Package 'BsplineQuantReg'

October 22, 2026

Type Package

Title 'Constrained Quantile Regression with B-Splines'

Version 0.2.6

Date 2026-10-19

Description Quantile regression with B-splines under shape constraints. The initial version with cubic splines, augmented with splines of degree 1 to 4, is now able to handle all degrees (degree >= 6 without any constraint).
Constraints for degrees 3 (monotone), 4 (monotone and convex) and 5 (convex) use the Karlin-Studden SOCP characterization for the sign of the polynomial (Karlin and Studden, 1966; Papp and Alizadeh, 2014), while other constraints are added as linear problems. The method for monotone constraints of cubic splines is described in Abbes (2025) <doi:10.5281/zenodo.17427913>. For quartic splines, see other references given in the README. All B-spline and polynomial functions have been rewritten for consistency. This package provides a fully independent original B-spline library for conversion between PP-form and B-spline representation, evaluation, differentiation, callable and non-callable objects, human-readable PP forms, and basis visualization, all based on De Boor's theory. It also extends to multiple knots to capture singularities. This feature is robust in the package, including for constrained regression. This R implementation is intended for demonstration and prototyping. An equivalent Python package is available at <https://pypi.org/project/BsplineQuantRegpy/>.

License GPL-3

URL https://github.com/alexandreabbes/BsplineQuantReg

BugReports https://github.com/alexandreabbes/BsplineQuantReg/issues

Depends R (>= 4.5.0)

Imports CVXR, ECOSolveR, utils

Suggests clarabel, cobs, quantreg, knitr, rmarkdown, testthat

VignetteBuilder knitr

Encoding UTF-8

Config/roxygen2/version 8.1.0

NeedsCompilation no

Author Alexandre Abbes [aut, cre]

Maintainer Alexandre Abbes <alexandre.abbes@proton.me>

Repository CRAN

Date/Publication 2026-10-22 12:40:50 UTC

Contents

apply_karlin_quadratic . . . 3
Bsplinetopp . . . 3
Bspline_base . . . 4
Bspline_base_deriv . . . 6
Bspline_deriv . . . 6
bspline_to_deriv_coeffs_cubic . . . 7
bspline_to_deriv_coeffs_lin . . . 8
bspline_to_deriv_coeffs_quad . . . 8
bspline_to_deriv_coeffs_quart . . . 9
bs_direct . . . 10
change_polynomial_base_taylor . . . 10
evalpp . . . 11
get_parameters . . . 11
make_spline . . . 12
makpp . . . 12
polyadd . . . 13
polyderiv . . . 13
polymul . . . 14
poly_eval . . . 15
print.callable_pp . . . 15
print.callable_spline . . . 16
print.non_callable_pp . . . 16
print.non_callable_spline . . . 17
quantile_spline . . . 17
reduce_pol . . . 19
setup_solver . . . 19
show_poly . . . 20
show_pp . . . 21
SplineConstantQuant . . . 22
SplineConstQuantRegBs1 . . . 23
SplineConstQuantRegBs2 . . . 24
SplineConstQuantRegBs3 . . . 25
SplineConstQuantRegBs4 . . . 26
SplineCubicQuant . . . 27
SplineLinearQuant . . . 29
SplinePolynQuant . . . 30

SplineQuadraticQuant 31
SplineQuarticQuant 32
Spline_der_knot 33
spline_eval 33
test_karlin_simple 34
view_basis 35

Index **36**

`apply_karlin_quadratic` *Apply Karlin-Studden constraints for a quadratic polynomial*

Description

For a quadratic polynomial p(u) = a*u^2 + b*u + c on [0,1], this function applies the Karlin-Studden SOCP constraints to ensure p(u) >= 0 or p(u) <= 0 on [0,1].

Usage

```
apply_karlin_quadratic(p2, p1, p0, z0, sign = 1)
```

Arguments

`p2`	Coefficient of u^2
`p1`	Coefficient of u
`p0`	Constant term
`z0`	Auxiliary SOCP variable
`sign`	Sign of the constraint (+1 for >= 0, -1 for <= 0)

Value

List of 'CVXR' constraints

`Bsplinetopp` *Convert a B-spline to Piecewise Polynomial (PP) form*

Description

Transforms a B-spline object (callable or non-callable) into a piecewise polynomial representation. The resulting PP object contains the polynomial coefficients for each interval between knots.

Usage

```
Bsplinetopp(Bspline, Bsbasis = NULL, callable = FALSE, verbose = FALSE)
```

Arguments

Bspline	A B-spline object (list, non_callable_spline, or callable_spline) containing at least 'coeff', 'degree', and 'knot'.
Bsbasis	Optional pre-computed B-spline basis object from Bspline_base(). If NULL, the basis is computed automatically.
callable	Logical; if TRUE, returns a callable function for evaluation. If FALSE (default), returns a non_callable_pp object. If a callable spline is given as input, then by default is return a callable pp.
verbose	Logical; if TRUE, print progress messages.

Value

A PP object (piecewise polynomial) with class: - "callable_pp" if callable = TRUE - "non_callable_pp" if callable = FALSE The object can be evaluated with evalpp() or directly if callable.

See Also

makpp, evalpp, Bspline_base

Examples

```
## Not run:

# Create a B-spline
sn <- c(0,0,0,0,1,2,3,4,5,5,5,5)
basis <- Bspline_base(sn, degree = 3)
basis$coeff <- runif(basis$n_splines)

# Convert to PP (non-callable)
pp <- Bsplinetopp(basis, callable = FALSE)
y <- evalpp(pp, seq(0, 5, length.out = 100))

# Convert to PP (callable)
pp_call <- Bsplinetopp(basis, callable = TRUE)
y <- pp_call(seq(0, 5, length.out = 100))

## End(Not run)
```

Bspline_base *Build B-spline basis in piecewise polynomial form Computes local polynomial coefficients for each B-spline basis function on each interval. Polynomials are expressed in the canonical basis Uses De Boor's recursion formula.*

Description

Build B-spline basis in piecewise polynomial form Computes local polynomial coefficients for each B-spline basis function on each interval. Polynomials are expressed in the canonical basis Uses De Boor's recursion formula.

Usage

```
Bspline_base(sn, degree = 3, der = 0, verbose = FALSE)
```

Arguments

sn	Extended knot vector (including endpoint repetitions) This means if t0..tkn it the set of knot then sn should be given as a vector with "degree" times t_0 and t_kn at the beginning and the ends. its length is number of intervals+1+2*degree.
degree	'B-spline' degree (default = 3 for cubic)
der	Derivative order (0 = original basis)
verbose	boolean FALSE (default) or TRUE.

Value

A list containing:

base	Coefficients in the local bases, in the form of an 3-d array [j,nu,coeff], j : the number of the spline in the basis, nu: the number of the interval in the extended notation, coeff : the coefficients in decreasing order Decresing (matlab) convention on the local bases (t-s_nu)^l, l=3.2.1 base[j,,] is a matrix of piecewise polynomial function compatible with the pp-form.
base0	Coefficients in canonical basis (centered at 0) (1, t, t^2, t^3) centered at the interval origin.
ext_knot	Extended knot vector
knot	Effective knot partition including ends)
degree	Spline degree
n_splines	Number of basis functions
deriv_order	Applied derivative order

Examples

```
sn <- c(0,0,0,0,1,2,3,4,5,5,5,5)
basis <- Bspline_base(sn, degree=3)
x=(0:(5*100))/100
y=bs_direct(basis,x)
matplot(x,t(y))
#or simple:
#view_basis(basis)
```

Bspline_base_deriv *Differentiate a B-spline basis*

Description

Computes the basis of order der derivatives of a B-spline basis.

Usage

```
Bspline_base_deriv(Bsbasis, der = 1, verbose = FALSE)
```

Arguments

Bsbasis	Object returned by Bspline_base or an equivalent coherent list with enough parameters.
der	Derivative order
verbose	boolean FALSE (default) or TRUE.

Value

A list similar to Bspline_base for the derivative basis

Bspline_deriv *Compute derivative coefficients of a B-spline*

Description

Given a B-spline of given degree, coefficients, knots, compute the coefficients of the derivative of order 'der' in the B-spline basis of degree d-der. Uses the standard algorithm for differentiating bsplines

Usage

```
Bspline_deriv(bspline, der = 1, callable = NULL, verbose = FALSE)
```

Arguments

bspline	callable_spline or non_callable_spline class list or function, or list with variable names 'degree', 'knot' ,'coeff'.
der	Derivative order (default = 1)
callable	Boolean, if TRUE returns a callable Bspline. If not provided
verbose	Boolean gives output messages when TRUE return the same class as input : 'callable' or 'non_callable'

Value

Coefficients of the derivative B-spline (degree = degree - der)

`bspline_to_deriv_coeffs_cubic` *normalized first and second derivative coefficients on each interval. for a cubic B-spline basis*

Description

normalized first and second derivative coefficients on each interval. for a cubic B-spline basis

Usage

```
bspline_to_deriv_coeffs_cubic(
  tn = NULL,
  degree = 3,
  x_values = 0,
  Bsbasis = NULL,
  verbose = FALSE
)
```

Arguments

`tn`	Knot vector (effective partition, not extended. OPtionnal if Bsbasis is given)
`degree`	Spline degree (default = 3)
`x_values`	Evaluation points for design matrix (0 = no evaluation)
`Bsbasis`	Bspline basis structure if already computed. Optional if knots are given.
`verbose`	boolean FALSE (default) or TRUE.

Value

A list containing (kn: Nb intervals, N=kn+3: Nb basis functions):

`d0`	Design matrix (if x_values provided)
`d1`	First derivative coefficients [a3, a2, a1] for each interval: (kn x N x 3) array
`d2`	Second derivative values at knot: ((kn+1) x N) matrix
`d3`	Third derivative values at knots: (kn x N) matrix

`bspline_to_deriv_coeffs_lin` *Derivative coefficients for linear B-spline Computes normalized first derivative coefficients for linear B-splines. For linear splines, the derivative is constant on each interval.*

Description

Derivative coefficients for linear B-spline Computes normalized first derivative coefficients for linear B-splines. For linear splines, the derivative is constant on each interval.

Usage

```
bspline_to_deriv_coeffs_lin(tn, degree = 1, x_values = 0, verbose = FALSE)
```

Arguments

`tn`	Knot vector (effective partition)
`degree`	Spline degree (should be 1)
`x_values`	Evaluation points for design matrix (0 = no evaluation)
`verbose`	logical; if TRUE, print progress messages

Value

A list containing:

`d0`	Design matrix (if x_values provided)
`d1`	First derivative values (constant per interval)

`bspline_to_deriv_coeffs_quad` *quadratic B-spline derivative coefficients*

Description

Computes normalized first and second derivative coefficients for quadratic B-splines on each interval.

Usage

```
bspline_to_deriv_coeffs_quad(tn, degree = 2, x_values = 0, verbose = FALSE)
```

Arguments

tn	Knot vector (effective partition)
degree	Spline degree (should be 2)
x_values	Evaluation points for design matrix (0 = no evaluation)
verbose	logical; if TRUE, print progress messages

Value

A list containing:

d0	Design matrix (if x_values provided)
d1	First derivative coefficients [a1, a0] for each interval
d2	Second derivative values (constant per interval)

---

bspline_to_deriv_coeffs_quart

Computes normalized first, second, and third derivative coefficients for quartic B-splines on each interval.

Description

Computes normalized first, second, and third derivative coefficients for quartic B-splines on each interval.

Usage

```
bspline_to_deriv_coeffs_quart(knot, degree = 4, x_values = 0)
```

Arguments

knot	Knot vector (effective partition)
degree	Spline degree (should be 4)
x_values	Evaluation points for design matrix

Value

A list containing:

d0	Design matrix (if x_values provided)
d1	First derivative coefficients [a3, a2, a1, a0] for each interval
d2	Second derivative coefficients [a2, a1, a0] for each interval
d3	Third derivative values at knot (linear constraints)

bs_direct *Direct evaluation of a B-spline basis*

Description

Computes the values of all B-spline basis functions at given points.

Usage

```
bs_direct(Basis, x_values = NULL, verbose = FALSE)
```

Arguments

Basis	Object returned by Bspline_base
x_values	Vector of evaluation points
verbose	set to TRUE increases verbosity

Value

Matrix of basis function values (n_splines x length(x_values))

change_polynomial_base_taylor *Change polynomial basis (Taylor expansion)*

Description

Converts a polynomial expressed in the basis (t-a)^k to its representation in the basis (t-b)^k using Taylor's formula. P(t) = sum c_k (t-a)^k P(t) = sum c"_k (t-b)^k

Usage

```
change_polynomial_base_taylor(coeffs_a, a, b)
```

Arguments

coeffs_a	Coefficients in basis centered at a (decreasing powers)
a	Original expansion point
b	New expansion point

Value

Coefficients in basis centered at b

evalpp *Evaluates a piecewise polynomial (PP form) function at given points.*

Description

Evaluates a piecewise polynomial (PP form) function at given points.

Usage

```
evalpp(p, x_values)
```

Arguments

p	List with components ext_knot (ext_knot) and coeff
x_values	Vector of evaluation points

Value

Function values at the requested points

get_parameters *Get parameters from a callable spline or a callable pp object*

Description

Get parameters from a callable spline or a callable pp object

Usage

```
get_parameters(x)
```

Arguments

x	A callable spline object

Value

A list with degree, knots, coefficients, and 'Bspline' object

make_spline *Create a callable spline object*

Description

Transforms a list container 'Bspline' that may result from any of the regression functions associated to 'quantile_spline' into a callable function that is the spline function with given knots and coefficients on the corresponding B-spline basis. It can be evaluated at any point of the knot range transfering the call to the 'spline_eval()' function, while preserving access to parameters. Outside the knots range, it extrapolates the side part of the spline.

Usage

```
make_spline(Bspline, verbose = FALSE, callable = TRUE)
```

Arguments

Bspline	A list containing at least (knot, coefficients, degree) like the one returned by quantile_spline or one of the degree-specific functions. If it already contains a 'spline' element, returns the object unchanged.
verbose	Boolean. If TRUE : output the attributes of the 'Bspline'
callable	is a Boolean flag that allows to return a simple spline list with class 'non_callable_spline'.

Value

Either a callable function can be called as 'Bspline(x)', with class ('callable_spline'), and all parameters as attributes or the list of parameters with an additional 'non_callable_spline' classe.

makpp *Build a piecewise polynomial (PP) form*

Description

Creates a PP structure from polynomial coefficients and knots. If callable = TRUE, returns a function that evaluates the PP.

Usage

```
makpp(coeff, tn = NULL, callable = FALSE, verbose = FALSE)
```

Arguments

coeff	matrix (coefficents) or pp polynomial. Coefficient matrix (kn x (degree+1))
tn	vector (knots) of length kn+1. Needed if coefficient is not pp
callable	Boolean; if TRUE, returns a callable function
verbose	Boolean

Value

A PP object or a callable function

`polyadd` *Polynomial addition*

Description

Adds two polynomials represented by coefficients in decreasing power order.

Usage

```
polyadd(p1, p2, verbose = FALSE)
```

Arguments

`p1`	First polynomial (coefficient vector)
`p2`	Second polynomial (coefficient vector)
`verbose`	boolean FALSE (default) or TRUE.

Value

Coefficient vector of the sum

Examples

```
polyadd(c(1, 1), c(1, -1)) # returns c(2, 0)
```

`polyderiv` *Polynomial derivative Computes the derivative of order* `der` *of a polynomial.*

Description

Polynomial derivative Computes the derivative of order `der` of a polynomial.

Usage

```
polyderiv(p, der = 1)
```

Arguments

`p`	Coefficient vector (decreasing powers)
`der`	Derivative order (default = 1)

Value

Coefficients of the derivative polynomial

Examples

```
# P(x) = x^2 -> P'(x) = 2x
polyderiv(c(1, 0, 0), 1) # returns c(2, 0)
```

`polymul` *Polynomial multiplication*

Description

Multiplies two polynomials represented by coefficients in decreasing power order.

Usage

```
polymul(p1, p2, ord = 0, verbose = FALSE)
```

Arguments

`p1`	First polynomial (coefficient vector, decreasing powers)
`p2`	Second polynomial (coefficient vector, decreasing powers)
`ord`	Unused (compatibility parameter)
`verbose`	boolean FALSE (default) or TRUE.

Value

Coefficient vector of the product polynomial (decreasing powers)

Examples

```
# (1 + x) * (1 + x) = 1 + 2x + x^2
polymul(c(1, 1), c(1, 1)) # returns c(1, 2, 1)
```

poly_eval *Evaluate polynomial*

Description

Evaluates a polynomial at one or more points.

Usage

```
poly_eval(p, xvalues)
```

Arguments

p	Coefficient vector (decreasing powers)
xvalues	vector at which to evaluate the polynomial

Value

Vector of same length as xvalues : polynomial values at the points xvalues

Examples

```
# P(x) = 1 + x + x^2
poly_eval(c(1, 1, 1), c(0, 1, 2)) # returns c(1, 3, 7)
```

print.callable_pp *Print method for callable_pp objects*

Description

Print method for callable_pp objects

Usage

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

Arguments

x	A callable_pp object
...	Additional arguments

print.callable_spline *Print method for callable spline*

Description

Print method for callable spline

Usage

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

Arguments

x	A callable spline object
...	Additional arguments

print.non_callable_pp *Print method for non_callable_pp objects*

Description

Print method for non_callable_pp objects

Usage

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

Arguments

x	A non_callable_pp object
...	Additional arguments

print.non_callable_spline *Print method for non_callable_spline results*

Description

Print method for non_callable_spline results

Usage

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

Arguments

x	Result object from quantile_spline when callable=FALSE
...	Additional arguments

quantile_spline *Unified Quantile Regression with B-Splines of Any Degree (1 to 4)*

Description

This function provides a unified interface for quantile regression with B-splines of degrees 1 to 4, with shape constraints (monotonicity, convexity, third derivative).

Usage

```
quantile_spline(
  xtab,
  ytab,
  knot = NULL,
  tau = 0.5,
  degree = 3,
  monot = 0,
  convcons = 0,
  der3cons = 0,
  solver = "CLARABEL",
  weight = NULL,
  verbose = FALSE,
  callable = TRUE,
  type_reg = "quantile"
)
```

Arguments

xtab	Predictor vector (x)
ytab	Response vector (y)
knot	Knot vector or number of knots (quantiles are then used)
tau	Quantile (between 0 and 1)
degree	Spline degree: 1 (linear), 2 (quadratic), 3 (cubic), or 4 (quartic)
monot	Monotonicity constraint vector per interval: 1 = increasing, -1 = decreasing, 0 = unconstrained. If scalar, repeated.
convcons	Convexity constraint vector: For degree 1: not available (ignored) For degree 2: per interval (1 = convex, -1 = concave) For degree 3: per knot (1 = convex, -1 = concave) For degree 4: per interval (1 = convex, -1 = concave)
der3cons	Third derivative constraint: For degree 1: not available (third derivative = 0) For degree 2: not available (third derivative = 0) For degree 3: per interval (1 = positive, -1 = negative) For degree 4: per knot (1 = positive, -1 = negative)
solver	'CVXR' solver to use (default = 'CLARABEL')
weight	Observation weights (default = 1 for all)
verbose	logical; if TRUE, print progress messages
callable	render the final container object callable: 'y=Bspline(x)' or 'y=Bspline(x,Bvalues)' for evaluation at x
type_reg	'quantile' or 'mean_square' type of regression

Value

A list containing coefficients, degree, and knots

References

- Abbes, A. (2025). *Quantile regression with cubic polynomial splines under shape constraints with applications*. Zenodo. doi:10.5281/zenodo.16999784
- Karlin, S., & Studden, W. J. (1966). *Tchebycheff Systems: With Applications in Analysis and Statistics*. Interscience.

See Also

SplineLinearQuant, SplineQuadraticQuant, SplineCubicQuant, SplineQuarticQuant

Examples

```
# Generate data
x <- seq(0, 1, length.out = 100)
y <- 2*x + sin(6*pi*x)/2 + rnorm(100, 0, 0.05)
knot <- quantile(x, probs = seq(0, 1, length.out = 10))

# Linear spline (degree 1) with increasing constraint
fit_lin <- quantile_spline(x, y, knot, tau = 0.5, degree = 1, monot = 1)
```

```
# Quadratic spline (degree 2) with convexity constraint
fit_quad <- quantile_spline(x, y, knot, tau = 0.5, degree = 2, convcons = 1)

# Cubic spline (degree 3) with both constraints
fit_cubic <- quantile_spline(x, y, knot, tau = 0.5, degree = 3,
                             monot = 1, convcons = 1)

# Quartic spline (degree 4) with all constraints
fit_quart <- quantile_spline(x, y, knot, tau = 0.5, degree = 4,
                             monot = 1, convcons = 1, der3cons = 1)
```

---

reduce_pol *Reduce polynomial*

Description

Removes leading zeros from a polynomial coefficient vector.

Usage

```
reduce_pol(p, verbose = FALSE)
```

Arguments

p	Polynomial coefficient vector(coefficients are in decreasing order)
verbose	Boolean FALSE (default) or TRUE.

Value

Reduced vector (without leading zeros)

Examples

```
reduce_pol(c(0,0, 1, 1))
```

---

setup_solver *Vérifier et configurer le solveur CVXR*

Description

Vérifier et configurer le solveur CVXR

Usage

```
setup_solver(solver = "OSQP", install = FALSE)
```

Arguments

solver	Nom du solveur ('OSQP', 'ECOS', 'SCS', 'CLARABEL')
install	Si TRUE, tente d'installer le solveur manquant

Value

Logical indiquant si le solveur est disponible

---

show_poly *Displays the equation of a polynomial in a given basis*

Description

Displays the equation of a polynomial in a given basis

Usage

```
show_poly(obj, a = 0, b = 0, digits = 4)
```

Arguments

obj	Polynomial object (coeff vector), decreasing power
a	: the base in which the coefficients are given is (x-a)^i, default 0 for canonical
b	: the base for output is (x-b)^i, default b=0
digits	Nombre de chiffres significatifs à afficher

Value

the polynomial written on the base (x-b)^i

Examples

```
# Simple polynomial
p <- c(3, -2, 1)  # 3 - 2x + x^2
show_poly(p) # or
show_poly(p,a=0, b=0)
# base not zero
p <- c(3, -2, 1)  # 3 - 2(x-2) + (x-2)^2
show_poly(p, a=2, b=2)
```

show_pp *Display Piecewise Polynomial (PP) in a human readble form*

Description

Display a Piecewise Polynomial (PP) object as readable equations

Usage

```
show_pp(ppol, local = TRUE, digits = 4, verbose = FALSE)
```

Arguments

ppol	A PP object (callable_pp, non_callable_pp) or spline (callable_spline, non_callable_spline)
local	Logical. If TRUE (default), uses the local basis (x-a)^i. If FALSE, uses the canonical basis 1, x, x², ...
digits	Number of significant digits for display (default: 4)
verbose	Logical. If TRUE, displays additional information

Details

This function takes a PP object (callable or non-callable) or a spline and displays the polynomial equations on each interval. Polynomials can be displayed in the canonical basis (1, x, x², ...) or in the local basis (1, (x-t_k), (x-t_k)², ...) centered at each knot t_k.

Value

A matrix or vector containing the formatted equations. If a single interval, returns a character vector. If multiple intervals, returns a matrix with each row: [interval, equation].

See Also

show_poly, Bsplinetopp, makpp

Examples

```
## Not run:
# Example with a simple PP
knot <- c(0, 0.5, 1)
coeff <- matrix(c(1, 2, 0.5, 0, 1, -1), nrow = 2, ncol = 3, byrow = TRUE)
pp <- makpp(coeff, knot)
show_pp(pp, local = FALSE)
# Output:
# [0.000, 0.500] 0.5x^2 + 2x + 1
# [0.500, 1.000] -1x^2 + 1x + 0

# With local basis
show_pp(pp, local = TRUE)
```

```
# Output:
# [0.000, 0.500] 0.5(x-0)^2 + 2(x-0) + 1
# [0.500, 1.000] -1(x-0.5)^2 + 1(x-0.5) + 0

# With a spline
sn <- c(0,0,0,0,1,2,3,4,5,5,5,5)
basis <- Bspline_base(sn, degree = 3)
basis$coeff <- runif(basis$n_splines)
show_pp(basis, local = TRUE, verbose = TRUE)

## End(Not run)
```

`SplineConstantQuant` *' Performs quantile/mean regression using constant B-splines with monotonicity constraints. No solver is needed, since for the degree 0, the B-splines in the basis are independent piece-wise constants.*

Description

' Performs quantile/mean regression using constant B-splines with monotonicity constraints. No solver is needed, since for the degree 0, the B-splines in the basis are independent piece-wise constants.

Usage

```
SplineConstantQuant(
  xtab,
  ytab,
  knot = NULL,
  tau = 0.5,
  monot = 0,
  weight = NULL,
  verbose = FALSE,
  type_reg = "quantile"
)
```

Arguments

`xtab`	Predictor vector (x)
`ytab`	Response vector (y)
`knot`	Knot vector or number of knots
`tau`	Quantile (between 0 and 1)
`monot`	Monotonicity constraint vector: Monotonicity is expressed at knots 1 ‖ k>0 = increasing, -1 ‖k<0 = decreasing, 0 = unconstrained. kn-1 constraints are taken into account, kn=length(knot)-1 If monot is a single number, it is repaeated as uniform coonstraint. if length(monot)<kn-1 then it is completed with zeros.

weight	Observation weights (default = 1 for all)
verbose	logical; if TRUE, print progress messages
type_reg	'quantile' or 'mean_square' type of regression i.e form of the objective.

Value

A list containing coefficients, degree, and knots

`SplineConstQuantRegBs1` *Wrapper function for linear spline quantile regression*

Description

This is a convenience wrapper for SplineLinearQuant.

Usage

```
SplineConstQuantRegBs1(
  xtab,
  ytab,
  knot,
  tau,
  monot = 0,
  solver = "CLARABEL",
  weight = NULL,
  verbose = FALSE
)
```

Arguments

xtab	Predictor vector (x)
ytab	Response vector (y)
knot	Knot vector or number of knots
tau	Quantile (between 0 and 1)
monot	Monotonicity constraint vector per interval: 1 = increasing, -1 = decreasing, 0 = unconstrained
solver	CVXR solver to use (default = "OSQP")
weight	Observation weights (default = 1 for all)
verbose	logical; if TRUE, print progress messages

Value

Same as SplineLinearQuant

SplineConstQuantRegBs2 *Wrapper function for quadratic spline quantile regression Previous version notation style This is a convenience wrapper for SplineQuadraticQuant.*

Description

Wrapper function for quadratic spline quantile regression Previous version notation style This is a convenience wrapper for SplineQuadraticQuant.

Usage

```
SplineConstQuantRegBs2(
  xtab,
  ytab,
  knot,
  tau,
  monot = 0,
  convcons = 0,
  solver = "CLARABEL",
  weight = NULL,
  verbose = FALSE
)
```

Arguments

xtab	Predictor vector (x)
ytab	Response vector (y)
knot	Knot vector or number of knots
tau	Quantile (between 0 and 1)
monot	Monotonicity constraint vector per interval: 1 = increasing, -1 = decreasing, 0 = unconstrained. Although constraints are set at knots; we apply them on each interval, at both extremities. kn+1 knots, kn constraints taken into account.
convcons	Convexity constraint vector per interval: 1 = convex, -1 = concave, 0 = unconstrained. kn constraints are considered for kn+1 knots.
solver	'CVXR' solver to use (default = 'CLARABEL')
weight	Observation weights (default = 1 for all)
verbose	logical; if TRUE, print progress messages

Value

Same as SplineQuadraticQuant

SplineConstQuantRegBs3 *Wrapper function for Cubic spline quantile regression*

Description

This is a convenience wrapper for SplineCubicQuant with convention of previous versions SplineConstQuantRegBs3.

Usage

```
SplineConstQuantRegBs3(
  xtab,
  ytab,
  knot,
  tau,
  monot = 0,
  convcons = 0,
  der3cons = 0,
  solver = "CLARABEL",
  weight = NULL,
  verbose = FALSE
)
```

Arguments

xtab	Predictor vector (x)
ytab	Response vector (y)
knot	Knot vector or number of knot (quantiles are then used)
tau	Quantile (between 0 and 1)
monot	Monotonicity constraint vector per interval: 1 = increasing, -1 = decreasing, 0 = unconstrained. If scalar, repeated.
convcons	Convexity constraint vector per knot: 1 = convex, -1 = concave, 0 = unconstrained. If scalar, repeated.
der3cons	Constraint on the 3rd derivative (on esach intervall: -1: negative, 0: no constraint, 1: positive constraint
solver	'CVXR' solver to use (default = 'CLARABEL')
weight	Observation weights (default = 1 for all)
verbose	boolean FALSE (default) or TRUE.

Value

Same as SplineCubicQuant

`SplineConstQuantRegBs4` *Wrapper function for quartic spline quantile regression*

Description

This is a convenience wrapper for SplineQuarticQuant that follows the same naming convention as SplineConstQuantRegBs3.

Usage

```
SplineConstQuantRegBs4(
  xtab,
  ytab,
  knot,
  tau,
  monot = 0,
  convcons = 0,
  der3cons = 0,
  solver = "CLARABEL",
  weight = NULL,
  verbose = FALSE
)
```

Arguments

`xtab`	Predictor vector (x)
`ytab`	Response vector (y)
`knot`	Knot vector or number of knot
`tau`	Quantile (between 0 and 1)
`monot`	Monotonicity constraint vector per interval: 1 = increasing, -1 = decreasing, 0 = unconstrained
`convcons`	Convexity constraint vector per interval: 1 = convex, -1 = concave, 0 = unconstrained
`der3cons`	Third derivative constraint vector at knot: 1 = positive third derivative, -1 = negative, 0 = unconstrained
`solver`	'CVXR' solver to use (default = "OSQP")
`weight`	Observation weights (default = 1 for all)
`verbose`	Logical; if TRUE, print progress messages

Value

Same as SplineQuarticQuant

SplineCubicQuant *Constrained quantile regression with cubic splines Performs quantile regression using cubic B-splines, with optional monotonicity constraints (via Karlin-Studden) and convexity constraints.*

Description

Constrained quantile regression with cubic splines Performs quantile regression using cubic B-splines, with optional monotonicity constraints (via Karlin-Studden) and convexity constraints.

Usage

```
SplineCubicQuant(
  xtab,
  ytab,
  knot = NULL,
  tau = 0.5,
  monot = 0,
  convcons = 0,
  der3cons = 0,
  solver = "CLARABEL",
  weight = NULL,
  verbose = FALSE,
  type_reg = "quantile"
)
```

Arguments

xtab	Predictor vector (x)
ytab	Response vector (y)
knot	Knot vector or number of knot (quantiles are then used)
tau	Quantile (between 0 and 1)
monot	Monotonicity constraint vector per interval: 1 = increasing, -1 = decreasing, 0 = unconstrained. If scalar, repeated.
convcons	Convexity constraint vector per knot: 1 = convex, -1 = concave, 0 = unconstrained. If scalar, repeated.
der3cons	Constraint on the 3rd derivative (on esach intervall: -1: negative, 0: no constraint, 1: positive constraint
solver	'CVXR' solver to use (default = 'CLARABEL')
weight	Observation weights (default = 1 for all)
verbose	boolean FALSE (default) or TRUE.
type_reg	'quantile' or 'mean_square' type of regression

Value

A list containing:

`coefficients`	B-spline coefficients (including y mean)
`degree`	Spline degree (always 3)
`ext_knot`	ext_knot vector used
`int_knot`	knot vector

References

- Abbes, A. (2025). *Quantile regression with cubic polynomial splines under shape constraints with applications* . Zenodo. doi:10.5281/zenodo.17427913
- de Boor, C. (1978). *A Practical Guide to Splines*. Springer-Verlag. doi:10.1007/97814612-63333
- Karlin, S., & Studden, W. J. (1966). *Tchebycheff Systems: With Applications in Analysis and Statistics*. Interscience.
- Koenker, R. G. (1978). Regression Quantiles. *Econometrica*, 46(1), 33-50. doi:10.2307/1913643
- Koenker, R. (2025). quantreg: Quantile Regression. R package version 5.99. https://CRAN.R-project.org/package=quantreg
- Ng, P., & Maechler, M. (2024). cobs: Constrained B-Splines. R package version 1.3-8. https://CRAN.R-project.org/package=cobs

See Also

Related 'R' packages:

- `quantreg` - Quantile regression with linear programming
- `cobs` - Constrained B-sines (linear and quadratic only)

Other implementations:

- MATLAB/Python versions: https://github.com/alexandreabbes/Constrained-Quantile-Regression-with-c
- Python package : https://pypi.org/project/BsplineQuantRegpy/

Examples

```
#optional set.seed(42)
x <- seq(0, 1, length=100)
y <- 2*x + sin(6*pi*x)/2 + rnorm(100, 0, 0.05)
knot <- as.numeric(quantile(x, probs=seq(0,1,length.out=10)))

# Median quantile regression without constraints
fit <- SplineConstQuantRegBs3(x, y, knot, tau=0.5)

# With increasing monotonicity constraint
fit_monot <- SplineConstQuantRegBs3(x, y, knot, tau=0.5, monot=1)
```

```
# With convexity constraint
fit_convex <- SplineConstQuantRegBs3(x, y, knot, tau=0.5, convcons=1)
```

SplineLinearQuant *Quantile regression with linear splines and monotonicity constraints*

Description

Performs quantile regression using linear B-splines with monotonicity constraints. For linear splines, the derivative is constant on each interval, so monotonicity is a simple linear constraint on the derivative.

Usage

```
SplineLinearQuant(
  xtab,
  ytab,
  knot = NULL,
  tau = 0.5,
  monot = 0,
  solver = "CLARABEL",
  weight = NULL,
  verbose = FALSE,
  type_reg = "quantile"
)
```

Arguments

xtab	Predictor vector (x)
ytab	Response vector (y)
knot	Knot vector or number of knots
tau	Quantile (between 0 and 1)
monot	Monotonicity constraint vector per interval: 1 = increasing, -1 = decreasing, 0 = unconstrained
solver	CVXR solver to use (default = "OSQP")
weight	Observation weights (default = 1 for all)
verbose	logical; if TRUE, print progress messages
type_reg	'quantile' or 'mean_square' type of regression i.e form of the objective.

Value

A list containing coefficients, degree, and knots

`SplinePolynQuant` *Quantile regression with general polynomial splines and shape constraints*

Description

Performs quantile regression using quartic (degree 4) B-splines with monotonicity (Karlin-Studden constraints on the cubic derivative), convexity (Karlin-Studden constraints on the quadratic second derivative), and third derivative constraints (linear at knot).

Usage

```
SplinePolynQuant(
  xtab,
  ytab,
  knot = NULL,
  convcons = 0,
  der3cons = 0,
  tau = 0.5,
  degree = 5,
  solver = "CLARABEL",
  weight = NULL,
  verbose = FALSE,
  type_reg = "quantile"
)
```

Arguments

`xtab`	Predictor vector (x)
`ytab`	Response vector (y)
`knot`	Knot vector or number of knot
`convcons`	not yet implemented Convexity constraint vector per interval: 1 = convex, -1 = concave, 0 = unconstrained
`der3cons`	Not yet implemented Third derivative constraint vector at knot: 1 = positive third derivative, -1 = negative, 0 = unconstrained
`tau`	Quantile (between 0 and 1)
`degree`	the degree of the polynomial spline degree>=5
`solver`	'CVXR' solver to use (default = "OSQP")
`weight`	Observation weights (default = 1 for all)
`verbose`	Logical; if TRUE, print progress messages
`type_reg`	'quantile' or 'mean_square' type of regression

Value

A list containing coefficients, degree, and knot

`SplineQuadraticQuant` *Quantile regression with quadratic splines and shape constraints*

Description

Performs quantile regression using quadratic B-splines with: - Monotonicity: Karlin-Studden constraints on the derivative (linear) - Convexity: Karlin-Studden constraints on the second derivative (constant)

Usage

```
SplineQuadraticQuant(
  xtab,
  ytab,
  knot = NULL,
  tau = 0.5,
  monot = 0,
  convcons = 0,
  solver = "CLARABEL",
  weight = NULL,
  verbose = FALSE,
  type_reg = "quantile"
)
```

Arguments

`xtab`	Predictor vector (x)
`ytab`	Response vector (y)
`knot`	Knot vector or number of knots
`tau`	Quantile (between 0 and 1)
`monot`	Monotonicity constraint vector per interval: 1 = increasing, -1 = decreasing, 0 = unconstrained. Although constraints are set at knots; we apply them on each interval, at both extremities. kn+1 knots, kn constraints taken into account.
`convcons`	Convexity constraint vector per interval: 1 = convex, -1 = concave, 0 = unconstrained. kn constraints are considered for kn+1 knots.
`solver`	'CVXR' solver to use (default = 'CLARABEL')
`weight`	Observation weights (default = 1 for all)
`verbose`	logical; if TRUE, print progress messages
`type_reg`	'quantile' or 'mean_square' type of regression,

Value

A list containing coefficients, degree, and knots

`SplineQuarticQuant` *Quantile regression with quartic splines and shape constraints*

Description

Performs quantile regression using quartic (degree 4) B-splines with monotonicity (Karlin-Studden constraints on the cubic derivative), convexity (Karlin-Studden constraints on the quadratic second derivative), and third derivative constraints (linear at knot).

Usage

```
SplineQuarticQuant(
  xtab,
  ytab,
  knot = NULL,
  tau = 0.5,
  monot = 0,
  convcons = 0,
  der3cons = 0,
  solver = "CLARABEL",
  weight = NULL,
  verbose = FALSE,
  type_reg = "quantile"
)
```

Arguments

`xtab`	Predictor vector (x)
`ytab`	Response vector (y)
`knot`	Knot vector or number of knot
`tau`	Quantile (between 0 and 1)
`monot`	Monotonicity constraint vector per interval: 1 = increasing, -1 = decreasing, 0 = unconstrained
`convcons`	Convexity constraint vector per interval: 1 = convex, -1 = concave, 0 = unconstrained
`der3cons`	Third derivative constraint vector at knot: 1 = positive third derivative, -1 = negative, 0 = unconstrained
`solver`	'CVXR' solver to use (default = "OSQP")
`weight`	Observation weights (default = 1 for all)
`verbose`	Logical; if TRUE, print progress messages
`type_reg`	'quantile' or 'mean_square' type of regression

Value

A list containing coefficients, degree, and knot

Spline_der_knot *Derivatives at knot of a B-spline*

Description

Computes derivative values of a B-spline at knot (efficient because it directly uses polynomial coefficients).

Usage

```
Spline_der_knot(Bsbase, der = 1)
```

Arguments

Bsbase	Object returned by Bspline_base
der	Derivative order (default = 1)

Value

Matrix of derivative values (n_splines x n_knot)

spline_eval *Evaluate a B-spline*

Description

Evaluates a spline (linear combination of B-splines) at given points.

Usage

```
spline_eval(Bspline, x_values = NULL, der = 0, Bvalues = NULL, verbose = FALSE)
```

Arguments

Bspline	Spline object (list with coefficients on the 'Bspline' basis, degree, extended knot) A 'Bspline' can also be rendered callable witk 'make_spline(BSpline)'
x_values	Vector of evaluation points. By default, evaluation is calculated at the knots
der	integer 0 (defalut)...degree is the order of derivative
Bvalues	: the values of the 'Bspline' basis evaluated at the values x this increases the speed by avoiding re-doing the same calculations of the basis for each spline with the same knots.
verbose	boollean, if TRUE output more messages

Value

vector of same length as x_values, with Spline values at the requested points

Examples

```
{ # Create and evaluate a spline
# This function is also used when a 'Bspline' object is rendered callable as a function
# with make_spline()
tn<-c(0,1,2,3,4,5)
x_values<-seq(0,5,length=100)
Bspline=list(degree=3, knot=tn,coeffs=runif(length(tn)+3-1))
y <- spline_eval(Bspline, x_values)
#alternatively compute the base before to optimize if needed
sn <- c(0,0,0,0,1,2,3,4,5,5,5,5)
Bsbase <- Bspline_base(sn, degree=3)
Bvalues <-bs_direct(Bsbase,x_values)
y <- spline_eval(Bspline=Bspline,x_values=x_values, Bvalues=Bvalues )}
```

---

test_karlin_simple *Comprehensive test function*

Description

Runs quantile regression tests with and without constraints, and displays results. Demo function. This example also shows that solution may be evaluated (extrapolation) out of the knot range

Usage

```
test_karlin_simple(degree = 3, seed = NULL, verbose = FALSE)
```

Arguments

degree	1, 2, 3 or 4 (default 3). The degree of the regression spline.
seed	(default=NULL) value for the random generator.
verbose	boolean FALSE (default) or TRUE.

`view_basis` *Visualize a B-spline functions basis Visualize a B-spline basis Plots all basis functions of a B-spline basis with improved styling.*

Description

Visualize a B-spline functions basis Visualize a B-spline basis Plots all basis functions of a B-spline basis with improved styling.

Usage

```
view_basis(BB, x_values = 0, main = NULL, view_knot = TRUE, add_knot = NULL)
```

Arguments

`BB`	Object returned by `Bspline_base`
`x_values`	Vector of evaluation points for plotting. If length is 1 (default = 0), generates 200 points in the knot range.
`main`	Title to print over the plot.
`view_knot`	Logical; if TRUE, adds vertical lines at knot positions.
`add_knot`	logical, same value as 'view_knot' for retro compatibility

Value

No return value, called for side effects (generates a plot)

Index

∗ **internal***
    apply_karlin_quadratic, 3

apply_karlin_quadratic, 3

bs_direct, 10
Bspline_base, *4*, 4
Bspline_base_deriv, 6
Bspline_deriv, 6
bspline_to_deriv_coeffs_cubic, 7
bspline_to_deriv_coeffs_lin, 8
bspline_to_deriv_coeffs_quad, 8
bspline_to_deriv_coeffs_quart, 9
Bsplinetopp, 3, *21*

change_polynomial_base_taylor, 10

evalpp, *4*, 11

get_parameters, 11

make_spline, 12
makpp, *4*, 12, *21*

poly_eval, 15
polyadd, 13
polyderiv, 13
polymul, 14
print.callable_pp, 15
print.callable_spline, 16
print.non_callable_pp, 16
print.non_callable_spline, 17

quantile_spline, 17

reduce_pol, 19

setup_solver, 19
show_poly, 20, *21*
show_pp, 21
Spline_der_knot, 33
spline_eval, 33
SplineConstantQuant, 22
SplineConstQuantRegBs1, 23
SplineConstQuantRegBs2, 24
SplineConstQuantRegBs3, 25
SplineConstQuantRegBs4, 26
SplineCubicQuant, *18*, 27
SplineLinearQuant, *18*, 29
SplinePolynQuant, 30
SplineQuadraticQuant, *18*, 31
SplineQuarticQuant, *18*, 32

test_karlin_simple, 34

view_basis, 35