Package ‘kedd’

October 31, 2015

Type Package
Title Kernel Estimator and Bandwidth Selection for Density and Its Derivatives
Version 1.0.3
Date 2015-10-30
Author Arsalane Chouaib Guidoum
Maintainer Arsalane Chouaib Guidoum <acguidoum@usthb.dz>
Depends R (>= 2.15.0)
Description Smoothing techniques and computing bandwidth selectors of the nth derivative of a probability density for one-dimensional data.
License GPL (>= 2)
LazyLoad yes
LazyData yes
Classification/MSC 62G05, 62G07, 65D10, 68N15
NeedsCompilation no
Repository CRAN
Date/Publication 2015-10-31 00:19:16

R topics documented:

kedd-package .................................................. 2
Claw, Bimodal, Kurtotic, Outlier, Trimodal ........................... 7
dkde .......................................................... 9
h.amise ....................................................... 13
h.bcv .......................................................... 15
h.ccv .......................................................... 17
h.mcv ......................................................... 19
h.mlcv ......................................................... 21
h.tcv .......................................................... 23
h.ucv .......................................................... 25
ekernel.conv .................................................. 27
Kernel Estimator and Bandwidth Selection for Density and Its Derivatives

Description

Smoothing techniques and computing bandwidth selectors of the r’th derivative of a probability density for one-dimensional data.

Details

Package: kedd
Type: Package
Version: 1.0.3
Date: 2015-10-30
License: GPL (>= 2)

There are four main types of functions in this package:

1. Compute the derivatives and convolutions of a kernel function (1-d).
2. Compute the kernel estimators for density and its derivatives (1-d).
3. Computing the bandwidth selectors (1-d).
4. Displaying kernel estimators.

Main Features

Convolutions and derivatives in kernel function:

In non-parametric statistics, a kernel is a weighting function used in non-parametric estimation techniques. The kernels functions $K(x)$ are used in derivatives of kernel density estimator to estimate $f_h^{(r)}(x)$, satisfying the following three requirements:
1. $\int_{R} K(x) \, dx = 1$
2. $\int_{R} x K(x) \, dx = 0$
3. $\mu_2(K) = \int_{R} x^2 K(x) \, dx < \infty$

Several types of kernel functions $K(x)$ are commonly used in this package: Gaussian, Epanechnikov, Uniform (rectangular), Triangular, Triweight, Tricube, Biweight (quartic), Cosine.

The function `kernel.fun` for kernel derivative $K^{(r)}(x)$ and `kernel.conv` for kernel convolution $K^{(r)} * K^{(r)}(x)$, where the write formally:

$$K^{(r)}(x) = \frac{d^r}{dx^r} K(x)$$

$$K^{(r)} * K^{(r)}(x) = \int_{-\infty}^{+\infty} K^{(r)}(y) K^{(r)}(x - y) \, dy$$

for $r = 0, 1, 2, \ldots$

**Estimators of r’th derivative of a density function:**

A natural estimator of the r’th derivative of a density function $f(x)$ is:

$$f_h^{(r)}(x) = \frac{d^r}{dx^r} \frac{1}{nh} \sum_{i=1}^{n} K \left( \frac{x - X_i}{h} \right) = \frac{1}{nh^{r+1}} \sum_{i=1}^{n} K^{(r)} \left( \frac{x - X_i}{h} \right)$$

Here, $X_1, X_2, \ldots, X_n$ is an i.i.d. sample of size $n$ from the distribution with density $f(x)$, $K(x)$ is the kernel function which we take to be a symmetric probability density with at least $r$ non zero derivatives when estimating $f^{(r)}(x)$, and $h$ is the bandwidth, this parameter is very important that controls the degree of smoothing applied to the data.

The case $(r = 0)$ is the standard kernel density estimator (e.g. Silverman 1986, Wolfgang 1991, Scott 1992, Wand and Jones 1995, Jeffrey 1996, Bowman and Azzalini 1997, Alexandre 2009), properties of such derivative estimators are well known e.g. Sheather and Jones (1991), Jones and Kappenman (1991), Wolfgang (1991). For the case $(r > 0)$, is derivative of kernel density estimator (e.g. Bhattacharya 1967, Schuster 1969, Alekseev 1972, Wolfgang et all 1990, Jones 1992, Stoker 1993) and for applications which require the estimation of density derivatives can be found in Singh (1977).

For r’th derivatives of kernel density estimator one-dimensional, the main function is `dkde`. For display, its plot method calls `plot.dkde`, and if to add a plot using `lines.dkde`.

```
R> data(trimodal)
R> dkde(x = trimodal, deriv.order = 0, kernel = "gaussian")

Data: trimodal (200 obs.); Kernel: gaussian
Derivative order: 0; Bandwidth 'h' = 0.1007
eval.points est.fx
Min. : -2.91274 Min. :0.0000066
```
Bandwidth selectors:

The most important factor in the r'th derivative kernel density estimate is a choice of the bandwidth $h$ for one-dimensional observations. Because of its role in controlling both the amount and the direction of smoothing, this choice is particularly important. We present the popular bandwidth selection (for more details see references) methods in this package:

- Optimal Bandwidth (AMISE); with $\text{deriv.order} \geq 0$, name of this function is $\text{h.amise}$. For display, its plot method calls $\text{plot.h.amise}$, and to add a plot used $\text{lines.h.amise}$.
- Maximum-likelihood cross-validation (MLCV); with $\text{deriv.order} = 0$, name of this function is $\text{h.mlcv}$. For display, its plot method calls $\text{plot.h.mlcv}$, and to add a plot used $\text{lines.h.mlcv}$.
- Unbiased cross validation (UCV); with $\text{deriv.order} \geq 0$, name of this function is $\text{h.ucv}$. For display, its plot method calls $\text{plot.h.ucv}$, and to add a plot used $\text{lines.h.ucv}$.
- Biased cross validation (BCV); with $\text{deriv.order} \geq 0$, name of this function is $\text{h.bcv}$. For display, its plot method calls $\text{plot.h.bcv}$, and to add a plot used $\text{lines.h.bcv}$.
- Complete cross-validation (CCV); with $\text{deriv.order} \geq 0$, name of this function is $\text{h.ccv}$. For display, its plot method calls $\text{plot.h.ccv}$, and to add a plot used $\text{lines.h.ccv}$.
- Modified cross-validation (MCV); with $\text{deriv.order} \geq 0$, name of this function is $\text{h.mcv}$. For display, its plot method calls $\text{plot.h.mcv}$, and to add a plot used $\text{lines.h.mcv}$.
- Trimmed cross-validation (TCV); with $\text{deriv.order} \geq 0$, name of this function is $\text{h.tcv}$. For display, its plot method calls $\text{plot.h.tcv}$, and to add a plot used $\text{lines.h.tcv}$.

R> data(trimodal)
R> h.bcv(x = trimodal, whichbcv = 1, deriv.order = 0, kernel = "gaussian")

Call: Biased Cross-Validation 1
Derivative order = 0
Data: trimodal (200 obs.); Kernel: gaussian
Min BCV = 0.004511636; Bandwidth 'h' = 0.4357812

R> h.ccv(x = trimodal, deriv.order = 1, kernel = "gaussian")

Call: Complete Cross-Validation
Derivative order = 1
Data: trimodal (200 obs.); Kernel: gaussian
Min CCV = 0.01985078; Bandwidth 'h' = 0.5828336

R> h.tcv(x = trimodal, deriv.order = 2, kernel = "gaussian")

Call: Trimmed Cross-Validation
Derivative order = 2
Data: trimodal (200 obs.); Kernel: gaussian
Min TCV = -295.563; Bandwidth 'h' = 0.08908582

R> h.ucv(x = trimodal, deriv.order = 3, kernel = "gaussian")

Call: Unbiased Cross-Validation
Derivative order = 3
Data: trimodal (200 obs.); Kernel: gaussian
Min UCV = -63165.18; Bandwidth 'h' = 0.1067236

For an overview of this package, see vignette("kedd").

Requirements

R version >= 2.15.0

Licence

This package and its documentation are usable under the terms of the "GNU General Public License", a copy of which is distributed with the package.

Author(s)

Arsalane Chouaib Guidoum <acguidoum@usthb.dz> (Dept. Probability and Statistics, USTHB, Algeria). Please send comments, error reports, etc. to the author via the addresses email mentioned above.

References


Claw, Bimodal, Kurtotic, Outlier, Trimodal


See Also

**ks, KernSmooth, sm, np, locfit, feature, GenKern.**

**Datasets**

A random sample of size 200 from the claw, bimodal, kurtotic, outlier and trimodal Gaussian density.
Usage

data(claw)
data(bimodal)
data(kurtotic)
data(outlier)
data(trimodal)

Format

Numeric vector with length 200.

Details

Generate 200 random numbers, distributed according to a normal mixture, using `rnormMix` in package `nor1mix`.

```r
## Claw density
claw <- rnormMix(n=200, MW.mm10)
plot(MW.mm10)

## Bimodal density
bimodal <- rnormMix(n=200, MW.mm7)
plot(MW.mm7)

## Kurtotic density
kurtotic <- rnormMix(n=200, MW.mm4)
plot(MW.mm4)

## Outlier density
outlier <- rnormMix(n=200, MW.mm5)
plot(MW.mm5)

## Trimodal density
trimodal <- rnormMix(n=200, MW.mm9)
plot(MW.mm9)
```

Source

Randomly generated a normal mixture with the function `rnormMix` in package `nor1mix`.

References

Derivatives of Kernel Density Estimator

Description

The (S3) generic function `dkde` computes the r'th derivative of kernel density estimator for one-dimensional data. Its default method does so with the given kernel and bandwidth \( h \) for one-dimensional observations.

Usage

```r
dkde(x, ...)  
## Default S3 method:  
dkde(x, y = NULL, deriv.order = 0, h, kernel = c("gaussian",  
  "epanechnikov", "uniform", "triangular", "triweight",  
  "tricube", "biweight", "cosine"), ...)  
```

Arguments

- `x`: the data from which the estimate is to be computed.
- `y`: the points of the grid at which the density derivative is to be estimated; the defaults are \( \tau \ast h \) outside of range(`x`), where \( \tau = 4 \).
- `deriv.order`: derivative order (scalar).
- `h`: the smoothing bandwidth to be used, can also be a character string giving a rule to choose the bandwidth, see `h.bcv`. The default `h.ucv`.
- `kernel`: a character string giving the smoothing kernel to be used, with default "gaussian".
- `...`: further arguments for (non-default) methods.

Details

A simple estimator for the density derivative can be obtained by taking the derivative of the kernel density estimate. If the kernel \( K(x) \) is differentiable \( r \) times then the r'th density derivative estimate can be written as:

\[
\hat{f}_h^{(r)}(x) = \frac{1}{nh^{r+1}} \sum_{i=1}^{n} K^{(r)} \left( \frac{x - X_i}{h} \right)
\]

where,

\[
K^{(r)}(x) = \frac{d^r}{dx^r} K(x)
\]

for \( r = 0, 1, 2, \ldots \)

The following assumptions on the density \( f^{(r)}(x) \), the bandwidth \( h \), and the kernel \( K(x) \):

1. The \((r + 2)\) derivative \( f^{(r+2)}(x) \) is continuous, square integrable and ultimately monotone.
2. \( \lim_{n \to \infty} h = 0 \) and \( \lim_{n \to \infty} nh^{2r+1} = \infty \) i.e., as the number of samples \( n \) is increased \( h \) approaches zero at a rate slower than \( 1/n^{2r+1} \).
3. $K(x) \geq 0$ and $\int_{\mathbb{R}} K(x)dx = 1$. The kernel function is assumed to be symmetric about the origin i.e., $\int_{\mathbb{R}} x K(r)(x)dx = 0$ for even $r$ and has finite second moment i.e., $\mu_2(K) = \int_{\mathbb{R}} x^2 K(x)dx < \infty$.

Some theoretical properties of the estimator $\hat{f}_h^{(r)}$ have been investigated, among others, by Bhat-tacharya (1967), Schuster (1969). Let us now turn to the statistical properties of estimator. We are interested in the mean squared error since it combines squared bias and variance.

The bias can be written as:

$$E\left[\hat{f}_h^{(r)}(x) - f^{(r)}(x)\right] = \frac{1}{2} h^2 \mu_2(K) f^{(r+2)}(x) + o(h^2)$$

The variance of the estimator can be written as:

$$\text{VAR}\left[\hat{f}_h^{(r)}(x)\right] = \frac{f(x) R\left(K^{(r)}\right)}{nh^{2r+1}} + o(1/nh^{2r+1})$$

with, $R\left(K^{(r)}\right) = \int_{\mathbb{R}} (K^{(r)}(x))^2 dx$.

The MSE (Mean Squared Error) for kernel density derivative estimators can be written as:

$$\text{MSE}\left(\hat{f}_h^{(r)}(x), f^{(r)}(x)\right) = \frac{f(x) R\left(K^{(r)}\right)}{nh^{2r+1}} + \frac{1}{4} h^4 \mu_2^2(K) f^{(r+1)}(x)^2 + o(h^4 + 1/nh^{2r+1})$$

It follows that the MSE-optimal bandwidth for estimating $\hat{f}_h^{(r)} S(x)$, is of order $n^{-1/(2r+5)}$. Therefore, the estimation of $\hat{f}_h^{(1)}(x)$ requires a bandwidth of order $n^{-1/7}$ compared to the optimal $n^{-1/5}$ for estimating $f(x)$ itself. It reveals the increasing difficulty in problems of estimating higher derivatives.

The MISE (Mean Integrated Squared Error) can be written as:

$$\text{MISE}\left(\hat{f}_h^{(r)}(x), f^{(r)}(x)\right) = \text{AMISE}\left(\hat{f}_h^{(r)}(x), f^{(r)}(x)\right) + o(h^4 + 1/nh^{2r+1})$$

where,

$$\text{AMISE}\left(\hat{f}_h^{(r)}(x), f^{(r)}(x)\right) = \frac{1}{nh^{2r+1}} R\left(K^{(r)}\right) + \frac{1}{4} h^4 \mu_2^2(K) R\left(f^{(r+2)}\right)$$

with: $R\left(f^{(r)}(x)\right) = \int_{\mathbb{R}} (f^{(r)}(x))^2 dx$.

The performance of kernel is measured by MISE or AMISE (Asymptotic MISE).

If the bandwidth $h$ is missing from dkde, then the default bandwidth is $h_{ucv}(x, \text{deriv.order}, \text{kernel})$ (Unbiased cross-validation, see $h_{ucv}$).

For more details see references.
Value

- x: data points - same as input.
- data.name: the deparsed name of the x argument.
- n: the sample size after elimination of missing values.
- kernel: name of kernel to use.
- deriv.order: the derivative order to use.
- h: the bandwidth value to use.
- eval.points: the coordinates of the points where the density derivative is estimated.
- est.fx: the estimated density derivative values.

Note

This function are available in other packages such as KernSmooth, sm, np, GenKern and locfit if deriv.order=0, and in ks package for Gaussian kernel only if 0 \leq deriv.order \leq 10.

Author(s)

Arsalane Chouaib Guidoum <acguidoum@usthb.dz>

References


See Also

`plot.dkde`, see `density` in package "stats" if deriv.order = 0, and `kdde` in package `ks`.

Examples

```r
## EXAMPLE 1: Simple example of a Gaussian density derivative

x <- rnorm(100)
dkde(x, deriv.order=0)  ## KDE of f
dkde(x, deriv.order=1)  ## KDDE of d/dx f
dkde(x, deriv.order=2)  ## KDDE of d^2/dx^2 f
dkde(x, deriv.order=3)  ## KDDE of d^3/dx^3 f
dev.new()
par(mfrow=c(2,2))
plot(dkde(x, deriv.order=0))
plot(dkde(x, deriv.order=1))
plot(dkde(x, deriv.order=2))
plot(dkde(x, deriv.order=3))

## EXAMPLE 2: Bimodal Gaussian density derivative
## show the kernels in the dkde parametrization

fx <- function(x) .5 * dnorm(x,-1.5,0.5) + .5 * dnorm(x,1.5,0.5)
fx1 <- function(x) .5 *(-4*x-6)* dnorm(x,-1.5,0.5) + .5 *(-4*x+6)* dnorm(x,1.5,0.5)

## 'h = 0.3' ; 'Derivative order = 0'
kernels <- eval(formals(dkde.default)$kernel)
dev.new()
plot(dkde(bimodal, h=0.3), sub=paste("Derivative order = 0",";","Bandwidth = 0.3 ", "Bimodal Gaussian Density"), main = "Bimodal Gaussian Density")
for(i in 2:length(kernels))
  lines(dkde(bimodal, h = 0.3, kernel = kernels[i]), col = i)
curve(fx, add=TRUE, lty=8)
legend("topright", legend = c(TRUE,kernels), col = c("black",seq(kernels)), lty = c(8,rep(1,length(kernels))), cex=0.7, inset = .015)

## 'h = 0.6' ; 'Derivative order = 1'
kernels <- eval(formals(dkde.default)$kernel)[-3]
dev.new()
plot(dkde(bimodal, deriv.order=1,h=0.6), main = "Bimodal Gaussian Density Derivative", sub=paste("Derivative order = 1",";","Bandwidth = 0.6"),ylim=c(-0.6,0.6))
for(i in 2:length(kernels))
  lines(dkde(bimodal,deriv.order=1, h = 0.6, kernel = kernels[i]), col = i)
curve(fx1, add=TRUE, lty=8)
legend("topright", legend = c(TRUE,kernels), col = c("black",seq(kernels)), lty = c(8,rep(1,length(kernels))), cex=0.7, inset = .015)
```
**h.amise**

**AMISE for Optimal Bandwidth Selectors**

**Description**

The (S3) generic function `h.amise` evaluates the asymptotic mean integrated squared error AMISE for optimal smoothing parameters \( h \) of \( r \)’th derivative of kernel density estimator one-dimensional.

**Usage**

```
h.amise(x, ...)  
```

# Default S3 method:
```
h.amise(x, deriv.order = 0, lower = 0.1 * hosL, upper = 2 * hosL,  
tol = 0.1 * lowerL, kernel = c("gaussian", "epanechnikov", "triweight",  
"tricube", "biweight", "cosine"), ...)  
```

**Arguments**

- **x**: vector of data values.
- **deriv.order**: derivative order (scalar).
- **lower, upper**: range over which to minimize. The default is almost always satisfactory. hosL (Over-smoothing) is calculated internally from an kernel, see details.
- **tol**: the convergence tolerance for `optimize`.
- **kernel**: a character string giving the smoothing kernel to be used, with default "gaussian".
- **...**: further arguments for (non-default) methods.

**Details**

`h.amise` asymptotic mean integrated squared error implements for choosing the optimal bandwidth \( h \) of a \( r \)’th derivative kernel density estimator.

We consider the following AMISE version of the \( r \)’th derivative of \( f \) the \( r \)’th derivative of the kernel estimate (see Scott 1992, pp 131):

\[
AMISE(h; r) = \frac{R(K^{(r)})}{nh^{2r+1}} + \frac{1}{4} h^4 \mu_2^2(K) R \left( f^{(r+2)} \right)
\]

The optimal bandwidth minimizing this function is:

\[
h^*_r = \left[ \frac{(2r + 1)R(K^{(r)})}{\mu_2^2(K) R \left( f^{(r+2)} \right)} \right]^{1/(2r+5)} n^{-1/(2r+5)}
\]

whereof

\[
\inf_{h>0} AMISE(h; r) = \frac{2r + 5}{4} R \left( K^{(r)} \right)^{\frac{4}{2r+5}} \left[ \frac{\mu_2^2(K) R \left( f^{(r+2)} \right)}{2r + 1} \right]^{\frac{2r+1}{2r+5}} n^{-\frac{4}{2r+5}}
\]
which is the smallest possible AMISE for estimation of \( f^{(r)}(x) \) using the kernel \( K(x) \), where 
\[
R(K^{(r)}) = \int_R K^{(r)}(x)^2 dx \quad \text{and} \quad \mu_2(K) = \int_R x^2 K(x) dx.
\]

The range over which to minimize is \( h \). Oversmoothing bandwidth, the default is almost always satisfactory. See George and Scott (1985), George (1990), Scott (1992, pp 165), Wand and Jones (1995, pp 61).

Value

- \( x \): data points - same as input.
- \( \text{data.name} \): the deparsed name of the \( x \) argument.
- \( n \): the sample size after elimination of missing values.
- \( \text{kernel} \): name of kernel to use.
- \( \text{deriv.order} \): the derivative order to use.
- \( h \): value of bandwidth parameter.
- \( \text{amise} \): the AMISE value.

Author(s)

Arsalane Chouaib Guidoum <acguidoum@usthb.dz>

References


See Also

- \texttt{plot.h.amise}, \texttt{see nmise} in package \texttt{sm} this function evaluates the mean integrated squared error of a density estimate (\texttt{deriv.order} = 0) which is constructed from data which follow a normal distribution.

Examples

```r
## Derivative order = 0

h.amise(kurtotic, deriv.order = 0)
```
## Description

The (S3) generic function `h.bcv` computes the biased cross-validation bandwidth selector of r'th derivative of kernel density estimator one-dimensional.

### Usage

```r
h.bcv(x, ...)  
## Default S3 method:  
h.bcv(x, whichbcv = 1, deriv.order = 0, lower = 0.1 * hos, upper = 2 * hos,  
tol = 0.1 * lower, kernel = c("gaussian","epanechnikov",  
"triweight","tricube","biweight","cosine"), ...)
```

### Arguments

- **x**: vector of data values.
- **whichbcv**: method selected, 1 = BCV1 or 2 = BCV2, see details.
- **deriv.order**: derivative order (scalar).
- **lower, upper**: range over which to minimize. The default is almost always satisfactory. hos (Over-smoothing) is calculated internally from an kernel, see details.
- **tol**: the convergence tolerance for `optimize`.
- **kernel**: a character string giving the smoothing kernel to be used, with default "gaussian".
- **...**: further arguments for (non-default) methods.

### Details

`h.bcv` biased cross-validation implements for choosing the bandwidth \( h \) of a r'th derivative kernel density estimator. if whichbcv = 1 then BCV1 is selected (Scott and George 1987), and if whichbcv = 2 used BCV2 (Jones and Kappenman 1991).

Scott and George (1987) suggest a method which has as its immediate target the AMISE (e.g. Silverman 1986, section 3.3). We denote \( \hat{\theta}_r(h) \) and \( \bar{\theta}_r(h) \) (Peter and Marron 1987, Jones and Kappenman 1991) by:

\[
\hat{\theta}_r(h) = \frac{(-1)^r}{n(n-1)h^{2r+1}} \sum_{i=1}^{n} \sum_{j=1; j \neq i}^{n} K^{(r)}(x_j-x_i) \frac{K^{(r)}(X_j-X_i)}{h}
\]

and
\[
\hat{\theta}_r(h) = \frac{(-1)^r}{n(n-1)h^{2r+1}} \sum_{i=1}^{n} \sum_{j=1, j \neq i}^{n} K^{(2r)} \left( \frac{X_i - X_j}{h} \right)
\]

Scott and George (1987) proposed using \( \hat{\theta}_r(h) \) to estimate \( f^{(r)}(x) \). Thus, \( \hat{h}_{BCV1}^{(r)} \), say, is the \( h \) that minimises:

\[
BCV1(h; r) = \frac{R(K^{(r)})}{nh^{2r+1}} + \frac{1}{4} \mu_2(K) h^4 \hat{\theta}_{r+2}(h)
\]

and we define \( \hat{h}_{BCV2}^{(r)} \) as the minimiser of (Jones and Kappenman 1991):

\[
BCV2(h; r) = \frac{R(K^{(r)})}{nh^{2r+1}} + \frac{1}{4} \mu_2(K) h^4 \bar{\theta}_{r+2}(h)
\]

where \( K^{(r)} * K^{(r)}(x) \) is the convolution of the \( r \)th derivative kernel function \( K^{(r)}(x) \) (see \texttt{kernel.conv} and \texttt{kernel.fun}); \( R(K^{(r)}) = \int_R K^{(r)}(x)^2 dx \) and \( \mu_2(K) = \int_R x^2 K(x) dx \).

The range over which to minimize is \( \hat{h} \) Oversmoothing bandwidth, the default is almost always satisfactory. See George and Scott (1985), George (1990), Scott (1992, pp 165), Wand and Jones (1995, pp 61).

**Value**

- \( x \): data points - same as input.
- \( \text{data.name} \): the deparsed name of the \( x \) argument.
- \( n \): the sample size after elimination of missing values.
- \( \text{kernel} \): name of kernel to use
- \( \text{deriv.order} \): the derivative order to use.
- \( \text{whichbcv} \): method selected.
- \( h \): value of bandwidth parameter.
- \( \text{min.bcv} \): the minimal BCV value.

**Author(s)**

Arsalane Chouaib Guidoum <acguidoum@usthb.dz>

**References**


**See Also**

plot.h.bcv, see bw.bcv in package "stats" and bcv in package MASS for Gaussian kernel only if deriv.order = 0, Hbcv for bivariate data in package ks for Gaussian kernel only if deriv.order = 0, kdeb in package locfit if deriv.order = 0.

**Examples**

```r
## EXAMPLE 1:

x <- rnorm(100)
h.bcv(x, whichbcv = 1, deriv.order = 0)
h.bcv(x, whichbcv = 2, deriv.order = 0)

## EXAMPLE 2:

## Derivative order = 0
h.bcv(kurtotic, deriv.order = 0)

## Derivative order = 1
h.bcv(kurtotic, deriv.order = 1)
```

---

**h.ccv**

*Complete Cross-Validation for Bandwidth Selection*

**Description**

The (S3) generic function h.ccv computes the complete cross-validation bandwidth selector of r'th derivative of kernel density estimator one-dimensional.

**Usage**

```r
h.ccv(x, ...)

## Default S3 method:

h.ccv(x, deriv.order = 0, lower = 0.1 * hos, upper = hos,
      tol = 0.1 * lower, kernel = c("gaussian", "triweight",
      "tricube", "biweight", "cosine"), ...)
```
 Arguments

- **x**: vector of data values.
- **deriv.order**: derivative order (scalar).
- **lower, upper**: range over which to minimize. The default is almost always satisfactory. *hos* (Over-smoothing) is calculated internally from a kernel, see details.
- **tol**: the convergence tolerance for *optimize*.
- **kernel**: a character string giving the smoothing kernel to be used, with default "gaussian".
- **...**: further arguments for (non-default) methods.

 Details

**h.ccv** complete cross-validation implements for choosing the bandwidth \( h \) of a \( r \)th derivative kernel density estimator.

Jones and Kappenman (1991) proposed a so-called complete cross-validation (CCV) in kernel density estimator. This method can be extended to the estimation of derivative of the density, basing our estimate of integrated squared density derivative (Peter and Marron 1987) on the \( \bar{\theta}_r(h) \)'s, we get the following, start from \( R \left( \hat{f}_h^{(r)} \right) - \bar{\theta}_r(h) \) as an estimate of MISE. Thus, \( \hat{h}_{CCV}^{(r)} \), say, is the \( h \) that minimises:

\[
CCV(h; r) = R \left( \hat{f}_h^{(r)} \right) - \bar{\theta}_r(h) + \frac{1}{2} \mu_2(K) h^2 \bar{\theta}_{r+1}(h) + \frac{1}{24} (6 \mu_2(K) - \delta(K)) h^4 \bar{\theta}_{r+2}(h)
\]

with

\[
R \left( \hat{f}_h^{(r)} \right) = \int \left( \hat{f}_h^{(r)}(x) \right)^2 dx = \frac{R(K^{(r)})}{nh^{2r+1}} + \frac{(-1)^r}{n(n-1)h^{2r+1}} \sum_{i=1}^n \sum_{j=1, j \neq i}^n K^{(r)}(x_i-x_j) \left( \frac{x_j - x_i}{h} \right)
\]

and

\[
\bar{\theta}_r(h) = \frac{(-1)^r}{n(n-1)h^{2r+1}} \sum_{i=1}^n \sum_{j=1, j \neq i}^n K^{(2r)}(x_i-x_j) \left( \frac{x_j - x_i}{h} \right)
\]

and \( K^{(r)} \ast K^{(r)}(x) \) is the convolution of the \( r \)th derivative kernel function \( K^{(r)}(x) \) (see *kernel.conv* and *kernel.fun*); \( R \left( K^{(r)} \right) = \int_R K^{(r)}(x)^2 dx \) and \( \mu_2(K) = \int_R x^2 K(x) dx, \delta(K) = \int_R x^4 K(x) dx \).

The range over which to minimize is *hos* Oversmoothing bandwidth, the default is almost always satisfactory. See George and Scott (1985), George (1990), Scott (1992, pp 165), Wand and Jones (1995, pp 61).

 Value

- **x**: data points - same as input.
- **data.name**: the deparsed name of the \( x \) argument.
- **n**: the sample size after elimination of missing values.
- **kernel**: name of kernel to use
- **deriv.order**: the derivative order to use.
- **h**: value of bandwidth parameter.
- **min.ccv**: the minimal CCV value.
Modified Cross-Validation for Bandwidth Selection

Description

The (S3) generic function h.mcv computes the modified cross-validation bandwidth selector of r'th derivative of kernel density estimator one-dimensional.

Usage

h.mcv(x, ...)  
## Default S3 method:
h.mcv(x, deriv.order = 0, lower = 0.1 * hos, upper = 2 * hos,  
tol = 0.1 * lower, kernel = c("gaussian", "epanechnikov", "triweight",  
"tricube", "biweight", "cosine"), ...)

Arguments

- **x**: vector of data values.
- **deriv.order**: derivative order (scalar).
- **lower, upper**: range over which to minimize. The default is almost always satisfactory. hos (Over-smoothing) is calculated internally from an kernel, see details.
- **tol**: the convergence tolerance for optimize.

Examples

```r
## Derivative order = 0
h.mcv(kurtotic, deriv.order = 0)

## Derivative order = 1
h.mcv(kurtotic, deriv.order = 1)
```
**kernel**

a character string giving the smoothing kernel to be used, with default "gaussian".

... further arguments for (non-default) methods.

**Details**

`h.mcv` modified cross-validation implements for choosing the bandwidth \( h \) of a \( r \)’th derivative kernel density estimator.

Stute (1992) proposed a so-called modified cross-validation (MCV) in kernel density estimation. This method can be extended to the estimation of derivative of a density, the essential idea based on approximated the problematic term by the aid of the Hajek projection (see Stute 1992). The minimization criterion is defined by:

\[
MCV(h; r) = \frac{R(K^{(r)})}{nh^{2r+1}} + \frac{(-1)^r}{n(n-1)h^{2r+1}} \sum_{i=1}^{n} \sum_{j=1; j \neq i}^{n} \psi^{(r)} \left( \frac{X_j - X_i}{h} \right)
\]

with

\[
\psi^{(r)}(c) = \left( K^{(r)} \ast K^{(r)} - K^{(2r)} - \mu_2(K)^2 K^{(2r+2)} \right)(c)
\]

and \( K^{(r)} \ast K^{(r)}(x) \) is the convolution of the \( r \)’th derivative kernel function \( K^{(r)}(x) \) (see `kernel.conv` and `kernel.fun`); \( R(K^{(r)}) = \int_R K^{(r)}(x)^2 dx \) and \( \mu_2(K) = \int_R x^2 K(x) dx \).

The range over which to minimize is \( h_0 \) Oversmoothing bandwidth, the default is almost always satisfactory. See George and Scott (1985), George (1990), Scott (1992, pp 165), Wand and Jones (1995, pp 61).

**Value**

- \( x \) data points - same as input.
- `data.name` the deparsed name of the `x` argument.
- \( n \) the sample size after elimination of missing values.
- `kernel` name of kernel to use
- `deriv.order` the derivative order to use.
- \( h \) value of bandwidth parameter.
- `min.mcv` the minimal MCV value.

**Author(s)**

Arsalane Chouaib Guidoum <acguidoum@usthb.dz>

**References**


See Also

plot.h.mcv.

Examples

## Derivative order = 0

h.mcv(kurtotic, deriv.order = 0)

## Derivative order = 1

h.mcv(kurtotic, deriv.order = 1)

---

h.mlcv

Maximum-Likelihood Cross-validation for Bandwidth Selection

Description

The (S3) generic function `h.mlcv` computes the maximum likelihood cross-validation (Kullback-Leibler information) bandwidth selector of a one-dimensional kernel density estimate.

Usage

h.mlcv(x, ...)

## Default S3 method:

h.mlcv(x, lower = 0.1, upper = 5, tol = 0.1 * lower,  
    kernel = c("gaussian", "epanechnikov", "uniform", "triangular",  
    "triweight", "tricube", "biweight", "cosine"), ...)

Arguments

- `x` vector of data values.
- `lower`, `upper` range over which to maximize. The default is almost always satisfactory.
- `tol` the convergence tolerance for `optimize`.
- `kernel` a character string giving the smoothing kernel to be used, with default "gaussian".
- `...` further arguments for (non-default) methods.

Details

`h.mlcv` maximum-likelihood cross-validation implements for choosing the optimal bandwidth `h` of kernel density estimator.
This method was proposed by Habbema, Hermans, and Van den Broeck (1971) and by Duin (1976). The maximum-likelihood cross-validation (MLCV) function is defined by:

\[ MLCV(h) = n^{-1} \sum_{i=1}^{n} \log \left[ \hat{f}_{h,i}(x) \right] \]

the estimate \( \hat{f}_{h,i}(x) \) on the subset \( \{X_j\}_{j \neq i} \) denoting the leave-one-out estimator, can be written:

\[ \hat{f}_{h,i}(X_i) = \frac{1}{(n-1)h} \sum_{j \neq i} K \left( \frac{X_j - X_i}{h} \right) \]

Define that \( h_{mlcv} \) as good which approaches the finite maximum of \( MLCV(h) \):

\[ h_{mlcv} = \arg \max_h MLCV(h) = \arg \max_h \left( n^{-1} \sum_{i=1}^{n} \log \left[ \sum_{j \neq i} K \left( \frac{X_j - X_i}{h} \right) \right] - \log((n-1)h) \right) \]

Value

- \( x \) data points - same as input.
- \( \text{data.name} \) the deparsed name of the \( x \) argument.
- \( n \) the sample size after elimination of missing values.
- \( \text{kernel} \) name of kernel to use
- \( h \) value of bandwidth parameter.
- \( \text{mlcv} \) the maximal likelihood CV value.

Author(s)

Arsalane Chouaib Guidoum <acguidoum@usthb.dz>

References


See Also

- `plot.h.mlcv`, see `lcv` in package `locfit`.

Examples

- `h.mlcv(bimodal)`
- `h.mlcv(bimodal, kernel ="epanechnikov")`
The (S3) generic function `h.tcv` computes the trimmed cross-validation bandwidth selector of r’th derivative of kernel density estimator one-dimensional.

Usage

```r
h.tcv(x, ...)  
## Default S3 method:  
h.tcv(x, deriv.order = 0, lower = 0.1 * hos, upper = 2 * hos,  
tol = 0.1 * lower, kernel = c("gaussian", "epanechnikov", "uniform",  
"triangular", "triweight", "tricube", "biweight", "cosine"), ...)  
```

Arguments

- **x**: vector of data values.
- **deriv.order**: derivative order (scalar).
- **lower, upper**: range over which to minimize. The default is almost always satisfactory. hos (Over-smoothing) is calculated internally from an kernel, see details.
- **tol**: the convergence tolerance for `optimize`.
- **kernel**: a character string giving the smoothing kernel to be used, with default "gaussian".
- **...**: further arguments for (non-default) methods.

Details

`h.tcv` trimmed cross-validation implements for choosing the bandwidth `h` of a r’th derivative kernel density estimator.

Feluch and Koronacki (1992) proposed a so-called trimmed cross-validation (TCV) in kernel density estimator, a simple modification of the unbiased (least-squares) cross-validation criterion. We consider the following "trimmed" version of "unbiased", to be minimized with respect to `h`:

\[
\int \left( \hat{f}_h^{(r)}(x) \right)^2 - 2 \frac{(-1)^r}{n(n-1)h^{2r+1}} \sum_{i=1}^{n} \sum_{j=1,j\neq i} K^{(2r)} \left( \frac{X_j - X_i}{h} \right) \chi (|X_i - X_j| > c_n)  
\]

where \(\chi(.)\) denotes the indicator function and \(c_n\) is a sequence of positive constants, \(c_n/h^{2r+1} \to 0\) as \(n \to \infty\), and

\[
\int \left( \hat{f}_h^{(r)}(x) \right)^2 = \frac{R(K^{(r)})}{nh^{2r+1}} + \frac{(-1)^r}{n(n-1)h^{2r+1}} \sum_{i=1}^{n} \sum_{j=1,j\neq i} K^{(r)} * K^{(r)} \left( \frac{X_j - X_i}{h} \right)  
\]
the trimmed cross-validation function is defined by:

$$TCV(h; r) = \frac{R(K^{(r)})}{nh^{2r+1}} + \frac{(-1)^r}{n(n-1)h^{2r+1}} \sum_{i=1}^{n} \sum_{j=1; j \neq i}^{n} \phi^{(r)} \left( \frac{X_j - X_i}{h} \right)$$

whit

$$\phi^{(r)}(c) = \left( K^{(r)} * K^{(r)} - 2K^{(2r)} \chi \left( |c| > c_n/h^{2r+1} \right) \right)(c)$$

here we take $c_n = 1/n$, for assure the convergence. Where $K^{(r)} * K^{(r)}(x)$ is the convolution of the $r$'th derivative kernel function $K^{(r)}(x)$ (see kernel.conv and kernel.fun).

The range over which to minimize is hos Oversmoothing bandwidth, the default is almost always satisfactory. See George and Scott (1985), George (1990), Scott (1992, pp 165), Wand and Jones (1995, pp 61).

Value

- `x` data points - same as input.
- `data.name` the deparsed name of the `x` argument.
- `n` the sample size after elimination of missing values.
- `kernel` name of kernel to use
- `deriv.order` the derivative order to use.
- `h` value of bandwidth parameter.
- `min.tcv` the minimal TCV value.

Author(s)

Arsalane Chouaib Guidoum <acguidoum@usthb.dz>

References


See Also

- `plot.h.tcv`

Examples

```r
## Derivative order = 0
h.tcv(kurtotic, deriv.order = 0)

## Derivative order = 1
h.tcv(kurtotic, deriv.order = 1)
```
h.ucv

Unbiased (Least-Squares) Cross-Validation for Bandwidth Selection

Description

The (S3) generic function h.ucv computes the unbiased (least-squares) cross-validation bandwidth selector of r'th derivative of kernel density estimator one-dimensional.

Usage

h.ucv(x, ..., 
## Default S3 method:
  h.ucv(x, deriv.order = 0, lower = 0.1 * hos, upper = 2 * hos, 
  tol = 0.1 * lower, kernel = c("gaussian", "epanechnikov", "uniform", 
  "triangular", "triweight", "tricube", "biweight", "cosine"), ...)

Arguments

  x vector of data values.
  deriv.order derivative order (scalar).
  lower, upper range over which to minimize. The default is almost always satisfactory. hos 
  (Over-smoothing) is calculated internally from an kernel, see details.
  tol the convergence tolerance for optimize.
  kernel a character string giving the smoothing kernel to be used, with default "gaussian".
  ... further arguments for (non-default) methods.

Details

h.ucv unbiased (least-squares) cross-validation implements for choosing the bandwidth h of a r'th derivative kernel density estimator.

Rudemo (1982) and Bowman (1984) proposed a so-called unbiased (least-squares) cross-validation (UCV) in kernel density estimator. An adaptation of unbiased cross-validation is proposed by Wolfgang et al. (1990) for bandwidth choice in the r'th derivative of kernel density estimator. The essential idea of this methods, for the estimation of f(r)(x) (r is derivative order), is to use the bandwidth h which minimizes the function:

\[ UCV(h; r) = \int \left( \frac{\hat{f}(r)(x)}{\hat{h}(r)} \right)^2 - 2n^{-1}(-1)^r \sum_{i=1}^n \hat{f}^{(2r)}(X_i) \]

The bandwidth minimizing this function is:

\[ \hat{h}^{(r)}_{ucv} = \arg \min_{h^{(r)}} UCV(h; r) \]
for \( r = 0, 1, 2, \ldots \)

where

\[
\int \left( \hat{f}_h^{(r)}(x) \right)^2 = \frac{R(K^{(r)})}{nh^{2(r+1)}} + \frac{(-1)^r}{n(n-1)h^{2r+1}} \sum_{i=1}^{n} \sum_{j=1,j \neq i}^{n} K^{(r)}*K^{(r)} \left( \frac{X_j - X_i}{h} \right)
\]

and \( K^{(r)}*K^{(r)}(x) \) is the convolution of the \( r \) th derivative kernel function \( K^{(r)}(x) \) (see kernel.conv and kernel.fun).

The estimate \( \hat{f}_h^{(2r)}(x) \) on the subset \( \{X_j\}_{j \neq i} \) denoting the leave-one-out estimator, can be written:

\[
\hat{f}_h^{(2r)}(X_i) = \frac{1}{n(n-1)h^{2r+1}} \sum_{j \neq i} K^{(2r)} \left( \frac{X_j - X_i}{h} \right)
\]

The function \( UCV(h;r) \) is unbiased cross-validation in the sense that \( E[UCV] = MISE[\hat{f}_h^{(r)}(x)] - R(f^{(r)}(x)) \) (see, Scott and George 1987). Can be simplified to give the computationally:

\[
UCV(h;r) = \frac{R(K^{(r)})}{nh^{2(r+1)}} + \frac{(-1)^r}{n(n-1)h^{2r+1}} \sum_{i=1}^{n} \sum_{j=1,j \neq i}^{n} \left( K^{(r)}*K^{(r)} - 2K^{(2r)} \right) \left( \frac{X_j - X_i}{h} \right)
\]

where \( R(K^{(r)}) = \int_{\mathbb{R}} K^{(r)}(x)^2 dx \).

The range over which to minimize is \( Hos \) Oversmoothing bandwidth, the default is almost always satisfactory. See George and Scott (1985), George (1990), Scott (1992, pp 165), Wand and Jones (1995, pp 61).

Value

- \( x \) data points - same as input.
- data.name the deparsed name of the x argument.
- \( n \) the sample size after elimination of missing values.
- kernel name of kernel to use
- deriv.order the derivative order to use.
- \( h \) value of bandwidth parameter.
- min.ucv the minimal UCV value.

Author(s)

Arsalane Chouaib Guidoum <acguidoum@usthb.dz>

References


**See Also**

`plot.h.ucv`, see `bw.ucv` in package "stats" and `ucv` in package MASS for Gaussian kernel only if `deriv.order = 0`, `hlscv` in package ks for Gaussian kernel only if `0 <= deriv.order <= 5`, `kdeb` in package locfit if `deriv.order = 0`.

**Examples**

```r
## Derivative order = 0
h.ucv(kurtotic, deriv.order = 0)

## Derivative order = 1
h.ucv(kurtotic, deriv.order = 1)
```

The (S3) generic function `kernel.conv` computes the convolution of r’th derivative for kernel function.
Usage

kernel.conv(x, ...)

## Default S3 method:
kernel.conv(x = NULL, deriv.order = 0, kernel = c("gaussian", "epanechnikov",
"uniform", "triangular", "triweight", "tricube",
"biweight", "cosine", "silverman"), ...)

Arguments

- x: points at which the convolution of kernel derivative is to be evaluated.
- deriv.order: derivative order (scalar).
- kernel: a character string giving the smoothing kernel to be used, with default "gaussian".
- ...: further arguments for (non-default) methods.

Details

The convolution of r'th derivative for kernel function is written \( K^{(r)} \ast K^{(r)} \). It is defined as the integral of the product of the derivative for kernel. As such, it is a particular kind of integral transform:

\[
K^{(r)} \ast K^{(r)}(x) = \int_{-\infty}^{+\infty} K^{(r)}(y)K^{(r)}(x-y)dy
\]

where:

\[
K^{(r)}(x) = \frac{d^r}{dx^r}K(x)
\]

for \( r = 0, 1, 2, \ldots \)

Value

- kernel: name of kernel to use.
- deriv.order: the derivative order to use.
- x: the n coordinates of the points where the convolution of kernel derivative is evaluated.
- kx: the convolution of kernel derivative values.

Author(s)

Arsalane Chouaib Guidoum <acguidoum@usthb.dz>

References


See Also

`plot.kernel.conv, kernapply` in package "stats" for computes the convolution between an input sequence, and `convolve` use the Fast Fourier Transform (`fft`) to compute the several kinds of convolutions of two sequences.

Examples

```r
kernels <- eval(formals(kernel.conv.default)$kernel)
kernels

## gaussian
kernel.conv(x = 0, kernel=kernels[1], deriv.order=0)
kernel.conv(x = 0, kernel=kernels[1], deriv.order=1)

## silverman
kernel.conv(x = 0, kernel=kernels[9], deriv.order=0)
kernel.conv(x = 0, kernel=kernels[9], deriv.order=1)
```

---

**kernel.fun**

**Derivatives of Kernel Function**

**Description**

The (S3) generic function `kernel.fun` computes the r'th derivative for kernel density.

**Usage**

```r
kernel.fun(x, ...)  # Default S3 method:
```

```r
kernel.fun(x = NULL, deriv.order = 0, kernel = c("gaussian","epanechnikov",
       "uniform", "triangular", "triweight", "tricube",
       "biweight", "cosine", "silverman"), ...)
```

**Arguments**

- `x` points at which the derivative of kernel function is to be evaluated.
- `deriv.order` derivative order (scalar).
- `kernel` a character string giving the smoothing kernel to be used, with default "gaussian".
- `...` further arguments for (non-default) methods.
Details

We give a short survey of some kernels functions \( K(x; r) \); where \( r \) is derivative order,

- Gaussian: \( K(x; \infty) = \frac{1}{\sqrt{2\pi}} \exp \left( -\frac{x^2}{2} \right) 1_{[-\infty, +\infty]} \)
- Epanechnikov: \( K(x; 2) = \frac{3}{4} (1 - x^2) 1_{(|x| \leq 1)} \)
- uniform (rectangular): \( K(x; 0) = \frac{1}{2} 1_{(|x| \leq 1)} \)
- triangular: \( K(x; 1) = (1 - |x|) 1_{(|x| \leq 1)} \)
- triweight: \( K(x; 6) = \frac{35}{32} (1 - x^2)^3 1_{(|x| \leq 1)} \)
- tricube: \( K(x; 9) = \frac{70}{61} (1 - |x|^3)^3 1_{(|x| \leq 1)} \)
- biweight: \( K(x; 4) = \frac{15}{16} (1 - x^2)^2 1_{(|x| \leq 1)} \)
- cosine: \( K(x; \infty) = \frac{\pi}{4} \cos \left( \frac{\pi}{2} x \right) 1_{(|x| \leq 1)} \)
- Silverman: \( K(x; r \mod 8)^1 = \frac{1}{2} \exp \left( -\frac{|x|}{\sqrt{2}} \right) \sin \left( \frac{|x|}{\sqrt{2}} + \frac{\pi}{4} \right) 1_{[-\infty, +\infty]} \)

The \( r \)'th derivative for kernel function \( K(x) \) is written:

\[
K^{(r)}(x) = \frac{d^r}{dx^r} K(x)
\]

for \( r = 0, 1, 2, \ldots \)

The \( r \)'th derivative of the Gaussian kernel \( K(x) \) is given by:

\[
K^{(r)}(x) = (-1)^r H_r(x) K(x)
\]

where \( H_r(x) \) is the \( r \)'th Hermite polynomial. This polynomials are set of orthogonal polynomials, for more details see, hermite.h.polynomials in package orthopolynom.

Value

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kernel</td>
<td>name of kernel to use.</td>
</tr>
<tr>
<td>deriv.order</td>
<td>the derivative order to use.</td>
</tr>
<tr>
<td>x</td>
<td>the n coordinates of the points where the derivative of kernel function is evaluated.</td>
</tr>
<tr>
<td>kx</td>
<td>the kernel derivative values.</td>
</tr>
</tbody>
</table>

Author(s)

Arsalane Chouaib Guidoum <acguidoum@usthb.dz>

References


\(^1\%\% = \mod x y\)
See Also

plot.kernel.fun, deriv and D in package "stats" for symbolic and algorithmic derivatives of simple expressions.

Examples

```r
kernels <- eval(formals(kernel.fun.default)$kernel)
kernels

## gaussian
kernel.fun(x = 0, kernel=kernels[1], deriv.order=0)
kernel.fun(x = 0, kernel=kernels[1], deriv.order=1)

## silverman
kernel.fun(x = 0, kernel=kernels[9], deriv.order=0)
kernel.fun(x = 0, kernel=kernels[9], deriv.order=1)
```

plot.dkde

Plot for Kernel Density Derivative Estimate

Description

The `plot.dkde` function loops through calls to the `dkde` function. Plot for kernel density derivative estimate for 1-dimensional data.

Usage

```r
## S3 method for class 'dkde'
plot(x, fx = NULL, ...)
## S3 method for class 'dkde'
lines(x, ...)
```

Arguments

- `x` object of class dkde (output from `dkde`).
- `fx` add to graphics the true density derivative (class :function), to compare it by the density derivative to estimate.
- `...` other graphics parameters, see `par` in package "graphics".

Details

The 1-d plot is a standard plot of a 1-d curve. If `!is.null(fx)` then a true density derivative is added.

Value

Plot of 1-d kernel density derivative estimates are sent to graphics window.
Author(s)
Arsalane Chouaib Guidoum <acguidoum@usthb.dz>

See Also
dkde, plot.density in package "stats" if deriv.order = 0.

Examples
plot(dkde(kurtotic, deriv.order=0, kernel="gaussian"), sub="")
lines(dkde(kurtotic, deriv.order=0, kernel="biweight"), col="red")

plot.h.amise  Plot for Asymptotic Mean Integrated Squared Error

Description
The plot.h.amise function loops through calls to the h.amise function. Plot for asymptotic mean integrated squared error function for 1-dimensional data.

Usage

## S3 method for class 'h.amise'
plot(x, seq.bws=NULL, ...)
## S3 method for class 'h.amise'
lines(x, seq.bws=NULL, ...)

Arguments

x  object of class h.amise (output from h.amise).
seq.bws  the sequence of bandwidths in which to compute the AMISE function. By default, the procedure defines a sequence of 50 points, from 0.15*hos to 2*hos (Over-smoothing).
...  other graphics parameters, see par in package "graphics".

Value
Plot of 1-d AMISE function are sent to graphics window.

kernel  name of kernel to use.
deriv.order  the derivative order to use.
seq.bws  the sequence of bandwidths.
amise  the values of the AMISE function in the bandwidths grid.
plot.h.bcv

Author(s)
Arsalane Chouaib Guidoum <acguidoum@usthb.dz>

See Also
h.amise.

Examples
plot(h.amise(bimodal,deriv.order=0))

plot.h.bcv Plot for Biased Cross-Validation

Description
The plot.h.bcv function loops through calls to the h.bcv function. Plot for biased cross-validation function for 1-dimensional data.

Usage
## S3 method for class 'h.bcv'
plot(x, seq.bws=NULL, ...)
## S3 method for class 'h.bcv'
lines(x,seq.bws=NULL, ...)

Arguments
x
object of class h.bcv (output from h.bcv).

seq.bws
the sequence of bandwidths in which to compute the biased cross-validation function. By default, the procedure defines a sequence of 50 points, from 0.15*hos to 2*hos (Over-smoothing).

... other graphics parameters, see par in package "graphics".

Value
Plot of 1-d biased cross-validation function are sent to graphics window.

kernel
name of kernel to use.

deriv.order
the derivative order to use.

seq.bws
the sequence of bandwidths.

bcv
the values of the biased cross-validation function in the bandwidths grid.

Author(s)
Arsalane Chouaib Guidoum <acguidoum@usthb.dz>
See Also

h.bcv.

Examples

## EXAMPLE 1:

```r
plot(h.bcv(trimodal, whichbcv = 1, deriv.order = 0), main="", sub="")
lines(h.bcv(trimodal, whichbcv = 2, deriv.order = 0), col="red")
legend("topright", c("BCV1", "BCV2"), lty=1, col=c("black", "red"), inset = .015)
```

## EXAMPLE 2:

```r
plot(h.bcv(trimodal, whichbcv = 1, deriv.order = 1), main="", sub="")
lines(h.bcv(trimodal, whichbcv = 2, deriv.order = 1), col="red")
legend("topright", c("BCV1", "BCV2"), lty=1, col=c("black", "red"), inset = .015)
```

plot.h.ccv  

*Plot for Complete Cross-Validation*

Description

The `plot.h.ccv` function loops through calls to the `h.ccv` function. Plot for complete cross-validation function for 1-dimensional data.

Usage

```r
## S3 method for class 'h.ccv'
plot(x, seq.bws=NULL, ...)  
## S3 method for class 'h.ccv'
lines(x, seq.bws=NULL, ...)
```

Arguments

- **x**: object of class `h.ccv` (output from `h.ccv`).
- **seq.bws**: the sequence of bandwidths in which to compute the complete cross-validation function. By default, the procedure defines a sequence of 50 points, from \(0.15\) to \(2\) (Over-smoothing).
- **...**: other graphics parameters, see `par` in package "graphics".

Value

Plot of 1-d complete cross-validation function are sent to graphics window.

- **kernel**: name of kernel to use.
- **deriv.order**: the derivative order to use.
- **seq.bws**: the sequence of bandwidths.
- **ccv**: the values of the complete cross-validation function in the bandwidths grid.
Author(s)
Arsalane Chouaib Guidoum <acguidoum@usthb.dz>

See Also
h.ccv.

Examples
```r
par(mfrow=c(2,1))
plot(h.ccv(trimodal, deriv.order=0), main="")
plot(h.ccv(trimodal, deriv.order=1), main="")
```

Description
The `plot.h.mcv` function loops through calls to the `h.mcv` function. Plot for modified cross-validation function for 1-dimensional data.

Usage
```r
## S3 method for class 'h.mcv'
plot(x, seq.bws=NULL, ...)  
## S3 method for class 'h.mcv'
lines(x, seq.bws=NULL, ...)
```

Arguments
- **x** object of class `h.mcv` (output from `h.mcv`).
- **seq.bws** the sequence of bandwidths in which to compute the modified cross-validation function. By default, the procedure defines a sequence of 50 points, from 0.15*hos to 2*hos (Over-smoothing).
- **...** other graphics parameters, see `par` in package "graphics".

Value
Plot of 1-d modified cross-validation function are sent to graphics window.

- **kernel** name of kernel to use.
- **deriv.order** the derivative order to use.
- **seq.bws** the sequence of bandwidths.
- **mcv** the values of the modified cross-validation function in the bandwidths grid.
Author(s)
Arsalane Chouaib Guidoum <acguidoum@usthb.dz>

See Also
h.mcv.

Examples
par(mfrow=c(2,1))
plot(h.mcv(trimodal, deriv.order=0), main="")
plot(h.mcv(trimodal, deriv.order=1), main="")

plot.h.mlcv  Plot for Maximum-Likelihood Cross-validation

Description
The plot.h.mlcv function loops through calls to the h.mlcv function. Plot for maximum-likelihood cross-validation function for 1-dimensional data.

Usage
## S3 method for class 'h.mlcv'
plot(x, seq.bws=NULL, ...)
## S3 method for class 'h.mlcv'
lines(x, seq.bws=NULL, ...)

Arguments
x  object of class h.mlcv (output from h.mlcv).
seq.bws  the sequence of bandwidths in which to compute the maximum-likelihood cross-validation function. By default, the procedure defines a sequence of 50 points, from 0.15*hos to 2*hos (Over-smoothing).
...  other graphics parameters, see par in package "graphics".

Value
Plot of 1-d maximum-likelihood cross-validation function are sent to graphics window.

kernel  name of kernel to use.
seq.bws  the sequence of bandwidths.
mlcv  the values of the maximum-likelihood cross-validation function in the bandwidths grid.
Author(s)
Arsalane Chouaib Guidoum <acguidoum@usthb.dz>

See Also
h.mlcv.

Examples
plot(h.mlcv(bimodal))
plot.h.ucv

See Also

h.tcv.

Examples

par(mfrow=c(2,1))
plot(h.tcv(trimodal, deriv.order=0), main="")
plot(h.tcv(trimodal, deriv.order=1), seq.bws=seq(0.1,0.5, length.out=50), main="")

plot.h.ucv

Plot for Unbiased Cross-Validation

Description

The plot.h.ucv function loops through calls to the h.ucv function. Plot for unbiased cross-validation function for 1-dimensional data.

Usage

## S3 method for class 'h.ucv'
plot(x, seq.bws=NULL, ...)
## S3 method for class 'h.ucv'
lines(x, seq.bws=NULL, ...)

Arguments

x object of class h.ucv (output from h.ucv).
seq.bws the sequence of bandwidths in which to compute the unbiased cross-validation function. By default, the procedure defines a sequence of 50 points, from 0.15*hos to 2*hos (Over-smoothing).
... other graphics parameters, see par in package "graphics".

Value

Plot of 1-d unbiased cross-validation function are sent to graphics window.

kernel name of kernel to use.
deriv.order the derivative order to use.
seq.bws the sequence of bandwidths.
ucv the values of the unbiased cross-validation function in the bandwidths grid.

Author(s)

Arsalane Chouaib Guidoum <acguidoum@usthb.dz>
plot.kernel.conv

See Also

h.ucv.

Examples

```r
par(mfrow=c(2,1))
plot(h.ucv(trimodal,deriv.order=0),seq.bws=seq(0.06,0.2,length=50))
plot(h.ucv(trimodal,deriv.order=1),seq.bws=seq(0.06,0.2,length=50))
```

Description

The `plot.kernel.conv` function loops through calls to the `kernel.conv` function. Plot for convolutions of r'th derivative kernel function one-dimensional.

Usage

```r
## S3 method for class 'kernel.conv'
plot(x, ...)
```

Arguments

- `x` object of class `kernel.conv` (output from `kernel.conv`).
- `...` other graphics parameters, see `par` in package "graphics".

Value

Plot of 1-d for convolution of r'th derivative kernel function are sent to graphics window.

Author(s)

Arsalane Chouaib Guidoum <acguidoum@usthb.dz>

See Also

`kernel.conv`.

Examples

```r
## Gaussian kernel

dev.new()
par(mfrow=c(2,2))
plot(kernel.conv(kernel="gaussian",deriv.order=0))
plot(kernel.conv(kernel="gaussian",deriv.order=1))
plot(kernel.conv(kernel="gaussian",deriv.order=2))
```
plot(kernel.conv(kernel="gaussian", deriv.order=3))

## Silverman kernel

dev.new()
par(mfrow=c(2,2))
plot(kernel.conv(kernel="silverman", deriv.order=0))
plot(kernel.conv(kernel="silverman", deriv.order=1))
plot(kernel.conv(kernel="silverman", deriv.order=2))
plot(kernel.conv(kernel="silverman", deriv.order=3))

---

**Description**

The `plot.kernel.fun` function loops through calls to the `kernel.fun` function. Plot for r'th derivative kernel function one-dimensional.

**Usage**

```r
## S3 method for class 'kernel.fun'
plot(x, ...)
```

**Arguments**

- `x` object of class `kernel.fun` (output from `kernel.fun`).
- `...` other graphics parameters, see `par` in package "graphics".

**Value**

Plot of 1-d for r'th derivative kernel function are sent to graphics window.

**Author(s)**

Arsalane Chouaib Guidoum <acguidoum@usthb.dz>

**See Also**

`kernel.fun`

**Examples**

```r
## Gaussian kernel

dev.new()
par(mfrow=c(2,2))
plot(kernel.fun(kernel="gaussian", deriv.order=0))
plot(kernel.fun(kernel="gaussian", deriv.order=1))
```
plot.kernel.fun

plot(kernel.fun(kernel="gaussian", deriv.order=2))
plot(kernel.fun(kernel="gaussian", deriv.order=3))

## Silverman kernel

dev.new()
par(mfrow=c(2,2))
plot(kernel.fun(kernel="silverman", deriv.order=0))
plot(kernel.fun(kernel="silverman", deriv.order=1))
plot(kernel.fun(kernel="silverman", deriv.order=2))
plot(kernel.fun(kernel="silverman", deriv.order=3))
Index

*Topic bandwidth selection
  h.amise, 13
  h.bcv, 15
  h.ccv, 17
  h.mcv, 19
  h.mlcv, 21
  h.tcvc, 23
  h.ucv, 25

*Topic datasets
  Claw, Bimodal, Kurtotic, Outlier, Trimodal, 7

*Topic density derivative
dkde, 9

*Topic kernel
  kernel.conv, 27
  kernel.fun, 29

*Topic nonparametric
dkde, 9
  h.amise, 13
  h.bcv, 15
  h.ccv, 17
  h.mcv, 19
  h.mlcv, 21
  h.tcvc, 23
  h.ucv, 25
  kernel.conv, 27
  kernel.fun, 29

*Topic package
  kedd-package, 2

*Topic plot
  plot.dkde, 31
  plot.h.amise, 32
  plot.h.bcv, 33
  plot.h.ccv, 34
  plot.h.mcv, 35
  plot.h.mlcv, 36
  plot.h.tcvc, 37
  plot.h.ucv, 38
  plot.kernel.conv, 39

  plot.kernel.fun, 40

*Topic smooth
dkde, 9
  h.amise, 13
  h.bcv, 15
  h.ccv, 17
  h.mcv, 19
  h.mlcv, 21
  h.tcvc, 23
  h.ucv, 25

  bcv, 17
  bimodal(Claw, Bimodal, Kurtotic, Outlier, Trimodal), 7
  bw.bcv, 17
  bw.ucv, 27

  claw(Claw, Bimodal, Kurtotic, Outlier, Trimodal), 7
  Claw, Bimodal, Kurtotic, Outlier, Trimodal, 7
  convolve, 29

  D, 31
  density, 12
  deriv, 31
dkde, 3, 9, 31, 32

  fft, 29
  function, 31

  h.amise, 4, 13, 32, 33
  h.bcv, 4, 9, 15, 33, 34
  h.ccv, 4, 17, 34, 35
  h.mcv, 4, 19, 35, 36
  h.mlcv, 4, 21, 36, 37
  h.tcvc, 4, 23, 37, 38
  h.ucv, 4, 9, 10, 25, 38, 39
  Hbcv, 17
  hermite.h.polynomials, 30
  hlscv, 27
INDEX

kdde, 12
kdeb, 17, 27
kedd (kedd-package), 2
kedd-package, 2
kernapply, 29
kernel.conv, 3, 16, 18, 20, 24, 26, 27, 39
kernel.fun, 3, 16, 18, 20, 24, 26, 29, 40
kurtotic (Claw, Bimodal, Kurtotic, Outlier, Trimodal), 7
lcv, 22
lines.dkde, 3
lines.dkde (plot.dkde), 31
lines.h.amise, 4
lines.h.amise (plot.h.amise), 32
lines.h.bcv, 4
lines.h.bcv (plot.h.bcv), 33
lines.h.ccv, 4
lines.h.ccv (plot.h.ccv), 34
lines.h.mcv, 4
lines.h.mcv (plot.h.mcv), 35
lines.h.mlcv, 4
lines.h.mlcv (plot.h.mlcv), 36
lines.h.tcv, 4
lines.h.tcv (plot.h.tcv), 37
lines.h.ucv, 4
lines.h.ucv (plot.h.ucv), 38

nmise, 14

optimize, 13, 15, 18, 19, 21, 23, 25
outlier (Claw, Bimodal, Kurtotic, Outlier, Trimodal), 7
par, 31–40
plot.density, 32
plot.dkde, 3, 12, 31, 31
plot.h.amise, 4, 14, 32, 32
plot.h.bcv, 4, 17, 33, 33
plot.h.ccv, 4, 19, 34, 34
plot.h.mcv, 4, 21, 35, 35
plot.h.mlcv, 4, 22, 36, 36
plot.h.tcv, 4, 24, 37, 37
plot.h.ucv, 4, 27, 38, 38
plot.kernel.conv, 29, 39, 39
plot.kernel.fun, 31, 40, 40
print.dkde (dkde), 9
print.h.amise (h.amise), 13
print.h.bcv (h.bcv), 15
print.h.ccv (h.ccv), 17
print.h.mcv (h.mcv), 19
print.h.mlcv (h.mlcv), 21
print.h.tcv (h.tcv), 23
print.h.ucv (h.ucv), 25

rnormix, 8
trimodal (Claw, Bimodal, Kurtotic, Outlier, Trimodal), 7

ucv, 27