Package 'lokern'

December 7, 2023

Version 1.1-10.1

Date 2023-12-07

Title Kernel Regression Smoothing with Local or Global Plug-in Bandwidth

Author Eva Herrmann <eherrmann@mathematik.tu-darmstadt.de> (F77 & S original); Packaged for R and enhanced by Martin Maechler

Maintainer Martin Maechler <maechler@stat.math.ethz.ch>

Imports grDevices, graphics, stats, utils, sfsmisc (>= 1.0-12)

Description Kernel regression smoothing with adaptive local or global plug-in bandwidth selection.

BuildResaveData no

URL https://curves-etc.r-forge.r-project.org/,
https://r-forge.r-project.org/R/?group_id=846,
https://r-forge.r-project.org/scm/viewvc.php/pkg/lokern/?root=curves-etc,
svn://svn.r-forge.r-project.org/svnroot/curves-etc/pkg/lokern

BugReports https://r-forge.r-project.org/R/?group_id=846

License GPL (>= 2)

NeedsCompilation yes

Repository CRAN

Date/Publication 2023-12-07 16:00:02 UTC

R topics documented:

glkerns *Kernel Regression Smoothing with Adaptive Plug-in Bandwidth*

Description

Nonparametric estimation of regression functions and their derivatives with kernel regression estimators and automatically adapted (**gl**obal) plug-in bandwidth.

Usage

```
glkerns(x, ...)

## Default S3 method:
glkerns(x, y=NULL, deriv = 0, n.out = 300, x.out=NULL, x.inOut = TRUE,
        korder= deriv + 2, hetero=FALSE, is.rand=TRUE,
        inputb = is.numeric(bandwidth) && all(bandwidth > 0),
        m1 = 400, xl=NULL, xu=NULL,
        s=NULL, sig=NULL, bandwidth=NULL, trace.lev = 0, ...)

## S3 method for class 'formula'
glkerns(formula, data, subset, na.action, ...)
```

Arguments

x	vector of design points, not necessarily ordered.
y	vector of observations of the same length as x.
deriv	order of derivative of the regression function to be estimated. Only deriv=0,1,2 are allowed for automatic smoothing, whereas deriv=0,1,2,3,4 is possible when smoothing with a global input bandwidth. The default value is deriv=0.
n.out	number of output design points where the function has to be estimated; default is n.out=300.
x.out	vector of output design points where the function has to be estimated. The default is an equidistant grid of n.out points from min(x) to max(x).
x.inOut	logical or character string indicating if x.out should contain the input x values. Note that this argument did not exist, equivalently to being FALSE, up to **lokern** version 1.0-9. In order for residuals() or fitted() methods to be applicable, it must be TRUE or a character string specifying one of the methodss of seqXtend (package **sfsmisc**). The default, TRUE corresponds to method "aim".
korder	nonnegative integer giving the kernel order k; it defaults to korder = deriv+2 or $k = \nu + 2$ where $k - \nu$ must be even. The maximal possible values are for automatic smoothing, $k \leq 4$, whereas for smoothing with input bandwidth, $k \leq 6$.

hetero	logical: if TRUE, heteroscedastic error variables are assumed for variance estimation, if FALSE the variance estimation is optimized for homoscedasticity. Default value is hetero=FALSE.
is.rand	logical: if TRUE (default), random x are assumed and the s-array of the convolution estimator is computed as smoothed quantile estimators in order to adapt this variability. If FALSE, the s-array is choosen as mid-point sequences as the classical Gasser-Mueller estimator, this will be better for equidistant and fixed design.
inputb	logical: if true, a local input bandwidth array is used; if FALSE (by default when bandwidth is not specified), a data-adaptive local plug-in bandwidths array is calculated and used.
m1	integer, the number of grid points for integral approximation when estimating the plug-in bandwidth. The default, 400, may be increased if a very large number of observations are available.
xl, xu	numeric (scalars), the lower and upper bounds for integral approximation and variance estimation when estimating the plug-in bandwidth. By default (when xl and xu are not specified), the 87% middle part of $[xmin, xmax]$ is used.
s	s-array of the convolution kernel estimator. If it is not given by input it is calculated as midpoint-sequence of the ordered design points for is.rand=FALSE or as quantiles estimators of the design density for is.rand=TRUE.
sig	variance of the error variables. If it is not given by input or if hetero=TRUE it is calculated by a nonparametric variance estimator.
bandwidth	*global* bandwidth for kernel regression estimation. If it is not given by input or if inputb=FALSE a data-adaptive global plug-in bandwidth is used instead.
trace.lev	integer indicating how much the internal (Fortran level) computations should be "traced", i.e., be reported. The default, 0, does not print anything.
formula	a formula of the form y ~ pred, specifying the response variable y and predictor variable pred which must be in data.
data	an optional matrix or data frame (or similar: see model.frame) containing the variables in the formula formula. By default the variables are taken from environment(formula).
subset	an optional vector specifying a subset of observations to be used.
na.action	a function which indicates what should happen when the data contain NAs. Defaults to getOption("na.action").
...	for the formula method: Optional arguments all passed to glkerns.default().

Details

This function calls an efficient and fast algorithm for automatically adaptive nonparametric regression estimation with a kernel method.

Roughly spoken, the method performs a local averaging of the observations when estimating the regression function. Analogously, one can estimate derivatives of small order of the regression function. Crucial for the kernel regression estimation used here is the choice of a global bandwidth. Too small bandwidths will lead to a wiggly curve, too large ones will smooth away important

details. The function glkerns calculates an estimator of the regression function or derivatives of the regression function with an automatically chosen global plugin bandwidth. It is also possible to use global bandwidths which are specified by the user.

Main ideas of the plugin method are to estimate the optimal bandwidths by estimating the asymptotically optimal mean integrated squared error optimal bandwidths. Therefore, one has to estimate the variance for homoscedastic error variables and a functional of a smooth variance function for heteroscedastic error variables, respectively. Also, one has to estimate an integral functional of the squared k-th derivative of the regression function ($k =$ `korder`) for the global bandwidth.

Here, a further kernel estimator for this derivative is used with a bandwidth which is adapted iteratively to the regression function. A convolution form of the kernel estimator for the regression function and its derivatives is used. Thereby one can adapt the s-array for random design. Using this estimator leads to an asymptotically minimax efficient estimator for fixed and random design. Polynomial kernels and boundary kernels are used with a fast and stable updating algorithm for kernel regression estimation. More details can be found in the references and previously at Biostats, University of Zurich under '`software/kernel.html`', but no longer.

Value

an object of class(es) `c("glkerns", "KernS")`, which is a list including used parameters and estimator, containing among others

`x`	vector of ordered design points.
`y`	vector of observations ordered with respect to `x`.
`bandwidth`	bandwidth which was used for kernel regression estimation.
`x.out`	vector of ordered output design points.
`est`	vector of estimated regression function or its derivative (at `x.out`).
`sig`	variance estimation which was used for calculating the plug-in bandwidth
`deriv`	derivative of the regression function which was estimated.
`korder`	order of the kernel function which was used.
`xl`	lower bound for integral approximation and variance estimation.
`xu`	upper bound for integral approximation and variance estimation.
`s`	vector of midpoint values used for the convolution kernel regression estimator.

Author(s)

- Eva Herrmann, TU Darmstadt(1995-1997): principal code (origianl Fortran and S+), see the references.
- Martin Maechler, 2001 ff: translated to R, created the package, refactored '`src/`', added class, methods (predict, plot ..), arguments, docu, tweaks, help, examples, etc.
- The `formula` method was added in 2014 after proposals by Andri Signorell.

References

- global plug-in bandwidth estimator:
Theo Gasser, Alois Kneip & Walter Koehler (1991) A flexible and fast method for automatic smoothing. *Journal of the American Statistical Association* **86**, 643–652. doi:10.2307/2290393

Muller, H.-G. (1984) Smooth optimum kernel estimators of densities, regression curves and modes. *The Annals of Statistics* **12**, 766–774.

- variance estimation:
T. Gasser, L. Sroka & C. Jennen-Steinmetz (1986) Residual and residual pattern in nonlinear regression. *Biometrika* **73**, 625–633.

- adapting heteroscedasticity:
E. Herrmann (1997) Local bandwidth choice in kernel regression estimation. *Journal of Graphical and Computational Statistics* **6**, 35–54.

- fast algorithm for kernel regression estimator:
T. Gasser & A. Kneip (1989) discussion of Buja, A., Hastie, T. and Tibshirani, R.: Linear smoothers and additive models, *The Annals of Statistics* **17**, 532–535.

B. Seifert, M. Brockmann, J. Engel & T. Gasser (1994) Fast algorithms for nonparametric curve estimation. *J. Computational and Graphical Statistics* **3**, 192–213.

- on the special kernel estimator for random design point:
E. Herrmann (1996) *On the convolution type kernel regression estimator*; Preprint 1833, FB Mathematik, Technische Universitaet Darmstadt; currently available from https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.32.6383

See Also

`lokerns` for **lo**cal bandwidth computation. `plot.KernS` documents all the `methods` for `"KernS"` classed objects.

The `demo` for computing derivatives, `demo("glk-derivs")`.

Examples

```
data(xSim)## linear plus an exponential peak, see help(xSim)
n <- length(xSim)
tt <- ((1:n) - 1/2)/n # equidistant x ==> is.rand = FALSE
gk <- glkerns(tt, xSim, is.rand = FALSE)
gk # print method
plot(gk) # nice plot() method
if(require("sfsmisc")) {
  TA.plot(gk)
} else { plot(residuals(gk) ~ fitted(gk)); abline(h = 0, lty=2) }
qqnorm(residuals(gk), ylab = "residuals(gk)")

cat("glkerns() bandwidth:",format(gk$bandwidth, dig=10),"\n")
## local bandwidth: fit is very similar :
(lk <- lokerns(tt, xSim, is.rand = FALSE))
nobs(lk)

cols <- c(gl="PaleGreen", lo="Firebrick")
plot(lk$x.out, lk$bandwidth, axes = FALSE, xlab="", ylab="",
     ylim=c(0,max(lk$bandwidth)), type="h", col = "gray90")
axis(4); mtext("bandwidth(s)", side=4)
lines(lk$x.out, lk$bandwidth, col = cols["lo"], lty = 3)
abline(     h = gk$bandwidth, col = cols["gl"], lty = 4)
par(new=TRUE)
```

```
plot(tt, xSim, main = "global and local bandwidth kernel regression")
lines(gk$x.out, gk$est, col = cols["gl"], lwd = 1.5)
lines(lk$x.out, lk$est, col = cols["lo"])
# the red curve (local bw) is very slightly better
legend(0.7,4.4, c("global bw","local bw"), col = cols, lwd=1)

## This should look
op <- par(mfrow = c(3,1), mar = .1 + c(4,4,2,1), oma = c(0,0,3,0),
          mgp = c(1.5, 0.6,0))
plot(gk, main = expression(paste("Data & ", hat(f))))
## calling extra plot() method
gk1 <- glkerns(tt, xSim, deriv = 1, is.rand = FALSE)
plot(gk1$x.out, gk1$est, col = "green", lwd = 1.5, type = "l",
     main = expression(widehat(paste(f,"'"))))
abline(h=0, col="gray", lty = 3)
gk2 <- glkerns(tt, xSim, deriv = 2, is.rand = FALSE)
plot(gk2$x.out, gk2$est, col = "orange", lwd = 1.5, type = "l",
     main = expression(widehat(paste(f,"''"))))
abline(h=0, col="gray", lty = 3)
mtext("Example from www.unizh.ch/biostat/..../kernf77.html",side=3,
      outer = TRUE, cex = 1, font = par("font.main"))

par(op)
data(cars)
plot(dist ~ speed, data = cars,
     main = "Global Plug-In Bandwidth")
## these two are equivalent
m1glk <- glkerns(dist ~ speed, data = cars)
m.glk <- glkerns(cars$ speed, cars$ dist)
lines(m.glk, col=2) # using the lines() method
mtext(paste("bandwidth = ", format(m.glk$bandwidth, dig = 4)))
ii <- names(m1glk) != "call"
stopifnot(all.equal(m1glk[ii], m.glk[ii], tol = 1e-15))
```

KernS-methods *Methods for ("KernS" classed) Results of lokerns() and glkerns()*

Description

Methods for results of `glkerns()` and `lokerns()` which are of (S3) class `"KernS"`.

Usage

```
## S3 method for class 'KernS'
fitted(object, ...)
## S3 method for class 'KernS'
plot(x, type = "l", lwd = 2.5, col = 3, ...)
## S3 method for class 'KernS'
predict(object, x, deriv = object[["deriv"]],
        korder = deriv+2, trace.lev = 0, ...)
```

```
## S3 method for class 'KernS'
print(x, digits = getOption("digits"), ...)
## S3 method for class 'KernS'
residuals(object, ...)
```

Arguments

x,object	an R object, of S3 class "KernS", typically result either from glkerns() or lokerns().
type, lwd, col	arguments for plot() *only* for the case when x$deriv is *not* 0.
deriv	integer, ≥ 0, specifiying order of derivative that should be predicted.
korder	nonnegative integer giving the kernel order; see lokerns or glkerns.
digits	number of significant digits, see print.
trace.lev	integer; level of tracing of Fortran level computations; see lokerns.
...	potentially further arguments passed to and from methods. For the plot(*, deriv=0) method, these are passed to plotDS from package **sfsmisc**.

Details

Note that fitted() and residuals() rely on x.inOut having been true or x.out having contained the data x, in the lokerns or glkerns call.

The plot() method calls plotDS from package **sfsmisc**.

predict(object, x, deriv) when either some x are not in x.out or deriv is not 0, basically recalls the original lokerns or glkerns function (keeping the bandwidths for lokerns).

Value

(differing, depending on the generic function)

See Also

glkerns, lokerns.

Examples

```
## "interesting" artificial data:
set.seed(47)
x <- sort(round(10*runif(250),2))
fn <- function(x) 5 - x/2 + 3*exp(-(x-5)^2)
y <- fn(x) + rnorm(x)/4
plot(x,y)
## Tracing the phases in the Fortran code: trace=1 gives some, trace=3 gives *much*
lof <- lokerns(x,y, trace=2)
plot(lof)
plot(lof, cex = 1/4)# maybe preferable
plot(fn, 0, 10, add=TRUE, col=adjustcolor("gray40",1/2), lwd=2, lty=2)
## Simpler, using the lines() method:
plot(x,y); lines(lof, lwd=2, col=2)
```

```
qqnorm(residuals(lof)) # hmm... overfitting?
stopifnot(all.equal(y, fitted(lof) + residuals(lof), tolerance = 1e-15),
          predict(lof)$y == fitted(lof))
lof$iter # negative ?
tt <- seq(0, 10, by=1/32)
## again with 'tracing' [not for the average user]
p0 <- predict(lof, x=tt,          trace=1)
p1 <- predict(lof, x=tt, deriv=1, trace=1)
p2 <- predict(lof, x=tt, deriv=2)
plot(p2, type="l"); abline(h=0, lty=3) # not satisfactory, but lokerns(*,deriv=2) is
lof2 <- lokerns(x,y, deriv=2)
plot(lof2, ylim = c(-12,4), main=
   "lokerns(*, deriv=2) -- much more smooth than predict(*,deriv=2)")
mtext("as lokerns(*, deriv=2) chooses larger bandwidths[] !")
lines(predict(lof2, x=tt), col=adjustcolor("tomato", 1/3), lwd=5)
lines(p2, col="gray50"); abline(h=0, lty=3)
## add 2nd derivative of underlying fn():
f2 <- fn; body(f2) <- D(D(body(fn), "x"),"x")
lines(tt, f2(tt), col="blue")
```

`lokerns` *Kernel Regression Smoothing with Local Plug-in Bandwidth*

Description

Nonparametric estimation of regression functions and their derivatives with kernel regression estimators and automatically adapted **local** plug-in bandwidth function.

Usage

```
lokerns(x, ...)

## Default S3 method:
lokerns(x, y=NULL, deriv = 0, n.out=300, x.out=NULL, x.inOut = TRUE,
        korder = deriv + 2, hetero=FALSE, is.rand=TRUE,
        inputb = is.numeric(bandwidth) && all(bandwidth > 0),
        m1 = 400, xl=NULL, xu=NULL,
        s=NULL, sig=NULL, bandwidth=NULL, trace.lev = 0, ...)

## S3 method for class 'formula'
lokerns(formula, data, subset, na.action, ...)
```

Arguments

`x`	vector of design points, not necessarily ordered.
`y`	vector of observations of the same length as `x`.

`deriv` order of derivative of the regression function to be estimated. Only deriv=0,1,2 are allowed for automatic smoothing, whereas deriv=0,1,2,3,4 is possible when smoothing with an input bandwidth array. The default value is deriv=0.

`n.out` number of output design points where the function has to be estimated; default is `n.out=300`.

`x.out` vector of output design points where the function has to be estimated. The default is an equidistant grid of n.out points from min(x) to max(x).

`x.inOut` logical or character string indicating if `x.out` should contain the input `x` values. Note that this argument did not exist, equivalently to being `FALSE`, up to **lokern** version `1.0-9`.

In order for `residuals()` or `fitted()` methods to be applicable, it must be `TRUE` or a character string specifying one of the `methodss` of `seqXtend` (package **sfsmisc**). The default, `TRUE` corresponds to method `"aim"`.

`korder` nonnegative integer giving the kernel order k; it defaults to `korder = deriv+2` or $k = \nu + 2$ where $k - \nu$ must be even. The maximal possible values are for automatic smoothing, $k \leq 4$, whereas for smoothing with input bandwidth array, $k \leq 6$.

`hetero` logical: if TRUE, heteroscedastic error variables are assumed for variance estimation, if FALSE the variance estimation is optimized for homoscedasticity. Default value is hetero=FALSE.

`is.rand` logical: if `TRUE` (default), random x are assumed and the s-array of the convolution estimator is computed as smoothed quantile estimators in order to adapt this variability. If FALSE, the s-array is choosen as mid-point sequences as the classical Gasser-Mueller estimator, this will be better for equidistant and fixed design.

`inputb` logical: if true, a local input bandwidth array is used; if `FALSE` (by default when `bandwidth` is not specified), a data-adaptive local plug-in bandwidths array is calculated and used.

`m1` integer, the number of grid points for integral approximation when estimating the plug-in bandwidth. The default, 400, may be increased if a very large number of observations are available.

`xl, xu` numeric (scalars), the lower and upper bounds for integral approximation and variance estimation when estimating the plug-in bandwidth. By default (when `xl` and `xu` are not specified), the 87% middle part of $[xmin, xmax]$ is used.

`s` s-array of the convolution kernel estimator. If it is not given by input it is calculated as midpoint-sequence of the ordered design points for `is.rand=FALSE` or as quantiles estimators of the design density for `is.rand=TRUE`.

`sig` variance of the error variables. If it is not given by input or if `hetero=TRUE` it is calculated by a nonparametric variance estimator.

`bandwidth` *local* bandwidth array for kernel regression estimation. If it is not given by input or if `inputb=FALSE` a data-adaptive local plug-in bandwidth array is used instead.

`trace.lev` integer indicating how much the internal (Fortran level) computations should be "traced", i.e., be reported. The default, `0`, does not print anything.

formula	a formula of the form y ~ pred, specifying the response variable y and predictor variable pred which must be in data.
data	an optional matrix or data frame (or similar: see model.frame) containing the variables in the formula formula. By default the variables are taken from environment(formula).
subset	an optional vector specifying a subset of observations to be used.
na.action	a function which indicates what should happen when the data contain NAs. Defaults to getOption("na.action").
...	for the formula method: Optional arguments all passed to lokerns.default().

Details

This function calls an efficient and fast algorithm for automatically adaptive nonparametric regression estimation with a kernel method.

Roughly spoken, the method performs a local averaging of the observations when estimating the regression function. Analogously, one can estimate derivatives of small order of the regression function. Crucial for the kernel regression estimation used here is the choice the local bandwidth array. Too small bandwidths will lead to a wiggly curve, too large ones will smooth away important details. The function lokerns calculates an estimator of the regression function or derivatives of the regression function with an automatically chosen local plugin bandwidth function. It is also possible to use a local bandwidth array which are specified by the user.

Main ideas of the plugin method are to estimate the optimal bandwidths by estimating the asymptotically optimal mean squared error optimal bandwidths. Therefore, one has to estimate the variance for homoscedastic error variables and a functional of a smooth variance function for heteroscedastic error variables, respectively. Also, one has to estimate an integral functional of the squared k-th derivative of the regression function ($k =$ korder) for the global bandwidth and the squared k-th derivative itself for the local bandwidths.

Some more details are in glkerns.

Value

an object of class(es) c("lokerns", "KernS"), which is a list including used parameters and estimator, containing among others

x	vector of ordered design points.
y	vector of observations ordered with respect to x.
bandwidth	local bandwidth array which was used for kernel regression estimation.
x.out	vector of ordered output design points.
est	vector of estimated regression function or its derivative (at x.out).
sig	variance estimation which was used for calculating the plug-in bandwidths if hetero=TRUE (default) and either inputb=FALSE (default) or is.rand=TRUE (default).
deriv	derivative of the regression function which was estimated.
korder	order of the kernel function which was used.
xl	lower bound for integral approximation and variance estimation.

xu	upper bound for integral approximation and variance estimation.
s	vector of midpoint values used for the convolution kernel regression estimator.

References

All the references in glkerns.

See Also

glkerns for **gl**obal bandwidth computation. plot.KernS documents all the methods for "KernS" classed objects.

Examples

```
data(cars)
lofit <- lokerns(dist ~ speed, data=cars)
lofit # print() method

if(require("sfsmisc")) {
  TA.plot(lofit)
} else { plot(residuals(lofit) ~ fitted(lofit)); abline(h = 0, lty=2) }
qqnorm(residuals(lofit), ylab = "residuals(lofit)")

## nice simple plot of data + smooth
plot(lofit)

(sb <- summary(lofit$bandwidth))
op <- par(fg = "gray90", tcl = -0.2, mgp = c(3,.5,0))
plot(lofit$band, ylim=c(0,3*sb["Max."]), type="h",#col="gray90",
     ann = FALSE, axes = FALSE)

boxplot(lofit$bandwidth, add = TRUE, at = 304, boxwex = 8,
        col = "gray90",border="gray", pars = list(axes = FALSE))
axis(4, at = c(0,pretty(sb)), col.axis = "gray")
par(op)
par(new=TRUE)
plot(dist ~ speed, data = cars,
     main = "Local Plug-In Bandwidth Vector")
lines(lofit, col=4, lwd=2)
mtext(paste("bandwidth in [",
            paste(format(sb[c(1,6)], dig = 3),collapse=","),
            "];  Median b.w.=",formatC(sb["Median"])))

## using user-specified bandwidth array
myBW <- round(2*lofit$bandwidth, 2)
(lofB <- lokerns(dist ~ speed, data=cars, bandwidth = myBW)) # failed (for a while)
## can use deriv=3 (and 4) here:
lofB3 <- lokerns(dist ~ speed, data=cars, bandwidth = myBW, deriv=3)
plot(lofB)
lines(lofB3, col=3)
stopifnot(inherits(lofB3, "KernS"), identical(lofB3$korder, 5L))
```

varNPreg *Nonparametric Variance Estimator*

Description

Estimates the error variance σ^2 nonparametrically in the model

$$Y_i = m(x_i) + E_i,$$

where $E_i \sim (0, \sigma^2)$, i.i.d.

Computes leave-one-out residuals (local linear approximation followed by reweighting) and their variance.

Usage

```
varNPreg(x, y)
```

Arguments

x	abscissae values, ordered increasingly.
y	observations at y[i] at x[i].

Value

A list with components

res	numeric; residuals at x[] of length n.
snr	explained variance of the true curve
sigma2	estimation of residual variance, $\hat{\sigma}^2$.

Note

This is an R interface to the resest Fortran subroutine, used in lokerns and glkerns, see their help pages for references and context.

Earlier version of the **lokern** package accidentally, contained varest() which has been an identical copy of varNPreg().

Author(s)

Martin Maechler

See Also

lokerns, glkerns.

Examples

```
x <- sort(runif(100))
y <- sin(pi*x) + rnorm(100)/10
str(ve <- varNPreg(x,y))
```

---

xSim *Simulated Linear plus Exponential Peak*

Description

This is simulated data, a linear plus an exponential peak. In similar form, data like this appears in the smoothing literature since at least the eighties.

Usage

```
data(xSim)
```

Format

A vector of 75 numbers between -3.1323 and 4.4505, all rounded to 4 digits after the decimal.

Source

https://www.biostat.uzh.ch/en/research/software/kernel.html

See Also

The example in glkerns replicates the computations and plots from the source given.

Examples

```
data(xSim)
plot(xSim, main = "`xSim' - N=75 simulated linear + peak")
```

Index