Package ‘JSODPsplines’

May 5, 2025

Title Resubstitution Method for Derivative Estimation Using P-Splines

Version 0.1.0

Description Tools for estimating derivatives of functions using P-splines.
The main feature is the 'resub' method, a novel approach developed to improve derivative estimation. The package also includes methods for penalized spline estimation, oracle estimation, and optimization of smoothing parameters using generalized cross-validation (GCV) and Mean Integrated Squared Error (MISE).

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Imports stats

Suggests knitr, rmarkdown, testthat

VignetteBuilder knitr

NeedsCompilation no

Author Christopher Odoom [aut, cre],
John Staudenmayer [aut]

Maintainer Christopher Odoom <odoomchristopher22@gmail.com>

Contents

Bbase *B-spline Basis Function*

Description

Creates a B-spline basis for a given set of values.

Usage

```
Bbase(x, xl = min(x), xr = max(x), nseg = 10, bdeg = 3)
```

Arguments

x	Numeric vector of values.
xl	Left boundary.
xr	Right boundary.
nseg	Number of segments.
bdeg	Degree of the B-spline.

Value

A list containing:

x	Numeric vector of input values.
xl	Left boundary.
xr	Right boundary.
nseg	Number of segments.
bdeg	Degree of the B-spline.
B	Matrix of B-spline basis functions.
knots	Vector of knot values.

Examples

```
# Example for Bbase
x <- seq(0, 1, length.out = 100)
result <- Bbase(x, nseg = 10, bdeg = 3)
matplot(x, result$B, type = "l", lty = 1, main = "B-spline Basis")
```

bbase.grid *B-spline Basis on a Grid*

Description

Creates a B-spline basis on a grid.

Usage

```
bbase.grid(x, dx, knots, bdeg)
```

Arguments

x	Numeric vector of values.
dx	Grid spacing.
knots	Knot values.
bdeg	Degree of the B-spline.

Value

A matrix representing the B-spline basis on the grid.

Examples

```
# Example for bbase.grid
x <- seq(0, 1, length.out = 100)
dx <- 0.1
knots <- seq(0, 1, length.out = 10)
deg <- 3
result <- bbase.grid(x, dx, knots, deg)
matplot(x, result, type = "l", lty = 1, main = "B-spline Basis on a Grid")
```

gcvlambda *Generalized Cross-Validation Criterion*

Description

Computes the GCV criterion for a given smoothing parameter lambda.

Usage

```
gcvlambda(lambda = 0, x, y, nseg = 35, pord = 3, bdeg = 4)
```

Arguments

lambda	Smoothing parameter.
x	Numeric vector of x values.
y	Numeric vector of y values.
nseg	Number of segments.
pord	Order of the penalty.
bdeg	Degree of the B-spline.

Value

The GCV criterion value.

Examples

```
# Example for gcvlambda
x <- seq(0, 1, length.out = 100)
y <- sin(2 * pi * x) + rnorm(100, sd = 0.1)
lambda <- 0.1
result <- gcvlambda(lambda, x, y, nseg = 10, pord = 2, bdeg = 3)
print(result)
```

---

mise.lambda.optim *MISE Lambda Optimization*

Description

Optimizes the Mean Integrated Squared Error (MISE) for a given lambda.

Usage

```
mise.lambda.optim(
  lambda = 0.1,
  x,
  y,
  r = 1,
  sig = 0.1,
  nseg = 35,
  pord = 2,
  bdeg = 35,
  f,
  fr = NULL
)
```

Arguments

`lambda`	Smoothing parameter.
`x`	Numeric vector of x values.
`y`	Numeric vector of y values.
`r`	Order of the derivative.
`sig`	Standard deviation of the noise.
`nseg`	Number of segments.
`pord`	Order of the penalty.
`bdeg`	Degree of the B-spline.
`f`	True function values.
`fr`	True derivative values (optional).

Value

A list containing:

mise	Optimized MISE value.
var	Variance component of the MISE.
sq.bias	Squared bias component of the MISE.
H	Matrix of fitted values.

Examples

```
# Example for mise.lambda.optim
x <- seq(0, 1, length.out = 100)
y <- sin(2 * pi * x) + rnorm(100, sd = 0.1)
lambda <- 0.1
sig <- 0.1
f <- sin(2 * pi * x)
result <- mise.lambda.optim(lambda, x, y, r = 1, sig = sig, nseg = 10, pord = 2, bdeg = 3, f = f)
print(result)
```

`naive.est.opt` *Naive Estimation of Derivative (Optimized)*

Description

Estimates the mean and derivative function using optimization to find the optimal smoothing parameter.

Usage

```
naive.est.opt(x, y, r, nseg = 35, bdeg = 4, pord = 2, x.grid)
```

Arguments

x	Numeric vector of x values.
y	Numeric vector of y values.
r	Order of the derivative.
nseg	Number of segments.
bdeg	Degree of the B-spline.
pord	Order of the penalty.
x.grid	Grid of x values for evaluation.

Value

A list containing:

fr.est	List of estimated derivative values.
f.hat	Estimated function values.
fg.hat	Estimated function values on the grid.
fr.hat	Estimated derivative values.

frg.hat	Estimated derivative values on the grid.
sig.hat	Estimated standard deviation of the noise.
lambda	Optimal smoothing parameter.
edf	Effective degrees of freedom.
tr	Trace of the smoothing matrix.

Examples

```
# Example for naive.est.opt
x <- seq(0, 1, length.out = 100)
y <- sin(2 * pi * x) + rnorm(100, sd = 0.1)
x.grid <- seq(0, 1, length.out = 200)
result <- naive.est.opt(x, y, r = 1, nseg = 10, bdeg = 3, pord = 2, x.grid = x.grid)
plot(x.grid, result$frg.hat, type = "l", col = "blue", main = "Naive Estimation")
points(x, y, col = "red")
```

---

oracle.est *Oracle Estimation of Derivative*

Description

Performs oracle estimation of the derivative function.

Usage

```
oracle.est(
  initial.lambda = 0.03,
  x,
  y,
  r,
  fr.grid,
  nseg = 35,
  pord = 2,
  bdeg = 5,
  x.grid
)
```

Arguments

initial.lambda	Initial value for the smoothing parameter.
x	Numeric vector of x values.
y	Numeric vector of y values.
r	Order of the derivative.
fr.grid	True derivative values on the grid.
nseg	Number of segments.
pord	Order of the penalty.
bdeg	Degree of the B-spline.
x.grid	Grid of x values for evaluation.

Value

A list containing:

`x.grid`	Grid of x values for evaluation.
`fr.hat`	Estimated derivative values.
`lambda`	Optimal smoothing parameter.
`frg.hat`	Estimated derivative values on the grid.

Examples

```
# Example for oracle.est
x <- seq(0, 1, length.out = 100)
y <- sin(2 * pi * x) + rnorm(100, sd = 0.1)
x.grid <- seq(0, 1, length.out = 200)
fr.grid <- cos(2 * pi * x.grid)  # True derivative
result <- oracle.est(initial.lambda = 0.1, x, y, r = 1, fr.grid = fr.grid, nseg = 10, pord = 2, bdeg = 3, x.grid =
plot(x.grid, result$frg.hat, type = "l", col = "blue", main = "Oracle Estimation")
points(x, y, col = "red")
```

`oracle.loss` *Oracle Loss Function*

Description

Computes the loss function for oracle estimation.

Usage

```
oracle.loss(
  lambda = 0.2,
  x,
  y,
  r,
  fr.grid,
  nseg = 35,
  pord = 2,
  bdeg = 5,
  x.grid
)
```

Arguments

`lambda`	Smoothing parameter.
`x`	Numeric vector of x values.
`y`	Numeric vector of y values.
`r`	Order of the derivative.
`fr.grid`	True derivative values on the grid.
`nseg`	Number of segments.
`pord`	Order of the penalty.
`bdeg`	Degree of the B-spline.
`x.grid`	Grid of x values for evaluation.

Value

The loss function value.

`pgams` *Penalized Spline Derivative Estimation*

Description

Estimates the derivative function using penalized splines.

Usage

```
pgams(x, y, lambda = 0.1, r = 0, x.grid, nseg = 35, pord = 2, bdeg = 3)
```

Arguments

`x`	Numeric vector of x values.
`y`	Numeric vector of y values.
`lambda`	Smoothing parameter.
`r`	Order of the derivative.
`x.grid`	Grid of x values for evaluation.
`nseg`	Number of segments.
`pord`	Order of the penalty.
`bdeg`	Degree of the B-spline.

Value

A list containing:

`x.grid`	Grid of x values for evaluation.
`f.hat`	Estimated function values.
`fg.hat`	Estimated function values on the grid.
`fr.hat`	Estimated derivative values.
`frg.hat`	Estimated derivative values on the grid.
`K`	Matrix of reparametrized parameters.
`M`	Matrix of smoothing parameters.
`Atilde`	Matrix of transformed basis functions.
`A`	Matrix of fitted values.
`lambda`	Smoothing parameter.

Examples

```
# Example for pgams
x <- seq(0, 1, length.out = 100)
y <- sin(2 * pi * x) + rnorm(100, sd = 0.1)
x.grid <- seq(0, 1, length.out = 200)
result <- pgams(x, y, lambda = 0.1, r = 1, x.grid = x.grid, nseg = 10, pord = 2, bdeg = 3)
plot(x.grid, result$frg.hat, type = "l", col = "blue", main = "Estimated Derivative")
points(x, y, col = "red")
```

plugin.est *Plug-in Estimation of Derivative*

Description

Performs one-step plug-in estimation of the derivative function.

Usage

```
plugin.est(x, y, r, nseg = 35, pord = 3, bdeg = 4, x.grid, fr = NULL)
```

Arguments

x	Numeric vector of x values.
y	Numeric vector of y values.
r	Order of the derivative.
nseg	Number of segments.
pord	Order of the penalty.
bdeg	Degree of the B-spline.
x.grid	Grid of x values for evaluation.
fr	Optional true derivative values.

Value

A list containing:

x.grid	Grid of x values for evaluation.
f.hat	Estimated function values.
fg.hat	Estimated function values on the grid.
fr.hat	Estimated derivative values.
frg.hat	Estimated derivative values on the grid.
lambda	Optimal smoothing parameter.
K	Matrix of reparametrized parameters.
M	Matrix of smoothing parameters.
Atilde	Matrix of transformed basis functions.
A	Matrix of fitted values.
sig.hat	Estimated standard deviation of the noise.

Examples

```
# Example for plugin.est
x <- seq(0, 1, length.out = 100)
y <- sin(2 * pi * x) + rnorm(100, sd = 0.1)
x.grid <- seq(0, 1, length.out = 200)
result <- plugin.est(x, y, r = 1, nseg = 10, pord = 2, bdeg = 3, x.grid = x.grid)
plot(x.grid, result$frg.hat, type = "l", col = "blue", main = "Plug-in Estimation")
points(x, y, col = "red")
```

`resub.est` *Iterative Re-substitution Estimation*

Description

Performs iterative re-substitution estimation of the derivative function.

Usage

```
resub.est(x, y, r, x.grid, nseg, pord, bdeg, tol = 1e-10, ITs = 10)
```

Arguments

`x`	Numeric vector of x values.
`y`	Numeric vector of y values.
`r`	Order of the derivative.
`x.grid`	Grid of x values for evaluation.
`nseg`	Number of segments.
`pord`	Order of the penalty.
`bdeg`	Degree of the B-spline.
`tol`	Tolerance for convergence.
`ITs`	Maximum number of iterations.

Value

A list containing:

`x.grid`	Grid of x values for evaluation.
`fr.hat`	Estimated derivative values.
`lambda`	Optimal smoothing parameter.
`frg.hat`	Estimated derivative values on the grid.

Examples

```
# Example for resub.est
x <- seq(0, 1, length.out = 100)
y <- sin(2 * pi * x) + rnorm(100, sd = 0.1)
x.grid <- seq(0, 1, length.out = 200)
result <- resub.est(x, y, r = 1, x.grid = x.grid, nseg = 10, pord = 2, bdeg = 3)
plot(x.grid, result$frg.hat, type = "l", col = "blue", main = "Resubstitution Estimation")
points(x, y, col = "red")
```

tpower *Truncated Power Function*

Description

Computes the truncated p-th power function.

Usage

```
tpower(x, t, p)
```

Arguments

x	Numeric vector of values to evaluate.
t	Knot value.
p	Degree of the polynomial.

Value

Numeric vector of evaluated values.

Examples

```
# Example for tpower
x <- seq(0, 1, length.out = 100)
t <- 0.5
p <- 2
result <- tpower(x, t, p)
plot(x, result, type = "l", col = "blue", main = "Truncated Power Function")
```

Index