Package 'JSODPsplines'

May 6, 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. Deault is 10.
`bdeg`	Degree of the B-spline. Default is 3.

Details

The function generates a B-spline basis matrix for the given input values. The basis is constructed using the specified degree and number of segments. The knots are generated based on the left and right boundaries and the number of segments.

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.

References

Eilers, P. H. C. & Marx, B. D. (1996). Flexible smoothing with B-splines and penalties. Statistical Science, 11(2), 89-121.

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.

Details

The function generates a B-spline basis matrix for the given input values on a specified grid. The basis is constructed using the specified degree and knot values. The grid is defined by the input values and the specified spacing.

Value

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

References

Eilers, P. H. C. & Marx, B. D. (1996). Flexible smoothing with B-splines and penalties. Statistical Science, 11(2), 89-121.

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.

Details

The function computes the GCV criterion value based on the residual sum of squares (RSS) and the effective degrees of freedom (EDF). The GCV criterion is a measure of the goodness of fit of the model, adjusted for the complexity of the model. It is used to select the optimal smoothing parameter by minimizing the GCV value. The function uses the fitted values from the penalized spline model to compute the RSS and EDF. The GCV criterion is defined as:

$$GCV(\lambda) = \frac{RSS(\lambda)}{(n - \text{EDF}(\lambda))^2}$$

where $RSS(\lambda)$ is the residual sum of squares and EDF(λ) is the effective degrees of freedom.

Value

The GCV criterion value.

References

Eilers, P. H. C. & Marx, B. D. (1996). Flexible smoothing with B-splines and penalties. Statistical Science, 11(2), 89-121.

See Also

`pgams`, `naive.est.opt`, `plugin.est`, `resub.est`

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)
# Example 2 for gcvlambda
x <- seq(0, 1, length.out = 100)
y <- 32 * exp(-8 * (1 - 2 * x)^2) * (1 - 2 * x) + rnorm(100, sd = 0.1)
lambda <- 0.1
result <- gcvlambda(lambda, x, y, nseg = 10, pord = 2, bdeg = 3)
print(result)
# Example 3 for gcvlambda
x <- seq(0, 1, length.out = 100)
y <- 32 * exp(-8 * (1 - 2 * x)^2) * (1 - 2 * x) + rnorm(100, sd = 0.1)
lambdas <- seq(0, 1, length.out = 10)
gcv_values <- sapply(lambdas, function(l) gcvlambda(l, x, y, nseg = 10, pord = 2, bdeg = 3))
plot(lambdas, gcv_values, type = "b", xlab = "Lambda", ylab = "GCV", main = "GCV vs Lambda")
abline(v = lambdas[which.min(gcv_values)], col = "red", lty = 2)
```

`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).

Details

The function computes the MISE for a given smoothing parameter lambda. It uses the B-spline basis to estimate the function and its derivative. The MISE is calculated as the sum of the variance and squared bias components. The variance component is based on the estimated smoothing parameter and the noise level. The squared bias component is based on the difference between the estimated and true function values. The function returns the optimized MISE value along with its components.

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.

References

Eilers, P. H. C. & Marx, B. D. (1996). Flexible smoothing with B-splines and penalties. Statistical Science, 11(2), 89-121.

See Also

pgams, naive.est.opt, plugin.est, resub.est

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)
# Example 2 for mise.lambda.optim
x <- seq(0, 1, length.out = 100)
y <- 32 * exp(-8 * (1 - 2 * x)^2) * (1 - 2 * x) + rnorm(100, sd = 0.1)
lambda <- 0.1
sig <- 0.1
f <- 32 * exp(-8 * (1 - 2 * x)^2) * (1 - 2 * 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 = NULL)
```

Arguments

`x`	Numeric vector of x values.
`y`	Numeric vector of y values.
`r`	Order of the derivative. The value of r must be greater than or equal to 1 since the function already estimates the mean function.
`nseg`	Number of segments.
`bdeg`	Degree of the B-spline.
`pord`	Order of the penalty.
`x.grid`	Grid of x values for evaluation. if NULL, it is generated based on the input x values.

Details

The function estimates the mean and derivative function using penalized splines. The B-spline basis is constructed based on the input values and the specified parameters. The smoothing parameter is optimized using the generalized cross-validation criterion. The function returns the estimated function values, derivative values, and other relevant matrices.

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.

References

Eilers, P. H. C. & Marx, B. D. (1996). Flexible smoothing with B-splines and penalties. Statistical Science, 11(2), 89-121.

Examples

```
# Example 1 for naive.est.opt
x <- seq(0, 1, length.out = 100)
f <- sin(2 * pi * x)
fprime <- cos(2 * pi * x)*(2 * pi)
set.seed(123)
y <- f + rnorm(100, sd = 0.1)
x.grid <- seq(0, 1, length.out = 200)
fprime.grid <- cos(2 * pi * x.grid)*(2 * pi)
result <- naive.est.opt(x, y, r = 1, nseg = 10, pord = 2, bdeg = 3, x.grid = x.grid)
plot(x.grid, result$frg.hat, type = "l", col = "blue", main = "Naive Estimation")
lines(x.grid, fprime.grid, col = "green", lty = 2)
legend("topright", legend = c("Estimated Derivative", "True Derivative"), col = c("blue", "green"), lty = 1:2)
# Example 2 for naive.est.opt
x <- seq(0, 1, length.out = 100)
f <- 32 * exp(-8 * (1 - 2 * x)^2) * (1 - 2 * x)
fprime <- (4096 * x^2 - 4096 * x + 960) * exp(-8 * (1 - 2 * x)^2)
set.seed(123)
y <- f + rnorm(100, sd = 0.1)
x.grid <- seq(0, 1, length.out = 200)
fprime.grid <- (4096 * x.grid^2 - 4096 * x.grid + 960) * exp(-8 * (1 - 2 * x.grid)^2)
result <- naive.est.opt(x, y, r = 1, nseg = 10, pord = 2, bdeg = 3, x.grid = x.grid)
plot(x.grid, result$frg.hat, type = "l", col = "blue", main = "Naive Estimation")
lines(x.grid, fprime.grid, col = "green", lty = 2)
legend("topright", legend = c("Estimated Derivative", "True Derivative"), col = c("blue", "green"), lty = 1:2)
```

`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. The value of r must be greater than or equal to 1 since the function already estimates the mean function.
`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. If NULL, it is generated based on the input x values.

Details

The function estimates the derivative using information about the true derivative. It uses the oracle loss function to optimize the smoothing parameter. It is assumed that the true derivative is known on the grid. This estimation is useful for evaluating the performance of the method since it provides a benchmark for the estimated derivative.

Value

A list containing:

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

References

Eilers, P. H. C. & Marx, B. D. (1996). Flexible smoothing with B-splines and penalties. Statistical Science, 11(2), 89-121.

Examples

```
# Example for oracle.est
x <- seq(0, 1, length.out = 100)
f <- sin(2 * pi * x)
fprime <- cos(2 * pi * x)*(2 * pi)
set.seed(123)
y <- f + rnorm(100, sd = 0.1)
x.grid <- seq(0, 1, length.out = 200)
fprime.grid <- cos(2 * pi * x.grid)*(2 * pi)
result <- oracle.est(initial.lambda = 0.1, x, y, r = 1, fr.grid = fprime.grid, nseg = 10, pord = 2, bdeg = 3, x.gr
plot(x.grid, result$frg.hat, type = "l", col = "blue", main = "Oracle Estimation")
lines(x.grid, fprime.grid, col = "green", lty = 2)
legend("topright", legend = c("Estimated Derivative", "True Derivative"), col = c("blue", "green"), lty = 1:2)
# Example 2 for oracle.est
x <- seq(0, 1, length.out = 100)
f <- 32 * exp(-8 * (1 - 2 * x)^2) * (1 - 2 * x)
fprime <- (4096 * x^2 - 4096 * x + 960) * exp(-8 * (1 - 2 * x)^2)
set.seed(123)
y <- f + rnorm(100, sd = 0.1)
x.grid <- seq(0, 1, length.out = 200)
fprime.grid <- (4096 * x.grid^2 - 4096 * x.grid + 960) * exp(-8 * (1 - 2 * x.grid)^2)
```

```
result <- oracle.est(initial.lambda = 0.1, x, y, r = 1, fr.grid = fprime.grid, nseg = 10, pord = 2, bdeg = 3, x.gr
plot(x.grid, result$frg.hat, type = "l", col = "blue", main = "Oracle Estimation")
lines(x.grid, fprime.grid, col = "green", lty = 2)
legend("topright", legend = c("Estimated Derivative", "True Derivative"), col = c("blue", "green"), lty = 1:2)
```

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.

Details

The function computes the loss function value based on the difference between the estimated derivative and the true derivative. It is used in the oracle estimation process to optimize the smoothing parameter.

Value

The loss function value.

References

Eilers, P. H. C. & Marx, B. D. (1996). Flexible smoothing with B-splines and penalties. Statistical Science, 11(2), 89-121.

See Also

pgams, naive.est.opt, plugin.est, resub.est

Examples

```
# Example for oracle.loss
x <- seq(0, 1, length.out = 100)
f <- sin(2 * pi * x)
fprime <- cos(2 * pi * x)*(2 * pi)
set.seed(123)
y <- f + rnorm(100, sd = 0.1)
x.grid <- seq(0, 1, length.out = 200)
fprime.grid <- cos(2 * pi * x.grid)*(2 * pi)
result <- oracle.loss(lambda = 0.1, x, y, r = 1, fr.grid = fprime.grid, nseg = 10, pord = 2, bdeg = 3, x.grid = x.g
print(result)
# Example 2 for oracle.loss
x <- seq(0, 1, length.out = 100)
f <- 32 * exp(-8 * (1 - 2 * x)^2) * (1 - 2 * x)
fprime <- (4096 * x^2 - 4096 * x + 960) * exp(-8 * (1 - 2 * x)^2)
set.seed(123)
y <- f + rnorm(100, sd = 0.1)
x.grid <- seq(0, 1, length.out = 200)
fprime.grid <- (4096 * x.grid^2 - 4096 * x.grid + 960) * exp(-8 * (1 - 2 * x.grid)^2)
result <- oracle.loss(lambda = 0.1, x, y, r = 1, fr.grid = fprime.grid, nseg = 10, pord = 2, bdeg = 3, x.grid = x.g
print(result)
```

---

pgams *Penalized Spline Derivative Estimation*

Description

Estimates the derivative function using penalized splines.

Usage

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

Arguments

x	Numeric vector of x values.
y	Numeric vector of y values.
lambda	Smoothing parameter. The default is 0.1.
r	Order of the derivative. The default is 0 which means estimating the mean function.
x.grid	Grid of x values for evaluation. if NULL, it is generated based on the input x values.

nseg	Number of segments. The default is 35.
pord	Order of the penalty. The default is 2.
bdeg	Degree of the B-spline. The default is 3.

Details

The function estimates the mean and derivative function using penalized splines. The B-spline basis is constructed based on the input values and the specified parameters. The smoothing parameter is used to control the amount of smoothing applied to the estimated function. The function returns the estimated function values, derivative values, and other relevant matrices.

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.

References

Eilers, P. H. C. & Marx, B. D. (1996). Flexible smoothing with B-splines and penalties. Statistical Science, 11(2), 89-121.

Examples

```
# Example  1 for pgams
x <- seq(0, 1, length.out = 100)
f<- sin(2 * pi * x)
fprime <- cos(2 * pi * x)*(2 * pi)
set.seed(123)
y <- f + rnorm(100, sd = 0.1)
points(x, y, col = "red")
x.grid <- seq(0, 1, length.out = 200)
fprime.grid <- cos(2 * pi * x.grid)*(2 * pi)
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")
lines(x.grid, fprime.grid, col = "green", lty = 2)
legend("topright", legend = c("Estimated Derivative", "True Derivative"), col = c("blue", "green"), lty = 1:2)
#' # Example 2 for pgams
x <- seq(0, 1, length.out = 100)
set.seed(123)
f <- 32 * exp(-8 * (1 - 2 * x)^2) * (1 - 2 * x)
fprime <- (4096 * x^2 - 4096 * x + 960) * exp(-8 * (1 - 2 * x)^2)
set.seed(123)
```

```
y <- f + rnorm(100, sd = 0.1)
points(x, y, col = "red")
x.grid <- seq(0, 1, length.out = 200)
fprime.grid <- (4096 * x.grid^2 - 4096 * x.grid + 960) * exp(-8 * (1 - 2 * x.grid)^2)
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")
lines(x.grid, fprime.grid, col = "green", lty = 2)
legend("topright", legend = c("Estimated Derivative", "True Derivative"), col = c("blue", "green"), lty = 1:2)
```

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)
```

Arguments

x	Numeric vector of x values.
y	Numeric vector of y values.
r	Order of the derivative. The value of r must be greater than or equal to 1 since the function already estimates the mean function.
nseg	Number of segments. The default is 35.
pord	Order of the penalty.The default is 2.
bdeg	Degree of the B-spline. The default is 3.
x.grid	Grid of x values for evaluation. If NULL, it is generated based on the input x values.

Details

The function estimates the mean and derivative function using penalized splines. The B-spline basis is constructed based on the input values and the specified parameters. It uses the mean integrated squared error (MISE) to optimize the smoothing parameter.

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.

References

Eilers, P. H. C. & Marx, B. D. (1996). Flexible smoothing with B-splines and penalties. Statistical Science, 11(2), 89-121.

Examples

```
# Example 1 for plugin.est
x <- seq(0, 1, length.out = 100)
f <- sin(2 * pi * x)
fprime <- cos(2 * pi * x)*(2 * pi)
set.seed(123)
y <- f + rnorm(100, sd = 0.1)
x.grid <- seq(0, 1, length.out = 200)
fprime.grid <- cos(2 * pi * x.grid)*(2 * pi)
result <- plugin.est(x, y, r = 1, nseg = 10, pord = 2, bdeg = 3, x.grid = x.grid)
plot(x.grid, result$frg.hat, type = "l", col = "blue", main = "Plug-in Estimation")
lines(x.grid, fprime.grid, col = "green", lty = 2)
legend("topright", legend = c("Estimated Derivative", "True Derivative"), col = c("blue", "green"), lty = 1:2)
#' # Example 2 for plugin.est
x <- seq(0, 1, length.out = 100)
f <- 32 * exp(-8 * (1 - 2 * x)^2) * (1 - 2 * x)
fprime <- (4096 * x^2 - 4096 * x + 960) * exp(-8 * (1 - 2 * x)^2)
set.seed(123)
y <- f + rnorm(100, sd = 0.1)
x.grid <- seq(0, 1, length.out = 200)
fprime.grid <- (4096 * x.grid^2 - 4096 * x.grid + 960) * exp(-8 * (1 - 2 * x.grid)^2)
result <- plugin.est(x, y, r = 1, nseg = 10, pord = 2, bdeg = 3, x.grid = x.grid)
plot(x.grid, result$frg.hat, type = "l", col = "blue", main = "Plug-in Estimation")
lines(x.grid, fprime.grid, col = "green", lty = 2)
legend("topright", legend = c("Estimated Derivative", "True Derivative"), col = c("blue", "green"), lty = 1:2)
```

`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. The value of r must be greater than or equal to 1 since the function already estimates the mean function.
`x.grid`	Grid of x values for evaluation. If NULL, it is generated based on the input x values.
`nseg`	Number of segments. The default is 35.
`pord`	Order of the penalty. The default is 2.
`bdeg`	Degree of the B-spline. The default is 3.
`tol`	Tolerance for convergence. The default is 1e-10. The tolerance is used to determine when the optimization has converged and it takes precedence over the maximum number of iterations.
`ITs`	Maximum number of iterations. The default is 10.

Details

The function estimates the mean and derivative function using penalized splines. The B-spline basis is constructed based on the input values and the specified parameters. It uses the mean integrated squared error (MISE) to optimize the smoothing parameter. This ivolves iteratively updating the estimated derivative function until convergence is reached.

Value

A list containing:

`x.grid`	Grid of x values for evaluation.
`f.hat`	Estimated function values using the improved smoothing parameter from iterations.
`fr.hat`	Estimated derivative values using iterative smoothing parameter.
`lambda`	Optimal smoothing parameter from iteration.
`frg.hat`	Estimated derivative values on the grid.

Examples

```
# Example 1 for resub.est
x <- seq(0, 1, length.out = 100)
f <- sin(2 * pi * x)
fprime <- cos(2 * pi * x)*(2 * pi)
y <- sin(2 * pi * x) + rnorm(100, sd = 0.1)
x.grid <- seq(0, 1, length.out = 200)
fprime.grid <- cos(2 * pi * x.grid)*(2 * pi)
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")
lines(x.grid, fprime.grid, col = "green", lty = 2)
legend("topright", legend = c("Estimated Derivative", "True Derivative"), col = c("blue", "green"), lty = 1:2)
#' # Example 2 for resub.est
x <- seq(0, 1, length.out = 100)
f <- 32 * exp(-8 * (1 - 2 * x)^2) * (1 - 2 * x)
fprime <- (4096 * x^2 - 4096 * x + 960) * exp(-8 * (1 - 2 * x)^2)
set.seed(123)
```

```
y <- f + rnorm(100, sd = 0.1)
x.grid <- seq(0, 1, length.out = 200)
fprime.grid <- (4096 * x.grid^2 - 4096 * x.grid + 960) * exp(-8 * (1 - 2 * x.grid)^2)
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")
lines(x.grid, fprime.grid, col = "green", lty = 2)
legend("topright", legend = c("Estimated Derivative", "True Derivative"), col = c("blue", "green"), lty = 1:2)
#' @references Eilers, P. H. C. & Marx, B. D. (1996). Flexible smoothing with B-splines and penalties. Statistic
```

Index