Package 'BsplineQuantReg'

July 24, 2026

Type Package

Title 'Constrained Quantile Regression with B-Splines'

Version 0.2.1

Date 2026-07-24

Description Quantile regression with B-splines under shape constraints. The initial version with cubic splines is now augmented with splines of degree 1 to 4. Constraints for degrees 3 (monotone) and 4 (monotone and convex) use the Karlin-Studden SOCP characterization for the sign of the polynomial, while other constraints applied at the knots are added as linear problems. The method for cubic splines is described in Abbes (2026) <doi:10.5281/zenodo.17427913>. Other formulations are simple consequences of the other given references. This R implementation is intended for demonstration and prototyping. All B-spline and polynomial functions have been rewritten for consistency. An equivalent Python package is available at <https://pypi.org/project/BsplineQuantRegpy/>.

License GPL-3

URL https://github.com/alexandreabbes/BsplineQuantReg, https://doi.org/10.5281/zenodo.17427913

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

Depends R (>= 3.5.0)

Imports CVXR

Suggests clarabel, ECOSolveR, cobs, quantreg

Encoding UTF-8

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Alexandre Abbes [aut, cre]

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

Repository CRAN

Date/Publication 2026-07-24 21:40:11 UTC

Contents

`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

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 = NULL, 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 begining 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 convention on the local bases (t-s_nu)^l, l=3..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_deriv *Differentiate a B-spline basis*

Description

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

Usage

```
Bspline_deriv(bspline, der = 2, verbose = FALSE)
```

Arguments

bspline	Object returned by Bspline_base
der	Derivative order
verbose	boolean FALSE (default) or TRUE.

Value

A list similar to Bspline_base for the derivative basis

`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, degree = 3, x_values = 0, verbose = FALSE)
```

Arguments

`tn`	Knot vector (effective partition, not extended)
`degree`	Spline degree (default = 3)
`x_values`	Evaluation points for design matrix (0 = no evaluation)
`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 to 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 to 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 *Convert quartic B-spline to derivative coefficients*

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, 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

`get_parameters` *Get parameters from a callable spline*

Description

Get parameters from a callable spline

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 'result' from the regression result 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 using the spline_eval() function, while preserving access to parameters.

Usage

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

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	If TRUE : output the attributes of the Bspline

Value

The same list with an additional 'spline' element containing the callable function. The function can be called as 'Bspline(x)' and has an optional 'Bvalues' parameter to accelerate multiple evaluations at the same x_values.

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_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.quantile_spline *Print method for quantile_spline results*

Description

Print method for quantile_spline results

Usage

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

Arguments

x	Result object from quantile_spline
...	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,
  tau,
  degree = 3,
  monot = 0,
  convcons = 0,
  der3cons = 0,
  solver = "HIGHS",
  weight = NULL,
  verbose = FALSE,
  callable = TRUE
)
```

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

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
set.seed(42)
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(coef in decreasing order)
verbose	boolean FALSE (default) or TRUE.

Value

Reduced vector (without leading zeros)

Examples

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

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 = "HIGHS",
  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 = "HIGHS",
  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 tcontsraints are set at knots; we apply them on each intervall, at both extremities. kn+1 knots, kn consraints 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 = "OSQP")
`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: négative, 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*

Description

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

Usage

```
SplineCubicQuant(
  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: négative, 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

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.16999784
- 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., & Bassett, 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

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 <- 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,
  tau,
  monot = 0,
  solver = "GUROBI",
  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

A list containing coefficients, degree, and knots

`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,
  tau,
  monot = 0,
  convcons = 0,
  solver = "GUROBI",
  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 tcontsraints are set at knots; we apply them on each intervall, at both extremities. kn+1 knots, kn consraints 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 = "OSQP")
`weight`	Observation weights (default = 1 for all)
`verbose`	logical; if TRUE, print progress messages

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,
  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

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, Bvalues = NULL)
```

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
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.

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 meake_spline()
tn<-c(0,1,2,3,4,5)
x_values<-seq(0,5,length=100)
Bspline=list(degree=3, knot=tn,coefficients=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 )}
```

---

view_basis *Visualize a B-spline functions basis*

Description

Plots all functions of a B-spline basis.

Usage

```
view_basis(BB, x_values = 0)
```

Arguments

BB	Object returned by Bspline_base
x_values	Vector of evaluation points for plotting (by default 100 points are computed in the knot range)

Value

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

Index