# Package 'kader'

October 4, 2017

**Type** Package

**Title** Kernel Adaptive Density Estimation and Regression

**Version** 0.0.8

**Date** 2017-10-04

**Maintainer** Gerrit Eichner <gerrit.eichner@math.uni-giessen.de>

**Description** Implementation of various kernel adaptive methods in nonparametric curve estimation like density estimation as introduced in Stute and Srihera (2011) <doi:10.1016/j.spl.2011.01.013> and Eichner and Stute (2013) <doi:10.1016/j.jspi.2012.03.011> for pointwise estimation, and like regression as described in Eichner and Stute (2012) <doi:10.1080/10485252.2012.760737>.

**Depends** R (>= 3.4.1)

**License** GPL-3

**Encoding** UTF-8

**LazyData** TRUE

**URL** http://github.com/GerritEichner/kader

**BugReports** http://github.com/GerritEichner/kader/issues

**RoxygenNote** 6.0.1

**Suggests** testthat

**Imports** grDevices, graphics, stats

**NeedsCompilation** no

**Author** Gerrit Eichner [aut, cre]

**Repository** CRAN

**Date/Publication** 2017-10-04 17:18:08 UTC

## R topics documented:

---

`adaptive_fnhat` *Specialized "Workhorse" Function for Kernel Adaptive Density Estimators*

---

## Description

Common specialized computational "workhorse" function to compute the kernel adaptive density estimators both in eq. (1.6) of Srihera & Stute (2011) and in eq. (4) of Eichner & Stute (2013) (together with several related quantities) with a $\sigma$ that minimizes the estimated MSE using an estimated $\theta$. This function is "specialized" in that it expects some pre-computed quantities (in addition to the point(s) at which the density is to be estimated, the data, etc.). In particular, the estimator of $\theta$ (which is typically the arithmetic mean of the data) is expected to be already "contained" in those pre-computed quantities, which increases the computational efficiency.

## Usage

```
adaptive_fnhat(x, data, K, h, sigma, Ai, Bj, fnx, ticker = FALSE,
  plot = FALSE, parlist = NULL, ...)
```

## Arguments

| | |
|---|---|
| `x` | Numeric vector $(x_1, \ldots, x_k)$ of location(s) at which the density estimate is to be computed. |
| `data` | Numeric vector $(X_1, \ldots, X_n)$ of the data from which the estimate is to be computed. |

| | |
|---|---|
| K | Kernel function with vectorized in- & output. |
| h | Numeric scalar, where (usually) $h = n^{-1/5}$. |
| sigma | Numeric vector $(\sigma_1, \ldots, \sigma_s)$ with $s \geq 1$. |
| Ai | Numeric matrix expecting in its i-th row $(x_i - X_1, \ldots, x_i - X_n)/h$, where (usually) $x_1, \ldots, x_k$ with $k =$ length(x) are the points at which the density is to be estimated for the data $X_1, \ldots, X_n$ with $h = n^{-1/5}$. |
| Bj | Numeric vector expecting $(-J(1/n), \ldots, -J(n/n))$ in case of the rank transformation method, but $(\hat{\theta} - X_1, \ldots, \hat{\theta} - X_n)$ in case of the non-robust Srihera-Stute-method. |
| fnx | Numeric vector expecting $(f_n(x_1), \ldots, f_n(x_k))$ with $f_n(x_i)$ the Parzen-Rosenblatt estimator at $x_i$, i.e., $f_n(x_i) =$ mean(K(Ai[i,]))/h where here typically h $= n^{-1/5}$. |
| ticker | Logical; determines if a 'ticker' documents the iteration progress through sigma. Defaults to FALSE. |
| plot | Logical or character or numeric and indicates if graphical output should be produced. Defaults to FALSE (i.e., no graphical output is produced). If it is a character string or a numeric value, graphical output will be written to numbered pdf-files (one for each element of x, in the current working directory) whose names start with the provided "value" after converting it into a character string followed by the index number of the pertaining x-element. (Parts of the graphical output are generated by minimize_MSEHat.) |
| parlist | A list of graphical parameters; affects only the pdf-files (if any are created at all). Default: NULL. |
| ... | Possible further arguments passed to minimize_MSEHat() (where they are currently ignored). |

## Details

The computational procedure in this function can be highly iterative because for each point in x (and hence for each row of matrix Ai) the MSE estimator is computed as a function of $\sigma$ on a (usually fine) $\sigma$-grid provided through sigma. This happens by repeated calls to bias_AND_scaledvar(). The minimization in $\sigma$ is then performed by minimize_MSEHat() using both a discrete grid-search and the numerical optimization routine implemented in base R's optimize(). Finally, compute_fnhat() yields the actual value of the density estimator for the adapted $\sigma$, i.e., for the MSE-estimator-minimizing $\sigma$. (If necessary the computation over the $\sigma$-grid is repeated after extending the range of the grid until the estimator functions for both bias and variance are *not constant* across the $\sigma$-grid.)

## Value

A list of as many lists as elements in x, each with components x, y, sigma.adap, msehat.min, discr.min.smaller, and sig.range.adj whose meanings are as follows:

| | |
|---|---|
| x | the n coordinates of the points where the density is estimated. |
| y | the estimate of the density value $f(x)$. |
| sigma.adap | Minimizer of MSE-estimator (from function minimize_MSEHat). |
| msehat.min | Minimum of MSE-estimator (from function minimize_MSEHat). |

| | |
|---|---|
| discr.min.smaller | TRUE iff the numerically found minimum was smaller than the discrete one (from function minimize |
| sig.range.adj | Number of adjustments of sigma-range. |

## References

Srihera & Stute (2011) and Eichner & Stute (2013): see kader.

## Examples

```
## Not run:
require(stats)

 # Kernel adaptive density estimators for simulated N(0,1)-data
 # computed on an x-grid using the rank transformation and the
 # non-robust method:
set.seed(2017);     n <- 100;     Xdata <- sort(rnorm(n))
x <- seq(-4, 4, by = 0.5);     Sigma <- seq(0.01, 10, length = 51)
h <- n^(-1/5)

x.X_h <- outer(x/h, Xdata/h, "-")
fnx <- rowMeans(dnorm(x.X_h)) / h   # Parzen-Rosenblatt estim. at
                                    # x_j, j = 1, ..., length(x).
 # non-robust method:
theta.X <- mean(Xdata) - Xdata
adaptive_fnhat(x = x, data = Xdata, K = dnorm, h = h, sigma = Sigma,
  Ai = x.X_h, Bj = theta.X, fnx = fnx, ticker = TRUE, plot = TRUE)

 # rank transformation-based method (requires sorted data):
negJ <- -J_admissible(1:n / n)   # rank trafo
adaptive_fnhat(x = x, data = Xdata, K = dnorm, h = h, sigma = Sigma,
  Ai = x.X_h, Bj = negJ, fnx = fnx, ticker = TRUE, plot = TRUE)

## End(Not run)
```

---

# bias_AND_scaledvar *Estimators of Bias and Scaled Variance*

---

## Description

"Workhorse" function for vectorized (in $\sigma$) computation of both the bias estimator and the scaled variance estimator of eq. (2.3) in Srihera & Stute (2011), and for the analogous computation of the bias and scaled variance estimator for the rank transformation method in the paragraph after eq. (6) in Eichner & Stute (2013).

## Usage

```
bias_AND_scaledvar(sigma, Ai, Bj, h, K, fnx, ticker = FALSE)
```

## Arguments

| | |
|---|---|
| sigma | Numeric vector $(\sigma_1, \ldots, \sigma_s)$ with $s \geq 1$. |
| Ai | Numeric vector expecting $(x_0 - X_1, \ldots, x_0 - X_n)/h$, where (usually) $x_0$ is the point at which the density is to be estimated for the data $X_1, \ldots, X_n$ with $h = n^{-1/5}$. |
| Bj | Numeric vector expecting $(-J(1/n), \ldots, -J(n/n))$ in case of the rank transformation method, but $(\hat{\theta} - X_1, \ldots, \hat{\theta} - X_n)$ in case of the non-robust Srihera-Stute-method. (Note that this the same as argument Bj of adaptive_fnhat!) |
| h | Numeric scalar, where (usually) $h = n^{-1/5}$. |
| K | Kernel function with vectorized in- & output. |
| fnx | $f_n(x_0) =$ mean(K(Ai))/h, where here typically $h = n^{-1/5}$. |
| ticker | Logical; determines if a 'ticker' documents the iteration progress through sigma. Defaults to FALSE. |

## Details

Pre-computed $f_n(x_0)$ is expected for efficiency reasons (and is currently prepared in function adaptive_fnhat).

## Value

A list with components BiasHat and VarHat.scaled, both numeric vectors of same length as sigma.

## References

Srihera & Stute (2011) and Eichner & Stute (2013): see kader.

## Examples

```
require(stats)

set.seed(2017);     n <- 100;     Xdata <- sort(rnorm(n))
x0 <- 1;      Sigma <- seq(0.01, 10, length = 21)

h <- n^(-1/5)
Ai <- (x0 - Xdata)/h
fnx0 <- mean(dnorm(Ai)) / h   # Parzen-Rosenblatt estimator at x0.

 # non-robust method:
Bj <- mean(Xdata) - Xdata
# # rank transformation-based method (requires sorted data):
# Bj <- -J_admissible(1:n / n)   # rank trafo

kader:::bias_AND_scaledvar(sigma = Sigma, Ai = Ai, Bj = Bj, h = h,
  K = dnorm, fnx = fnx0, ticker = TRUE)
```

`bias_ES2012` *Bias Estimator of Eichner & Stute (2012)*

## Description

Bias estimator $Bias_n(\sigma)$, vectorized in $\sigma$, on p. 2540 of Eichner & Stute (2012).

## Usage

```
bias_ES2012(sigma, h, xXh, thetaXh, K, mmDiff)
```

## Arguments

| | |
|---|---|
| `sigma` | Numeric vector $(\sigma_1, \ldots, \sigma_s)$ with $s \geq 1$ with values of the scale parameter $\sigma$. |
| `h` | Numeric scalar for bandwidth $h$ (as "contained" in `thetaXh` and `xXh`). |
| `xXh` | Numeric vector expecting the pre-computed h-scaled differences $(x - X_1)/h$, $\ldots$, $(x - X_n)/h$ where $x$ is the single (!) location for which the weights are to be computed, the $X_i$'s are the data values, and $h$ is the numeric bandwidth scalar. |
| `thetaXh` | Numeric vector expecting the pre-computed h-scaled differences $(\theta - X_1)/h$, $\ldots$, $(\theta - X_n)/h$ where $\theta$ is the numeric scalar location parameter, and the $X_i$'s and $h$ are as in `xXh`. |
| `K` | A kernel function (with vectorized in- & output) to be used for the estimator. |
| `mmDiff` | Numeric vector expecting the pre-computed differences $m_n(X_1) - m_n(x), \ldots, m_n(X_n) - m_n(x)$. |

## Details

The formula can also be found in eq. (15.21) of Eichner (2017). Pre-computed $(x - X_i)/h$, $(\theta - X_i)/h$, and $m_n(X_i) - m_n(x)$ are expected for efficiency reasons (and are currently prepared in function `kare`).

## Value

A numeric vector of the length of `sigma`.

## References

Eichner & Stute (2012) and Eichner (2017): see `kader`.

## See Also

`kare` which currently does the pre-computing.

## Examples

```
require(stats)

 # Regression function:
m <- function(x, x1 = 0, x2 = 8, a = 0.01, b = 0) {
 a * (x - x1) * (x - x2)^3 + b
}
 # Note: For a few details on m() see examples in ?nadwat.

n <- 100       # Sample size.
set.seed(42)   # To guarantee reproducibility.
X <- runif(n, min = -3, max = 15)      # X_1, ..., X_n   # Design.
Y <- m(X) + rnorm(length(X), sd = 5)   # Y_1, ..., Y_n   # Response.

h <- n^(-1/5)
Sigma <- seq(0.01, 10, length = 51)   # sigma-grid for minimization.
x0 <- 5   # Location at which the estimator of m should be computed.

 # m_n(x_0) and m_n(X_i) for i = 1, ..., n:
mn <- nadwat(x = c(x0, X), dataX = X, dataY = Y, K = dnorm, h = h)

 # Estimator of Bias_x0(sigma) on the sigma-grid:
(Bn <- bias_ES2012(sigma = Sigma, h = h, xXh = (x0 - X) / h,
  thetaXh = (mean(X) - X) / h, K = dnorm, mmDiff = mn[-1] - mn[1]))

## Not run:
 # Visualizing the estimator of Bias_n(sigma) at x on the sigma-grid:
plot(Sigma, Bn, type = "o", xlab = expression(sigma), ylab = "",
  main = bquote(widehat("Bias")[n](sigma)~~"at"~~x==.(x0)))

## End(Not run)
```

---

`compute_fnhat` *"Unified" Function for Kernel Adaptive Density Estimators*

---

## Description

"Unified" function to compute the kernel density estimator both of Srihera & Stute (2011) and of Eichner & Stute (2013).

## Usage

```
compute_fnhat(x, data, K, h, Bj, sigma)
```

## Arguments

| | |
|---|---|
| `x` | Numeric vector with the location(s) at which the density estimate is to be computed. |

| | |
|---|---|
| data | Numeric vector $(X_1, \ldots, X_n)$ of the data from which the estimate is to be computed. |
| K | A kernel function (with vectorized in- & output) to be used for the estimator. |
| h | Numeric scalar for bandwidth $h$. |
| Bj | Numeric vector expecting $(-J(1/n), \ldots, -J(n/n))$ as produced in fnhat_SS2011 in case of the rank transformation method (using an admissible rank transformation as implemented by J_admissible), but $(\hat{\theta} - X_1, \ldots, \hat{\theta} - X_n)$ as produced in fnhat_ES2013 in case of the non-robust method. |
| sigma | Numeric scalar for value of scale parameter $\sigma$. |

## Details

Implementation of both eq. (1.6) in Srihera & Stute (2011) for given and fixed scalars $\sigma$ and $\theta$, and eq. (4) in Eichner & Stute (2013) for a given and fixed scalar $\sigma$ and for a given and fixed rank transformation (and, of course, for fixed and given location(s) in $x$, data $(X_1, \ldots, X_n)$, a kernel function $K$ and a bandwidth $h$). The formulas that the computational version implemented here is based upon are given in eq. (15.3) and eq. (15.9), respectively, of Eichner (2017). This function rests on preparatory computations done in fnhat_SS2011 or fnhat_ES2013.

## Value

A numeric vector of the same length as x with the estimated density values from eq. (1.6) of Srihera & Stute (2011) or eq. (4) of Eichner & Stute (2013).

## Note

In case of the rank transformation method the data are expected to be sorted in increasing order.

## References

Srihera & Stute (2011), Eichner and Stute (2013), and Eichner (2017): see kader.

## Examples

```
require(stats)

 # The kernel density estimators for simulated N(0,1)-data and a single
 # sigma-value evaluated on a grid using the rank transformation and
 # the non-robust method:
set.seed(2017);     n <- 100;     Xdata <- rnorm(n)
xgrid <- seq(-4, 4, by = 0.1)
negJ <- -J_admissible(1:n / n)                # The rank trafo requires
compute_fnhat(x = xgrid, data = sort(Xdata),   # sorted data!
  K = dnorm, h = n^(-1/5), Bj = negJ, sigma = 1)

theta.X <- mean(Xdata) - Xdata    # non-robust method
compute_fnhat(x = xgrid, data = Xdata, K = dnorm, h = n^(-1/5),
  Bj = theta.X, sigma = 1)
```

## cuberoot *Cube-root that retains its argument's sign*

### Description

Computes $(x_1^{1/3}, \ldots, x_n^{1/3})$ with $x_i^{1/3}$ being negative if $x_i < 0$.

### Usage

```
cuberoot(x)
```

### Arguments

| | |
|---|---|
| x | Numeric vector. |

### Value

Vector of same length and mode as input.

### Examples

```
kader:::cuberoot(x = c(-27, -1, -0, 0, 1, 27))

curve(kader:::cuberoot(x), from = -27, to = 27)
```

## epanechnikov *Epanechnikov kernel*

### Description

Vectorized evaluation of the Epanechnikov kernel.

### Usage

```
epanechnikov(x)
```

### Arguments

| | |
|---|---|
| x | Numeric vector. |

### Value

A numeric vector of the Epanechnikov kernel evaluated at the values in x.

## Examples

```
kader:::epanechnikov(x = c(-sqrt(6:5), -2:2, sqrt(5:6)))

curve(kader:::epanechnikov(x), from = -sqrt(6), to = sqrt(6))
```

---

`fnhat_ES2013` *Robust Kernel Density Estimator of Eichner & Stute (2013)*

---

## Description

Implementation of eq. (4) in Eichner & Stute (2013) for a given and fixed scalar $\sigma$, for rank transformation function $J$ (and, of course, for fixed and given location(s) in $x$, data $(X_1, \ldots, X_n)$, a kernel function $K$, and a bandwidth $h$).

## Usage

```
fnhat_ES2013(x, data, K, h, ranktrafo, sigma)
```

## Arguments

| | |
|---|---|
| `x` | Numeric vector with the location(s) at which the density estimate is to be computed. |
| `data` | Numeric vector $(X_1, \ldots, X_n)$ of the data from which the estimate is to be computed. Missing values are not allowed and entail an error. |
| `K` | A kernel function to be used for the estimator. |
| `h` | Numeric scalar for bandwidth $h$. |
| `ranktrafo` | A function used for the rank transformation. |
| `sigma` | Numeric scalar for value of scale parameter $\sigma$. |

## Details

The formula upon which the computational version implemented here is based is given in eq. (15.9) of Eichner (2017). This function does mainly only a simple preparatory computation and then calls `compute_fnhat` which does the actual work.

## Value

An object with class "density" whose underlying structure is a list containing the following components (as described in `density`), so that the `print` and `plot` methods for `density`-objects are immediately available):

| | |
|---|---|
| `x` | the n coordinates of the points where the density is estimated. |
| `y` | the estimated density values from eq. (4) in Eichner & Stute (2013). |
| `bw` | the bandwidth used. |

| | |
|---|---|
| n | the sample size. (Recall: missing or infinite values are not allowed here.) |
| call | the call which produced the result. |
| data.name | the deparsed name of the x argument. |
| has.na | logical, for compatibility (always FALSE). |

Additionally:

| | |
|---|---|
| ranktrafo | as in Arguments. |
| sigma | as in Arguments. |

## References

Eichner & Stute (2013) and Eichner (2017): see kader.

## See Also

fnhat_SS2011.

## Examples

```
require(stats);   require(grDevices);   require(datasets)

 # Simulated N(0,1)-data and one sigma-value
set.seed(2016);     n <- 100;     d <- rnorm(n)
xgrid <- seq(-4, 4, by = 0.1)
(fit <- fnhat_ES2013(x = xgrid, data = d, K = dnorm, h = n^(-1/5),
  ranktrafo = J2, sigma = 1) )

plot(fit, ylim = range(0, dnorm(0), fit$y), col = "blue")
curve(dnorm, add = TRUE);   rug(d, col = "red")
legend("topleft", lty = 1, col = c("blue", "black", "red"),
  legend = expression(hat(f)[n], phi, "data"))


 # The same data, but several sigma-values
sigmas <- seq(1, 4, length = 4)
(fit <- lapply(sigmas, function(sig)
  fnhat_ES2013(x = xgrid, data = d, K = dnorm, h = n^(-1/5),
    ranktrafo = J2, sigma = sig)) )

ymat <- sapply(fit, "[[", "y")
matplot(x = xgrid, y = ymat, type = "l", lty = 1, col = 2 + seq(sigmas),
  ylim = range(0, dnorm(0), ymat), main = "", xlab = "", ylab = "Density")
curve(dnorm, add = TRUE);   rug(d, col = "red")
legend("topleft", lty = 1, col = c("black", "red", NA), bty = "n",
  legend = expression(phi, "data", hat(f)[n]~"in other colors"))


 # Old-Faithful-eruptions-data and several sigma-values
d <- faithful$eruptions;     n <- length(d);     er <- extendrange(d)
xgrid <- seq(er[1], er[2], by = 0.1);    sigmas <- seq(1, 4, length = 4)
(fit <- lapply(sigmas, function(sig)
```

```
   fnhat_ES2013(x = xgrid, data = d, K = dnorm, h = n^(-1/5),
     ranktrafo = J2, sigma = sig)) )

ymat <- sapply(fit, "[[", "y");     dfit <- density(d, bw = "sj")
plot(dfit, ylim = range(0, dfit$y, ymat), main = "", xlab = "")
rug(d, col = "red")
matlines(x = xgrid, y = ymat, lty = 1, col = 2 + seq(sigmas))
legend("top", lty = 1, col = c("black", "red", NA), bty = "n",
  legend = expression("R's est.", "data", hat(f)[n]~"in other colors"))
```

---

`fnhat_SS2011` *(Non-robust) Kernel Density Estimator of Srihera & Stute (2011)*

---

## Description

Implementation of eq. (1.6) in Srihera & Stute (2011) for given and fixed scalars $\sigma$ and $\theta$ (and, of course, for fixed and given location(s) in $x$, data $(X_1, \ldots, X_n)$, a kernel function $K$ and a bandwidth $h$).

## Usage

```
fnhat_SS2011(x, data, K, h, theta, sigma)
```

## Arguments

| | |
|---|---|
| `x` | Numeric vector with the location(s) at which the density estimate is to be computed. |
| `data` | Numeric vector $(X_1, \ldots, X_n)$ of the data from which the estimate is to be computed. Missing or infinite values are not allowed and entail an error. |
| `K` | A kernel function to be used for the estimator. |
| `h` | Numeric scalar for bandwidth $h$. |
| `theta` | Numeric scalar for value of location parameter $\theta$. |
| `sigma` | Numeric scalar for value of scale parameter $\sigma$. |

## Details

The formula upon which the computational version implemented here is based is given in eq. (15.3) of Eichner (2017). This function does mainly only a simple preparatory computation and then calls `compute_fnhat` which does the actual work.

## Value

An object with class "density" whose underlying structure is a list containing the following components (as described in `density`), so that the `print` and `plot` methods for `density`-objects are immediately available):

| | |
|---|---|
| x | the n coordinates of the points where the density is estimated. |
| y | the estimated density values from eq. (1.6) in Srihera & Stute (2011). |
| bw | the bandwidth used. |
| n | the sample size. (Recall: missing or infinite values are not allowed here.) |
| call | the call which produced the result. |
| data.name | the deparsed name of the x argument. |
| has.na | logical, for compatibility (always FALSE). |

Additionally:

| | |
|---|---|
| theta | as in Arguments. |
| sigma | as in Arguments. |

## References

Srihera & Stute (2011) and Eichner (2017): see kader.

## See Also

fnhat_ES2013.

## Examples

```
require(stats);   require(grDevices);    require(datasets)

 # Simulated N(0,1)-data and one sigma-value
set.seed(2017);     n <- 100;     d <- rnorm(n)
xgrid <- seq(-4, 4, by = 0.1)
(fit <- fnhat_SS2011(x = xgrid, data = d, K = dnorm, h = n^(-1/5),
  theta = mean(d), sigma = 1))

plot(fit, ylim = range(0, dnorm(0), fit$y), col = "blue")
curve(dnorm, add = TRUE);   rug(d, col = "red")
legend("topleft", lty = 1, col = c("blue", "black", "red"),
  legend = expression(tilde(f)[n], phi, "data"))

 # The same data, but several sigma-values
sigmas <- seq(1, 4, length = 4)
(fit <- lapply(sigmas, function(sig)
  fnhat_SS2011(x = xgrid, data = d, K = dnorm, h = n^(-1/5),
    theta = mean(d), sigma = sig)))

ymat <- sapply(fit, "[[", "y")
matplot(x = xgrid, y = ymat, type = "l", lty = 1, col = 3:6,
  ylim = range(0, dnorm(0), ymat), main = "", xlab = "", ylab = "Density")
curve(dnorm, add = TRUE);   rug(d, col = "red")
legend("topleft", lty = 1, col = c("black", "red", NA), bty = "n",
  legend = expression(phi, "data", tilde(f)[n]~"in other colors"))

 # Old-Faithful-eruptions-data and several sigma-values
d <- faithful$eruptions;     n <- length(d);     er <- extendrange(d)
```

```
xgrid <- seq(er[1], er[2], by = 0.1);    sigmas <- seq(1, 4, length = 4)
(fit <- lapply(sigmas, function(sig)
   fnhat_SS2011(x = xgrid, data = d, K = dnorm, h = n^(-1/5),
     theta = mean(d), sigma = sig)))

ymat <- sapply(fit, "[[", "y");     dfit <- density(d, bw = "sj")
plot(dfit, ylim = range(0, dfit$y, ymat), main = "", xlab = "")
rug(d, col = "red")
matlines(x = xgrid, y = ymat, lty = 1, col = 3:6)
legend("top", lty = 1, col = c("black", "red", NA), bty = "n",
  legend = expression("R's est.", "data", tilde(f)[n]~"in other colors"))
```

---

J1 *J1*

---

## Description

Eq. (15.16) in Eichner (2017) as a result of Cardano's formula.

## Usage

```
J1(u, cc = sqrt(5/3))
```

## Arguments

| | |
|---|---|
| u | Numeric vector. |
| cc | Numeric constant, defaults to $\sqrt{(5/3)}$. |

## Details

Using, for brevity's sake, $J_{1a}(u, c) := -q_c(u)$ and $J_{1b}(u, c) := J_{1a}(u, c)^2 + p_c^3$, the definition of $J_1$ reads:

$J_1(u, c) := [J_{1a}(u, c) + \sqrt{(J_{1b}(u, c))}]^{1/3} + [J_{1a}(u, c) - \sqrt{(J_{1b}(u, c))}]^{1/3}$.

For implementation details of $q_c(u)$ and $p_c$ see qc and pc, respectively.

For further mathematical details see Eichner (2017) and/or Eichner & Stute (2013).

## Value

Vector of same length and mode as u.

## Note

Eq. (15.16) in Eichner (2017), and hence $J_1(u, c)$, requires $c$ to be in $[\sqrt{(5/3)}, 3)$. If cc does not satisfy this requirement a warning (only) is issued.

## See Also

J_admissible.

## Examples

```
u <- seq(0, 1, by = 0.01)
c0 <- expression(sqrt(5/3))
c1 <- expression(sqrt(3) - 0.01)
cgrid <- c(1.35, seq(1.4, 1.7, by = 0.1))
cvals <- c(eval(c0), cgrid, eval(c1))

Y <- sapply(cvals, function(cc, u) J1(u, cc = cc), u = u)
cols <- rainbow(ncol(Y), end = 9/12)
matplot(u, Y, type = "l", lty = "solid", col = cols,
  ylab = expression(J[1](u, c)))
abline(h = 0)
legend("topleft", title = "c", legend = c(c0, cgrid, c1),
  lty = 1, col = cols, cex = 0.8)
```

---

J2 *J2*

---

## Description

Eq. (20) in Eichner (2017) (based on "Bronstein's formula for k = 3")

## Usage

```
J2(u, cc = sqrt(5))
```

## Arguments

| | |
|---|---|
| u | Numeric vector. |
| cc | Numeric constant, defaults to $\sqrt{5}$. |

## Details

$J_2(u, c) = 2\sqrt{(-p_c)} * sin(1/3 * arcsin(q_c(u)/(-p_c)^{3/2}))$

For implementation details of $q_c(u)$ and $p_c$ see qc and pc, respectively.

For further mathematical details see Eichner (2017) and/or Eichner & Stute (2013).

## Value

Vector of same length and mode as u.

## Note

Eq. (20) in Eichner (2017), and hence $J_2(u, c)$, requires $c$ to be in $(\sqrt{3}, \sqrt{5}]$. If cc does not satisfy this requirement (only) a warning is issued.

The default `cc = sqrt(5)` yields the optimal rank transformation.

## See Also

J_admissible.

## Examples

```
u <- seq(0, 1, by = 0.01)
c0 <- expression(sqrt(3) + 0.01)
c1 <- expression(sqrt(5))
cgrid <- seq(1.85, 2.15, by = 0.1)
cvals <- c(eval(c0), cgrid, eval(c1))

Y <- sapply(cvals, function(cc, u) J2(u, cc = cc), u = u)
cols <- rainbow(ncol(Y), end = 9/12)
matplot(u, Y, type = "l", lty = "solid", col = cols,
  ylab = expression(J[2](u, c)))
abline(h = 0)
legend("topleft", title = "c", legend = c(c0, cgrid, c1),
  lty = 1, col = cols, cex = 0.8)
```

---

# J_admissible *Admissible Rank Transformations of Eichner & Stute (2013)*

---

## Description

This is just a wrapper for the functions J1, J2, and $u- > \sqrt{3} * (2u - 1)$ which implement the admissible transformations for the three cases for $c$. For mathematical details see eq. (15.16) and (15.17) in Eichner (2017) and/or Eichner & Stute (2013).

## Usage

```
J_admissible(u, cc = sqrt(5))
```

## Arguments

| | |
|---|---|
| u | Numeric vector. |
| cc | Numeric constant, defaults to $\sqrt{5}$. |

## Details

Basically, for cc in $[\sqrt(5/3), \sqrt{5}]$:

J_admissible(u, cc) = J1(u, cc) if cc $< \sqrt{3}$,

J_admissible(u, cc) = J2(u, cc) if cc $> \sqrt{3}$, and

J_admissible(u, cc) = sqrt(3) * (2*u - 1) if cc $= \sqrt{3}$.

## Value

Vector of same length and mode as u.

## Note

The admissible rank transformations require $c$ to be in $[\sqrt{(5/3)}, \sqrt{5}]$. If cc does not satisfy this requirement a warning (only) is issued. The default `cc = sqrt(5)`, i.e., $c = \sqrt{5}$, yields the optimal rank transformation.

## See Also

J1 and J2.

## Examples

```
par(mfrow = c(1, 2), mar = c(3, 3, 0.5, 0.5), mgp = c(1.7, 0.7, 0))
example(J1)
example(J2)
```

---

kade *Kernel Adaptive Density Estimator*

---

## Description

Wrapper function which does some preparatory calculations and then calls the actual "workhorse" functions which do the main computations for kernel adaptive density estimation of Srihera & Stute (2011) or Eichner & Stute (2013). Finally, it structures and returns the obtained results. Summarizing information and technical details can be found in Eichner (2017).

## Usage

```
kade(x, data, kernel = c("gaussian", "epanechnikov", "rectangular"),
  method = c("both", "ranktrafo", "nonrobust"), Sigma = seq(0.01, 10, length
  = 51), h = NULL, theta = NULL, ranktrafo = J2, ticker = FALSE,
  plot = FALSE, parlist = NULL, ...)
```

## Arguments

| | |
|---|---|
| x | Vector of location(s) at which the density estimate is to be computed. |
| data | Vector $(X_1, \ldots, X_n)$ of the data from which the estimate is to be computed. NAs or infinite values are removed (and a warning is issued). |

| | |
|---|---|
| kernel | A character string naming the kernel to be used for the adaptive estimator. This must partially match one of "gaussian", "rectangular" or "epanechnikov", with default "gaussian", and may be abbreviated to a unique prefix. (Currently, this kernel is also used for the initial, non-adaptive Parzen-Rosenblatt estimator which enters into the estimators of bias and variance as described in the references.) |
| method | A character string naming the method to be used for the adaptive estimator. This must partially match one of "both", "ranktrafo" or "nonrobust", with default "both", and may be abbreviated to a unique prefix. |
| Sigma | Vector of value(s) of the scale parameter $\sigma$. If of length 1 no adaptation is performed. Otherwise considered as the initial grid over which the optimization of the adaptive method will be performed. Defaults to `seq(0.01, 10, length = 51)`. |
| h | Numeric scalar for bandwidth $h$. Defaults to NULL and is then internally set to $n^{-1/5}$. |
| theta | Numeric scalar for value of location parameter $\theta$. Defaults to NULL and is then internally set to the arithmetic mean of $x_1, \ldots, x_n$. |
| ranktrafo | Function used for the rank transformation. Defaults to `J2` (with its default `cc = sqrt(5)`). |
| ticker | Logical; determines if a 'ticker' documents the iteration progress through `Sigma`. Defaults to FALSE. |
| plot | Logical or character or numeric and indicates if graphical output should be produced. Defaults to FALSE (i.e., no graphical output is produced) and is passed to `adaptive_fnhat()` which does the actual work. For details on how it is processed see there. |
| parlist | A list of graphical parameters that is passed to `adaptive_fnhat()`; see there. Default: `NULL`. |
| ... | Further arguments possibly passed down. Currently ignored. |

## Value

In the case of only one method a data frame whose components have the following names and meanings:

| | |
|---|---|
| x | x_0. |
| y | Estimate of f(x_0). |
| sigma.adap | The found minimizer of the MSE-estimator, i.e., the adaptive smoothing parameter value. |
| msehat.min | The found minimum of the MSE-estimator. |
| discr.min.smaller | TRUE iff the numerically found minimum was smaller than the discrete one. |
| sig.range.adj | Number of adjustments of sigma-range. |

In the case of both methods a list of two data frames of the just described structure.

## References

Srihera & Stute (2011), Eichner & Stute (2013), and Eichner (2017): see `kader`.

## Examples

```
require(stats)

 # Generating N(0,1)-data
set.seed(2017);     n <- 80;     d <- rnorm(n)

 # Estimating f(x0) for one sigma-value
x0 <- 1
(fit <- kade(x = x0, data = d, method = "nonrobust", Sigma = 1))

 # Estimating f(x0) for sigma-grid
x0 <- 1
(fit <- kade(x = x0, data = d, method = "nonrobust",
  Sigma = seq(0.01, 10, length = 10), ticker = TRUE))

## Not run:
 # Estimating f(x0) for sigma-grid and Old-Faithful-eruptions-data
x0 <- 2
(fit <- kade(x = x0, data = faithful$eruptions, method = "nonrobust",
  Sigma = seq(0.01, 10, length = 51), ticker = TRUE, plot = TRUE))

## End(Not run)
```

---

kader *Kernel Adjusted Density Estimation and Regression*

---

## Description

Package of functions to compute kernel estimators for

- nonparametric density estimation using a data-adjusted kernel or an appropriate rank-transformation, and for
- nonparametric regression using a data-adjusted kernel.

## Details

The functions are based on the theory laid out in the following papers:


- Srihera, R., Stute, W. (2011): Kernel adjusted density estimation. Statistics and Probability Letters 81, 571 - 579, URL http://dx.doi.org/10.1016/j.spl.2011.01.013.
- Eichner, G., Stute, W. (2012): Kernel adjusted nonparametric regression. Journal of Statistical Planning and Inference 142, 2537 - 2544, URL http://dx.doi.org/10.1016/j.jspi.2012.03.011.
- Eichner, G., Stute, W. (2013): Rank Transformations in Kernel Density Estimation. Journal of Nonparametric Statistics 25(2), 427 - 445, URL http://dx.doi.org/10.1080/10485252.2012.760737.

A very brief summary of the three papers above and sort of a vignette is presented in Eichner, G. (2017): Kader - An R package for nonparametric kernel adjusted density estimation and regression. In: Ferger, D., et al. (eds.): From Statistics to Mathematical Finance, Festschrift in Honour of Winfried Stute. Springer International Publishing. To appear in Jan. 2018. DOI then(!) presumably: 10.1007/978-3-319-50986-0.

---

`kare` *Kernel Adaptive Regression Estimator*

---

## Description

Wrapper function which does some preparatory calculations and then calls the actual "workhorse" functions which do the main computations for kernel adaptive regression estimation of Eichner & Stute (2012). Finally, it structures and returns the obtained results. Summarizing information and technical details can be found in Eichner (2017).

## Usage

```
kare(x.points, data, kernel = c("gaussian", "epanechnikov", "rectangular"),
  Sigma = seq(0.01, 10, length = 51), h = NULL, theta = NULL)
```

## Arguments

| | |
|---|---|
| `x.points` | Vector of location(s) at which the regression estimate is to be computed. |
| `data` | Data frame or list with one component named `x` which contains the vector of regressor values $x_1, \ldots, x_n$ and one named `y` which holds the vector of pertaining response values $y_1, \ldots, y_n$ (in the corresponding order) of the data from which the estimate is to be computed at the values given in `x.points`. Pairs $(x_i, y_i)$ with `NA` or an infinite value in a least one of their elements are removed (and a warning is issued). |
| `kernel` | A character string naming the kernel to be used for the adaptive estimator. This must partially match one of "gaussian", "rectangular" or "epanechnikov", with default "gaussian", and may be abbreviated to a unique prefix. (Currently, this kernel is also used for the initial, non-adaptive Nadaraya-Watson regression estimator which enters into the estimators of bias and variance as described in the references.) |
| `Sigma` | Vector of value(s) of the scale parameter $\sigma$. If of length 1 no adaptation is performed. Otherwise considered as the grid over which the optimization of the adaptive method will be performed. Defaults to `seq(0.01, 10, length = 51)`. |
| `h` | Numeric scalar for bandwidth $h$. Defaults to NULL and is then internally set to $n^{-1/5}$. |
| `theta` | Numeric scalar for value of location parameter $\theta$. Defaults to NULL and is then internally set to the arithmetic mean of $x_1, \ldots, x_n$. |

## Value

If `length(x.points)` = 1, a list of eight components with the following names and meanings:

| | |
|---|---|
| x | Scalar $x$-value in x.points at which the regression estimator was computed. |
| y | Estimated scalar value of $m(x)$ at point in x.points. |
| sigma.adap | The found scalar minimizer of the MSE-estimator, i.e., the adaptive smoothing parameter value. |
| msehat.min | The found scalar minimum of the MSE-estimator. |
| Sigma | Vector with the $\sigma$-grid on which the minimization process was performed. |
| Bn | Vector with the estimator of bias on that $\sigma$-grid. |
| Vn2 | Ditto for the variance. |
| MSE | Ditto for the MSE. |

If length(x.points) > 1, a list with the same component names as above, but then

| | |
|---|---|
| x | Vector x.points with x-values at which the regression estimator was computed. |
| y | Vector of estimated values of $m(x)$ at the x-values in x.points. |
| sigma.adap | Vector of the found minimizers of the MSE-estimator, i.e., the adaptive smoothing parameter values. |
| msehat.min | Vector of the found minima of the MSE-estimator. |
| Sigma | Vector with the $\sigma$-grid on which the minimization process was performed. |
| Bn | (length(Sigma) by length(x.points))-matrix with the estimated values of the bias on the $\sigma$-grid in their co |
| Vn2 | Ditto for the variance. |
| MSE | Ditto for the MSE. |

## References

Eichner & Stute (2012) and Eichner (2017): see kader.

## Examples

```
require(stats)

 # Regression function:
m <- function(x, x1 = 0, x2 = 8, a = 0.01, b = 0) {
 a * (x - x1) * (x - x2)^3 + b
}
 # Note: For a few details on m() see examples in ?nadwat.

x0 <- 5   # The point x_0 at which the MSE-optimal kernel adjusted
 # nonparametric estimation of m should take place. (Recall: for m's
 # default values a minimum is at 2, a point of inflection at 4, and
 # a saddle point an 8; an "arbitrary" point would, e.g., be at 5.)

n <- 100   # Sample size.
sdeps <- 1   # Std. dev. of the \epsilon_i: \sqrt(Var(Y|X=x))
             # (here: constant in x).
design.ctr <- x0 + 0.5   # "centre" and "scale" of the design, i.e.,
design.scl <- 1  # in the normal scenario below, expected value and
                 # std. dev. of the distribution of the x_i's.

set.seed(42)   # To guarantee reproducibility.
x <- rnorm(n, mean = design.ctr, sd = design.scl)   # x_1, ..., x_n
```

```
Y <- m(x) + rnorm(length(x), sd = sdeps)            # Y_1, ..., Y_n
data <- data.frame(x = x, y = Y)

 # Computing the kernel adaptive regression estimator values
 #*********************************************************
x.points <- seq(-3.3 * design.scl, 3.3 * design.scl, length = 101) +
   design.ctr  # x-grid on which to draw and estimate the regr. fct. m.

Sigma <- seq(0.01, 10, length = 51)   # \sigma-grid for minimization.
fit <- kare(x.points = x0, data = data, Sigma = Sigma)

## Not run:
 # Grafical display for the current data set
 #******************************************
 # Storing the curent settings of the graphics device
 # and changing its layout for the three plots to come:
op <- par(mfrow = c(3, 1), mar = c(3, 3, 2, 0.1)+0.1,
   mgp = c(1.5, 0.5, 0), tcl = -0.3, cex.main = 2)

 # The scatter plot of the "raw data":
plot(y ~ x, data = data, xlim = range(data$x, x.points),
   ylim = range(data$y, fit$y, na.rm = TRUE),
   main = bquote(n == .(n)), xlab = "x", ylab = "y")

 # The "true" regression function m:
lines(x.points, m(x.points), lty = 2)

 # The MSE-optimal kernel adjusted nonparametric regression estimator
 # at x_0, i.e., the point (x_0, \hat m_n(x_0)):
points(fit$x, fit$y, col = "red", pch = 4, cex = 2)

 # The legend for the "true" regression function m and for the point
 # (x_0, \hat m_n(x_0)):
legend("topleft", lty = c(2, NA), pch = c(NA, 4),
 col = c("black", "red"), bty = "n", cex = 1.2,
 legend = c(as.expression(bquote(paste("m  with  ",
                                       sigma(paste(Y, "|", X == x))
                                  == .(sdeps)))),
            as.expression(bquote(paste(hat(m)[n](x[0]), "  at  ",
                                       x[0] == .(x0))))))

 # Visualizing the estimators of (Bias_n(sigma))^2 and
 # Var_n(sigma) at x0 on the sigma-grid:
with(fit,
  matplot(Sigma, cbind(Bn*Bn, Vn2), type = "l", lty = 1:2,
   col = c("black", "red"), xlab = expression(sigma), ylab = ""))

 # The legend for (Bias_n(sigma))^2 and Var_n(sigma):
legend("topleft", lty = 1:2, col = c("black", "red"), bty = "n",
  legend = c(expression(paste(widehat(plain(Bias))[n]^2, (sigma))),
             expression(widehat(plain(Var))[n](sigma))),
  cex = 1.2)
```

```
 # Visualizing the estimator of MSE_n(sigma) at x0 on the sigma-grid
 # together with the point indicating the detected minimum, and a legend:
plot(fit$Sigma, fit$MSE, type = "l",
 xlab = expression(sigma), ylab = "")
points(fit$sigma.adap, fit$msehat.min, pch = 4, col = "red", cex = 2)
legend("topleft", lty = c(1, NA), pch = c(NA, 4),
 col = c("black", "red"), bty = "n", cex = 1.2,
 legend = c(expression(widehat(plain(MSE))[n](sigma)),
            substitute(group("(", list(plain(Minimizer),
                                       plain(Minimum)), ")")
                       == group("(", list(x, y), ")"),
                       list(x = signif(fit$sigma.adap, 4),
                            y = signif(fit$msehat.min, 4)))))

par(op) # Restoring the previous settings of the graphics device.

## End(Not run)
```

---

kfn_vectorized *Convolution of Kernel Function K with fn*

---

## Description

Vectorized evaluation of the convolution of the kernel function K with fn.

## Usage

```
kfn_vectorized(u, K, xixj, h, sig)
```

## Arguments

| | |
|---|---|
| u | Numeric vector. |
| K | Kernel function with vectorized in- & output. |
| xixj | Numeric matrix. |
| h | Numeric scalar. |
| sig | Numeric scalar. |

## Details

Vectorized (in u) evaluation of - a more explicit representation of - the integrand $K(u) * f_n(\ldots - h^2/\sigma * u)$ which is used in the computation of the bias estimator before eq. (2.3) in Srihera & Stute (2011). Also used for the analogous computation of the respective bias estimator in the paragraph after eq. (6) in Eichner & Stute (2013).

## Value

A vector of $(K * f_n)(u)$ evaluated at the values in u.

## Note

An alternative implementation could be `K(u) * sapply(h/sig * u, function(v) mean(K(xixj - v))) / h`

## Examples

```
require(stats)

set.seed(2017);   n <- 100;   Xdata <- rnorm(n)
x0 <- 1;          sig <- 1;   h <- n^(-1/5)

Ai <- (x0 - Xdata)/h
Bj <- mean(Xdata) - Xdata   # in case of non-robust method
AiBj <- outer(Ai, Bj/sig, "+")

ugrid <- seq(-10, 10, by = 1)
kader:::kfn_vectorized(u = ugrid, K = dnorm, xixj = AiBj, h = h, sig = sig)
```

---

`minimize_MSEHat` *Minimization of Estimated MSE*

---

## Description

Minimization of the estimated MSE as function of $\sigma$ in four steps.

## Usage

```
minimize_MSEHat(VarHat.scaled, BiasHat.squared, sigma, Ai, Bj, h, K, fnx,
  ticker = FALSE, plot = FALSE, ...)
```

## Arguments

| | |
|---|---|
| `VarHat.scaled` | Vector of estimates of the scaled variance (for values of $\sigma$ in `sigma`). |
| `BiasHat.squared` | Vector of estimates of the squared bias (for values of $\sigma$ in `sigma`). |
| `sigma` | Numeric vector $(\sigma_1, \ldots, \sigma_s)$ with $s \geq 1$. |
| `Ai` | Numeric vector expecting $(x_0 - X_1, \ldots, x_0 - X_n)/h$, where (usually) $x_0$ is the point at which the density is to be estimated for the data $X_1, \ldots, X_n$ with $h = n^{-1/5}$. |
| `Bj` | Numeric vector expecting $(-J(1/n), \ldots, -J(n/n))$ in case of the rank transformation method, but $(\hat{\theta} - X_1, \ldots, \hat{\theta} - X_n)$ in case of the non-robust Srihera-Stute-method. (Note that this the same as argument `Bj` of `adaptive_fnhat`!) |
| `h` | Numeric scalar, where (usually) $h = n^{-1/5}$. |
| `K` | Kernel function with vectorized in- & output. |
| `fnx` | $f_n(x_0) =$ `mean(K(Ai))/h`, where here typically $h = n^{-1/5}$. |

| | |
|---|---|
| ticker | Logical; determines if a 'ticker' documents the iteration progress through sigma. Defaults to FALSE. |
| plot | Should graphical output be produced? Defaults to FALSE. |
| ... | Currently ignored. |

## Details

Step 1: determine first (= smallest) maximizer of VarHat.scaled (!) on the grid in sigma. Step 2: determine first (= smallest) minimizer of estimated MSE on the $\sigma$-grid LEFT OF the first maximizer of VarHat.scaled. Step 3: determine a range around the yet-found (discrete) minimizer of estimated MSE within which a finer search for the "true" minimum is continued using numerical minimization. Step 4: check if the numerically determined minimum is indeed better, i.e., smaller than the discrete one; if not keep the first.

## Value

A list with components sigma.adap, msehat.min and discr.min.smaller whose meanings are as follows:

| | |
|---|---|
| sigma.adap | Found minimizer of MSE estimator. |
| msehat.min | Found minimum of MSE estimator. |
| discr.min.smaller | TRUE iff the numerically found minimum was smaller than the discrete one. |

## Examples

```
require(stats)

set.seed(2017);     n <- 100;     Xdata <- sort(rnorm(n))
x0 <- 1;      Sigma <- seq(0.01, 10, length = 11)

h <- n^(-1/5)
Ai <- (x0 - Xdata)/h
fnx0 <- mean(dnorm(Ai)) / h   # Parzen-Rosenblatt estimator at x0.

 # For non-robust method:
Bj <- mean(Xdata) - Xdata
#  # For rank transformation-based method (requires sorted data):
# Bj <- -J_admissible(1:n / n)   # rank trafo

BV <- kader:::bias_AND_scaledvar(sigma = Sigma, Ai = Ai, Bj = Bj,
  h = h, K = dnorm, fnx = fnx0, ticker = TRUE)

kader:::minimize_MSEHat(VarHat.scaled = BV$VarHat.scaled,
  BiasHat.squared = (BV$BiasHat)^2, sigma = Sigma, Ai = Ai, Bj = Bj,
  h = h, K = dnorm, fnx = fnx0, ticker = TRUE, plot = FALSE)
```

`mse_hat` *MSE Estimator*

## Description

Vectorized (in $\sigma$) function of the MSE estimator in eq. (2.3) of Srihera & Stute (2011), and of the analogous estimator in the paragraph after eq. (6) in Eichner & Stute (2013).

## Usage

```
mse_hat(sigma, Ai, Bj, h, K, fnx, ticker = FALSE)
```

## Arguments

| | |
|---|---|
| `sigma` | Numeric vector $(\sigma_1, \ldots, \sigma_s)$ with $s \geq 1$. |
| `Ai` | Numeric vector expecting $(x_0 - X_1, \ldots, x_0 - X_n)/h$, where (usually) $x_0$ is the point at which the density is to be estimated for the data $X_1, \ldots, X_n$ with $h = n^{-1/5}$. |
| `Bj` | Numeric vector expecting $(-J(1/n), \ldots, -J(n/n))$ in case of the rank transformation method, but $(\hat{\theta} - X_1, \ldots, \hat{\theta} - X_n)$ in case of the non-robust Srihera-Stute-method. (Note that this the same as argument `Bj` of `adaptive_fnhat`!) |
| `h` | Numeric scalar, where (usually) $h = n^{-1/5}$. |
| `K` | Kernel function with vectorized in- & output. |
| `fnx` | $f_n(x_0) =$ `mean(K(Ai))/h`, where here typically $h = n^{-1/5}$. |
| `ticker` | Logical; determines if a 'ticker' documents the iteration progress through `sigma`. Defaults to FALSE. |

## Value

A vector with corresponding MSE values for the values in `sigma`.

## See Also

For details see `bias_AND_scaledvar`.

## Examples

```
require(stats)

set.seed(2017);     n <- 100;     Xdata <- sort(rnorm(n))
x0 <- 1;      Sigma <- seq(0.01, 10, length = 11)

h <- n^(-1/5)
Ai <- (x0 - Xdata)/h
fnx0 <- mean(dnorm(Ai)) / h   # Parzen-Rosenblatt estimator at x0.
```

```
 # non-robust method:
theta.X <- mean(Xdata) - Xdata
kader:::mse_hat(sigma = Sigma, Ai = Ai, Bj = theta.X,
  h = h, K = dnorm, fnx = fnx0, ticker = TRUE)

 # rank transformation-based method (requires sorted data):
negJ <- -J_admissible(1:n / n)   # rank trafo
kader:::mse_hat(sigma = Sigma, Ai = Ai, Bj = negJ,
  h = h, K = dnorm, fnx = fnx0, ticker = TRUE)
```

---

nadwat *The Classical Nadaraya-Watson Regression Estimator*

---

## Description

In its arguments x and dataX vectorized function to compute the classical Nadaraya-Watson estimator (as it is $m_n$ in eq. (1.1) in Eichner & Stute (2012)).

## Usage

```
nadwat(x, dataX, dataY, K, h)
```

## Arguments

| | |
|---|---|
| x | Numeric vector with the location(s) at which the Nadaraya-Watson regression estimator is to be computed. |
| dataX | Numeric vector $(X_1, \ldots, X_n)$ of the x-values from which (together with the pertaining y-values) the estimate is to be computed. |
| dataY | Numeric vector $(Y_1, \ldots, Y_n)$ of the y-values from which (together with the pertaining x-values) the estimate is to be computed. |
| K | A kernel function (with vectorized in- & output) to be used for the estimator. |
| h | Numeric scalar for bandwidth $h$. |

## Details

Implementation of the classical Nadaraya-Watson estimator as in eq. (1.1) in Eichner & Stute (2012) at given location(s) in x for data $(X_1, Y_1), \ldots, (X_n, Y_n)$, a kernel function $K$ and a bandwidth $h$.

## Value

A numeric vector of the same length as x.

## Examples

```
require(stats)

 # Regression function: a polynomial of degree 4 with one maximum (or
 # minimum), one point of inflection, and one saddle point.
 # Memo: for p(x) = a * (x - x1) * (x - x2)^3 + b the max. (or min.)
 # is at x = (3*x1 + x2)/4, the point of inflection is at x =
 # (x1 + x2)/2, and the saddle point at x = x2.
m <- function(x, x1 = 0, x2 = 8, a = 0.01, b = 0) {
 a * (x - x1) * (x - x2)^3 + b
 }
 # Note: for m()'s default values a minimum is at x = 2, a point
 # of inflection at x = 4, and a saddle point at x = 8.

n <- 100       # Sample size.
set.seed(42)   # To guarantee reproducibility.
X <- runif(n, min = -3, max = 15)      # X_1, ..., X_n
Y <- m(X) + rnorm(length(X), sd = 5)   # Y_1, ..., Y_n

x <- seq(-3, 15, length = 51)   # Where the Nadaraya-Watson estimator
                                # mn of m shall be computed.
mn <- nadwat(x = x, dataX = X, dataY = Y, K = dnorm, h = n^(-1/5))

plot(x = X, y = Y);   rug(X)
lines(x = x, y = mn, col = "blue")  # The estimator.
curve(m, add = TRUE, col = "red")   # The "truth".
```

---

pc *pc*

---

## Description

Coefficient $p_c$ of eq. (15.15) in Eichner (2017).

## Usage

```
pc(cc)
```

## Arguments

| | |
|---|---|
| cc | Numeric vector. |

## Details

p_c = 1/5 * (3c^2 - 5) / (3 - c^2) * c^2.

For further details see p. 297 f. in Eichner (2017) and/or Eichner & Stute (2013).

## Value

Vector of same length and mode as cc.

## Note

$p_c$ should be undefined for $c = \sqrt{3}$, but pc is here implemented to return Inf in each element of its return vector for which the corresponding element in cc contains R's value of sqrt(3).

## Examples

```
c0 <- expression(sqrt(5/3))
c1 <- expression(sqrt(3) - 0.01)
cgrid <- seq(1.325, 1.7, by = 0.025)
cvals <- c(eval(c0), cgrid, eval(c1))

plot(cvals, pc(cvals), xaxt = "n", xlab = "c", ylab = expression(p[c]))
axis(1, at = cvals, labels = c(c0, cgrid, c1), las = 2)
```

---

# qc *qc*

---

## Description

Coefficient $q_c(u)$ of eq. (15.15) in Eichner (2017).

## Usage

```
qc(u, cc = sqrt(5/3))
```

## Arguments

| | |
|---|---|
| u | Numeric vector. |
| cc | Numeric constant, defaults to $\sqrt{(5/3)}$. |

## Details

$q_c(u) = 2/5 * c^5/(3 - c^2) * (1 - 2 * u)$

For further details see p. 297 f. in Eichner (2017) and/or Eichner & Stute (2013).

## Value

Vector of same length and mode as u.

## Note

$q_c(u)$ should be undefined for $c = \sqrt{3}$, but `qc` is here implemented to return `Inf * (1 - 2*u)` if `cc` contains R's value of `sqrt(3)`.

## Examples

```
u <- c(0, 1)   # seq(0, 1, by = 0.1)
c0 <- expression(sqrt(5/3))
c1 <- expression(sqrt(3) - 0.05)
cgrid <- seq(1.4, 1.6, by = 0.1)
cvals <- c(eval(c0), cgrid, eval(c1))

Y <- sapply(cvals, function(cc, u) qc(u, cc = cc), u = u)
cols <- rainbow(ncol(Y), end = 9/12)
matplot(u, Y, type = "l", lty = "solid", col = cols,
  ylab = expression(q[c](u)))
abline(h = 0, lty = "dashed")
legend("topright", title = "c", legend = c(c0, cgrid, c1),
  lty = 1, col = cols, cex = 0.8)
```

---

# rectangular *Rectangular kernel*

---

## Description

Vectorized evaluation of the rectangular kernel.

## Usage

```
rectangular(x, a = -0.5, b = 0.5)
```

## Arguments

| | |
|---|---|
| `x` | Numeric vector. |
| `a` | Numeric scalar: lower bound of kernel support; defaults to -0.5. |
| `b` | Numeric scalar: upper bound of kernel support; defaults to 0.5. |

## Value

A numeric vector of the rectangular kernel evaluated at the values in `x`.

## Examples

```
kader:::rectangular(x = seq(-1, 1, by = 0.1))

curve(kader:::rectangular(x), from = -1, to = 1)
```

---

`var_ES2012` *Variance Estimator of Eichner & Stute (2012)*

---

## Description

Variance estimator $Var_n(\sigma)$, vectorized in $\sigma$, on p. 2540 of Eichner & Stute (2012).

## Usage

```
var_ES2012(sigma, h, xXh, thetaXh, K, YmDiff2)
```

## Arguments

| | |
|---|---|
| `sigma` | Numeric vector $(\sigma_1, \ldots, \sigma_s)$ with $s \geq 1$ with values of the scale parameter $\sigma$. |
| `h` | Numeric scalar for bandwidth $h$ (as "contained" in `thetaXh` and `xXh`). |
| `xXh` | Numeric vector expecting the pre-computed h-scaled differences $(x - X_1)/h$, $\ldots$, $(x - X_n)/h$ where $x$ is the single (!) location for which the weights are to be computed, the $X_i$'s are the data values, and $h$ is the numeric bandwidth scalar. |
| `thetaXh` | Numeric vector expecting the pre-computed h-scaled differences $(\theta - X_1)/h$, $\ldots$, $(\theta - X_n)/h$ where $\theta$ is the numeric scalar location parameter, and the $X_i$'s and $h$ are as in `xXh`. |
| `K` | A kernel function (with vectorized in- & output) to be used for the estimator. |
| `YmDiff2` | Numeric vector of the pre-computed squared differences $(Y_1 - m_n(x))^2$, $\ldots$, $(Y_n - m_n(x))^2$. |

## Details

The formula can also be found in eq. (15.22) of Eichner (2017). Pre-computed $(x - X_i)/h$, $(\theta - X_i)/h$, and $(Y_i - m_n(x))^2$ are expected for efficiency reasons (and are currently prepared in function `kare`).

## Value

A numeric vector of the length of `sigma`.

## References

Eichner & Stute (2012) and Eichner (2017): see `kader`.

## See Also

kare which currently does the pre-computing.

## Examples

```
require(stats)

 # Regression function:
m <- function(x, x1 = 0, x2 = 8, a = 0.01, b = 0) {
 a * (x - x1) * (x - x2)^3 + b
}
 # Note: For a few details on m() see examples in ?nadwat.

n <- 100       # Sample size.
set.seed(42)   # To guarantee reproducibility.
X <- runif(n, min = -3, max = 15)      # X_1, ..., X_n   # Design.
Y <- m(X) + rnorm(length(X), sd = 5)   # Y_1, ..., Y_n   # Response.

h <- n^(-1/5)
Sigma <- seq(0.01, 10, length = 51)   # sigma-grid for minimization.
x0 <- 5   # Location at which the estimator of m should be computed.

mnX  <- nadwat(x = X, dataX = X, dataY = Y, K = dnorm, h = h) # m_n(X_i)
                                                       # for i = 1, ..., n.
 # Estimator of Var_x0(sigma) on the sigma-grid:
(Vn <- var_ES2012(sigma = Sigma, h = h, xXh = (x0 - X) / h,
  thetaXh = (mean(X) - X) / h, K = dnorm, YmDiff2 = (Y - mnX)^2))

## Not run:
 # Visualizing the estimator of Var_n(sigma) at x0 on the sigma-grid:
plot(Sigma, Vn, type = "o", xlab = expression(sigma), ylab = "",
  main = bquote(widehat("Var")[n](sigma)~~"at"~~x==.(x0)))

## End(Not run)
```

---

weights_ES2012 *Weights $W\_ni$ of Eichner & Stute (2012)*

---

## Description

Function, vectorized in its first argument sigma, to compute the "updated" weights $W_{ni}$ in eq. (2.1) of Eichner & Stute (2012) for the kernel adjusted regression estimator.

## Usage

```
weights_ES2012(sigma, xXh, thetaXh, K, h)
```

## Arguments

| | |
|---|---|
| sigma | Numeric vector $(\sigma_1, \ldots, \sigma_s)$ with $s \geq 1$ with values of the scale parameter $\sigma$. |
| xXh | Numeric vector expecting the pre-computed h-scaled differences $(x - X_1)/h$, $\ldots$, $(x - X_n)/h$ where $x$ is the single (!) location for which the weights are to be computed, the $X_i$'s are the data values, and $h$ is the numeric bandwidth scalar. |
| thetaXh | Numeric vector expecting the pre-computed h-scaled differences $(\theta - X_1)/h$, $\ldots$, $(\theta - X_n)/h$ where $\theta$ is the numeric scalar location parameter, and the $X_i$'s and $h$ are as in xXh. |
| K | A kernel function (with vectorized in- & output) to be used for the estimator. |
| h | Numeric scalar for bandwidth $h$ (as "contained" in thetaXh and xXh). |

## Details

Note that it is not immediately obvious that $W_{ni}$ in eq. (2.1) of Eichner & Stute (2012) is a function of $\sigma$. In fact, $W_{ni} = W_{ni}(x; h, \theta, \sigma)$ as can be seen on p. 2542 ibid. The computational version implemented here, however, is given in (15.19) of Eichner (2017). Pre-computed $(x - X_i)/h$ and $(\theta - X_i)/h$, $i = 1, \ldots, n$ are expected for efficiency reasons (and are currently prepared in function kare).

## Value

If length(sigma) > 1 a numeric matrix of the dimension length(sigma) by length(xXh) with elements $(W_{ni}(x; h, \theta, \sigma_r))$ for $r = 1, \ldots,$ length(sigma) and $i = 1, \ldots,$ length(xXh); otherwise a numeric vector of the same length as xXh.

## References

Eichner & Stute (2012) and Eichner (2017): see kader.

## See Also

bias_ES2012 and var_ES2012 which both call this function, and kare which currently does the pre-computing.

## Examples

```
require(stats)

 # Regression function:
m <- function(x, x1 = 0, x2 = 8, a = 0.01, b = 0) {
 a * (x - x1) * (x - x2)^3 + b
}
 # Note: For a few details on m() see examples in ?nadwat.

n <- 100       # Sample size.
set.seed(42)   # To guarantee reproducibility.
X <- runif(n, min = -3, max = 15)      # X_1, ..., X_n   # Design.
Y <- m(X) + rnorm(length(X), sd = 5)   # Y_1, ..., Y_n   # Response.
```

```
h <- n^(-1/5)
Sigma <- seq(0.01, 10, length = 51)   # sigma-grid for minimization.
x0 <- 5   # Location at which the estimator of m should be computed.

 # Weights (W_{ni}(x; \sigma_r))_{1<=r<=length(Sigma), 1<=i<=n} for
 # Var_n(sigma) and Bias_n(sigma) each at x0 on the sigma-grid:
weights_ES2012(sigma = Sigma, xXh = (x0 - X) / h,
  thetaXh = (mean(X) - X) / h, K = dnorm, h = h)
```

# Index