") # knot positions
```

---

diffn — *Finite differences*

Description

Calculate dy/dx where x,y are first and the rest of columns in the entry matrix 'm'

Usage

```
diffn(m, ndiff = 1L)
```

Arguments

m	2- or more-column numeric matrix
ndiff	Integer scalar, order of finite difference (1 by default)

Value

Numeric matrix, first column is midpoints of x, the second and following are dy/dx dmat *Differentiation matrix*

Description

Calculate matrix for obtaining coefficients of first-derivative B-spline. They can be calculated as `dqw=Md %*% qw`. Here, dqw are coefficients of the first derivative, Md is the matrix returned by this function, and qw are the coefficients of differentiated B-spline.

Usage

```
dmat(nqw = NULL, xk = NULL, n = NULL, f = NULL, same_xk = FALSE)
```

Arguments

`nqw`	Integer scalar, row number of qw matrix (i.e. degree of freedom of a B-spline)
`xk`	Numeric vector, knot positions
`n`	Integer scalar, B-spline polynomial order
`f`	Function from which previous parameters can be retrieved. If both f and any of previous parameters are given then explicitly set parameters take precedence over those retrieved from f.
`same_xk`	Logical scalar, the same meaning as in `dbsp`

Value

Numeric matrix of size `nqw-1 x nqw`

ibsp *Indefinite integral of B-spline*

Description

Indefinite integral of B-spline

Usage

```
ibsp(f, const = 0, nint = 1L)
```

Arguments

`f`	Function, B-spline such as returned by `smbsp()` or `par2bsp()`
`const`	Numeric scalar or vector of length `ncol(qw)` where qw is weight matrix of f. Defines starting value of weights for indefinite integral (0 by default).
`nint`	Integer scalar >= 0, defines how many times to take integral (1 by default)

Details

If f is B-spline, then following identity is held: Dbsp(ibsp(f)) is identical to f. Generally, it does not work in the other sens: ibsp(Dbsp(f)) is not f but not very far. If we can get an appropriate constant C=f(min(x)) then we can assert that ibsp(Dbsp(f), const=C) is the same as f.

Value

Function calculating requested integral

`iknots` *Estimate internal knot positions equalizing jumps in n-th derivative*

Description

Normalized total variation of n-th finite differences is calculated for each column in `y` then averaged. These averaged values are fitted by a linear spline to find knot positions that equalize the jumps of n-th derivative.
NB. This function is used internally in `(fit)smbsp()` and a priori has no interest to be called directly by user.

Usage

```
iknots(x, y, nki = 1L, n = 3L)
```

Arguments

`x`	Numeric vector
`y`	Numeric vector or matrix
`nki`	Integer scalar, number of internal knots to estimate (1 by default)
`n`	Integer scalar, polynomial order of B-spline (3 by default)

Value

Numeric vector, estimated knot positions `ipk` *Intervals of points in knot intervals*

Description

Find first and last+1 indexes iip s.t. x[iip] belongs to interval starting at xk[iik]

Usage

```
ipk(x, xk)
```

Arguments

`x`	Numeric vector, abscissa points (must be non decreasing)
`xk`	Numeric vector, knots (must be non decreasing)

Value

Integer matrix of size `(2 x length(xk)-1)`. Indexes are 0-based

`jacw` *Knot Jacobian of B-spline with weights*

Description

Knot Jacobian of B-spline with weights

Usage

```
jacw(jac, qws)
```

Arguments

`jac`	Numeric array, such as returned by `bsc(..., cjac=TRUE)`
`qws`	Numeric matrix, each column is a set of weights forming a B-spline. If qws is a vector, it is coerced to 1-column matrix.

Value

Numeric array of size `nx x ncol(qw) x nk`, where `nx=dim(jac)[1]` and nk is the number of knots `dim(jac)[3]+n+1` (n being polynomial order).

par2bsp — *Convert parameters to B-spline function*

Description

Convert parameters to B-spline function

Usage

```
par2bsp(n, qw, xk, covqw = NULL, sdy = NULL, sdqw = NULL)
```

Arguments

`n`	Integer scalar, polynomial order of B-splines
`qw`	Numeric vector or matrix, coefficients of B-splines, one set per column in case of matrix
`xk`	Numeric vector, knots
`covqw`	Numeric Matrix, covariance matrix of qw (can be estimated in `smbsp`).
`sdy`	Numeric vector, SD of each y column (can be estimated in `smbsp`).
`sdqw`	Numeric Matrix, SD of qw thus having the same dimension as qw (can be estimated in `smbsp`).

Value

Function, calculating B-splines at arbitrary points and having interface `f(x, select)` where `x` is a vector of abscissa points. Parameter `select` is passed to `qw[, select, drop=FALSE]` and can be missing. This function will return a matrix of size `length(x)` x `ncol(qw)` if `select` is missing. Elsewhere, a number of column will depend on `select` parameter. Column names in the result matrix will be inherited from `qw`.

parr — *Polynomial formulation of B-spline*

Description

Polynomial formulation of B-spline

Usage

```
parr(xk, n = 3L)
```

Arguments

`xk`	Numeric vector, knots
`n`	Integer scalar, polynomial order (3 by default)

Value

Numeric 3D array, the first index runs through n+1 polynomial coefficients; the second – through n+1 supporting intervals; and the last one through nk-n-1 B-splines (here nk=length(xk)). Knot interval of length 0 will have corresponding coefficients set to 0.

`pbsc` *Polynomial B-spline Calculation of Basis Matrix*

Description

Polynomial B-spline Calculation of Basis Matrix

Usage

```
pbsc(x, xk, coeffs)
```

Arguments

`x`	Numeric,vector, abscissa points
`xk`	Numeric vector, knots
`coeffs`	Numeric 3D array, polynomial coefficients such as calculated by `parr`

Details

Polynomials are calculated recursively by Cox-de Boor formula. However, it is not applied to final values but to polynomial coefficients. Multiplication by a linear functions gives a raise of polynomial degree by 1.
Polynomial coefficients stored in the first dimension of `coeffs` are used as in the following formula `p[1]*x^n + p[1]*x^(n-1) + ... + p[n+1]`.
Resulting matrix is the same as returned by `bsc(x, xk, n=dim(coeffs)[1]-1)`

Value

Numeric matrix, basis vectors, one per column. Row number is `length(x)`.

See Also

`bsc`

Examples

```
n=3
x=seq(0, 5, length.out=101)
xk=c(rep(0, n+1), 1:4, rep(5, n+1))
# cubic polynomial coefficients
coeffs=parr(xk)
# basis matrix
```

```
m=pbsc(x, xk, coeffs)
matplot(x, m, t="l")
stopifnot(all.equal.numeric(c(m), c(bsc(x, xk))))
```

smbsp *Smoothing B-spline of order n >= 0*

Description

Optimize smoothing B-spline coefficients (smbsp) and knot positions (fitsmbsp) such that residual squared sum is minimized for all y columns.

Usage

```
smbsp(
  x,
  y,
  n = 3L,
  xki = NULL,
  nki = 1L,
  lieq = NULL,
  monotone = 0,
  positive = 0,
  mat = NULL,
  estSD = FALSE,
  tol = 1e-10
)

fitsmbsp(
  x,
  y,
  n = 3L,
  xki = NULL,
  nki = 1L,
  lieq = NULL,
  monotone = 0,
  positive = 0,
  control = list(),
  estSD = FALSE,
  tol = 1e-10
)
```

Arguments

x	Numeric vector, abscissa points
y	Numeric vector or matrix or data.frame, ordinate values to be smoothed (one set per column in case of matrix or data.frame)

n
: Integer scalar, polynomial order of B-splines (3 by default)

xki
: Numeric vector, strictly internal B-spline knots, i.e. lying strictly inside of `x` bounds. If NULL (by default), they are estimated with the help of `iknots()`. This vector is used as initial approximation during optimization process. Must be non decreasing if not NULL.

nki
: Integer scalar, internal knot number (1 by default). When nki==0, it corresponds to polynomial regression. If `xki` is not NULL, this parameter is ignored.

lieq
: List, equality constraints to respect by the smoothing spline, one list item per y column. By default (NULL), no constraint is imposed. Constraints are given as a 2-column matrix `(xe, ye)` where for each xe, an ye value is imposed. If a list item is NULL, no constraint is imposed on corresponding y column.

monotone
: Numeric scalar or vector, if `monotone > 0`, resulting B-spline weights must be increasing; if `monotone < 0`, B-spline weights must be decreasing; if `monotone == 0` (default), no constraint on monotonicity is imposed. If 'monotone' is a vector it must be of length `ncol(y)`, in which case each component indicates the constraint for corresponding column of y.

positive
: Numeric scalar, if `positive > 0`, resulting B-spline weights must be >= 0; if `positive < 0`, B-spline weights must be decreasing; if `positive == 0` (default), no constraint on positivity is imposed. If 'positive' is a vector it must be of length `ncol(y)`, in which case each component indicates the constraint for corresponding column of y.

mat
: Numeric matrix of basis vectors, if NULL it is recalculated by `bsc()`. If provided, it is the responsibility of the user to ensure that this matrix be adequate to xki vector.

estSD
: Logical scalar, if TRUE, indicates to calculate: SD of each y column, covariance matrix and SD of spline coefficients. All these values can be retrieved with bsppar() call (FALSE by default). These estimations are made under assumption that all y points have uncorrelated noise. Optional constraints are not taken into account of SD.

tol
: Numerical scalar, relative tolerance for small singular values that should be considered as 0 if `s[i] <= tol*s[1]`. This parameter is ignored if estSD=FALSE (1.e-10 by default).

control
: List, passed through to `nlsic()` call

Details

If constraints are set, we use `nlsic::lsie_ln()` to solve a least squares problem with equality constraints in least norm sens for each y column. Otherwise, `nlsic::ls_ln_svd()` is used for the whole y matrix. The solution of least squares problem is a vector of B-splines coefficients `qw`, one vector per `y` column. These vectors are used to define B-spline function which is returned as the result.

NB. When `nki >= length(x)-n-1` (be it from direct setting or calculated from `length(xki)`), it corresponds to spline interpolation, i.e. the resulting spline will pass exactly by (x,y) points (well, up to numerical precision).

Border and external knots are fixed, only strictly internal knots can move during optimization. The optimization process is constrained to respect a minimal distance between knots as well as to bound them to x range. This is done to avoid knots getting unsorted during iterations and/or going outside of a meaningful range.

Value

Function, smoothing B-splines respecting optional constraints (generated by `par2bsp()`).

See Also

`bsppar` for retrieving parameters of B-spline functions; `par2bsp` for generating B-spline function; `iknots` for estimation of knot positions

Examples

```
 x=seq(0, 1, length.out=11)
 y=sin(pi*x)+rnorm(x, sd=0.1)
 # constraint B-spline to be 0 at the interval ends
 fsm=smbsp(x, y, nki=1, lieq=list(rbind(c(0, 0), c(1, 0))))
 # check parameters of found B-splines
 bsppar(fsm)
 plot(x, y) # original "measurements"
 # fine grained x
 xfine=seq(0, 1, length.out=101)
 lines(xfine, fsm(xfine)) # fitted B-splines
 lines(xfine, sin(pi*xfine), col="blue") # original function
 # visualize knot positions
 xk=bsppar(fsm)$xk
 points(xk, fsm(xk), pch="x", col="red")
# fit broken line with linear B-splines
x1=seq(0, 1, length.out=11)
x2=seq(1, 3, length.out=21)
x3=seq(3, 4, length.out=11)
y1=x1+rnorm(x1, sd=0.1)
y2=-2+3*x2+rnorm(x2, sd=0.1)
y3=4+x3+rnorm(x3, sd=0.1)
x=c(x1, x2, x3)
y=c(y1, y2, y3)
plot(x, y)
f=fitsmbsp(x, y, n=1, nki=2)
lines(x, f(x))
```

Index