Package ‘lokern’

February 20, 2015

Version 1.1-6
Date 2014-11-18
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 sfsmisc (>= 1.0-12)
Description Kernel regression smoothing with adaptive local or global plug-in bandwidth selection.
BuildResaveData no
License GPL (>= 2)
NeedsCompilation yes
Repository CRAN
Date/Publication 2014-11-18 15:41:50

R topics documented:

   glkerns ...................................................... 2
   KernS-methods .............................................. 6
   lokerns ..................................................... 7
   varNPreG ................................................... 10
   xSim ......................................................... 11

Index 13
Nonparametric estimation of regression functions and their derivatives with kernel regression estimators and automatically adapted (global) plug-in bandwidth.

Usage

```r
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) & bandwidth > 0,
ml = 400, xl=NULL, xu=NULL,
s=NULL, sig=NULL, bandwidth=NULL, trace.lev = 0)
```

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 methods 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 = ν + 2 where k − ν must be even. The maximal possible values are for automatic smoothing, k ≤ 4, whereas for smoothing with input bandwidth, k ≤ 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.

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 on http://www.biostat.uzh.ch/research/software/kernel.html

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.

References

- global plug-in bandwidth estimator:


- variance estimation:

- adapting heteroscedasticity:

- fast algorithm for kernel regression estimator:


- on the special kernel estimator for random design point:

See Also

`lokerns` for local bandwidth computation. `plot.KernS` documents all the methods for "KernS" classed objects.
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 )
} qnorm(residuals(gk), ylab = "residuals(gk)"

local bandwidth: fit is very similar :
(lk <- lokerns(tt, xSim, is.rand = FALSE))

nobs(lk)

cols <- c("PaleGreen", "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$bandwidth, col = cols["lo"], lty = 3)
abline(h = lk$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"], lty = 3)

# 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(1,3,1))
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")
m.glk <- glkerns(cars$speed, cars$dist)
lines(m.glk$x.out, m.glk$est, col=2)
Methods for results of `glkerns()` and `lokerns()` which are of (S3) class "KernS".

**Usage**

```r
## 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, $\geq 0$, specifying 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

gl kerns, lokerns.

Examples

```r
## "interesting" artificial data:
set.seed(47)
x <- sort(round(10*runif(250),2))
fX <- 5 - x/2 + 3*exp(-(x-5)^2)
y <- fX + rnorm(fX)/4
plot(x,y)
lof <- lokerns(x,y)
plot(lof)
plot(lof, cex = 1/4)# maybe preferable
## 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))
tt <- seq(0, 10, by=1/32)
p0 <- predict(lof, x=tt)
p1 <- predict(lof, x=tt, deriv=1)
p2 <- predict(lof, x=tt, deriv=2)
plot(p2, type="l"); abline(h=0, lty=3) # not satisfactory:
lof2 <- lokerns(x,y, deriv=2)
plot(lof2, main="lokerns(*, deriv=2) -- much more smooth than predict(*,deriv=2)")
lines(p2); abline(h=0, lty=3)
```

lokerns

Kernel Regression Smoothing with Local Plug-in Bandwidth

Description

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

Usage

```r
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) && bandwidth > 0,
m1 = 400, x1=NULL, xu=NULL,
s=NULL, sig=NULL, bandwidth=NULL, trace.lev = 0)
```
Arguments

\( x \) vector of design points, not necessarily ordered.
\( y \) vector of observations of the same length as \( x \).
\( \text{deriv} \) order of derivative of the regression function to be estimated. Only \( \text{deriv}=0,1,2 \) are allowed for automatic smoothing, whereas \( \text{deriv}=0,1,2,3,4 \) is possible when smoothing with an input bandwidth array. The default value is \( \text{deriv}=0 \).
\( \text{n.out} \) number of output design points where the function has to be estimated; default is \( \text{n.out}=300 \).
\( \text{x.out} \) vector of output design points where the function has to be estimated. The default is an equidistant grid of \( \text{n.out} \) points from \( \min(x) \) to \( \max(x) \).
\( \text{x.inOut} \) logical or character string indicating if \( \text{x.out} \) should contain the input \( x \) values. Note that this argument did not exist, equivalently to being \( \text{FALSE} \), up to \text{lokern} version 1.0-9.

In order for \( \text{residuals}() \) or \( \text{fitted}() \) methods to be applicable, it must be \( \text{TRUE} \) or a character string specifying one of the methods of \text{seqxtend} (package \text{sfsmisc} ). The default, \( \text{TRUE} \) corresponds to method "aim".

\( \text{korder} \) nonnegative integer giving the kernel order; it defaults to \( \text{korder} = \text{deriv}+2 \) 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, \( k \leq 6 \).
\( \text{hetero} \) logical: if \( \text{TRUE} \), heteroscedastic error variables are assumed for variance estimation, if \( \text{FALSE} \) the variance estimation is optimized for homoscedasticity. Default value is \( \text{hetero}={\text{FALSE}} \).
\( \text{is.rand} \) logical: if \( \text{TRUE} \) (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 \( \text{FALSE} \), the \( s \)-array is choosen as mid-point sequences as the classical Gasser-Mueller estimator, this will be better for equidistant and fixed design.
\( \text{inputb} \) logical: if \( \text{true} \), a local input bandwidth array is used; if \( \text{FALSE} \) (by default when \( \text{bandwidth} \) is not specified), a data-adaptive local plug-in bandwidths array is calculated and used.
\( \text{m1} \) 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.
\( \text{xL}, \text{xU} \) numeric (scalars), the lower and upper bounds for integral approximation and variance estimation when estimating the plug-in bandwidth. By default (when \( \text{xL} \) and \( \text{xU} \) are not specified), the 87% middle part of \( [\text{min}, \text{max}] \) is used.
\( \text{s} \) \( s \)-array of the convolution kernel estimator. If it is not given by input it is calculated as mid-point-sequence of the ordered design points for \( \text{is.rand}={\text{FALSE}} \) or as quantiles estimators of the design density for \( \text{is.rand}={\text{TRUE}} \).
\( \text{sig} \) variance of the error variables. If it is not given by input or if \( \text{hetero}={\text{TRUE}} \) it is calculated by a nonparametric variance estimator.
\( \text{bandwidth} \) \text{local} bandwidth array for kernel regression estimation. If it is not given by input or if \( \text{inputb}={\text{FALSE}} \) 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.

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 \( (k = \text{korder}) \) 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 \( x \).
\[ \text{bandwidth} \] local bandwidth array which was used for kernel regression estimation.
\[ x.out \] vector of ordered output design points.
\[ \text{est} \] vector of estimated regression function or its derivative (at \( x.out \)).
\[ \text{sig} \] variance estimation which was used for calculating the plug-in bandwidths if \( \text{hetero}=\text{TRUE} \) (default) and either \( \text{inputb}=\text{FALSE} \) (default) or \( \text{is.rand}=\text{TRUE} \) (default).
\[ \text{deriv} \] derivative of the regression function which was estimated.
\[ \text{korder} \] order of the kernel function which was used.
\[ x1 \] 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 global bandwidth computation. `plot.KernS` documents all the methods for "KernS" classed objects.

Examples

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

---

**varNpreg**

Nonparametric Variance Estimator

**Description**

Estimates the error variance $\sigma^2$ 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, $\sigma^2$.

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**


**See Also**

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

**Examples**

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

*Topic datasets
  xSim, 11
*Topic nonparametric
  varNPreg, 10
*Topic regression
  varNPreg, 10
*Topic smooth
  glkerns, 2
  lokerns, 7
*Topic utilities
  KernS-methods, 6
fitted, 2, 8
fitted.KernS (KernS-methods), 6

glkerns, 2, 6, 7, 9–12
KernS-methods, 6
lines.KernS (KernS-methods), 6
lokerns, 4, 6, 7, 7, 11
plot.KernS, 4, 10
plot.KernS (KernS-methods), 6
plotDS, 6
predict.KernS (KernS-methods), 6
print, 6
print.KernS (KernS-methods), 6

residuals, 2, 8
residuals.KernS (KernS-methods), 6
seqXtend, 2, 8

varest (varNPreg), 10
varNPreg, 10

xSim, 11